

**Material Didáctico de Java
Lenguaje de Programación II**

**Carlos Alberto Vanegas
Ingeniero de Sistemas
Especialista en Ingeniería de Software
Maestría en Ingeniería de Sistemas
Profesor Universidad Distrital Francisco José de Caldas**

**Universidad Distrital Francisco José de Caldas
Facultad Tecnológica - Sistematización de Datos
Bogotá, Julio / 2005**

PREFACIO

Capitulos del Libro

En este manual encontraras 14 capitulos distribuidos de la siguiente forma:

1. **Capitulo 1 “Lenguaje de programación Java”**: Habla sobre los conceptos básicos del lenguaje de programación Java, sus palabras reservadas, tipos de datos, operadores y la compilación y ejecución de un programa en Java.
2. **Capitulo 2 “estructuras de control”**: maneja los conceptos propios de las estructuras de decisión *if*, *else*, *switch- case*, y de las estructuras repetitivas *while*, *for*, *do-while*, así como una serie de ejemplos que permiten reforzar los conceptos y al final del capitulo encontrará unos ejercicios para resolver.
3. **Capitulo 3 “Arreglos”**: Se habla sobre la estructuración de datos en arreglos con elementos de un mismo tipo; aquí también se presentan ejemplos aplicados al tema, como también ejercicios para practicar.
4. **Capitulo 4 “Métodos”**: Aquí se habla sobre los diferentes métodos propios de java, así como también la creación de métodos creados por el usuario, representado por medio de unos ejemplos prácticos.
5. **Capitulo 5 “Cadenas y caracteres”**: en este capitulo encontramos algunas de las funciones de Java que permiten el manejo de cadenas y caracteres, explicados a través de ejemplos.
6. **Capitulo 6 “Interfaz Grafica de Usuario”**: Se habla sobre algunos objetos que permiten mejorar la interfaz de un programa presentados por medio de un ejemplo para cada objeto.
7. **Capitulo 7 “Gráficos en Java”**: Hablamos sobre los objetos gráficos: el dibujo de caracteres, cadenas, el control de color y letra y el dibujo de líneas, rectángulos, ovalos, arcos y polígonos.
8. **Capitulo 8 “Manejo de Eventos (Mouse y Teclado)”**: Se explica algunos eventos para el manejo del mouse y el teclado especificado por medio de ejemplos.
9. **Capitulo 9 “Programación orientada a Objetos con Java”**: Se manejan los conceptos de programación Orientada a Objetos utilizando el lenguaje de programación Java en cuanto a: constructores, sobrecarga de métodos, herencia, polimorfismo, clases abstractas e interfaces.
10. **Capitulo 10 “Manejo de Excepciones”**: nos introducimos al manejo de las excepciones que se pueden generar en un programa por posibles errores del sistema.
11. **Capitulo 11 “Hilos y animación en Java”**: en este capitulo abordamos el tema de la animación en Java, utilizando el concepto de hilos.
12. **Capitulo 12 “Manejo de Datos (Archivos) a través de los flujos de Java”**: Se maneja el concepto de flujo de datos, utilizando los conceptos de archivos secuenciales y aleatorios.
13. **Capitulo 13 “Trabajo con Redes”**: En este capitulo se habla sobre la comunicación de programas a través de las redes de computadores, manejando las conexión de sockets de flujo y datagramas.
14. **Capitulo 14 “Bases de Datos con Java (JDBC)”**: Se habla del soporte que tiene java para la manipulación de Bases de Datos.

TABLA DE CONTENIDO

Introducción	6
1. Lenguaje de Programación Java	7
1.1 Consideraciones importantes de Java	7
1.2 Entorno de Java	8
1.3 Palabras reservadas de Java	8
1.4 Tipos de variables	9
1.5 Operadores	9
1.5.1 Aritméticos	9
1.5.2 De igualdad y relacionales	9
1.6 Creación de un primer programa en Java	10
1.6.1 Programa Autónomo	10
1.6.2 Subprograma Applet	11
2. Estructuras de Control	13
2.1 Estructura de alternativa simple if	13
2.2 Estructura de alternativa compuesta if - else	15
2.3 Estructura de alternativa múltiple switch- case	16
2.4 Estructura de repetición for	19
2.4 Estructura de repetición while	20
2.5 Estructura de repetición do - while	21
2.6 Estructuras de control anidadas	22
2.7 Ejercicios	24
3. Arreglos	25
3.1 Arreglos Unidimensionales	25
3.2 Arreglos con múltiples subíndices (Matrices)	27
3.3 Ejercicios	28
4. Métodos	29
4.1 Métodos que se ejecutan automáticamente	29
4.2 Métodos predefinidos de la clase Math	29
4.3 Métodos de usuario que no reciben parámetros, ni retornan valores	30
4.4 Métodos de usuario que reciben parámetros pero no retornan valores	31
4.5 Métodos de usuario que reciben parámetros y retornan valores	32
4.6 Métodos recursivos	34
4.7 Paso de Arreglos a Métodos	35
4.8 Ejercicios	36
5. Cadenas y caracteres	38
5.1 Clase String	38
5.2 Clase StringTokenizer	42
5.3 Ejercicios	43
6. Interfaz Gráfica de Usuario	44
6.1 Label (etiqueta)	44
6.2 Button (boton)	44
6.3 TextField (campo de texto)	44

6.4 TextArea (area de texto)	45
6.5 Choice (cuadro combinado)	45
6.6 Checkbox (casillas de verificación)	45
6.7 CheckboxGroup (Grupo de Botones)	46
6.8 List (Lista)	46
6.9 ScrollBar (Barra de desplazamiento)	46
6.10 Administrador de Diseños	46
6.10.1 FlowLayout	47
6.10.2 BorderLayout	48
6.10.3 GridLayout	49
6.11 Ejemplos de Aplicación	50
6.12 Ejercicios	56
7. Gráficos en Java	57
7.1 Graficando cadenas de caracteres, caracteres y bytes	57
7.2 Colores en Java	58
7.3 Los tipos de Fuente en Java	58
7.4 Dibujo de figuras en Java	61
7.4.1 Polígonos	64
7.5 Marcos (Frame)	65
7.6 Menús	69
7.7 Cuadros de Diálogo	72
7.8 Ejercicios	73
8. Manejo de eventos (Mouse y Teclado)	74
8.1 Eventos de Mouse	74
8.2 Eventos del Teclado	77
8.3 Ejercicios	80
9. Programación Orientada a Objetos con Java	81
9.1 Modificadores de control de acceso	82
9.2 Constructores	83
9.2.1 Constructores sin parámetros	83
9.2.2 Constructores con parámetros	84
9.3 Empleo de la referencia this	86
9.4 Sobrecarga de métodos	86
9.5 Sobrecarga de constructores	87
9.6 Herencia y poliformismo	88
9.7 Clases abstractas	91
9.8 Interfaces	92
9.9 Ejercicios	94
10. Manejo de Excepciones	96
10.1 Conceptos	96
10.2 Ejercicios	99
11. Hilos y Animación en Java	100
11.1. Hilos (Thread)	100
11.2. Hilos con Interfaz Runnable	100
11.3. Animación	101

11.4 Ejercicios	107
12. Manejo de Datos (archivos) a través de los flujos de Java.....	108
12.1. Archivos y Flujos.....	108
12.1.1 Creación de un archivo de acceso secuencial.....	108
12.1.2 Lectura de un archivo secuencial	111
12.2 Archivos de acceso aleatorio	115
12.2.1 Creación de un archivo de acceso aleatorio.....	116
12.2.2 Leer datos de un archivo aleatorio.....	120
12.3 Ejercicios	122
13. Trabajo con redes	124
13.1 Manipulación de URL	124
13.2 Conexiones de Sockets de flujos (cliente /servidor).....	125
13.3 Utilización de Socket de datagrama (Servidor – Cliente).....	132
13.4 Ejercicios	136
14. Bases de datos con Java (JDBC)	137
14.1 Que es una Base de Datos	138
14.2 Tipos de Bases de Datos (BD)	138
14.2.1 Relacionales	138
14.2.2 Enfoque Orientado a Objetos	138
14.3 Lenguaje de consulta estructurado (S.Q.L.)	139
14.3.1 Comandos.....	139
14.3.2 Cláusulas	139
14.3.3 Operadores Lógicos	140
14.3.4 Operadores de Comparación	140
14.3.5 Funciones de Agregado	141
14.4 Acceder a una Base de Datos.....	141
14.4.1 Establecer una Conexión.....	141
14.4.2 Seleccionar Tablas.....	142
14.5 Ejercicios	149
Bibliografía.....	150
Infografía	151

Introducción

Java es un lenguaje de programación que cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. Una de las principales características de Java es que es independiente de la plataforma. Esto quiere decir que si se hace un programa en Java puede funcionar en cualquier computador, esto es una ventaja significativa para programadores pues antes tenían que hacer un programa para cada sistema operativo. El lenguaje de programación Java, logra esa independencia gracias a su máquina virtual que hace de puente entre el sistema operativo y el programa de Java y posibilita que este último se entienda perfectamente.

La Máquina Virtual de Java es un módulo de software que funciona como interprete (programa que ejecuta cada una de las instrucciones y declaraciones que encuentra conforme va analizando el programa que le ha sido dado de entrada sin producir un programa objeto o ejecutable) de la representación binaria intermedia. La MVJ examina cada uno de los códigos binarios (bytecodes) que se han compilado y los ejecuta en el computador donde reside el interprete. El código Java puede ser ejecutado en cualquier plataforma que disponga de la máquina virtual de Java. Eso quiere decir, que un computador necesita solamente de la MVJ para ejecutar cualquier programa hecho bajo este lenguaje.

Java presenta muchas características que lo diferencian de lenguajes similares como C++, empezando por las posibilidades de ejecución.

Básicamente un programa en Java puede ejecutarse como:

- Stand Alone: Aplicación independiente.
- Applet: Una aplicación especial que se ejecuta en el navegador del cliente.
- Aplicación: programa que se guarda y se ejecuta en el equipo local del usuario.
- Servlet: Una aplicación especial sin Interfaz que se ejecuta en un servidor.

1. Lenguaje de Programación Java

Inicialmente para trabajar con el lenguaje de programación Java es necesario tener el Kit de desarrollo Java (JDK 1.2 en adelante) que se distribuye gratuitamente en la dirección de Internet www.javasoft.com ó www.java.sun.com de Sun Microsystems. Este kit incluye unas clases predefinidas, es decir, el API (Application Programming Interface). de Java. También incluye el compilador de Java y el JRE (Java Runtime Environment).

El JRE incluye los elementos necesarios para hacer funcionar programas Java en nuestro computador. Principalmente nos instala la Máquina Virtual de Java y los plugins (componentes) necesarios para el/los navegador/es instalado(s) en nuestro sistema. El JRE también se puede descargar independientemente del entorno de desarrollo, pues se distribuye en un paquete llamado J2RE.

En la siguiente tabla se enumeran los programas que se encuentran en el JDK:

Producto	Propósito
java	Interpretador que se usa para ejecutar programas autónomos
javac	Compilador Java
javadoc	Generador de documentos Java
javah	Generador de archivos C que crea encabezados y archivos fuente para definición de clases
javap	Desensamblador de clases Java
jdb	Depurador de Java

Los programas en Java contienen *clases* y *métodos*. Podemos crear nuestras propias clases y métodos para crear un programa en Java, pero la mayoría de los programadores aprovechan las clases y los métodos que existen en las bibliotecas de las clases de Java. Estas Bibliotecas de clases las podemos encontrar en la dirección de Internet www.java.sun.com, en la sección API & Language Documentation (Java 2 Platform API Specification (NO FRAMES)).

1.1 Consideraciones importantes de Java

Algunas de las consideraciones más importantes que debemos tener en cuenta para la realización de programas en Java son:

- Todos los programas inician con la palabra reservada *class*.
`public class nombre_clase.....`
- El nombre de la clase debe iniciar con Mayúscula (Estandarización de la codificación).
`public class Primera....`
- El nombre del archivo del código fuente debe ser igual al nombre de la clase y terminan con la extensión *.java* (para Java es diferente la letra a de la letra A).
- Los nombre de los métodos se deben iniciar con minúscula (Estandarización de la codificación).
`public void mi_Metodo(parámetros)`
- Toda instrucción termina con punto y coma (" ; ").
`System.out.println("Hola...");`

1.2 Entorno de Java

Los sistemas Java generalmente consta de varias partes: un entorno, el lenguaje, la interfaz de programación de aplicaciones (API, Applications Programming Interface) de Java y diversas bibliotecas de clases.

Los programas de Java pasan por 5 fases antes de ejecutarse, estas son:

- **Fase 1:** Editar el Archivo: escritura del código en un editor de texto almacenando es un dispositivo de almacenamiento como un disco. Los programas en Java termina con la extensión ".java", Ejemplo: Primero.java.
- **Fase 2:** El programador ejecuta el comando *javac* para compilar el programa. El compilador de Java traduce el programa Java a códigos de Bytes, que es el lenguaje que entiende el interpretador de Java. Si el programa se compila correctamente se producirá un archivo cuya extensión será ".class", ejemplo : *javac Primero.java*.
- **Fase 3:** Antes de que un programa pueda ejecutarse, es necesario colocarlo en la memoria. Esto lo hace el cargador de clases que toma el archivo ".class" que contiene los códigos de Bytes y los transfiere a la memoria. El cargador de clases comienza a cargar el archivos ".class" en dos situaciones:
 - *java Primero* :Invoca el interprete *java* para el programa Primero y hace que el cargador de clases cargue la información del programa (llamada aplicación) o programa autónomo.
 - *appletviewer Primero.html* : cuando el programa es un Applet de Java, este se carga en un navegador de la World Wide Web. Se requiere un ".html" para que *appletviewer* pueda invocar a un Applet.
- **Fase 4:** Antes de que se ejecute el interprete o el *appletviewer*, que ejecuta los códigos de bytes, estos son verificados por el verificador de códigos de bytes . Esto asegura que los códigos de bytes son válidos y no violan las restricciones de seguridad de Java.
- **Fase 5:** El computador controlado por su CPU, interpreta el programa, un código de bytes a la vez.

1.3 Palabras reservadas de Java

Las palabras reservadas de Java son aquellas que pertenecen al programa en si y no pueden declararse como nombre de variables, métodos o clases. Todas las palabras reservadas de Java son en minúscula.

abstract	boolean	break	byte	case	cast
catch	char	class	cons	continue	default
do	double	else	extends	final	finally
float	for	future	generic	goto	if
implements	import	inner	instanceof	in	interface
long	native	new	null	operator	outer
package	private	protected	public	rest	return
short	static	super	switch	synchronized	this
throw	throws	transient	Try	var	unsigned
virtual	void	volatile	while		

1.4 Tipos de variables

Un tipo variable especifica la clase de valor (numérico, alfanumérico, lógica) que puede almacenar, así como las operaciones que se pueden realizar con ella. Para asignarle un valor a una variable se utiliza el operador de asignación de java "igual" (=).

Tipo	Rango que puede almacenar
boolean	Verdadero ó Falso
byte	Valores en el rango de -128 a 127
char	Alfanuméricos(letras, números, símbolos)
double	Valores en el rango de -1.7×10^{-308} a 1.7×10^{308}
float	Valores en el rango de -3.4×10^{-38} a 3.4×10^{38}
int	Valores en el rango de -32.768 a 32767
long	Valores en el rango de -2.147.483.648 a 2.147.483.647

1.5 Operadores

1.5.1 Aritméticos

En algunas ocasiones dentro de un proceso de un programa es necesario realizar operaciones aritméticas con los valores que contienen las variables, Java cuenta con los siguientes operadores:

Operadores de Java	Operador Aritmético	Expresión en Java
Suma	+	F+7
Resta	-	F-7
Multiplicación	*	B*m
División	/	X/y
Residuo	%	R%s

1.5.2 De igualdad y relacionales

Operadores de Igualdad	En Java	Ej. En Java	Significado en Java
=	==	x==y	X es igual a Y
#	!=	x!=y	X es diferente de Y
Operadores Relacionales			
>	>	y>x	Y es mayor que X
>=	>=	y>=x	Y es mayor o igual que X
<	<	y<x	Y es menor X
<=	<=	y<=x	Y es menor o igual que X

1.6 Creación de un primer programa en Java

Java es un lenguaje de programación con el que pueden crearse aplicaciones autónomas y applets (basados en un explorador). Los programas autónomos son parecidos a aquellos que pueden realizarse con C++. Los Applets son independientes del dispositivo, es decir un mismo Applet puede ejecutarse desde un PC con Windows/x, Mac o Linux, ya que el compilador de Java genera código de máquina virtual el cual se puede ejecutar desde cualquier plataforma. Los programas autónomos o applets se pueden digitar en un editor de texto como: Wordpad, Edit, Block de Notas, etc. Estos deben tener la extensión ".java".

Listo.....empecemos a jugar con Java; realizaremos un primer programa que imprima en pantalla el siguiente mensaje: " Bienvenido a la programación en Java". Dicho programa debe realizarse como un programa Autónomo y un Applet.

1.6.1 Programa Autónomo

- **Programa Fuente**

```
1      class PrimeroAuto{
2          public static void main(String args[])
3              {
4              System.out.println("Bienvenido a la programación en Java");
5          }
6      }
```

Línea 1: Se define la clase con el nombre "PrimeroAuto"

Línea 2: Se declara el método principal main (todo programa autónomo debe contener el método principal main)

Línea 3: Se crean las instrucciones de lo que va a realizar el programa. En este programa utilizamos la clase System y su método "out" que permite realizar la impresión de un texto en la pantalla por medio de println, terminando con punto y coma la instrucción.

Es necesario guardar el archivo con el nombre definido en la clase, en este caso Primero y con extensión ".java" (se debe tener en cuenta como está escrito el nombre de la clase porque para el compilador de Java "PrimeroAuto" es diferente de "primeroAuto").

- **Compilación y ejecución del programa autónomo**

- Desde el prompt de la interfaz de comandos debe digitar:
javac <nombre del programa con extensión>< <enter>
javac PrimeroAuto.java <enter>
- Si el programa no genera ningún error de compilación este creará un archivo del mismo nombre pero con extensión .class
(PrimeroAuto.class)
- Para ejecutar el programa debemos digitar:
java <nombre del programa> <enter>
java PrimeroAuto <enter>
- Al ejecutarse en la pantalla se imprimirá el siguiente mensaje:
" Bienvenido a la programación en Java "

1.6.2 Subprograma Applet

- **Programa Fuente**

```
1. import java.applet.Applet; //para poderlo ejecutar en un explorador
2. import java.awt.Graphics; //maneja la interfaz gráfica
3. public class PrimeroApp extends Applet{
4.     public void paint (Graphics g)
5.     {
        g.drawString("Bienvenido a la programación en Java",25,25);
    }
}
```

Línea 1 y 2: se importan las clases definidas en Java en el archivo especial llamado librería de clases o paquete (Class library o packet) ubicados en la carpeta java\lib.

Línea 3: indica al compilador que se está creando un Applet llamado Primero que hereda de la clase Applet. La palabra reservada *public* al comienzo de las declaraciones permiten al navegador correr el Applet. Si se omite este no puede correr el Applet.

Línea 4: se crea un método llamado *paint* (pintar), el cual se ejecuta automáticamente, con una lista de parámetros que sirven para que el método reciba la información que necesita para que realice las tareas especificadas.

Línea 5: utiliza el método *drawString* de la clase *Graphics* instanciando un objeto *g*, para dibujar una cadena de caracteres contenidas entre comillas, como también las coordenadas donde aparecerá en pantalla en este caso 25,25, terminando la instrucción con punto y coma (;).

- **Compilación y ejecución del subprograma Applet**

- Desde el prompt de la interfaz de comandos debemos digitar:
javac <nombre del programa con extensión> <enter>
javac PrimeroApp.java <enter>
- Si el programa no genera ningún error de compilación este creará un archivo del mismo nombre pero con extensión .class
(PrimeroApp.class)
- Luego debemos crear un archivo del mismo nombre pero con extensión html (PrimeroApp.html) desde cualquier editor de texto, con las siguientes instrucciones:
<html>
<Applet code="<nombre del programa con extensión .class"> width=300
height=200></applet>
</html>

Donde height y width indican el tamaño de la ventana en pixeles.
En nuestro ejemplo sería:

```
<html>
<Applet code=" PrimeroApp.class" width=300 height=200></applet>
</html>
```

- luego de creado el programa html, para ejecutarlo debemos digitar:
appletviewer <nombre del programa con extensión html> <enter>
appletviewer Primero.html <enter>

- Otra forma de ejecutar el Applet, es abrir el archivo ".html" desde un navegador de Internet.
- En la pantalla se imprimirá el siguiente mensaje:
" Bienvenido a la programación en Java "

2. Estructuras de Control

2.1 Estructura de alternativa simple if

La estructura de alternativa simple **if** ejecuta una determinada acción cuando se cumple una condición. La selección **if** evalúa la condición y:

- ♦ Si la condición es verdadera, entonces ejecuta la acción **if** (o acciones caso de ser S1 una acción compuesta y constar de varias acciones) y continua con el resto del programa.
- ♦ Si la condición es falsa, entonces no entra al **if**, pero continua con el resto del programa.

```
if (condición verdadera)
    { S1 }
Resto del programa
```

Ejemplo: crear un programa autónomo y un applet que capture un numero, si el numero es igual a 5, deberá imprimir "Entre al if".

1) Programa Applet

```
1  import java.awt.*;
2  import java.applet.Applet;
3  public class IfSimple extends Applet{
4  Label solicita; // objeto que permite imprimir texto en pantalla
5  TextField entrada; //objeto que permite capturar datos desde teclado
6  int numero; // variable para almacenar valores enteros
   //método para preparar componentes de interfaz gráfica de usuario e inicializar
   variables.
7  public void init()
   {
8      solicita= new Label("Teclee un entero y Enter");
        //crear un objeto de tipo Label
9      entrada=new TextField(10); // crear un objeto de tipo TextField
10     add(solicita); // colocar el objeto solicita en el Applet
11     add(entrada); //colocar el objeto entrada en el Applet
   }
   //método que permite la impresión de texto en la pantalla
12  public void paint (Graphics g)
   {
13     if(numero==5)
14         g.drawString("Entre al IF",100,90); // si la cond es verdadera
   }

   //método que ejecuta la acción que ha realizado un usuario con el teclado.
15  public boolean action(Event e,Object o)
16  {
        // realizar una conversión de tipo de dato String a un tipo de dato int.
17     numero=Integer.parseInt(entrada.getText());
        // llamar nuevamente al método paint
18     repaint();
19     return true; // indica que la acción del usuario se proceso
   }
}
```

Línea 4: La clase *Label* permite crear un objeto donde se puede mostrar un texto no editable.

Línea 5: La clase *TextField* permite crear un objeto donde un usuario introduce datos mediante teclado. También puede mostrar información.

Línea 7: el método *init()* que se ejecuta automáticamente y se utiliza para inicializar los objetos asignándoles espacio en memoria.

Línea 8 y 9: se asigna espacio de memoria con la palabra reservada *new* a cada objeto.

Línea 10 y 11: adiciona los objetos en el Applet.

Línea 15: el método *action* se ejecuta automáticamente cuando se realiza una acción con el teclado. Al final del método siempre debe existir un *return* que finalice dicho proceso por ser un método que retorna un tipo de datos.

Línea 17: el método *parseInt()* de la clase *Integer*, permite convertir un *string* en un numero entero.

Línea 18: método que llama nuevamente a *paint()*.

2) programa Autónomo

//paquete que permite el manejo entrada/salida

```
1  import java.io.*;
2  public class IfSimpleAuto2
3  {
4      public static void main(String args[])
5      {
6          //try -catch permite el manejo de excepciones y errores para programas autónomos
7          try{
8              //Clase que me permite crear un objeto para que capture datos desde el teclado
9              BufferedReader linea = new BufferedReader(
10                  new InputStreamReader( System.in ) );
11              System.out.println("Digite un numero:");
12              // creación de un objeto de tipo String
13              String lin = linea.readLine();
14              //con equals se compara dos Objetos de tipo String
15              if (lin.equals("5"))
16                  System.out.println("Entre al If");
17              System.out.println("Es número digitado fue:"+lin);
18              }catch(IOException e) {
19                  System.out.println("Error: " + e);
20              }
21          }
22      }
```

Línea 1: paquete que me permite el manejo de entradas – salidas de flujos de información en un autónomo.

Línea 4: el *try – catch* permite el manejo de excepciones, donde una excepción es la indicación de un error en la ejecución de un programa ó del sistema operativo.

Línea 5: la clase *BufferedReader* permite leer de teclado una línea de información.

Línea 7: el objeto *lin* de tipo *String* guarda lo digitado por medio del teclado.

Línea 8: se utiliza el método *equals* para realizar una comparación.

Línea 11: se envía al método *catch*, el parámetro que indicara el error que se produce en un instante dado en la ejecución de un programa.

2.2 Estructura de alternativa compuesta if - else

Esta estructura permite elegir entre dos opciones o alternativas posibles, es decir, que realiza una acción S1, si la condición es verdadera y otra acción S2, si la condición es falsa.

```
if (condición verdadera )
{ S1}
else
{ S2}
Resto del programa
```

Ejemplo: Crear un programa Autónomo y un Applet que capture dos números desde el teclado, si el primero es mayor que el segundo imprimir "Soy un loco" sino imprimir "Soy un Genio".

1) Applet

```
import java.awt.*;
import java.applet.Applet;
public class IfCompuestoApp extends Applet{
    Label solicita, solicita1;
    TextField entrada, entrada1;
    int número, número1; //para almacenar valores digitados
    public void init()
    {
        solicita= new Label("Teclee un entero y enter");
        entrada=new TextField(10);
        solicita1= new Label("Teclee un entero y enter");
        entrada1=new TextField(10);
        add(solicita); // poner solicitud en el applet
        add(entrada); //poner la entrada en el applet
        add(solicita1); // poner solicitud en el applet
        add(entrada1); //poner la entrada en el applet
    }
    public void paint (Graphics g)
    {
        if(numero>numero1)
            g.drawString("Soy un loco",100,90);
        else
            g.drawString("Soy un Genio",100,90);
    }
    public boolean action(Event e,Object o)
    {
        if(e.target==entrada1)
        {
            número=Integer.parseInt(entrada.getText()); //obtener numero
            número1=Integer.parseInt(entrada1.getText()); //obtener numero
            repaint();
        }
        return true; // indica que la acción del usuario se proceso
    }
}
```

2) Autónomo

```
import java.io.*;
public class IfCompuestoAuto
{
    public static void main(String args[])
    {
        try{
            int num,num1;
            BufferedReader linea = new BufferedReader(
                new InputStreamReader( System.in ) );
            BufferedReader linea1 = new BufferedReader(
                new InputStreamReader( System.in ) );

            System.out.println("Digite un numero:");
            String lin = linea.readLine();
            System.out.println("Digite un numero:");
            String lin1 = linea1.readLine();
            num=Integer.parseInt(lin);
            num1=Integer.parseInt(lin1);
            if (num>num1)
                System.out.println("Soy un loco");
            else
                System.out.println("Soy un Genio");
            System.out.println("Es numero fue:"+lin);
        }catch(IOException e) {
            System.out.println("Error: " + e);
        }
    }
}
```

2.3 Estructura de alternativa múltiple switch- case

En algunos casos es necesario más de dos alternativas de solución a un problema. Esto se podría resolver por estructuras alternativas simples o compuestas anidadas; sin embargo, este método si el número de alternativas es grande puede plantear serios problemas de escritura del algoritmo y naturalmente de legibilidad.

La estructura de alternativa múltiple evalúa una expresión que puede tomar n valores distintos, 1, 2, 3, 4,..., n. Según el valor de la expresión, se realizará una de las n acciones.

```
switch(variable){
    case valor1:
        acciones1;
        break,
    case valor2:
        acciones2;
        break,
    case valor3:
        acciones3;
        break,
    .....:
    .....:
```

```

        default:
            otras_acciones;
    }

```

Ejemplo: crear un programa autónomo y un applet que capture un carácter e imprima el tipo de vocal.

1) Applet

```

import java.awt.*;
import java.applet.Applet;
public class SwitchApp extends Applet{
    Label solicita, vocal_ingresada;
    TextField entrada; //introducir valores
    public void init()
    {
        solicita= new Label("Teclee un caracter y Enter");
        vocal_ingresada= new Label();

        entrada=new TextField(2);

        add(solicita); // poner solicitud en el applet
        add(entrada); //poner la entrada en el applet
        add(vocal_ingresada);
    }
    public boolean action(Event e, Object o)
    {
        String val=o.toString();
        char vocal=val.charAt(0); //para obtener cada carácter de la cadena
        entrada.setText(""); //borra campo de texto
        switch(vocal){
            case 'A':case 'a':
                vocal_ingresada.setText("es una ." +vocal );
                break;
            case 'E':case 'e':
                vocal_ingresada.setText("es una ." +vocal );
                break;
            case 'I':case 'i':
                vocal_ingresada.setText("es una ." +vocal );
                break;
            case 'O':case 'o':
                vocal_ingresada.setText("es una ." +vocal );
                break;
            case 'U':case 'u':
                vocal_ingresada.setText("es una ." +vocal );
                break;
            default:
                vocal_ingresada.setText("No es una Vocal" );
                break;
        }
        repaint();
        return true; // indica que la acción del usuario se proceso
    }
}

```

Se utilizo el método *toString* de la clase *Object* para obtener la cadena de caracteres capturada en la caja de texto entrada.

```
String val=o.toString();
```

2) Autónomo

```
import java.io.*;
public class SwitchAuto
{
    public static void main(String args[])
    {
        try{
            char vocal;
            BufferedReader linea = new BufferedReader(
                new InputStreamReader( System.in ) );
            System.out.println("Digite un caracter:");
            String lin = linea.readLine();
            vocal=lin.charAt(0);
            switch(vocal){
                case 'A':case 'a':
                    System.out.println("es una :"+vocal);
                    break;
                case 'E':case 'e':
                    System.out.println("es una :"+vocal );
                    break;
                case 'I':case 'i':
                    System.out.println ("es una :"+vocal );
                    break;
                case 'O':case 'o':
                    System.out.println("es una :"+vocal );
                    break;
                case 'U':case 'u':
                    System.out.println("es una :"+vocal );
                    break;
                default:
                    System.out.println("No es una Vocal" );
                    break;
            }
        }catch(IOException e) {
            System.out.println("Error: " + e);
        }
    }
}
```

Utilizamos en los dos programa el método *charAt(n)* de la clase *String*, el cual me permite seleccionar un carácter en una posición *n* de una cadena de caracteres. En nuestro ejemplo utilizamos *charAt* en la posición 0 (*charAt(0)*), lo que permite extraer de la caja de texto entrada el carácter en la posición cero.

2.4 Estructura de repetición for

La estructura de repetición **for** ejecuta una acción mientras una condición sea verdadera.

```
for(inicialización; condición _ verdadera; incremento ó decremento)
{
    acción;
}
```

Ejemplo: crear un programa autónomo y un applet que imprima los primeros 10 números enteros así como su suma.

1) Applet

```
import java.awt.*;
import java.applet.Applet;
public class CicloForApp extends Applet{
    int suma=0;
    public void paint(Graphics g)
    {
        int cuenta,pos=25;
        for ( cuenta = 1; cuenta <= 10; cuenta++ ) {
            g.drawString(Integer.toString(cuenta),pos,25);
            pos+=10;
            suma=suma+cuenta;
        }
        g.drawString("la suma de los números es:"+ suma, pos+10,40);
    }
}
```

Utilizamos el método *toString* de la clase *Integer* que permite convertir un numero a carácter.

2) Autónomo

```
import java.io.*;
public class CicloForAuto
{
    public static void main(String args[])
    {
        try{
            int i,suma=0;
            for (i=0; i<10;i++)
            {
                suma=suma+i;
                System.out.println(Integer.toString(i));
            }
            System.out.println("La suma de los números es:"+suma);
        }catch(Exception e)
        { System.out.println("Error: " + e); }
    }
}
```

2.4 Estructura de repetición while

La estructura repetitiva **while** permite repetir una acción mientras la condición sea verdadera. Cuando se ejecuta la instrucción **while**, se evalúa la condición, si se evalúa como falsa, ninguna acción se toma y el programa prosigue en la siguiente instrucción después del ciclo. Si la condición es verdadera, entonces se ejecuta la acción en el ciclo. Este proceso se repite una y otra vez mientras la condición sea verdadera.

```
while( condición_verdadera)
{
    acción;
    incremento ó decremento;
}
```

Ejemplo: Crear un programa autónomo y un applet que imprima los primeros 10 números así como su suma.

1) Applet

```
import java.awt.*;
import java.applet.Applet;
public class CicloWhileApp extends Applet{
    public void paint(Graphics g)
    { int cuenta=1,pos=25;
      int suma=0;
      while (cuenta <= 10 ) {
          g.drawString(Integer.toString(cuenta),pos,25);
          pos+=10;
          suma=suma+cuenta;
          cuenta++;
      }
      g.drawString("la suma de los números es:" + suma, pos+10,40);
    }
}
```

2) Autónomo

```
public class CicloWhileAuto
{
    public static void main(String args[])
    { try{
        int i=0,suma=0;
        while (i<=9)
        {
            suma=suma+i;
            System.out.println(Integer.toString(i));
            i++;
        }
        System.out.println("La suma de los números es:" + suma);
    } catch (Exception e)
    { System.out.println("Error: " + e); }
}
```

2.5 Estructura de repetición do - while

En ocasiones es necesario que un ciclo de repetición se ejecute por lo menos una vez antes de evaluar la condición. La estructura **do - while** ejecuta una acción hasta que se cumpla una condición determinada que se comprueba al final del ciclo. La estructura de repetición **do - while** se repite mientras el valor de la condición sea falsa.

```
do {  
    accion;  
    incremento ó decremento  
}while(condición_falsa);
```

Ejemplo: crear un programa autónomo y un applet que imprima los primeros 10 números así como su suma.

1) Applet

```
import java.awt.*;  
import java.applet.Applet;  
public class CicloDoWhileApp extends Applet{  
    int suma=0;  
    public void paint(Graphics g)  
    {  
        int cuenta=1,pos=25;  
        do {  
            g.drawString(Integer.toString(cuenta),pos,25);  
            pos+=10;  
            suma=suma+cuenta;  
            cuenta++;  
        }while(cuenta <= 10);  
        g.drawString("La suma de los números es:"+ suma, pos+10,40);  
    }  
}
```

2) Autónomo

```
public class CicloDoWhileAuto  
{  
    public static void main(String args[])  
    {  
        try{  
            int i=0,suma=0;  
            do  
            {  
                suma=suma+i;  
                System.out.println(Integer.toString(i));  
                i++;  
            }while(i<=9);  
            System.out.println("La suma de los números es:"+suma);  
        }catch(Exception e)  
        { System.out.println("Error: " + e); }  
    }  
}
```

2.6 Estructuras de control anidadas

Las estructuras de control tanto alternativas como de repetición se pueden anidar, es decir, se puede insertar una estructura dentro de otra. La estructura interna debe estar incluida totalmente dentro de la estructura externa.

<pre>if (condición) { if(condición) { acción1; } else { acción2; } } else { if(condición) { if(condición) { acción; } } }</pre>	<pre>while (condición) { while(condición) { acción1; incremento; } for(inicio; cond ;inc) { acción2; for(inicio; cond ;inc) { acción; } } if(condición) { if(condición) { acción; } } }</pre>
---	---

Ejemplo 1: Crear un applet que imprima veinte números aleatoriamente en forma de matriz.

```
import java.awt.*;
import java.applet.Applet;
public class ForRandom extends Applet{
    public void paint(Graphics g)
    {
        int x=25,y=25,valor;
        for(int i=1;i<=20;i++)
        {
            valor=(int)(Math.random()*6)+1;
            g.drawString(Integer.toString(valor),x,y);
            if(i%5!=0)
                x+=40;
            else
            {
                x=25; y+=15;
            }
        }
    }
}
```

Ejemplo 2: Crear un applet que lea 3 números enteros e imprima el mayor y el menor de los números.

```
import java.awt.*;
import java.applet.Applet;
public class CiclosAppl extends Applet
{
    Label texto1, texto2, texto3;
    TextField entrada1, entrada2, entrada3; //introducir valores
    int numero1, numero2, numero3; //almacenar valores introducidos
    int my=0, mn=0;
    public void init()
    {
        Solicita1= new Label("Teclee un entero");
        Entrada1=new TextField(10);
        Solicita2= new Label("Teclee un entero");
        Entrada2=new TextField(10);
        Solicita3= new Label("Teclee un entero");
        Entrada3=new TextField(10);
        add(solicita1);
        add(entrada1);
        add(solicita2);
        add(entrada2);
        add(solicita3);
        add(entrada3);
    }
    public void paint (Graphics g)
    {
        if(numero1>numero2 && numero1>numero3)
        {
            my=numero1;
            if(numero2<numero3)
                mn=numero2;
            else
                mn=numero3;
        }
        if(numero2>numero1 && numero2>numero3)
        {
            my=numero2;
            if(numero1<numero3)
                mn=numero1;
            else
                mn=numero3;
        }
        if(numero3>numero2 && numero3>numero1)
        {
            my=numero3;
            if(numero1<numero2)
                mn=numero1;
            else
                mn=numero2;
        }
        g.drawString("El mayor es:"+my,100,90);
        g.drawString("El menor es:"+mn,100,120);
    }
    public boolean action(Event e, Object o)
    {

```

```

        if(e.target==entrada1)
            numero1=Integer.parseInt(entrada1.getText()); //obtener numero
        if(e.target==entrada2)
            numero2=Integer.parseInt(entrada2.getText()); //obtener numero
        if(e.target==entrada3)
            numero3=Integer.parseInt(entrada3.getText()); //obtener numero
    }
    return true; // indica que la acción del usuario se proceso
}
}

```

2.7 Ejercicios

Realizar los siguientes programas tanto en applet como en autónomo.

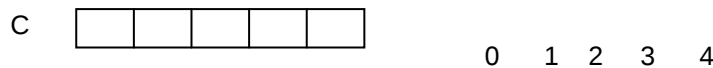
1. Hacer un programa que capture un numero por teclado e imprima si es par o impar.
2. Hacer un programa que capture tres números e imprima cual es el mayor, el del medio y el menor.
3. Escribir un programa que capture un número e imprima si es menor de 15, si es mayor de 50 o si está entre 16 y 49.
4. Hacer un programa que capture dos números e imprima la suma, la resta, la división, la multiplicación y el residuo de la división.
5. Hacer un programa que pida al usuario digitar en hora militar la hora, los minutos, los segundos e imprima la hora, los minutos y los segundos un segundo después.
6. Escribir un programa que sume los números enteros de 1 a 100, utilizando las estructura de repetición for, while, do while.
7. Escribir un programa que capture un numero e imprima si dicho numero es primo o no.
8. Escribir un programa que permita leer una cadena de caracteres e imprimir cuantas vocales existen de cada una.
9. Escribir un programa que permita imprimir todos los números primos entre 3 y 999 inclusive.
10. Escribir un programa para imprimir la suma de los números impares menores o iguales que n. donde n es un numero digitado por teclado.

3. Arreglos

3.1 Arreglos Unidimensionales

Un arreglo es un grupo de posiciones de memoria contiguas, las cuales tiene el mismo nombre y el mismo tipo. Para referirnos a un elemento del arreglo especificamos el nombre del arreglo y el número de la posición de ese elemento en el arreglo. El primer elemento del arreglo es el elemento número cero(0). El número de la posición en corchetes recibe el nombre de subíndice. El subíndice debe ser un entero.

- Declaración de un arreglo
`int c[];`
- Asignación de espacio de almacenamiento :
`c= new int[5];`



- La longitud de un arreglo se determina con la expresión:
`c.length` // donde c es el nombre del arreglo
- Todos elementos del arreglo siempre se inicializan en cero (0).
- Los arreglos se pueden inicializar directamente.

Ejemplo 1: crear un applet que imprima 10 números, los subíndices del arreglo, como también la suma de los 10 números.

```
import java.awt.*;
import java.applet.Applet;
public class Arreglos1 extends Applet
{
    int total=0;
    int M[]={125,14,25,36,85,47,96,33,258,478}; Arreglo inicializado directamente
    public void paint(Graphics g)
    {
        int y=25;
        g.drawString("Posición",25,y);
        g.drawString("Valor",100,y);
        for(int i=0;i<M.length;i++)
        {
            y+=15;
            total=total+M[i];
            g.drawString(String.valueOf(i),25,y);
            g.drawString(String.valueOf(M[i]),100,y);
        }
        g.drawString("la suma de los numeros del arreglo es:"+total,120,y+15);
    }
}
```

Ejemplo 2: crear un Applet que capture 10 números y los ordene de mayor a menor

```
import java.awt.*;
import java.applet.Applet;
public class Ordenar extends Applet{
    Label solicita; // solicitar entrada del usuario
    TextField entrada; //introducir valores
    int numero; //almacenar valores introducidos
    int aux=0,temp,my=0,mn=0,i;
    int m[];
    public void init()
    {
        if(my==0)
        {
            m=new int[10];
            my=1;
        }
        solicita= new Label("Teclee un entero y enter");
        entrada=new TextField(10);
        add(solicita); // poner solicitud en el applet
        add(entrada); //poner la entrada en el applet
    }
    public void paint (Graphics g)
    {
        int x=35,y=135;
        for(i=0;i<10;i++)
        {
            for(int j=i;j<10;j++)
            {
                if (m[i]< m[j])
                {
                    temp=m[i];
                    m[i]=m[j];
                    m[j]=temp;
                }
            }
        }
        if (aux>=10)
        {
            for(int j=0;j<aux;j++)
            {
                g.drawString(String.valueOf(m[j]),x,y);
                x=x+35;
            }
        }
    }
    public boolean action(Event e,Object o)
    { while(aux<10){
        if(e.target==entrada)
        {
            numero=Integer.parseInt(entrada.getText()); //obtener numero
            showStatus(Integer.toString(numero));
            m[aux]=numero;
            entrada.setText("");
        }
    }
}
```

```

        repaint();
        aux++;
    }
}
entrada.setEditable(false);
return true; // indica que la acción del usuario se proceso
}
}

```

3.2 Arreglos con múltiples subíndices (Matrices)

Las matrices consisten en información dispuesta en filas y columnas. Para identificar un elemento en particular debemos especificar el número de la fila y el número de la columna. Su formato es:

- Declaración de una matriz
`int c[][];`
- Asignación de espacio de almacenamiento :
`c= new int[2][5];`

C	0					
	1					
		0	1	2	3	4

- La longitud de la matriz se determina con la expresión:
`c.length`
- Todos elementos de la matriz siempre se inicializan en cero (0).

Ejemplo : crear un applet donde se imprima los valores de una matriz de 3 x 3.

```

import java.awt.*;
import java.applet.Applet;
public class ArregloMulti extends Applet
{
    int M[][]={{3,45,66},{87,94,23},{13,22,88}};
    public void paint(Graphics g)
    {
        int x=25, y=40;
        g.drawString("Valores del Arreglo M",25,25);
        for(int i=0; i<M.length;i++)
        {
            for(int j=0;j<b[i].length;j++)
            {
                g.drawString(String.valueOf(M[i][j]),x,y);
                x+=20;
            }
            x=25;
            y+=15;
        }
    }
}

```

3.3 Ejercicios

Realizar los siguientes programas tanto en applet como en autónomo.

1. Leer una cadena e imprimir la cadena inversamente
2. Hacer un programa que le permita insertar o eliminar elementos de un arreglo (los elementos deben mostrarse como se ingresaron).
3. Crear un programa que lea 10 números utilizando un arreglo, luego imprima el arreglo sin números repetidos.
4. Crear un programa que utilice dos arreglos de 5 posiciones y realice las siguientes operaciones: unión e intersección entre los arreglos.
5. Leer una matriz de M filas x N columnas calcular e imprimir la suma de los elementos de la matriz, los múltiplos de tres (3).
6. Se Tiene un arreglo A de N elementos, calcular:
 - La suma de las diferencias entre números adyacentes
 - La suma de los números positivos
7. Se Tiene un arreglo unidimensional C de N elementos, calcular:
 - El numero de valores impares
 - El numero de valores pares
 - La cantidad de ceros.
8. Leer una cadena de caracteres, digitar el carácter que se quiera eliminar e imprimir la cadena resultante.
9. Los numeros astromg o cubos perfectos, son aquellos que sumados los cubos de sus digitos nos dan el mismo numero. Por ejemplo 153 es un cubo perfecto , pues $(1)^3 + (5)^3 + (3)^3 = 153$. Escriba un programa que dado un numero entero , diga si es o no es, un cubo perfecto.
10. Escribir un algoritmo que convierta los números arábigos en romanos y viceversa (I=1, V=5, X=10, L=50, C=100 y m=1000).

4. Métodos

Un método es un subprograma de un programa principal que realiza tareas específicas. Los programas que se escriben en Java generalmente se combinan con métodos nuevos o creados por el programador y por métodos predefinidos en las diferentes clases del API de Java, los cuales forman parte del desarrollador SJK. Los API de Java ofrecen una gran colección de clases y métodos que nos permiten realizar cálculos matemáticos, manipular caracteres, manipular cadenas, realizar entradas/salidas, verificar errores, gráficos, etc.

Un método se invoca, es decir, se le pide que realice una tarea designada. La llamada al método se especifica con el nombre del método. Los métodos de algunas clases se invocan automáticamente.

En los métodos todas las variables declaradas en las definiciones de métodos son variables locales; solo se conocen en el método que las define. Casi todos los métodos tienen una lista de parámetros que permiten comunicar información entre métodos. Los parámetros de un método son variables locales.

El formato de una definición de métodos es:

```
[modificador][tipo_valor_devuelto] [nombre_metodo] (lista-parámetros)
{
    declaraciones y enunciados;
}
```

- Los modificadores de un método pueden ser: public, private
- El tipo de valor devuelto puede ser un tipo de dato, el nombre de una clase, un valor nulo (void).
- Los parámetros son variables que reciben información que van a ser procesada en el método.

4.1 Métodos que se ejecutan automáticamente

En los applet que se han trabajado, se han utilizado métodos que se invocan automáticamente, por ejemplo, los métodos de la clase Applet:

- **init():** Este método inicializa un Applet. Las acciones que por lo regular se realizan aquí incluyen la inicialización de variables y componentes de la interfaz gráfica del usuario.
- **paint(Graphics g):** Este método se invoca para dibujar en la Applet después de que el método init termina de ejecutarse y se ha comenzado a ejecutarse el método start(); también se invoca automáticamente cada vez que la Applet necesita redibujarse.

4.2 Métodos predefinidos de la clase Math

La clase Math contiene métodos que nos permiten realizar cálculos matemáticos comunes, algunos son:

- **sin(double a):** retorna el seno de un ángulo
- **cos(double a):** retorna el coseno de un ángulo
- **max(int a, int b):** retorna el valor máximo entre *a* y *b*
- **min(int a, int b):** retorna el valor mínimo entre *a* y *b*
- **ceil (double a):** retorna el valor entero superior de un double, ej: 13.45 =13

- floor(double a): retorna el valor entero inferior de un double, ej: 13.45 =14
- pow(double a, double b): retorna la potencia de a elevado a la b
- random(): retorna un valor entero positivo entre 0.0 y 1.0
- sqrt(n): retorna la raíz cuadrada de n
- tan(): retorna la tangente de un ángulo

Ejemplo 1: Crear un Applet que utilice algunos de los métodos de la clase Math

```
import java.applet.Applet;
import java.awt.Graphics;
public class MetMath extends Applet{
    public void paint (Graphics g)
    {
        g.drawString("Métodos Existentes en la clase MATH",25,15);
        g.drawString("Valor Absoluto de -250 :"+Math.abs(-250),25,40);
        g.drawString("Redondeo de x al entero superior ( 12.8) :"+Math.ceil(12.8),25,55);
        g.drawString("El coseno de 0.0 :"+Math.cos(0.0),25,70);
        g.drawString("La raíz cuadrada de un numero:5 :"+Math.sqrt(5),25,85);
        g.drawString("La potencia de N(y) : 5(5) :"+Math.pow(5,5),25,100);
        g.drawString("Calcula el mínimo de 2 números (10,15) :"+Math.min(10,15),25,115);
        g.drawString("Calcula el máximo de dos números (10,15) :"+Math.max(10,15),25,130);
        g.drawString("El seno de 0.0 :"+Math.sin(0.0),25,145);
        g.drawString("Redondeo de x al entero inferior : (12.8) :"+Math.floor(12.8),25,160);
    }
}
```

4.3 Métodos de usuario que no reciben parámetros, ni retornan valores

Cuando un método no retorna valores el tipo de valor devuelto debe ser *void*, ya que esta palabra reservada tiene un valor nulo. La lista de parámetros no debe contener ningún tipo de dato. Si desea incluir un parámetro este debería ser de tipo *void*:

```
void funcion()
{
    acciones;
}

ó

void funcion(void)
{
    acciones;
}
```

Ejemplo: Hacer un programa autónomo y un applet que utilice un método que no reciba ni retorne valores.

a) Autónomo

```
class Mimetodo
{
    void mensaje()
    {
```

```

        System.out.println("Hola mi gente");
    }
}
public class MetodoNrNr
{
    public static void main(String args[])
    {
        Mimetodo m=new Mimetodo();
        m.mensaje();
    }
}

```

b) Applet

```

import java.awt.*;
import java.applet.*;
public class MetodoNrNr extends Applet
{ int x,y,total;
    public void paint(Graphics g)
    { método();
      g.drawString("La suma en el método es :"+total, 20, 20);
    }
    void método()
    {
        x=9;
        y=7;
        total=x+y;
    }
}

```

4.4 Métodos de usuario que reciben parámetros pero no retornan valores

Cuando un método no retorna valores el tipo de valor devuelto debe ser *void*, ya que esta palabra reservada tiene un valor nulo. La lista de parámetros debe contener los diferentes tipos de datos.

```

void funcion(tipo x, tipo y, etc.....)
{
    Acciones;
}

```

Ejemplo: crear un Applet utilizando métodos que capture tres números y obtenga el mayor de los tres.

```

import java.awt.*;
import java.applet.Applet;
public class Maximo extends Applet{
    Label solicita,solicita1,solicita2,resultado; // solicitar entrada del usuario
    TextField entrada,entrada1,entrada2,resp; //introducir valores
    int numero,numero1,numero2,may; //almacenar valores introducidos
    public void init()
    {
        solicita= new Label("Teclee un entero y enter");
    }
}

```

```

    entrada=new TextField(10);
    solicita1= new Label("Teclee un entero y enter");
    entrada1=new TextField(10);
    solicita2= new Label("Teclee un entero y enter");
    entrada2=new TextField(10);
    resultado= new Label("El valor máximo es:");
    resp=new TextField(10);
    resp.setEditable(false);
    add(solicita2); // poner solicitud en el Applet
    add(entrada2); //poner la entrada en el Applet
    add(solicita); // poner solicitud en el Applet
    add(entrada); //poner la entrada en el Applet
    add(solicita1); // poner solicitud en el Applet
    add(entrada1); //poner la entrada en el Applet
    add(resultado); // poner solicitud en el Applet
    add(resp); //poner la entrada en el Applet
}
public void máximo (int x, int y ,int z)
{
    if(x>y && x>z)
        may=x;
    if(y>x && y>z)
        may=y;
    if(z>y && z>x)
        may=z;
}
public boolean action(Event e, Object o)
{
    numero=Integer.parseInt(entrada.getText()); //obtener numero
    numero1=Integer.parseInt(entrada1.getText()); //obtener numero
    numero2=Integer.parseInt(entrada2.getText()); //obtener numero
    maximo(numero,numero1,numero2);
    resp.setText(Integer.toString(may));
    return true; // indica que la acción del usuario se proceso
}
}

```

4.5 Métodos de usuario que reciben parámetros y retornan valores

Cuando un método retorna valores, el tipo de valor devuelto debe ser diferente de *void*. La lista de parámetros debe contener los diferentes tipos de datos. Esta clase de métodos debe contener la palabra reservada *return*, la cual sirve para devolver el valor a la función principal. Solamente se puede devolver un único valor.

```

<tipo de dato> funcion(tipo x, tipo y, etc.....)
{
    acciones;
    return valor;
}

```

Ejemplo 1: crear un Applet que capture dos números e imprima el producto de estos por el método de las sumas sucesivas.

```
import java.awt.*;
import java.applet.Applet;
public class SumasSucesivas extends Applet
{
    Label solicita,solicita1,resultado,resultado1; // solicitar entrada del usuario
    TextField entrada,entrada1,producto; //introducir valores
    int numero,numero1,valor;
    public void init()
    {
        solicita= new Label("Teclee el multiplicando :");
        entrada=new TextField(10);
        solicita1= new Label("Teclee el Multiplicador :");
        entrada1=new TextField(10);
        resultado= new Label("El producto es:");
        producto=new TextField(10);
        producto.setEditable(false);
        add(solicita);// poner solicitud en el applet
        add(entrada); //poner la entrada en el applet
        add(solicita1);// poner solicitud en el applet
        add(entrada1); //poner la entrada en el applet
        add(resultado);// poner solicitud en el applet
        add(producto); //poner la entrada en el applet
    }
    public boolean action(Event e,Object o)
    {
        if(e.target==entrada)
            entrada1.requestFocus();
        if(e.target==entrada1)
        {
            numero=Integer.parseInt(entrada.getText());//obtener numero
            numero1=Integer.parseInt(entrada1.getText());//obtener numero
            valor=miproducto(numero,numero1);
            producto.setText(""+valor);
            repaint();
        }
        return true; // indica que la acción del usuario se proceso
    }
    public int miproducto(int x,int y)
    { int z=0;
      for(int i=0;i<y;i++)
          z=z+x;
      return z;
    }
}
```

Ejemplo 2: Crear un Applet que capture un número y determine si es perfecto o no. Un número es perfecto si la suma de sus factores incluido el 1 es igual al número.

```
import java.awt.*;
import java.applet.*;
public class Unperfecto extends Applet
{
    TextField entrada;
    Label solicita;
```

```

int numero;
boolean per;
public void init()
{
    setLayout(null);
    entrada=new TextField(5);
    solicita=new Label("Digite un Numero : ");
    add(solicita);
    solicita.setBounds(20,20,150,20);
    add(entrada);
    entrada.setBounds(170,20,40,20);
}
public void paint(Graphics g)
{
    if(per==true)
        g.drawString("El numero :"+numero +" es perfecto",20,100);
    else
        g.drawString("El numero:"+numero +" no es perfecto",20,100);
}
public boolean action(Event e, Object o)
{
    if(e.target==entrada)
    {
        numero=Integer.parseInt(entrada.getText());
        per=perfecto(numero);
        repaint();
    }
    return true;
}
public boolean perfecto(int n)
{
    int sf=1,s=0;
    for(int i=2;i<n;i++)
    {
        if(n%i==0)
            sf=sf+i;
    }
    if(sf==n)
        return true;
    else
        return false; }
}

```

4.6 Métodos recursivos

Un método recursivo es aquel que se llama a sí mismo ya sea directamente o indirectamente a través de otro método.

Ejemplo: Utilizando recursividad crear un Applet que realice la serie de fibonacci, la cual se define:

```

Fibonacci(0)=0
Fibonacci(1)=1
Fibonacci(n)=Fibonacci(n-1)+Fibonacci(n-2)

```

```

import java.awt.*;
import java.applet.Applet;
public class Fibonacci extends Applet
{
    Label numero,resultado;
    TextField num,resp;
    public void init()
    {
        numero= new Label("Digite un numero entero y pulse Enter");
        num=new TextField("0",10);
        resultado= new Label("El valor de Fibonacci es:");
        resp=new TextField("0",10);
        add(numero);
        add(num);
        add(resultado);
        add(resp);
    }
    public boolean action(Event e, Object o)
    {
        long numeros,valf;
        numeros=Long.parseLong(num.getText());
        showStatus("Calculando...");
        valf=fibonacci(numeros);
        showStatus("Listo...");
        resp.setText(Long.toString(valf));
        return true;    }
    long fibonacci(long n)
    {
        if(n==0 || n==1)
            return n;
        else
            return fibonacci(n-1)+fibonacci(n-2);
    }
}

```

4.7 Paso de Arreglos a Métodos

Para pasar un arreglo a un método se debe especificar el nombre del arreglo sin los corchetes:

```

int arreglo[];
arreglo = new int [24];

```

la llamada al método sería :

```

traerArreglo(arreglo)

```

Y el método lo recibiría así:

void traerArreglo(b[]), donde b sería el nombre con el cual manejaría el arreglo en el método.

Ejemplo: crear un Applet que imprima 10 números de un arreglo ordenándolo, utilizando el paso de arreglos a métodos.

```

import java.awt.*;
import java.applet.*;

```

```

public class ArregloFuncion extends Applet
{
    int M[]={3,45,66,87,94,23,13,22,88,100};
    int aux;
    public void paint(Graphics g)
    {
        pintar(g,"Arreglo Original",M,25,25);
        ordenar();
        pintar(g,"Arreglo Ordenado",M,25,100);
    }
    public void ordenar()
    {
        for(int i=0;i<M.length;i++)
        {
            for(int j=i;j<M.length;j++)
            {
                if(M[i]>M[j])
                {
                    aux=M[i];
                    M[i]=M[j];
                    M[j]=aux;
                }
            }
        }
    }

    public void pintar(Graphics g, String texto, int b[],int x, int y)
    {
        g.drawString(texto,x,y);
        x+=15;
        y+=15;
        for(int i=0; i<b.length;i++)
        { g.drawString(String.valueOf(b[i]),x,y);
          x+=20;
        }
    }
}

```

4.8 Ejercicios

Realizar los siguientes programas tanto en applet como en autónomo utilizando métodos creado por el usuario ó del API de java..

1. Leer un arreglo C de N elementos, calcular:
 - El numero de datos repetidos en el arreglo
 - El numero de valores impares
 - El numero de valores pares
 - La cantidad de ceros.
2. Leer dos arreglos A y B. Forme un nuevo arreglo C cuyos elementos comprendan la unión de A y B. Por unión se entiende que C contenga una copia de cada valor distinto encontrado en todo A y en todo B. Se debe imprimir cada arreglo, C debe de estar ordenado de Menor a Mayor.

3. Leer una cadena de caracteres, digitar el carácter que se quiera eliminar e imprimir la cadena resultante.
4. Diseñar el programa para imprimir la suma de los números impares menores o iguales que n.
5. Programa que reciba un número entero y retorne el número con sus dígitos invertidos. Ejemplo: dado el número 7631 el método deberá retornar 1367.
6. Programa que calcule las raíces de una ecuación de segundo grado. El discriminante es $(b^2 - 4*a*c)$. Se deben tener en cuenta todas las posibles validaciones.
7. Realizar un programa que decida si dos números son amigos. Dos números son amigos si la suma de los divisores del primer numero, excluido el, es igual al segundo numero, y viceversa; es decir, si la suma de los divisores del segundo numero, excluido el, es igual al primer numero.
8. Los numeros astromg o cubos perfectos, son aquellos que sumados los cubos de sus digitos nos dan el mismo numero. Por ejemplo 153 es un cubo perfecto , pues $(1)^3 + (5)^3 + (3)^3$ es igual a 153. Escriba un programa que dado un numero entero , diga si es o no es, un cubo perfecto.
9. Hacer un programa que lea una cadena de caracteres S y un factor de multiplicación N, cuya función sea generar la cadena dada N veces. Ej:

!hola! 3

deberá imprimir:

!hola! !hola! !hola!
10. Hacer un programa que lea un numero no mayor de 1000 e imprima ese numero en letras.

5. Cadenas y caracteres

Todo programa de Java se compone de una secuencia de caracteres que al agruparse son interpretados por el computador como una serie de instrucciones que sirven para realizar una tarea.

Una cadena en Java es un conjunto de caracteres y se crean como un objeto de la clase `String`(cadena), las cadenas se escriben encerradas entre comillas, ejemplo:

```
"Rosa"  
"Bogotá - Cundinamarca"
```

Un `String` se puede asignar a una referencia `String` en una declaración:
`String nombre="Cristian"`

```
String str = "abc";
```

Es equivalente a:

```
char data[] = {'a', 'b', 'c'};  
String str = new String(data);
```

5.1 Clase String

La clase `String` cuenta con varios métodos que sirven para manipular y realizar operaciones con cadena de caracteres, a continuación se explicara algunos métodos y su utilización:

- **`charAt(n)`:** Recibe un argumento entero que se utiliza como número de posición (índice) y devuelve el carácter que esta en esa posición.
Ejemplo:

```
String cadena="Colombia"  
char toma_caracter=cadena.charAt(3);  
el valor de la variable toma_caracter es igual a:"o";
```
- **`compareTo()`:** permite comparar dos cadenas de caracteres, retornando un valor entero. Si el valor es cero(0), las dos cadenas son iguales. Si el valor es entero positivo, el `String` que invoca es mayor. Si el valor es entero negativo, el `String` que invoca es menor.
Ejemplo:

```
(string_invoca).compareTo(string_invocado)
```
- **`compareToIgnoreCase()`:** al igual que la anterior, pero no tiene en cuenta mayúsculas ó minúsculas.
- **`equals()`:** sirve para comparar dos cadenas

```
string1.equals(string2)
```
- **`equalsIgnoreCase()`:** sirve para comparar dos cadenas, pero no tiene en cuenta mayúsculas ó minúsculas.
- **`length()`:** retorna la longitud de una cadena

```
String cadena="Colombia"  
int valor=cadena.length();  
valor =8
```
- **`replace()`:** reemplaza un carácter por otro carácter en una cadena.

```
String cadena="Colombia"  
String valor=cadena.replace('o','x');
```

```
valor ="Cxlxmbia"
```

- **substring():** retorna una nueva cadena a partir de otra cadena. Se puede retorna la nueva cadena desde una un subíndice inicial hasta un subíndice final.

```
String cadena="Colombia"  
String valor=cadena.substring(2);  
valor ="ombia"  
String valor=cadena.substring(2,2);  
valor ="om"
```
- **toCharArray():** convierte una cadena de caracteres en un arreglo tipo char.

```
String cadena="Colombia"  
char valor[]=cadena.toCharArray();  
valor[] ={'c','o','l','o','m','b','i','a'};
```
- **startsWith:** busca si una cadena inicia con una cadena especifica.
- **endsWith:** busca si una cadena termina con una cadena especifica.
- **concat:** sirve para unir dos cadenas de caracteres.

```
String cadena="Colombia"  
String cadena2="Viva ";  
String unir= cadena2.concat(cadena);  
unir="Viva Colombia";
```
- **toLowerCase():** convierte una cadena escrita en mayúsculas a minúsculas

```
String cadena="COLOMBIA"  
cadena.toLowerCase();  
cadena="colombia";
```
- **toUpperCase():** convierte una cadena escrita en minúsculas a mayúsculas

```
String cadena="colombia"  
cadena.toUpperCase();  
cadena="COLOMBIA";
```
- **regionMatches():** compara si una subcadena de una cadena están en la misma posición de otra cadena. Contiene cuatro argumentos: el primero es el índice inicial, el segundo es la cadena de caracteres, el tercero es la posición inicial a comparar y el último es el número de caracteres a comparar. Si contiene el argumento *true*, ignora mayúsculas y minúsculas.
- **indexOf(primer índice):** busca un carácter o un grupo de caracteres en una cadena en una posición inicial.
- **lastIndexOf(último índice):** busca un carácter o un grupo de caracteres en una cadena en una posición final.

Ejemplo 1: Applet que permita manipular los diferentes métodos de comparación de cadenas: equals, equalsIgnoreCase, compareTo, regionMatches.

```
import java.awt.*;  
import java.applet.*;
```

```
public class CompararCadenas extends Applet  
{  
    String a1,a2,a3,a4;
```

```

public void init()
{
    a1=new String("hola");
    a2=new String("Universidad");
    a3=new String("Facultad Tecnologica");
    a4=new String("facultad tecnologica");
}
public void paint(Graphics g)
{
    g.drawString("a1 = " + a1,25,25);
    g.drawString("a2 = " + a2,25,40);
    g.drawString("a3 = " + a3,25,55);
    g.drawString("a4 = " + a4,25,70);
    if(a1.equals("hola"))
        g.drawString("a1 es igual a \"hola\"",25,100);
    else
        g.drawString("a1 no es igual a \"hola\"",25,100);
    if(a3.equalsIgnoreCase(a4))
        g.drawString("a3 es igual a a4",25,130);
    else
        g.drawString("a3 no es igual a a4",25,130);

    //compareTo
    //Devuelve 0 si los dos String son iguales
    // negativo si el String que invoca a compareTo es menor
    //positivo si el String que invoca a compareTo es mayor
    g.drawString("a1.compareTo(a2) es"+ a1.compareTo(a2),25,160);
    g.drawString("a2.compareTo(a1) es"+ a2.compareTo(a1),25,175);
    g.drawString("a1.compareTo(a1) es"+ a1.compareTo(a1),25,190);
    g.drawString("a3.compareTo(a4) es"+ a3.compareTo(a4),25,205);
    g.drawString("a4.compareTo(a3) es"+ a4.compareTo(a3),25,220);
    //utilizar regionMatches** para comparar subcadenas de una cadena
    if(a3.regionMatches(0,a4,0,5))
        g.drawString("los primeros 5 caracteres de a3 y a4 coinciden",25,250);
    else
        g.drawString("los primeros 5 caracteres de a3 y a4 no coinciden",25,250);

    if(a3.regionMatches(true,0,a4,0,5))
        g.drawString("los primeros 5 caracteres de a3 y a4 coinciden",25,265);
    else
        g.drawString("los primeros 5 caracteres de a3 y a4 no coinciden",25,265);
}
}

```

Ejemplo 2: applet que utilice los métodos startsWith(comienza) y endsWith(termina) de la clase String

```

import java.awt.*;
import java.applet.*;
public class InicioFinCadenas extends Applet
{
    String cadena[]={ "inicio", "iniciando", "finalizo", "finalizando" };
    public void paint(Graphics g)
    {

```

```

int y=25;
//probar el método startsWith
for (int i=0;i<cadena.length;i++)
    if(cadena[i].startsWith("ini"))
    {
        g.drawString("["+cadena[i]+" comienza con \"ini\",25,y);
        y+=15;
    }
//comenzando en la posición 2
for (int i=0;i<cadena.length;i++)
    if(cadena[i].startsWith("ici",2))
    {
        g.drawString("["+cadena[i]+" comienza con \"ici\" en la posición 2\",25,y);
        y+=15;
    }
//probar el método endsWith
y+=15;
for (int i=0;i<cadena.length;i++)
    if(cadena[i].endsWith("ando"))
    {
        g.drawString("["+cadena[i]+" termina con \"ando\",25,y);
        y+=15;
    }
}
}

```

Ejemplo 3: applet que localice subcadenas de caracteres en una cadena de caracteres utilizando los métodos `indexOf` y `lastIndexOf`.

```

import java.awt.*;
import java.applet.*;

public class LocalizaCadenas extends Applet
{
    String palabra="esternocleidomastoideoesternocleidomastoideo";
    public void paint(Graphics g)
    {
        //con indexOf para encontrar un carácter
        g.drawString("'e' se encuentra en palabra"+ palabra.indexOf((int) 'c'),25,40);
        g.drawString("'l' se encuentra en palabra"+ palabra.indexOf((int) 'l',1),25,55);
        g.drawString("'u' se encuentra en palabra"+ palabra.indexOf((int) 'u'),25,70);
        //con lastIndexOf para encontrar un carácter en una cadena
        g.drawString("La ultima 'e' se encuentra en:"+ palabra.lastIndexOf((int) 'c'),25,85);
        g.drawString("La ultima 'l' se encuentra en:"+ palabra.lastIndexOf((int) 'l'),25,100);
        g.drawString("La ultima 'u' se encuentra en:"+ palabra.lastIndexOf((int) 'u'),25,115);
        //con indexOf para encontrar una subcadena
        g.drawString("'cleido' se encuentra en palabra"+ palabra.indexOf("cleido"),25,130);
        g.drawString("'cleido' se encuentra en palabra"+ palabra.indexOf("cleido",9),25,145);
        g.drawString("'gracias' se encuentra en palabra"+ palabra.indexOf("gracias"),25,160);
        //con lastindexOf para encontrar una subcadena
        g.drawString("La ultima 'cleido' se encuentra en:"+ palabra.lastIndexOf("cleido"),25,175);
        g.drawString("La ultima 'gracias' se encuentra en:"+
        palabra.lastIndexOf("gracias"),25,190);    }}

```

5.2 Clase StringTokenizer

La clase StringTokenizer descompone una oración en palabras individuales (unidades lexicográficas - tokens en inglés). Esta clase se encuentra en el paquete de Java: java.util. Sus principales métodos son:

- **countTokens:** devuelve el número de palabras contenidas en la cadena que se va a descomponer.
- **hasMoreTokens:** determina si existen más palabras en la cadena que se está descomponiendo.
- **nextToken:** devuelve la siguiente palabra de una cadena.

Ejemplo: Applet que utilice alguno de los métodos de la clase StringTokenizer para descomponer una cadena de caracteres.

```
import java.awt.*;
import java.applet.*;
import java.util.*;
```

```
public class DescomponeTexto extends Applet {
    Label texto;
    TextField entrada;
```

// área de texto: Son similares a los campos de texto; pueden servir para digitar o mostrar textos. En el Ejemplo siguiente se especifica que el área de texto tiene 10 filas y 30 columnas.

```
    TextArea salida;
```

```
    public void init()
    {
        texto= new Label("Teclee un párrafo y presione Enter");
        entrada = new TextField(50);
        salida= new TextArea(10,30);
        salida.setEditable(false);
        add(texto);
        add(entrada);
        add(salida);
    }
```

```
    public boolean action(Event e, Object o)
    {
        String descompuesto=o.toString();
```

```
        StringTokenizer palabra= new StringTokenizer(descompuesto);
        salida.setText("");
```

```
        /* del método appendText(anexar texto) de la clase TextArea sirve para añadir el String
        concatenado que se especifica como argumento al texto que ya está en el área de texto.*/
        salida.appendText("Número de elementos:" +
        palabra.countTokens()+ "\n Las unidades lexicográficas son:\n");
        while (palabra.hasMoreTokens())
            salida.appendText(palabra.nextToken()+"\n");
        return true; }}
```

5.3 Ejercicios

1. Hacer un programa que permita capturar dos cadenas por teclado e imprimir si la primera cadena es menor o igual o mayor que la segunda.
2. Leer un carácter y deducir si esta situado antes o después de la letra m en orden alfabético
3. Leer dos caracteres y deducir si estan en orden alfabético
4. Hacer un programa que permita comparar dos cadenas digitadas por teclado. El programa deberá solicitar el numero de caracteres a comparar y el índice inicial de la comparación. No se debe tener en cuenta minúsculas ni mayúsculas.
5. hacer un programa que permita leer una palabra y la imprima en un área de texto invertida.
6. Escribir un programa que lea una línea de texto, la divida en tokens o unidades lexicográficas e imprima cada palabra inversamente.
7. Hacer un programa que lea una línea de texto e imprima cada palabra en orden alfabético.
8. Escriba un programa que lea una línea de texto y pida un carácter para buscar e imprima cuantas veces se encuentra ese carácter en la línea de texto como también las palabras que contiene ese carácter.
9. Hacer un programa que lea una cadena e imprima en un área de texto todas las palabras que la tercera letra sea una b.
10. Hacer un programa que lea una serie de cadenas e imprima en un texto de área todas aquellas aparezca la palabra do.

6. Interfaz Gráfica de Usuario

Una interfaz gráfica con el usuario (GUI) se representa por una serie de objetos de fácil interpretación y visualización para el usuario. La GUI es un objeto visual con el que el usuario puede interactuar a través del ratón o el teclado.

Las clases para crear componentes GUI forman parte del paquete java.awt.

6.1 Label (etiqueta)

Es un área en la que se puede mostrar un texto no editable. Sus principales métodos son:

- `Label()`: Construye un rotulo vacío, no se muestra texto.
`Label texto=new Label();`
- `Label (String)`: Construye un rotulo que muestra el texto predefinido.
`Label texto=new Label("hola");`
- `Label (String variable)`: Construye un rotulo que muestra el texto predefinido.
`String texto="hola";`
`Label etiqueta=new Label(texto);`
- `Label (String, int)`: crea una etiqueta con texto predeterminado y la alineación indicada por el argumento int. Las variables de clase que se usan para establecer la alineación son: `Label.RIGHT`, `Label.LEFT` y `Label.CENTER`.
`Label etiqueta=new Label("hola", Label.CENTER);`
- `getText()` : obtiene el texto de una etiqueta
- `setText( String s)`: coloca un texto en una etiqueta.

6.2 Button (boton)

Es un objeto que activa un evento cuando se hace un clic en el. Sus principales métodos son:

- `Button ()`: crea un botón sin texto
`Button boton=new Button();`
- `Button (String s)` : construye un botón para pulsar con texto.
`Button boton=new Button("Mi Boton");`
- `getLabel()` : obtiene el texto del botón
- `setLabel( String )`: coloca un texto en el botón.

6.3 TextField (campo de texto)

Es un objeto en que un usuario puede digitar datos mediante teclado. El TextField también puede mostrar información.

- `TextField()`: construye un objeto campo de texto vacío.
`TextField campo=new TextField();`

- `TextField(int columnas)`: construye un objeto `TextField` vacío con el número de columnas especificadas.
`TextField campo=new TextField(" ",30);`
- `TextField(String s, int columnas)`: construye un objeto `TextField` que muestra un texto predefinido y un número de columnas predeterminadas.
`. TextField campo=new TextField("hola",30);`
- `TextField(String s)`: construye un objeto con un texto predefinido en un campo de texto.
`. TextField campo=new TextField("hola");`

6.4 TextArea (area de texto)

Permite crear un objeto con un área para manipular múltiples líneas de texto.

- `TextArea()`: construye un objeto con área de texto vacío.
`TextArea campo=new TextArea();`
- `TextArea(int filas, int columnas)`: construye un objeto `TextArea` vacío con el número de filas y columnas especificadas.
`TextArea campo=new TextArea(12,30);`
- `TextArea(String s)`: construye un objeto `TextArea` que muestra un texto predefinido.
`. TextArea campo=new TextArea("hola");`
- `TextArea(String s, int filas, int columnas)`: construye un objeto con un área de texto predefinido , con filas y columnas definidas.
`. TextArea campo=new TextArea("hola",5,10);`
- `append(String)`: permite adicionar un texto a un área de texto
- `getRows()`: retorna el numero de filas de un área de texto.
- `getColumns()`: retorna el numero de columnas de un área de texto.

6.5 Choice (cuadro combinado)

Permite crear objeto donde el usuario puede escoger una alternativa dentro de varias opciones..

- `Choice()`: construye un objeto de opción..

6.6 Checkbox (casillas de verificación)

Sirve para crear casillas de verificación. Estas son casillas de estado, es decir, las casillas tienen un valor de encendido/apagado ó verdadero/falso.

- `Checkbox()`: casilla de verificación sin texto.
- `Checkbox(String)`: casilla de verificación con texto

6.7 CheckboxGroup (Grupo de Botones)

Es un grupo de botones en los que solo un botón del grupo puede ser verdadero (true), de modo que la selección de un botón obligue a los demás botones a ser false. Para crear el grupo de botones utilizamos la clase CheckboxGroup y Checkbox.

- CheckboxGroup(): crea una instancia de un grupo de botones.

6.8 List (Lista)

Crea una serie de elementos de los cuales el usuario puede escoger uno o más.

- List(): crea un objeto de tipo lista.
- List(int): crea un objeto de tipo lista, indicando cuanto elementos serán visibles.
- add(String) : adiciona elementos a una lista
- getItemCount(): cuenta los elementos que contiene una lista
- getItems(): obtiene el elementos de una lista
- remove(int posición): quita el elementos en la posición indicada
- remove (String s): quita el primer elemento que concuerde con el texto especificado.
- removeAll(): quita todos los elementos de una lista.

6.9 ScrollBar (Barra de desplazamiento)

Una Barra de Desplazamiento es un componente que recorre un intervalo de valores enteros. Las barras de Desplazamiento tienen una orientación horizontal o vertical. La posición relativa del cuadro de desplazamiento indica el valor actual de la barra de desplazamiento.

- ScrollBar(): crear un objeto ScrollBar en forma vertical
- ScrollBar(int orientación): crear un objeto ScrollBar con la orientación especifica.
- ScrollBar(int orientación, int valor, int visible, int minimo, int maximo): crear un objeto ScrollBar con las siguientes condiciones:
 - int orientación: tipo de orientación (vertical o horizontal)
 - int valor: valor inicial de la barra de desplazamiento
 - int visible: tamaño del cuadro de desplazamiento
 - int minimo: valor mínimo que puede tomar la barra de desplazamiento
 - int máximo: valor máximo de la barra de desplazamiento.

6.10 Administrador de Diseños

Las interfaces graficas de los usuarios requieren que cada componente se coloque en un lugar exacto. Los administradores de diseños acomodan los componentes en un contenedor. Hasta ahora se ha trabajado con el administrador de diseños FlowLayout que se usa por omisión en los applets. Analizaremos los siguientes:

6.10.1 FlowLayout

Los componente GUI se colocan de izquierda a derecha en el orden que se agregan al contenedor. Este administrador se puede utilizar por alineación centrada(por omisión), a la derecha y a la izquierda. Su formato es:

FlowLayout(alineación, distancia_horizontal, distancia_vertical)

- alineación: es la alineación especificada por el usuario (CENTER, LEFT, RIGHT)
- distancia_horizontal: valor entero que permite obtener la distancia en pixeles entre los objetos en forma horizontal. (se puede omitir).
- distancia_vertical: valor entero que permite obtener la distancia en pixeles entre los objetos en forma vertical. (se puede omitir).

Ejemplo 1: Programa que utiliza el administrador de diseños FlowLayout, sin utilizar parámetros

```
import java.awt.*;
import java.applet.*;
public class Centrada extends Applet
{
    private TextField t1,t2,t3,t4;
    public void init()
    {
        setLayout(new FlowLayout(FlowLayout.CENTER)); // por omision
        // cambie el CENTER por LEFT ó RIGHT
        t1=new TextField("Centrado");
        t1.setEditable(false);
        add(t1);
        t2=new TextField("Centrado");
        t2.setEditable(false);
        add(t2);
        t3=new TextField("Centrado");
        t3.setEditable(false);
        add(t3);
        t4=new TextField("Centrado");
        t4.setEditable(false);
        add(t4);
    }
}
```

Ejemplo 2: Programa que utiliza el administrador de diseños FlowLayout, utilizando parámetros

```
import java.awt.*;
import java.applet.*;
public class Centrada extends Applet
{
    private TextField t1,t2,t3,t4;
    public void init()
    {
        setLayout(new FlowLayout(FlowLayout.RIGHT,5,10));
        // cambie el CENTER por LEFT ó RIGHT
        t1=new TextField("Centrado");
```

```

t1.setEditable(false);
add(t1);
t2=new TextField("Centrado");
t2.setEditable(false);
add(t2);
t3=new TextField("Centrado");
t3.setEditable(false);
add(t3);
t4=new TextField("Centrado");
t4.setEditable(false);
add(t4);
}}

```

6.10.2 BorderLayout

Este administrador de diseños acomoda los componentes en cinco áreas: North, South, West, East y Center. El administrador BorderLayout permite dos parámetros de distancia (horizontal y vertical).

```
setLayout(new BorderLayout( distancia_horizontal, distancia_vertical))
```

- `distancia_horizontal`: valor entero que permite obtener la distancia en pixeles entre los objetos en forma horizontal. (se puede omitir).
- `distancia_vertical`: valor entero que permite obtener la distancia en pixeles entre los objetos en forma vertical. (se puede omitir).

Ejemplo 1: programa que permita utilizar el administrador de diseños BorderLayout. sin parámetros

```

import java.awt.*;
import java.applet.*;
public class Bordes extends Applet
{
    private Button centro,este,norte,sur,oeste;
    public void init()
    {
        setLayout(new BorderLayout());
        centro=new Button("Boton central");
        oeste=new Button("Boton oeste");
        este=new Button("Boton este");
        sur=new Button("Boton sur");
        norte=new Button("Boton norte");

        add("South",sur);
        add("North",norte);
        add("East",este);
        add("Center",centro);
        add("West",oeste);
    }
}

```

Ejemplo 2: programa que permita utilizar el administrador de diseños BorderLayout. con parámetros

```
import java.awt.*;
import java.applet.*;
public class Bordes extends Applet
{
    private Button centro,este,norte,sur,oeste;
    public void init()
    {
        setLayout(new BorderLayout(20,15));
        centro=new Button("Boton central");
        oeste=new Button("Boton oeste");
        este=new Button("Boton este");
        sur=new Button("Boton sur");
        norte=new Button("Boton norte");

        add("South",sur);
        add("North",norte);
        add("East",este);
        add("Center",centro);
        add("West",oeste);
    }
}
```

6.10.3 GridLayout

Divide el contenedor en una cuadrícula que permite colocar los objetos en filas y columnas. Permite cuatro parámetros.

setLayout(new GridLayout(filas, columnas, distancia_horizontal, distancia_vertical))

- filas: un valor entero para el número de filas
- columnas: un valor entero para el número de columnas
- distancia_horizontal: valor entero que permite obtener la distancia en pixeles entre los objetos en forma horizontal. (se puede omitir).
- distancia_vertical: valor entero que permite obtener la distancia en pixeles entre los objetos en forma vertical. (se puede omitir).

Ejemplo 1: programa que permita utilizar el administrador de diseños GridLayout sin espaciado.

```
import java.awt.*;
import java.applet.*;
public class Bordes extends Applet
{
    private Button centro,este,norte,sur,oeste;
    public void init()
    {
        setLayout(new GridLayout(2,3));
        centro=new Button("Boton central");
        oeste=new Button("Boton oeste");
```

```

        este=new Button("Boton este");
        sur=new Button("Boton sur");
        norte=new Button("Boton norte");
        add(sur);
        add(norte);
        add(este);
        add(centro);
        add(oeste);
    }
}

```

Ejemplo 2: programa que permita utilizar el administrador de diseños GridLayout. con espaciado

```

import java.awt.*;
import java.applet.*;
public class Bordes extends Applet
{
    private Button centro,este,norte,sur,oeste;
    public void init()
    {
        setLayout(new GridLayout(2,3,20,25));
        centro=new Button("Boton central");
        oeste=new Button("Boton oeste");
        este=new Button("Boton este");
        sur=new Button("Boton sur");
        norte=new Button("Boton norte");

        add(sur);
        add(norte);
        add(este);
        add(centro);
        add(oeste);
    }
}

```

6.11 Ejemplos de Aplicación

Ejemplo 1: realizar un programa que utilice los componente gráficos de usuario: TextField, Label, Button

```

import java.awt.*;
import java.applet.*;
public class Gui1 extends Applet
{
    TextField texto,texto1,texto2,texto3;
    Label etiqueta, etiqueta1;
    Button boton, boton1;
    public void init()
    {

```

```

String cadena="Label con texto de una variable";
texto=new TextField();
texto1=new TextField(30);
texto2=new TextField("muestra de texto");
texto3=new TextField("muestra de texto",30);
etiqueta =new Label("Label con texto predefinido");
etiqueta1=new Label(cadena);
boton=new Button();
boton1=new Button("texto en un boton");
add(texto);
add(texto2);
add(texto1);
add(texto3);
add(etiqueta);
add(etiqueta1);
add(boton);
add(boton1);
}
}

```

Ejemplo 2: realizar un programa que utilice el componente gráfico de usuario: Checkbox,

```

import java.awt.*;
import java.applet.*;
public class Gui2 extends Applet
{
    Label t,t1,t2,t3,t4;
    Checkbox c1=new Checkbox("Perro");
    Checkbox c2=new Checkbox("Gato");
    Checkbox c3=new Checkbox("Pescado");
    Checkbox c4=new Checkbox("Bufalo");
    Checkbox c5=new Checkbox("Elefante");
    public void init()
    {
        t=new Label();
        t1=new Label();
        t2=new Label();
        t3=new Label();
        t4=new Label();
        add(c1);
        c1.setState(true);// inicializar el Checkbox C1 como true
        add(c2);
        add(c3);
        add(c4);
        add(c5);
        add(t);
        add(t1);
        add(t2);
        add(t3);
        add(t4);
    }
    public boolean action(Event e, Object o)
    { String cadena;
      if(e.target instanceof Checkbox){

```

```

        if(c1.getState()==true) {
            cadena=c1.getLabel();
            t.setText(cadena);
        }
        else
            t.setText("");
        if(c2.getState()==true) {
            cadena=c2.getLabel();
            t1.setText(cadena);
        }
        else
            t1.setText("");
        if(c3.getState()==true) {
            cadena=c3.getLabel();
            t2.setText(cadena);
        }
        else
            t2.setText("");
        if(c4.getState()==true) {
            cadena=c4.getLabel();
            t3.setText(cadena);
        }
        else
            t3.setText("");
        if(c5.getState()==true) {
            cadena=c5.getLabel();
            t4.setText(cadena);
        }
        else
            t4.setText("");
    }
    return true;
}
}

```

Ejemplo 3: realizar un programa que utilice el componente gráfico de usuario: `CheckboxGroup`,

```

import java.awt.*;
import java.applet.*;
public class Gui3 extends Applet
{
    Label etiqueta;
    CheckboxGroup grupo=new CheckboxGroup();
    Checkbox c1=new Checkbox("Perro",grupo,false);
    Checkbox c2=new Checkbox("Gato",grupo,false);
    Checkbox c3=new Checkbox("Pescado",grupo, true);
    Checkbox c4=new Checkbox("Bufalo",grupo,false);
    Checkbox c5=new Checkbox("Elefante",grupo,false);
    public void init()
    {
        etiqueta=new Label();
        add(c1);
    }
}

```

```

        add(c2);
        add(c3);
        add(c4);
        add(c5);
        add(etiqueta);
    }
    public boolean action(Event e, Object o)
    {
        if(e.target instanceof Checkbox){
            if(e.target==c1)
                etiqueta.setText("Perro");
            if(e.target==c2)
                etiqueta.setText("Gato");
            if(e.target==c3)
                etiqueta.setText("Pescado");
            if(e.target==c4)
                etiqueta.setText("Bufalo");
            if(e.target==c5)
                etiqueta.setText("Elefante");
        }
        return true;
    }
}

```

Ejemplo 4: realizar un programa que utilice los componentes gráficos de usuario: List y Choice.

```

import java.awt.*;
import java.applet.*;
public class Gui4 extends Applet
{
    List ciudad;
    Choice paises;
    Label etiqueta,etiqueta1;
    public void init()
    {
        etiqueta=new Label();
        etiqueta1=new Label();
        ciudad=new List(5,false);
        paises=new Choice();
        //agregar datos a la lista
        ciudad.addItem("Cucuta");
        ciudad.addItem("Bucaramanga");
        ciudad.addItem("Cali");
        ciudad.addItem("Bogotá");
        ciudad.addItem("Pasto");
        ciudad.addItem("Neiva");
        //agregar datos al boton de opcion
        paises.addItem("Colombia");
        paises.addItem("brasil");
        paises.addItem("Ecuador");
        paises.addItem("Bolivia");
        paises.addItem("Argentina");
        paises.addItem("Uruguay");
    }
}

```

```

add(ciudad);
add(paises);
add(etiqueta);
add(etiqueta1);
}
public boolean action(Event e, Object o)
{
    if(e.target==ciudad) {
        etiqueta.setText(ciudad.getSelectedItem());
    }
    if(e.target==paises ) {
        etiqueta1.setText(paises.getSelectedItem());
    }
    return true;
}
}

```

Ejemplo 5: programa que utilice dos componentes gráficos de usuario TextArea y se pueda marcar un texto del primero y copiarla en el segundo.

```

import java.awt.*;
import java.applet.*;
public class Texto extends Applet
{
    private TextArea t1,t2;
    private Button b;
    public void init()
    {
        String s="El amor el es principio\n"+
            " de todo la razón de todo\n "+
            "El fin de todo...";
        t1=new TextArea(5,20);
        t1.setText(s);//agregar texto al area 1
        t2=new TextArea(5,20);
        b=new Button("Copiar...");
        setLayout(new FlowLayout(FlowLayout.LEFT,5,5));
        add(t1);
        add(b);
        add(t2);
    }
    public boolean action(Event e, Object o)
    {
        if(e.target==b)
        {
            t2.setText(t1.getSelectedText());
            return true;
        }
        return false;
    }
}

```

Ejemplo 6: programa que utilice el componente gráficos de usuario ScrollBar, para seleccionar los diferentes colores de Java.

```

import java.awt.*;
import java.applet.*;
public class MyApp extends Applet
{
    Scrollbar rojo, verde, azul;
    Canvas c;
    Panel p;
    TextField t1, t2, t3;
    Label l1;
    int ro=0, ve=0, az=0;
    public void init()
    {
        l1 = new Label("Colores personalizados");
        t1 = new TextField (10);
        t1.setEditable(false);
        t2 = new TextField (10);
        t2.setEditable(false);
        t3 = new TextField (10);
        t3.setEditable(false);
        rojo=new Scrollbar(Scrollbar.VERTICAL, 100, 0, 0, 256);
        verde=new Scrollbar(Scrollbar.VERTICAL, 100, 0, 0, 256);
        azul = new Scrollbar(Scrollbar.VERTICAL, 100, 0, 0, 256);
        c=new Canvas();
        c.resize(100,100);
        p=new Panel();// marco
        p.add("Center",rojo);
        p.add("Center",verde);
        p.add("Center",azul);
        p.add("Center",c);
        add("Center",p);
        add(t1);
        add(t2);
        add(t3);
    }
    public boolean handleEvent(Event e)
    {
        if(e.target instanceof Scrollbar){
            if(e.target==rojo)
                ro = rojo.getValue();
            else if (e.target == verde)
                ve = verde.getValue();
            else
                az = azul.getValue();
            c.setBackground(new Color(ro, ve, az));
            t1.setText("rojo = "+ ro);
            t2.setText("verde = "+ ve);
            t3.setText("azul = "+ az);
            c.repaint();
            return true;
        }
        return super.handleEvent(e);
    }
}

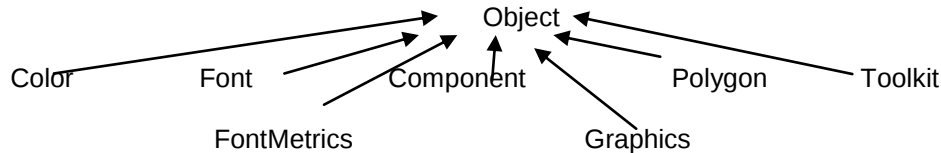
```

6.12 Ejercicios

1. Hacer un programa que capture en una caja de texto una palabra, por medio de un botón imprima en un área de texto la cantidad de letra del alfabeto que contiene dicha palabra.
2. Hacer un programa que al pulsar un botón aparezca una caja de texto y se capture una cadena, y dicha cadena la imprima en una etiqueta en forma inversa.
3. Hacer un programa que permita escoger una opción de un cuadro combinado, e imprimirla en un área de texto.
4. Hacer un programa que permita escoger una opción de un cuadro combinado, e insertarla a otro cuadro combinado por medio de un botón.
5. Hacer un programa que permita escoger una opción de un cuadro combinado, e insertarla a una lista por medio de un botón.
6. Hacer un programa que contenga 5 cuadros de verificación, al escoger una opción, dicha opción se deberá ir insertando en un cuadro de lista.
7. Hacer un programa que utilice un Scrollbar, que permita seleccionar una letra del alfabeto e imprimirla en un área de texto.
8. Hacer un programa que utilice un Scrollbar, que permita seleccionar un numero entre 1 y 255, dicho numero se deberá imprimir en un cuadro de lista, así como su valor en ascci.
9. Hacer un programa que permita capturar en un campo de texto un valor entero e imprima en una etiqueta el valor correspondiente en letras. (nota: valor entre 1 y 999).
10. Hacer un programa que permita capturar en un campo de texto una cadena de caracteres e imprima en un área de texto, el valor ascci de cada una de las letras que contenga la cadena.

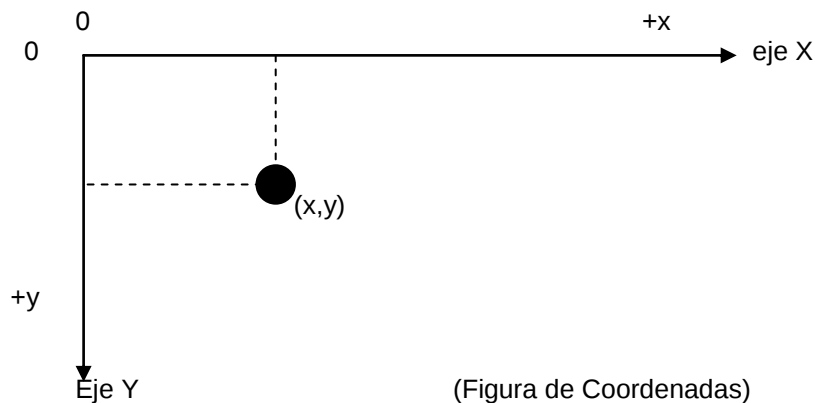
7. Gráficos en Java

Las clases que componen el paquete **java.awt** contiene las capacidades gráficas para dibujar en la pantalla, una muestra de la jerarquía de clases de java.awt es:



La clase **Color** contiene métodos y constantes para manipular colores. La clase **Font** (fuente) contiene métodos y constantes para manipular tipos de letra. La clase **FontMetrics** (métricas de fuentes) contiene métodos para obtener información acerca de las fuentes. La clase **Polygon** contiene métodos para crear polígonos. La clase **Graphics** contiene métodos para dibujar cadenas, líneas, rectángulos y otras figuras. La clase **Toolkit** (juego de herramientas) proporciona métodos para obtener información gráfica de un sistema.

El sistema de coordenadas de Java, identifica los puntos posibles en la pantalla. Por omisión la esquina superior izquierda de la pantalla tiene coordenadas (0,0). Un par de coordenadas se componen de una coordenada x (coordenada horizontal) y una coordenada y (coordenada vertical). La coordenada x es la distancia horizontal hacia la derecha partiendo de la esquina superior izquierda. La coordenada y es la distancia vertical hacia abajo partiendo de la esquina superior izquierda. El eje x describe todas las coordenadas horizontales y el eje y todas las coordenadas verticales.



7.1 Graficando cadenas de caracteres, caracteres y bytes

Para graficar cadena de caracteres, caracteres, bytes, se utilizan los siguientes métodos de la clase **Graphics**:

- **drawString**: Dibuja un String. Este método recibe tres argumentos: el String que va a dibujar, una coordenada x y una coordenada y.
- **drawChar**: dibuja una serie de caracteres. Este método recibe cinco argumentos. El primer argumento es un arreglo de caracteres. El segundo argumento especifica el subíndice en el arreglo del primer carácter que se dibujará. El tercer argumento

especifica el numero de caracteres a dibujar. Los dos últimos argumentos especifican las coordenadas donde se comenzara a dibujar.

- **drawBytes:** dibuja una serie de bytes. Recibe cinco argumentos. El primero es un arreglo de Bytes. El segundo argumento especifica el subíndice en arreglo del primer byte que se dibujara. El tercer argumento especifica el numero de elementos a dibujar. Los dos últimos argumentos especifican las coordenadas donde se comenzara a dibujar.

7.2 Colores en Java

Todos los colores se crean a partir de un valor RGB (red/green/Blue). Un valor RGB se crea con tres partes, cada una de las cuales puede ser un entero en el intervalo de 0 a 255 ó un valor flotante en el intervalo de 0.0 a 1.0. La primera define la cantidad de rojo, la segunda la cantidad de verde y la tercera la cantidad de azul.

- **Método getRed():** devuelve la cantidad de rojo.
- **Método getGreen():** devuelve la cantidad de verde.
- **Método getBlue():** devuelve la cantidad de azul.

El color vigente se cambia con:

```
g.setColor(new Color(red,green,blue));
```

El método setColor establece como color vigente el objeto COLOR que se construyó con los valores red, green, blue.

Otra forma de establecer el color vigente es con:

```
private Color variable;
```

En el método init()

```
variable=Color.blue // color a mostrar
```

Y en el método paint()

```
g.setColor(variable)
```

7.3 Los tipos de Fuente en Java

La clase Font contiene los métodos y constantes de fuentes. El constructor Font recibe tres argumentos: nombre de la fuente, estilo de la fuente y tamaño de la fuente. El nombre de la fuente es cualquier fuente que reconozca el sistema, como Courier, Helvética y TimesRoman. El estilo de la fuente puede ser Font.PLAIN, Font.ITALIC, Font.BOLD. Los estilos de la fuente se pueden combinar (Font.ITALIC + Font.BOLD) . El tamaño de la fuente se mide en puntos. Un punto es 1/72 de pulgada.

El método setFont establece la fuente vigente. Este recibe un objeto Font como argumento.

Con frecuencia es necesario obtener información acerca de la fuente actual, como el nombre de la fuente, su estilo y su tamaño. Los métodos Font que se utilizan para obtener información sobre las fuentes son:

- **getStyle:** devuelve un entero que representa el estilo vigente

```
variable=fuente.getStyle();
```

- **getSize**: devuelve un entero que representa el tamaño vigente
variable=fuente.getSize();
- **getName**: devuelve un entero que representa el nombre de fuente vigente
variable=fuente.getName();
- **getFamily**: devuelve la familia de la fuente vigente
g.drawString("DE familia:+ fuente.getFamily(),x,y);

Otros métodos permiten obtener las métricas de las fuentes.

- **getFont()** (Graphics): devuelve un objeto Font que representa a la fuente actual.
g.getFont().toString();
- **getFontMetrics()** (FontMetrics): Devuelve las métricas de fuentes vigentes.
- **getFontlist()** (Toolkit): crea una lista de las fuentes disponibles de un sistema
- **getAscent()** (FontMetrics): Devuelve un valor que representa la subida en puntos
g.getFontMetrics().getAscent();
- **getDescent()** (FontMetrics): Devuelve un valor que representa la bajada en puntos
g.getFontMetrics().getDescent();
- **getLeading()** (FontMetrics): Devuelve un valor que representa la interlinea de una fuentes
g.getFontMetrics().getLeading();
- **getHeight()** (FontMetrics): Devuelve un valor que representa la altura de una fuente en puntos
g.getFontMetrics().getHeight();
- **getDefaultToolkit()** (Toolkit): Obtiene el juego de herramientas por omisión del sistema actual.

Ejemplo 1: programa que manipule los colores de tipo entero y los tipos de fuentes de java

```
import java.awt.*;
import java.applet.*;
public class ColorCaracter extends Applet
{
    int rojo,verde,azul;
    Font f1,f2,f3;
    String s="Colombia patria querida";
    char c[]={'p','e','r','r','i','t','o'};
    byte b[]={'c','a','s','t','o','r'};
    Color co;
    public void init()
    {
        rojo=100;
        azul=255;
        verde=125;
        f1= new Font("TimesRoman",Font.BOLD,12);
        f2= new Font("Courier",Font.ITALIC,24);
        f3= new Font("Helvética",Font.PLAIN,14);
        co=Color.blue;
    }
    public void paint(Graphics g)
    {
        g.setFont(f1);
        g.setColor(co);
        g.drawString(s, 20, 20);
    }
}
```

```

        g.setFont(f2);
        g.drawChars(c,2,3,20,45);
        g.setFont(f3);
        g.drawBytes(b,0,5,20,60);
    }
}

```

Ejemplo 2: programa que manipule los colores de tipo flotante y los tipos de fuentes de java

```

import java.awt.*;
import java.applet.*;
public class ColorCaracter extends Applet
{
    int rojo,verde,azul;
    Font f1,f2,f3;
    String s="Colombia patria querida";
    char c[]={'p','e','r','r','i','t','o'};
    byte b[]={'c','a','s','t','o','r'};
    Color co ;
    public void init()
    {
        rojo=0.34f;
        azul=0.88f;
        verde=0.20f;
        f1= new Font("TimesRoman",Font.BOLD,12);
        f2= new Font("Courier",Font.ITALIC,24);
        f3= new Font("Helvética",Font.PLAIN,14);
        co=Color.red;
    }
    public void paint(Graphics g)
    {
        g.setFont(f1);
        g.setColor(co);
        g.drawString(s, 20, 20);
        g.setFont(f2);
        g.drawChars(c,2,3,20,45);
        g.setFont(f3);
        g.drawBytes(b,0,5,20,60);
    }
}

```

Ejemplo 3: Programa que utilice algunos de los métodos de la clase Font

```

import java.awt.*;
import java.applet.*;
public class ColorCaracter3 extends Applet
{
    Font f1,f2;
    int rojo=121, verde=215, azul=88 ;
    public void init()
    {
        f1=new Font("TimesRoman",Font.BOLD,18);

```

```

        f2=new Font("Helvetica",Font.PLAIN,16);
    }
    public void paint(Graphics g)
    {
        g.setColor(new Color(rojo,verde,azul));
        g.setFont(f1);
        int estilo, tamaño;
        String s, nombre;
        estilo =f1.getStyle();
        if(estilo==Font.PLAIN)
            s="Normal ";
        else if(estilo==Font.BOLD)
            s="Negrita ";
        else if(estilo==Font.ITALIC)
            s="Cursiva ";
        else //bold+italic
            s="Negrita cursiva ";
        tamaño=f1.getSize(); //determina el tamaño actual de la fuente
        s+=" de "+tamaño+"puntos";
        nombre=f1.getName(); //determina el nombre de la fuente
        g.drawString("La fuentes es:"+nombre,20,60);
        g.drawString("el tipo de fuente es: "+s ,20,80);
        g.drawString("pertenece a la familia:"+f1.getFamily(),20,100);
        g.setColor(new Color(28,80,125));
        g.setFont(f2);
        g.drawString(" Lista de Fuentes en el sistema",20,160);
        String fuente[]=Toolkit.getDefaultToolkit().getFontList();
        for(int i=0;i<fuente.length;i++)
            g.drawString(fuente[i],10,i*10+200);
    }
}

```

7.4 Dibujo de figuras en Java

Con la clase Graphics se pueden dibujar líneas, rectángulos, óvalos, arcos y polígonos. Los métodos que utilizaremos serán:

- **drawLine(x, y, x1, y1):** sirve para dibujar una línea, sus argumentos son:

- x = coordenada x del primer punto
- y = coordenada y del primer punto
- x1 = coordenada x del segundo punto
- y1 = coordenada y del segundo punto.

- **drawRect(x, y, w, a):** sirve para dibujar un rectángulo, sus argumentos son:

- x = coordenada x superior izquierda
- y = coordenada y superior izquierda
- w = ancho
- a = altura.

- **fillRect(x,y,w,a):** sirve para rellenar un rectángulo, sus argumentos son:

- x = coordenada x superior izquierda
 - y = coordenada y superior izquierda
 - w = ancho
 - a = altura.
- **ClearRect(x,y,w,a):** sirve para dibuja en el color de segundo plano vigente sobre el rectángulo especificado, sus argumentos son:
- x = coordenada x superior izquierda
 - y = coordenada y superior izquierda
 - w = ancho
 - a = altura.
- **drawRoundRect(x,y,w,a,aw,aa):** sirve para dibujar un rectángulo con esquinas redondeadas, sus argumentos son:
- x = coordenada x
 - y = coordenada y
 - w = ancho
 - a = altura.
 - aw = ancho del arco
 - aa = altura del arco
- **fillRoundRect(x,y,w,a,aw,aa):** sirve para rellenar un rectángulo con esquinas redondeadas, sus argumentos son:
- x = coordenada x
 - y = coordenada y
 - w = ancho
 - a = altura.
 - aw = ancho del arco
 - aa = altura del arco
- **draw3DRect(x,y,w,a, b):** sirve para dibujar un rectángulo tridimensional, sus argumentos son:
- x = coordenada x de esquina superior izquierda
 - y = coordenada y de esquina superior izquierda
 - w = ancho
 - a = altura.
 - b = realizada si es true y sumida si es false
- **fill3DRect(x,y,w,a, b):** sirve para rellenar un rectángulo tridimensional, sus argumentos son:
- x = coordenada x de esquina superior izquierda
 - y = coordenada y de esquina superior izquierda
 - w = ancho
 - 7. a = altura.
 - 8. b = realizada si es true y sumida si es false
- **drawOval(x,y,w,a):** sirve para dibujar un ovalo, sus argumentos son:
- x = coordenada x
 - y = coordenada y
 - w = ancho
 - a = altura.
- **fillOval(x,y,w,a):** sirve para rellenar un ovalo, sus argumentos son:

- x = coordenada x
 - y = coordenada y
 - w = ancho
 - a = altura.
- **drawArc(x,y,w,a,ai,af):** sirve para dibujar un arco, sus argumentos son:
- x = coordenada x
 - y = coordenada y
 - w = ancho del arco
 - a = altura. Del arco
 - ai = ángulo inicial
 - af = ángulo final
- **fillArc(x,y,w,a,ai,af):** sirve para rellenar un arco, sus argumentos son:
- x = coordenada x
 - y = coordenada y
 - w = ancho del arco
 - a = altura. Del arco
 - ai = ángulo inicial
 - af = ángulo final

Ejemplo 1: programa que utilice los métodos de dibujo de figuras geométricas, con la clase Graphics

```
import java.awt.*;
import java.applet.*;
public class Dibujos extends Applet
{
    public void paint(Graphics g)
    {
        g.setColor(Color.red);
        // trazar una línea
        g.drawLine(330,10,400,95);
        // Dibujar un rectángulo en la posición 20,25
        g.setColor(Color.blue);
        g.drawRect(20,25,100,100);
        // Dibujar el rectángulo relleno en la posición 150,25
        g.setColor(Color.yellow);
        g.fillRect(150,25,100,100);
        // Dibujar un rectángulo redondeado en 20,155
        g.setColor(Color.black);
        g.drawRoundRect(20,155,50,50,10,20);
        // Dibujar un rectángulo redondeado relleno en 120,155
        g.setColor(Color.magenta);
        g.fillRoundRect(120,155,80,100,70,70);
        // Dibujar un rectángulo redondeado en 220,185
        g.setColor(Color.blue);
        g.drawRoundRect(220,185,100,20,70,70);
        // Dibujar un cuadrado relleno en 340,155
        g.setColor(Color.orange);
        g.fillRoundRect(340,155,80,80,0,0);
        // Dibujar un circulo en 44,155
        g.drawRoundRect(440,155,50,50,50,50);
        // Dibujar un rectángulo en tercera dimensión en
```

```

        g.draw3DRect(20,350,100,100,true);
// Dibujar un rectángulo en tercera dimensión en
        g.draw3DRect(140,350,100,100,false);
// Dibujar un rectángulo en tercera dimensión en
        g.fill3DRect(260,350,100,100,true);
// Dibujar un rectángulo en tercera dimensión en
        g.fill3DRect(400,350,100,100,false);
        g.setColor(Color.red);
// Dibujar ovalo sin relleno
        g.drawOval(380,15,100,70);
// Dibujar un ovalo con relleno
        g.fillOval(500,15,70,130);
// dibujar un arco
        g.drawArc(580,55,80,80,0,180);
        g.drawArc(550,100,80,80,0,110);
//dibujar un arco con relleno
        g.fillArc(550,355,70,80,0,270);
        g.fillArc(400,70,70,80,0,-110);
        g.fillArc(550,115,80,140,0,-360);
    }
}

```

7.4.1 Polígonos

Los polígonos son figuras de varios lados, para dibujar o rellenar un polígono se requieren tres argumentos:

- un arreglo de enteros que contiene coordenadas x
- un arreglo de enteros que contiene coordenadas y
- Y el número de puntos del polígono

Si se especifica un punto final diferente del primer punto, se produce un polígono abierto, en el que el último punto no está conectado con el primero.

- **drawPolygon(xv[],yv[],p):** sirve para dibujar un polígono, sus argumentos son:

- x v[] = vector de coordenadas x
- yv[] = vector de coordenadas y
- p = números de puntos

- **fillPolygon(xv[],yv[],p):** sirve para rellenar un polígono, sus argumentos son:

- x v[] = vector de coordenadas x
- yv[] = vector de coordenadas y
- p = números de puntos

Ejemplo: Programa que utilice los métodos de la clase Polygon para dibujar polígonos

```

import java.awt.*;
import java.applet.*;
public class Polígono extends Applet
{
// coordenadas primer polígono
private int xvalor[]={20,40,50,30,20,15,20};
private int yvalor[]={20,20,30,50,50,30,20};
//coordenadas segundo polígono

```

```

private int xvalor2[]={70,90,100,80,70,65,60,70};
private int yvalor2[]={70,70,80,100,100,80,60,70};
//coordenadas tercer polígono
private int xvalor3[]={120,140,150,190};
private int yvalor3[]={10,40,50,30};
//crea referencias a polígonos
private Polygon p1,p2,p3;
public void init()
{
    // crear objetos polígonos
    p1=new Polygon(xvalor, yvalor, 7);
    p2=new Polygon();
    p3=new Polygon();
    //agregar puntos a p2
    p2.addPoint(165,105);
    p2.addPoint(175,120);
    p2.addPoint(270,170);
    p2.addPoint(200,190);
    p2.addPoint(130,150);
    p2.addPoint(165,105);
    //agregar puntos a p3
    p3.addPoint(240,50);
    p3.addPoint(260,70);
    p3.addPoint(250,90);
}

public void paint(Graphics g)
{
    //dibujar un polígono de 8 puntos
    g.drawPolygon(xvalor2,yvalor2,8);
    //dibujar un polígono
    g.drawPolygon(p1);
    //dibujar un polígono relleno
    g.fillPolygon(p2);
    //dibujar un polígono relleno
    g.fillPolygon(p3);
    //dibujar un polígono relleno de 4 puntos
    g.fillPolygon(xvalor3,yvalor3,4);
    // copiar una área de 100 x 100 a 140,10
    g.copyArea(0,0,100,100,140,10);
}
}

```

7.5 Marcos (Frame)

Un marco es una ventana con barra de título y borde. Los marcos se crean con la clase Frame, que extiende de la clase Windows. La clase Windows contiene métodos para manejar ventanas. Los marcos generalmente se usan para construir aplicaciones con ventanas (programas que no requieren de un Navegador para ejecutarse), aunque también se pueden ejecutar en un Applet. Algunos métodos son:

- `resize()`: permite obtener el tamaño del marco
`nombre_marco.resize(300,200);`

- `setResizable(boolean)`: permite redimensionar ó no un marco, por omisión los marcos son redimensionables. El valor boolean *false* no permite la redimensión.
`nombre_marco.setResizable(false);`
- `setTitle`: permite colocarle un titulo al marco
- `show()`: permite Mostrar un marco
`nombre_marco.show();`
- `hide()`: oculta un marco
- `dispose()`: libera la memoria utilizada por un marco.

Ejemplo 1: Hacer un programa que muestre un marco desde un Applet con botones que permita cambiar de color el fondo del marco.

```
import java.awt.*;
import java.applet.*;

public class Marcos extends Applet
{
    private MostrarMarco f;
    private Button b;
    public void init()
    {
        String s="Pulse para ver un marco";
        b=new Button(s);
        add(b);
    }
    public boolean action (Event e, Object o)
    {
        if(e.target==b)
        {
            String s="Este marco realiza una acción";
            if(f!=null)
            {
                f.hide();
                f.dispose();
            }
            f=new MostrarMarco(s);
            f.resize(300,200);
            f.setResizable(false);
            f.show();
        }
        return true;
    }
}

class MostrarMarco extends Frame
{
    private Button a,b,c,d;
    public MostrarMarco(String s)
    {
        super(s);
        a=new Button("Amarillo");
        b=new Button("Rojo");
```

```

c=new Button("Azul");
d=new Button("Verde");
add("North",a);
add("East",b);
add("South",c);
add("West",d);
}
public boolean handleEvent(Event e)
{
    if(e.id==Event.WINDOW_DESTROY)
    {
        hide();
        dispose();
        return true;
    }
    return super.handleEvent(e);
}
public boolean action(Event e, Object o)
{
    if(e.target==a)
        setBackground(Color.yellow);
    else if(e.target==b)
        setBackground(Color.red);
    else if(e.target==c)
        setBackground(Color.blue);
    else
        setBackground(Color.green);
    repaint();
    return true;
}
}

```

Ejemplo 2: Hacer un programa autónomo muestre un marco con botones que permita cambiar de color el fondo del marco.

```

import java.awt.*;
import java.awt.event.*;
public class Marcos3 extends Frame{
    private MostrarMarco f;
    private Button b;
    private int numero;
    private String titulomarco,titulo;
    public Marcos3()
    {
        this("Marcos en Java");
        // metodo para liberar los recursos utilizados por el marco
        this.addWindowListener (new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                dispose();
                System.exit(0);
            }
        });
    }
    public Marcos3(String t)

```

```

{
    setTitle ("Sistematización de Datos");
    String s="Pulse para ver el marco";
    numero=0;
    titulo=t;
    b=new Button(s);
    add("South",b);
    resize(300,100);
    show();
}
public void init()
{
    String s="Pulse para ver un marco";
    b=new Button(s);
    add(b); }
public boolean handleEvent (Event e)
{
    if(e.id==Event.WINDOW_DESTROY)
    {
        action(e,e.arg);
        return true;
    }
    else if(e.id==Event.WINDOW_DESTROY)
    {
        borrar(this);
        System.exit(0);
        return true;
    }
    return super.handleEvent(e);
}
public boolean action (Event e, Object o)
{
    if(e.target==b)
    {
        if(f!=null)
            borrar(f);
        numero++;
        titulomarco=titulo+ " " + String.valueOf(numero);
        f=new MostrarMarco(titulomarco);
    }
    return true;
}
public void borrar (Frame w)
{
    w.hide();
    w.dispose();
}
public static void main(String a[])
{
    Marcos3 mio;
    if(a.length==0)
        mio=new Marcos3();
    else
        mio=new Marcos3(a[0] );
}

```

```

    }
}
class MostrarMarco extends Frame
{
    private Button a,b,c,d;

    public MostrarMarco(String s)
    {
        super(s);
        a=new Button("Amarillo");
        b=new Button("Rojo");
        c=new Button("Azul");
        d=new Button("Verde");
        add("North",a);
        add("East",b);
        add("South",c);
        add("West",d);
        resize(200,200);
        show();
    }
    public boolean handleEvent(Event e)
    {
        if(e.id==Event.WINDOW_DESTROY)
        {
            hide();
            dispose();
            return true;
        }
        return super.handleEvent(e);
    }
    public boolean action(Event e, Object o)
    {
        if(e.target==a)
            setBackground(Color.yellow);
        else if(e.target==b)
            setBackground(Color.red);
        else if(e.target==c)
            setBackground(Color.blue);
        else
            setBackground(Color.green);
        repaint();
        return true;
    }
}

```

7.6 Menús

Los menús son una parte integral de la interfaz de usuario. Actualmente los menús pueden usarse con objetos Frame. Para trabajar los menús utilizaremos las clases y métodos que a continuación se explican:

- MenuBar: clase que permite crear una barra de menús
- MenuItem: clase que permite crear elementos de un menú

- Menu: clase que permite crear los menús que se agregan a la barra de Menús
- CheckboxMenuItem: clase que permite crear un elemento de menú con casilla de verificación, contiene dos estados (activo ó desactivo).
- setMenuBar: método de la clase Frame que permite fijar la barra de menús en el marco.
- setState(boolean): para determinar el estado de un casilla de verificación (false o true).

Ejemplo: Hacer un programa autónomo que cree un menú y simule un editor de texto donde se pueda cambiar el tipo y el color de la fuente.

```
import java.awt.*;
import java.applet.*;
public class Menucito extends Frame{
    TextArea texto;
    Font fte;
    Color colores;
    CheckboxMenuItem casilla;
    public Menucito()
    {
        super("Editor de Texto");
        texto=new TextArea();
        add("Center",texto);
        fte=new Font("TimerRoman",Font.PLAIN,14);
        setFont(fte);
        texto.setForeground(Color.black);
        //crear la barra de menús
        MenuBar barra=new MenuBar();
        // Crear menús
        Menu archivo=new Menu("Archivo");
        Menu edicion=new Menu("Edición");
        Menu ver=new Menu("Ver");
        Menu fuente=new Menu("Fuente");
        Menu color=new Menu("Color");
        //construir submenus de archivo
        archivo.add(new MenuItem("Nuevo..."));
        archivo.add(new MenuItem("Cerrar"));
        archivo.add(new MenuItem("Salir"));
        //construir submenús de edición
        edicion.add(new MenuItem("Copiar"));
        edicion.add(new MenuItem("Cortar"));
        edicion.add(new MenuItem("Pegar"));
        //construir submenus de color
        color.add(new MenuItem("Negro"));
        color.add(new MenuItem("Azul"));
        //construir submenús de fuentes
        fuente.add(new MenuItem("Times Roman"));
        fuente.add(new MenuItem("Courier"));
        //construir submenus de ver
        ver.add(fuente);
        ver.add(new MenuItem("-"));
        ver.add(color);
    }
}
```

```

        ver.add(new MenuItem("-"));
//Crea un elemento de menú de cuadro de verificación
        casilla=new CheckboxMenuItem("Solo leer");
//se agrega al menú con
        ver.add(casilla);
// se usa el método setState(establecer estado) de
//CheckboxMenuItem para establecer el estado boolean
// de un elemento de menú de casilla de verificación
        casilla.setState(false);
        barra.add(archivo);
        barra.add(edicion);
        barra.add(ver);
// fijar la barra de menú
        setMenuBar(barra);
        resize(300,200);
        show();
    }
    public boolean handleEvent(Event e)
    {
// para destruir el marco de la pantalla (Window_destroy)
        if(e.id==Event.WINDOW_DESTROY)
        {
            hide();
            dispose();
            System.exit(0);
            return true;
        }
        return super.handleEvent(e);
    }
    public boolean action(Event e, Object o)
    {
        if(e.target instanceof MenuItem)
        {
            if(e.arg.equals("Times Roman"))
                fte=new Font("TimesRoman",Font.PLAIN,12);
            else if(e.arg.equals("Courier"))
                fte=new Font("Courier",Font.ITALIC,12);
            else if(e.arg.equals("Negro"))
                colores=Color.black;
            else if(e.arg.equals("Azul"))
                colores=Color.blue;
            else if(e.arg.equals(casilla.getLabel()))
                texto.setEditable(!casilla.getState());
            texto.setForeground(colores);
            texto.setFont(fte);
        }
        return true;
    }
    public static void main(String args[])
    {
        Menucito e;
        e=new Menucito();
    }
}

```

7.7 Cuadros de Diálogo

Un cuadro de dialogo es una ventana sin borde y con barra de titulo. Los cuadros de dialogo se usan para obtener información del usuario o para mostrar información al usuario. Estos pueden ser modales o sin modo. Los cuadros de dialogo modales no permiten acceder a ninguna otra ventana de la aplicación en tanto no se cierre el cuadro de dialogo. Un cuadro de dialogo no modal permite acceder a otras ventanas mientras se observa el cuadro de dialogo. Utilizaremos:

- `FileDialog`: clase que permite cargar el cuadro de dialogo respectivo
- `LOAD`: método que permite cargar el cuadro de dialogo Abrir...
- `SAVE`: método que permite cargar el cuadro de dialogo Guardar Como...

Ejemplo: programa que permita mostrar los cuadros de dialogo "Abrir" y "Guardar como".

```
import java.awt.*;
import java.applet.*;

public class Dialogo extends Frame
{
    Contiene a;
    MenuItem item,item2;
    public Dialogo()
    {
        super("Cuadros de Dialogo");
        //crear la barra de menús
        MenuBar barra=new MenuBar();
        Menu archivo= new Menu("Archivo");
        Menu ayuda= new Menu("Ayuda");
        item=new MenuItem("Abrir...");
        item2=new MenuItem("Guardar como...");
        archivo.add(item);
        archivo.add(item2);
        barra.add(archivo);
        barra.add(ayuda);
        // fijar la barra de menú
        setMenuBar(barra);
        resize(200,100);
        show();
    }

    public boolean handleEvent(Event e)
    {
        if(e.id==Event.WINDOW_DESTROY)
        {
            hide();
            dispose();
            System.exit(0);
            return true;
        }
        return super.handleEvent(e);
    }
    public boolean action(Event e, Object o)
```

```

        {
            if(e.target instanceof MenuItem)
                if(e.arg.equals(item.getLabel()))
                {
                    a=new Contiene(this, FileDialog.LOAD);
                }
            else if(e.arg.equals(item2.getLabel()))
                a=new Contiene(this, FileDialog.SAVE);
            return true;
        }
    public static void main(String args[])
    {
        Dialogo d;
        d=new Dialogo();
    }
}

class Contiene extends FileDialog{
    public Contiene(Frame marco, int tipo)
    {
        super(marco,(tipo== FileDialog.LOAD ? "Abrir..." : "Guardar como... ")+" : cuadro de dialogo",tipo);
        resize(400,250);
        show();
    }
}

```

7.8 Ejercicios

1. Escriba un programa que dibuje diez círculos concéntricos. Los círculos deben estar separados por 5 píxeles utilizando drawRoundRect.
2. Escriba un programa que dibuje líneas de longitud y color aleatorio.
3. Escriba un programa que dibuje un espiral utilizando drawArc.
4. Escriba un programa que dibuje 10 palabras con fuente aleatoria de diferente tamaño.
5. Escriba un programa que capture una palabra e imprima dicha palabra en forma aleatoria y de diferente color.
6. Escriba un programa que dibuje un tablero de ajedrez
7. Escriba un programa que dibuje un cubo
8. Escriba un programa que capture el número de triángulos que deben dibujarse. Dicho triángulos deben ser de diferente color.
9. Escriba un programa que lea un par de coordenada, el radio y dibuje el círculo, además de imprimir el diámetro, la circunferencia y el área del círculo.
10. Escriba un programa que simule un protector de pantalla. El programa deberá dibujar 50 líneas al azar y después limpiar la pantalla y viceversa.

8. Manejo de eventos (Mouse y Teclado)

8.1 Eventos de Mouse

Un evento de Mouse ocurre cuando el usuario interactúa con el mouse. Todos los métodos reciben tres argumentos: un evento, una coordenada x y una coordenada y. Las coordenadas especifican donde ocurrió un evento. Los métodos del mouse existentes son:

- mouseDown: se llama cada vez que se pulsa un botón del Mouse.
- mouseUp: se llama cuando se suelta un botón del Mouse.
- mouseMove: para manejar el movimiento del Mouse sin oprimir ningún botón.
- mouseDrag: maneja los movimientos hechos con el botón presionado
- mouseEnter: es llamado cuando el puntero del Mouse entra a un Applet
- mouseExit: es llamado cuando el puntero del Mouse sale de un Applet

Ejemplo 1: programa que utilice los eventos mouseDown y mouseUp, para imprimir las coordenadas del mouse.

```
import java.awt.*;
import java.applet.*;
public class Mouse extends Applet
{
    private int xu,yu;
    private boolean primero;
    private Font f;
    public void init()
    {
        primero=true;
        f = new Font("TimesRoman",Font.BOLD,14);
    }

    public void paint(Graphics g)
    {
        //Estos enunciados no se ejecutaran la primera vez que se invoque paint
        if(primero==false)
        {
            String s="Mouse Arriba en";
            g.setFont(f);
            s+=(" "+xu+" "+yu+"!");
            g.drawString(s,xu+50,yu+50);
        }
    }
    public boolean mouseDown(Event e , int x, int y)
    {
        showStatus("Mouse Abajo en (" +x+" "+y+"");
        return true;
    }
    public boolean mouseUp(Event e , int x, int y)
    {
        primero=false;
        xu=x;
        yu=y;
        repaint();
        return true;
    }
}
```

```

    }
}

```

Ejemplo 2: programa que utilice eventos del mouse, que permita dibujar puntos en un applet.

```

import java.awt.*;
import java.applet.*;
public class PintarMouse extends Applet
{
    private int x,y;
    private boolean primero;
    public void init()
    {
        primero=true;
    }
    public void paint(Graphics g)
    {
        if(!primero)
            g.fillOval(x,y,6,6);
    }
    public void update(Graphics g)
    {
        paint(g);
    }
    public boolean mouseDrag(Event e, int xx, int yy)
    {
        x=xx;
        y=yy;
        primero =false;
        repaint();
        return true;
    }
}

```

Ejemplo 3: Programa que permita dibujar un ovalo con el mouse en un mantel (Canvas).

```

import java.awt.*;
import java.applet.*;

class MiOvalo extends Canvas
{
    private int ancho, altura;
    private int x,y;
    public void paint(Graphics g)
    {
        g.drawOval(x,y,ancho,altura);
    }
    public boolean mouseDown(Event e, int x1,int y1)
    {
        x=x1;
        y=y1;
        return true;
    }
    public boolean mouseUp(Event e, int x1, int y1)
    {
        ancho=Math.abs(x1-x);
    }
}

```

```

        altura=Math.abs(y1-y);
        x=Math.min(x,x1);
        y=Math.min(y,y1);
        repaint();
        return true;
    }
}
public class MoverOvalo extends Applet
{
    private MiOvalo c;
    public void init()
    {
        c=new MiOvalo();
        c.resize(300,200);
        c.setBackground(Color.yellow);
        add(c);
    }
}

```

Ejemplo 4: programa que permita dibujar 10 puntos en la pantalla

```

import java.awt.*;
import java.applet.*;
public class BolitasMouse extends Applet
{
    final int puntos =10;
    int xbolita[]=new int[puntos];
    int ybolita[]=new int[puntos];
    int cuentabolitas=0;

    public void init()
    {
        setBackground(Color.white);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.blue);
        for(int i=0; i<cuentabolitas;i++)
            g.fillOval(xbolita[i]-10,ybolita[i]-10,20,20);
    }
    public boolean mouseDown(Event e , int x, int y)
    {
        if(cuentabolitas<puntos)
        {
            adicionarpuntos(x, y);
            return true;
        }
        else
            return false;
    }
    public void adicionarpuntos(int x, int y)
    {
        xbolita[cuentabolitas]=x;
        ybolita[cuentabolitas]=y;
    }
}

```

```

        cuentabolitas++;
        repaint();
    }
}

```

8.2 Eventos del Teclado

Cada vez que el usuario presiona o suelta una tecla se genera un evento de teclado. Los métodos de Teclado son:

- keyDown: ocurre cuando se pulsa una tecla
- keyUp: ocurre cuando se suelta una tecla

Ejemplo 1: programa que utilizando eventos de teclado verifique la tecla que fue pulsada por un usuario..

```

import java.awt.*;
import java.applet.*;
public class Teclado extends Applet
{
    private Font f;
    private String letra;
    private boolean primero;
    public void init()
    {
        f=new Font("Courier", Font.BOLD,72);
        primero=true;
    }

    public void paint(Graphics g)
    {
        g.setFont(f);
        if(!primero)
            g.drawString(letra, 75, 70);
    }
    public boolean keyDown(Event e, int llave)
    {
        letra=String.valueOf((char)llave);
        primero=false;
        repaint();
        return true;
    }
    public boolean keyUp(Event e, int llave)
    {
        showStatus("Tecla pulsada :"+(char)llave+".");
        return true;
    }
}

```

Ejemplo 2: programa que permita identificar si lo pulsado es una tecla especial o es una tecla normal.

```
import java.applet.*;
import java.awt.*;
public class Dialogo extends Applet
{
    Font H,T,C,S;
    String letra;
    boolean primero;
    TextField texto,a;
    public void init()
    {
        H=new Font("Helvetica", Font.PLAIN, 92);
        T=new Font("TimesRoman", Font.BOLD, 92);
        C=new Font("Courier", Font.ITALIC, 92);
        S=new Font("Symbol", Font.ITALIC, 92);
        primero= true;
        texto=new TextField(25);
        texto.setEditable(false);
        add(texto);
    }

    public boolean handleEvent(Event e)
    {
        //determina cual evento de tecla ocurri 
        switch(e.key)
        {
            case Event.TAB:
                texto.setText("tecla tabulador");
                return true;
            case Event.ENTER:
                texto.setText("tecla Enter");
                return true;
            case Event.PAUSE:
                texto.setText("tecla PAUSE");
                return true;
            case Event.INSERT:
                texto.setText("tecla insert");
                return true;
            case Event.DELETE:
                texto.setText("tecla Suprimir");
                return true;
            case Event.PGUP:
                texto.setText("tecla Repag");
                return true;
            case Event.PGDN:
                texto.setText("tecla Avpag");
                return true;
            case Event.END:
                texto.setText("tecla Fin");
                return true;
            case Event.RIGHT:
                texto.setText("tecla de flecha derecha ");
                return true;
        }
    }
}
```

```

case Event.LEFT:
texto.setText("tecla de flecha izquierda");
return true;
case Event.UP:
texto.setText("tecla de flecha arriba");
return true;
case Event.DOWN:
texto.setText("tecla de flecha de abajo");
return true;
case Event.HOME:
texto.setText("tecla de inicio");
return true;
case Event.F1:case Event.F2:
case Event.F3:case Event.F4:
case Event.F5:case Event.F6:
case Event.F7:case Event.F8:
case Event.F9:case Event.F10:
case Event.F11:case Event.F12:
texto.setText("Una tecla de funcion F1-F12");
return true;
default:
texto.setText("no es una tecla especial");
letra=String.valueOf((char)e.key);
primero=false;
repaint();
return true;
}
}
public void paint(Graphics g)
{
g.drawString("se pulso ",75,90);
g.setColor(Color.darkGray);
g.fillRect(40,100,400,110);
g.setColor(Color.lightGray);
g.fillOval(20,100,440,110);
g.setColor(Color.magenta);
g.fillRect(40,100,400,10);
g.fillRect(40,200,400,10);
g.setColor(Color.yellow);
g.setFont(H);
if(!primero)
{g.drawString(letra,100,180);}
g.setColor(Color.blue);
g.setFont(T);
if(!primero)
{g.drawString(letra, 200,180);}
g.setColor(Color.red);
g.setFont(C);
if(!primero)
{g.drawString(letra, 300,180);}
}
}

```

8.3 Ejercicios

1. Escriba un programa que permita a un usuario dibujar un rectángulo con el Mouse. Muestre en la barra de estado el área del triángulo, utilice la formula: $\text{Área} = \text{ancho} * \text{altura}$.
2. Escriba un programa que permita al usuario escoger las siguientes figuras: óvalos, arcos, líneas, rectángulos. La figura escogida deberá dibujarse y en la barra de estado deberá imprimir las coordenadas del Mouse.
3. Escriba un programa que permita dibujar una figura por medio del teclado. Las figura deberá dibujar al pulsar las siguientes teclas: c dibuja un círculo, o dibuja un ovalo, r dibuja un rectángulo, y l dibuja una línea.
4. Escriba un programa que permita dibujar una figura. La figura se debe escoger por medio de un cuadro combinado (Choice), la opción de relleno por medio de botones de opcion (CheckBox), el color deseado por medio de un List (lista).
5. Escriba un programa que dibuje un cuadrado, Conforme el Mouse se mueva el cuadrado deberá dibujarse en la dirección del Mouse.
6. Escriba un programa que permita escoger una figura, por medio de las teclas: r rellenar la figura, s sin relleno, n color de la figura naranja, o color de la figura rojo, a color de la figura azul.
7. Hacer un programa que permita la usuario escoger la figura a dibujarse 20 veces en forma aleatoria.
8. hacer un programa que muestre en 10 cuadros cada uno con color diferente. El usuario por medio de un list escogerá una figura. Con el Mouse el usuario seleccionará un cuadro y la figura escogida deberá cambiar al color escogido.
9. Hacer un programa que permita dibujar con el Mouse. Además incluya una barra de herramientas que permita cambiar lo dibujado por otro color. La barra de herramientas se debe generar con botones de opción.
10. Hacer un programa que permita dibujar una figura. Con el Mouse el usuario podrá escoger una nueva posición y la figura deberá redibujarse en esta nueva posición.

9. Programación Orientada a Objetos con Java

La Programación Orientada a Objetos (P.O.O) encapsula datos (atributos) y métodos (comportamientos) en objetos y trata de encontrar solución a problemas utilizando los siguientes conceptos:

- **Objetos:** entidades provistas de datos (propiedades, atributos) y comportamiento (funcionalidad, programas, métodos). Corresponden a los objetos reales del mundo que nos rodea.
- **Abstracción:** Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características.
- **Encapsulación:** También llamada "ocultación de la información", esto asegura que los objetos no pueden cambiar el estado interno de otros objetos de manera inesperada; solamente los propios métodos internos del objeto pueden acceder a su estado. Cada tipo de objeto expone una interfaz a otros objetos que especifica cómo otros objetos pueden interactuar con él.
- **Clases:** conjuntos de objetos que comparten propiedades y comportamientos.
- **Polimorfismo:** programas diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, aunque el significado del programa varíe según el objeto al que se aplica.
- **Herencia:** Organiza y facilita el polimorfismo y la encapsulación permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementar su comportamiento.
- **Método:** es un programa asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena mediante un "mensaje".
- **Mensaje:** una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros.
- **Propiedad, atributo o variable:** datos asociados a un objeto o a una clase de objetos.

La P.O.O. expresa un programa como un conjunto de objetos, que se comunican entre ellos para realizar tareas; Siendo un objeto una unidad que contiene datos y los métodos que operan sobre esos datos. Los objetos tienen la propiedad de ocultamiento de la información, esto significa que los objetos saben comunicarse entre sí a través de interfaces (normalmente no se permite que un objeto sepa como están implementados otros objetos). La unidad de programación de Java es la clase con la cual se crean objetos con características similares.

Los objetos pueden ser activados mediante la recepción de mensajes. Un mensaje es simplemente una petición para que un objeto se comporte de una determinada manera, ejecutando un método. La técnica de enviar mensajes se conoce como paso de mensajes.

Ejemplo: Se tiene un objeto obj1 con los siguientes datos: nombre_alumno y curso, con los métodos: leer_nombre e imprimir. Si el objeto obj1 recibe el mensaje imprimir, esto se expresa:

```
obj1.imprimir()
```

La sentencia anterior se lee: "enviar mensaje imprimir al objeto obj1". El objeto obj1 reacciona al mensaje ejecutando el método de igual nombre que el mensaje.

Una clase es un tipo definido por el usuario que determina las estructuras de datos y las operaciones asociadas con ese tipo. Cada vez que se construye un objeto de una clase, se crea una instancia de esa clase. En general, los términos objetos e instancias de una clase se pueden utilizar indistintamente. Una clase es una colección de objetos similares y un objeto es una instancia de una definición de una clase. La comunicación con el objeto se realiza a través del paso de mensajes. El envío a una instancia de una clase produce la ejecución de un método.

Los datos o variables definidos en una clase se llaman *variables de instancias*. El código esta contenido en los *métodos*. Los métodos y las variables de instancias definidas en una clase son los *miembros* de la clase.

Un programa orientado a objetos es una colección de clases. Necesitará una método principal que cree objetos y comience la ejecución mediante la invocación de sus métodos. Esta organización conduce a separar partes diferentes de una aplicación en distintos archivos. La idea consiste en colocar la descripción de la clase para cada una de ellas en un archivo separado. El método principal también se coloca en un archivo independiente. El compilador ensamblará el programa completo a partir de los archivos independientes en una única unidad. En realidad, cuando se ejecuta un programa orientado a objetos, ocurren tres acciones:

1. Se crean los objetos cuando se necesitan
2. Los mensajes se envían desde unos objetos y se reciben en otros
3. Se borran los objetos cuando ya no son necesarios y se recupera la memoria ocupada por ellos.

Los atributos son las cosas individuales que diferencian una clase de objetos de otros y determinan la apariencia, estado y otras cualidades de esa clase. Por ejemplo una clase Persona podría incluir:

- Nombre :Carlos, José, Rosa, Claudia
- Apellido : Ramírez, Pinzón, Maldonado, Torres
- ciudad_nacimiento : bogota, cali, medellin, bucaramanga
- Edad : 19, 20, 30, 45
- Sexo : Femenino, masculino

Toda clase debe contener una definición de variables o métodos precedida por un modificador de acceso a los miembros; los modificadores de acceso a miembros pueden aparecer varias veces y en cualquier orden en una definición de una clase.

9.1 Modificadores de control de acceso

La encapsulación de datos proporciona el atributo de *control de acceso*. A través de la encapsulación se puede controlar que partes de un programa pueden acceder a los miembros de una clase para prevenir cambios no deseables en los datos. El acceso a un miembro de una clase se determina con el especificador de acceso que modifica su declaración. Los modificadores de acceso de Java son **public**(público), **private**(privado) y **protected** (protegido). El especificador de acceso protected sólo se utiliza cuando se trabaja herencia.

Cuando un miembro de una clase tiene el especificador public, ese miembro puede ser accedido por cualquier parte del programa. Cuando un miembro es private, ese miembro sólo puede ser accedido por otros miembros de esa clase.

Cuando no se utiliza ningún especificador de acceso, por defecto los miembros de una clase son públicos dentro de su propio paquete. Un especificador precede al resto de la especificación de tipo del miembro de la clase, es decir, debe iniciar una sentencia de declaración de un miembro.

Ejemplo:

```
public int j;  
private double x;  
public void mifuncion(int a, int b)  
private int mifuncion()
```

9.2 Constructores

Se llama constructor a un método que tiene el mismo nombre de la clase, el cual se inicializa automáticamente cuando se crea un objeto de la clase. Los constructores pueden recibir argumentos pero no pueden devolver un valor.

9.2.1 Constructores sin parámetros

En algunos casos es necesario crear varios objetos que se inicialicen siempre con valores predeterminados, es decir, cada vez que se cree un nuevo objeto este se inicializaría con los mismos valores.

Ejemplo: Hacer un programa que inicialice las variables alto, ancho y profundidad por medio de un constructor e imprima el volumen de una caja.

- Autónomo

```
class Caja  
{  
    double alto, ancho, profundidad;  
    //definimos el constructor  
    Caja(){  
        alto=10;  
        ancho=20;  
        profundidad=15;  
    }  
    double volumen()  
    {  
        return alto*ancho*profundidad;  
    }  
}  
class PruebaCaja{  
    public static void main(String args[])  
    {  
        Caja caja1=new Caja();  
        double volumen;  
        volumen= caja1.volumen();  
        System.out.println("El volumen de la caja es:"+volumen);  
    }  
}
```

- Applet

```
import java.applet.*;  
import java.awt.*;  
class Caja  
{
```

```

double alto, ancho, profundidad;
//definimos el constructor
Caja(){
    alto=10;
    ancho=20;
    profundidad=15;
}
double volumen()
{
    return alto*ancho*profundidad;
}
}
public class PruebaCajaAp extends Applet{
    Caja caja1;
    double volum;
    public void init()
    {
        caja1=new Caja();
    }
    public void paint(Graphics g)
    {
        volum= caja1.volumen();
        g.drawString("El volumen de la caja es:"+volum,20,20);
    }
}

```

9.2.2 Constructores con parámetros

Existen casos en que es necesario crear varios objetos que se inicialicen con diferentes valores, como se puede apreciar en el ejemplo anterior cada vez que se crea un nuevo objeto este se inicializaría con los mismos valores, la solución es que el constructor tenga parámetros.

Ejemplo : Hacer un programa que inicialice las variables alto, ancho y profundidad por medio de un constructor con parámetros para tres cajas diferentes e imprima el volumen de cada una de las cajas.

- Autónimo

```

class Caja
{
    double alto, ancho, profundidad;
    //definimos el constructor
    Caja(double a, double l, double p){
        alto =a;
        ancho =l;
        profundidad =p;
    }
    double volumen()
    {
        return alto*ancho*profundidad;
    }
}

```

```

class PruebaCaja{
    public static void main(String args[])
    {
        Caja caja1 =new Caja(10,5,12);
        Caja caja2 =new Caja(5,15,12);
        Caja caja3 =new Caja(20,5,20);

        double volumen;
        volumen= caja1.volumen();
        System.out.println("El volumen de la caja 1 es:"+volumen);
        volumen= caja2.volumen();
        System.out.println("El volumen de la caja 2 es:"+volumen);
        volumen= caja3.volumen();
        System.out.println("El volumen de la caja 3 es:"+volumen);
    }
}

```

- **Applet**

```

import java.applet.*;
import java.awt.*;
class Caja
{
    double alto, ancho, profundidad;
    //definimos el constructor
    Caja(double a, double l, double p){
        alto =a;
        ancho =l;
        profundidad =p;
    }
    double volumen()
    {
        return alto*ancho*profundidad;
    }
}
public class PruebaCajaAp extends Applet{
    Caja caja1,caja2,caja3;
    double volum;
    public void init()
    {
        caja1=new Caja(10,5,12);
        caja2=new Caja(12,15,10);
        caja2=new Caja(20,15,20);
    }
    public void paint(Graphics g)
    {
        volum= caja1.volumen();
        g.drawString("El volumen de la caja 1 es :"+volum,20,20);
        volum= caja2.volumen();
        g.drawString("El volumen de la caja 2 es:"+volum,40,20);
        volum= caja3.volumen();
        g.drawString("El volumen de la caja 3 es:"+volum,60,20);
    }
}

```

9.3 Empleo de la referencia this

Algunas veces un método necesita hacer referencia al objeto que lo invocó. Java permite esto con la palabra clave *this*, que puede ser utilizada dentro de cualquier método para referirse al objeto actual., Ejemplo:

```
class Caja{
    double alto, ancho, profundidad;
    Caja(double a, double l, double p){
        this.alto =a;
        this.ancho =l;
        this.profundidad =p;
    }
    double funcion(double valor)
    {
        double alto;
        alto=valor;
        //la clausula this se refiere a la variable double alto de la clase Caja
        this.alto=alto;
    }
}
```

Aunque en este ejemplo la utilización de *this* es redundante es perfectamente valida. En java es ilegal declarar dos variables locales con el mismo nombre dentro de un mismo ámbito. En los parámetros del constructor de la clase Caja se utilizaron nombre de las variables distintas a las que se habían creado anteriormente, con *this* podemos utilizar los mismos nombre en los parámetros del constructor o de cualquier método.

```
    Caja(double alto, double ancho, double profundidad){
        this.alto =a;
        this.ancho =l;
        this.profundidad =p;
    }
```

9.4 Sobrecarga de métodos

La sobrecarga de métodos se refiere a la creación de métodos con el mismo nombre pero con diferente lista de parámetros en la misma clase. Cuando invoca un método sobrecargado, Java utiliza como apoyo el número de parámetros como guía para determinar la versión del método sobrecargado que realmente debe llamar.

Ejemplo 1: Hacer un programa donde se utilice la sobrecarga de métodos para la suma de tres números.

```
import java.awt.*;
import java.applet.*;

class Prueba
{
    int sumar(int a, int b)
```

```

{
    return a+b;
}

int sumar(int a,int b, int c)
{
    return a+b+c;
}
}
public class Sobrecarga extends Applet
{
    Prueba objeto;
    public void init()
    {
        objeto=new Prueba();
    }
    public void paint(Graphics g)
    {
        g.drawString("La suma de 10 + 20 es :"+objeto.sumar(10,20), 20, 20);
        g.drawString("La suma de 10 + 20 + 30 es :"+objeto.sumar(10,20,30), 20, 40);
    }
}

```

9.5 Sobrecarga de constructores

Además de sobrecargas métodos normales, también se pueden sobrecargar los constructores, De hecho las clases que se implementan en la realidad la sobrecarga de constructores es la norma y no la excepción.

```

import java.awt.*;
import java.applet.*;

class Prueba
{
    int x=0,y=0,z=0;
    Prueba(int a, int b)
    {
        x=a;
        y=b;
    }
    Prueba(int a, int b, int c)
    {
        x=a;
        y=b;
        z=c;
    }
    int operaciones()
    {
        return x+y+z;
    }
}
public class SobrecargaConstructores extends Applet
{

```

```

Prueba objeto, objeto1;
public void init()
{
    objeto=new Prueba(10,15);
    objeto1=new Prueba(20,25,30);
}
public void paint(Graphics g)
{
    g.drawString("La suma de 10 + 15 es :"+objeto.operaciones(), 20, 20);
    g.drawString("La suma de 20 + 25+30 es :"+objeto1.operaciones(), 20, 40);
}
}

```

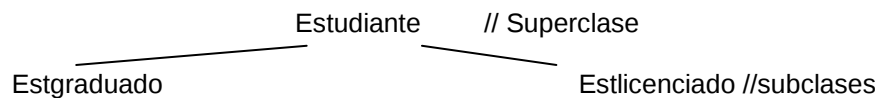
9.6 Herencia y poliformismo

La herencia es la posibilidad de crear una nueva clase a partir de una clase existente, la clase nueva hereda todos los atributos y comportamientos de una clase existente. Después podemos agregar atributos y comportamientos o supeditar los comportamientos de la superclase a fin de adaptar la clase a nuestras necesidades.

El poliformismo nos permite escribir programas para manejar una amplia variedad de clases interrelacionadas existentes y por especificar.

Al crear una nueva clase, en lugar de escribir variables y métodos totalmente nuevos, el programador puede indicar que la nueva clase debe heredar las variables y los métodos de una superclase previamente definida.

Para poder acceder a las variables de la superclase estas previamente deben estar con el modificar de acceso protected:



Ejemplo 1: clase que permite mostrar los valores de las coordenadas de X y Y.

```

public class Puntos{
    protected double x,y;
    public Puntos(double a, double b){
        fijapuntos(a,b);    }
    public void fijapuntos(double a,double b)
    {
        x=a;
        y=b;
    }
    public double mostrarx() {return x;}
    public double mostry() {return y;}
    public String cadena(){
        return "["+x+ "," +y + "];"}
}

```

- Applet que utiliza la clase punto para visualizar las coordenadas X y Y.

```

import java.awt.*;
import java.applet.Applet;
//utiliza la clase Puntos
public class Herencia extends Applet
{
    private Puntos p;

    public void init()
    {
        p=new Puntos(7.3,12.5);
    }
    public void paint(Graphics g)
    {
        g.drawString("La coordenada X es:"+p.mostrarx(),25,25);
        g.drawString("La coordenada Y es:"+p.mostrary(),25,40);
        p.fijapuntos(10,10);
        g.drawString("La nueva posición de p es:"+p.cadena(),25,65);
    }
}

```

Ejemplo 2: Crear una clase Circulo que herede de la clase Punto, donde se pueda obtener el área y el radio de un círculo.

```

public class Circulo extends Puntos{
    protected double radio;
    public Circulo(){
        super(0,0);
        fijarradio(0); }
    public Circulo(double r, double a, double b)
    {
        super(a,b);
        fijarradio(r); }

    public void fijarradio(double r)
    {
        radio=(r>=0.0 ? r:0.0);
    }
    public double obtradio() {return radio;}
    public double area() {return 3.14159*radio*radio;}
    public String cadena(){
        return "Centro="+super.cadena()+ ";Radio = " +radio;}
}

```

- Applet que utilice la clase Circulo para mostrar las coordenadas X y Y, el área y el radio.

```

import java.awt.*;
import java.applet.Applet;
//utiliza la clase Puntos y Circulo
public class Herencia1 extends Applet
{
    private Circulo c;
    public void init()
    {

```

```

        c=new Circulo(7.3,12.5,4.22);
    }
    public void paint(Graphics g)
    {
        g.drawString("La coordenada X es:"+c.mostrarx(),25,25);
        g.drawString("La coordenada Y es:"+c.mostrary(),25,40);
        c.fijarradio(3.4);
        c.fijapuntos(3,4);
        g.drawString("La nueva posición y el radio de c son:",25,65);
        g.drawString(c.cadena(),40,80);
        g.drawString("El área es: "+ c.area(),25,100);
    }
}

```

Ejemplo 3: Crear una clase Cilindro que herede de la clase Circulo, donde se pueda obtener el área, altura y el volumen de un cilindro.

```

public class Cilindro extends Circulo{
    protected double altura;
    public Cilindro(double h, double r, double a,double b)
    {
        super(r,a,b);
        fijaaltura(h); }

    public void fijaaltura(double h)
    {
        altura=(h>=0 ?h:0);
    }
    public double obtaltura() {
        return altura;}
    public double area() {
        return 2*super.area()+2*3.14159*radio*altura;}
    public double volumen (){
        return super.area()*altura;}
    public String cadena(){
        return super.cadena()+ ";Altura = " +altura;}
}

```

Ejemplo: Applet que utilice la clase Cilindro para mostrar las coordenadas X y Y, el radio, la altura y el volumen de un cilindro.

```

import java.awt.*;
import java.applet.Applet;
//utiliza la clase Puntos, circulo, cilindro
public class Herencia2 extends Applet
{
    private Cilindro c;
    public void init()
    {
        c=new Cilindro(7.3,12.5,4.22,2.3);
    }
    public void paint(Graphics g)
    {
        g.drawString("La coordenada X es:"+c.mostrarx(),25,25);

```

```

g.drawString("La coordenada Y es:"+c.mostrary(),25,40);
g.drawString("El radio es:"+c.obtradio(),25,55);
g.drawString("La altura es :"+c.obtaltura(),25,70);
c.fijaaltura(7);
c.fijaradio(3.4);
c.fijapuntos(3,4);
g.drawString("La nueva posición del radio y la altura:",25,85);
g.drawString(c.cadena(),40,115);
g.drawString("El área es: "+ c.area(),25,130);
g.drawString("El Volumen es: "+ c.volumen(),25,145);
}
}

```

9.7 Clases abstractas

En ocasiones se debe definir una superclase que declare una estructura de una abstracción dada sin proporcionar una implementación completa de cada método, es decir a veces crear una superclase que sólo defina una forma generalizada que será compartida por todas las subclases, dejando a cada subclase la tarea de completar los detalles. La superclase determina la naturaleza de los métodos que las clases deben implementar. La clase y los métodos se deben declarar con la palabra clave *abstract*. No pueden haber objetos de una clase abstracta, es decir, que no se pueden crear instancias de dichas clases directamente con el operador *new*, como tampoco se pueden declarar constructores abstract o métodos abstract estáticos. Cualquier subclase de una clase abstracta debe implementar todos los métodos abstractos de la superclase o ser declarada también como abstract.

Ejemplo: Hacer un programa que utilice clases abstractas para hallar el área de un Rectángulo y de un Triángulo.

```

abstract class Figura{
    double diametro1;
    double diametro2;
    Figura(double a, double b)
    {
        diametro1 =a;
        diametro2 =b;
    }
    abstract double area();
}
class Rectangulo extends Figura{
    Rectangulo(double a, double b) {
        super(a, b);
    }
    double area(){
        System.out.println("Dentro del método área para un objeto Rectángulo.");
        return diametro1* diametro2;
    }
}
class Triangulo extends Figura{
    Triangulo(double a, double b)
    {
        super(a, b);
    }
}

```

```

    }
    // Sobrescribir el método área para un Triangulo
    double area()
    {
        System.out.println("Dentro del método área para un objeto Triangulo.");
        return (diametro1* diametro2)/2;
    }
}
class Pruebaabstract{
    public static void main(String args[])
    {
        // Figura f =new Figura (10,15); esta instrucción es incorrecta
        Rectangulo r =new Rectangulo (9,5);
        Triangulo t = new Triangulo(10,8);
        Figura figurita; //esto es correcto, no se crea ningún objeto
        figurita =r;
        System.out.println("El área del Rectángulo es :" +figurita.area());
        figurita =t;
        System.out.println("El área del Triangulo es :" +figurita.area());}}

```

9.8 Interfaces

Los métodos abstractos son útiles cuando se quiere que cada implementación de la clase parezca y funcione igual, pero necesita que se cree una nueva clase para utilizar los métodos abstractos.

Un interface contiene una colección de métodos que se implementan en otro lugar. Los métodos de una clase son *public*, *static* y *final*.

La principal diferencia entre *interface* y *abstract* es que un interface proporciona un mecanismo de encapsulación de los protocolos de los métodos sin forzar al usuario a utilizar la herencia.

Por ejemplo:

```

public interface VideoClip {
    // comienza la reproduccion del video
    void play();
    // reproduce el clip en un bucle
    void bucle();
    // detiene la reproduccion
    void stop();
}

```

Las clases que quieran utilizar el interface VideoClip utilizarán la palabra *implements* y proporcionarán el código necesario para implementar los métodos que se han definido para el interface:

```

class MiClase implements VideoClip {
    void play() {
        <código>
    }
    void bucle() {
        <código>
    }
}

```

```

void stop() {
    <código>
}

```

Al utilizar implements para el interface es como si se hiciese una acción de copiar-y-pegar del código del interface, con lo cual no se hereda nada, solamente se pueden usar los métodos.

La ventaja principal del uso de interfaces es que una clase **interface** puede ser implementada por cualquier número de clases, permitiendo a cada clase compartir el interfaz de programación sin tener que ser consciente de la implementación que hagan las otras clases que implementen el **interface**. La interface debe ser un archivo individual con extensión “.java”.

Ejemplo: Hacer un programa que utilice una interface Figura para hallar el área de un Rectángulo y de un Triángulo.

- **interface Figura**

```

public interface Figura
{
    public double area();
}

```

- **clase PruebaInterface**

```

class Rectangulo implements Figura{
    double d1,d2;
    public Rectangulo(double a, double b) {
        d1=a;
        d2=b;
    }
    public double area(){
        System.out.println("Dentro del método área para un objeto Rectángulo.");
        return d1* d2;
    }
}
class Triangulo implements Figura{
    double d1,d2;
    public Triangulo(double a, double b)
    {
        d1=a;
        d2=b;
    }
    // Sobrescribir el método área para un Triangulo
    public double area()
    {
        System.out.println("Dentro del método área para un objeto Triangulo.");
        return (d1* d2)/2;
    }
}
class PruebaInterface{
    public static void main(String args[])
    {

```

```

// Figura f =new Figura (10,15); esta instrucción es incorrecta
Rectangulo r =new Rectangulo (9,5);
Triangulo t = new Triangulo(10,8);
Figura figurita; //esto es correcto, no se crea ningún objeto
figurita =r;
System.out.println("El área del Rectángulo es :" +figurita.area());
figurita =t;
System.out.println("El área del Triangulo es :" +figurita.area());
}
}

```

9.9 Ejercicios

1. Cree las clases que contenga la siguiente jerarquía:
 - a. Clase Empleado que contiene , nombre, apellidos, cedula
 - b. Clase EmpleadoAsalariado que es una subclase de Empleado y contiene el salario mensual
 - c. Clase EmpleadoPorHoras que es una subclase de empleado y contiene sueldo y número de horas trabajadas
 - d. Clase EmpleadoPorComision que es una subclase de Empleado y contiene tasa de comisiones y ventas brutas
 - e. Clase EmpleadoBaseComision que es una subclase de EmpleadoPorComision y contiene un salario base.

Con la jerarquia anterior crear un programa que muestre toda la información asociada con cada objeto (incluyendo información heredada).

2. Cree un programa que contenga una superclase Figura donde se herede las figuras de rectángulos, cuadrados, triángulos y círculos. El usuario podrá escoger la posición, la figura, y los caracteres de relleno que se usaran para dibujar cada figura.
3. Realizar un programa utilizando herencia que decida si dos números son amigos. Dos números son amigos si la suma de los divisores del primer numero, excluido el, es igual al segundo numero, y viceversa; es decir, si la suma de los divisores del segundo numero, excluido el, es igual al primer numero.
4. Los numeros astromg o cubos perfectos, son aquellos que sumados los cubos de sus digitos nos dan el mismo numero. Por ejemplo 153 es un cubo perfecto , pues $(1)^3 + (5)^3 + (3)^3 = 153$. Escriba un programa utilizando herencia que dado un numero entero , diga si es o no es, un cubo perfecto.
5. Hacer un programa utilizando herencia que lea un numero no mayor de 1000 e imprima ese numero en letras.
6. Crear una clase llamada Rectangulo que contenga los atributos longitud y ancho, cada uno de los cuales debe ser inicializado con 1. debe contener métodos para calcular el perímetro y el área del rectángulo, los métodos deben verificar que la longitud y la altura sean numeros flotantes entre 0.0 y 20.0. Deberá escribir un programa que permita utilizar la clase Rectangulo.
7. Crear un JApplet que dibuje líneas, rectángulos, óvalos al azar. Para este fin, cree un conjunto de clases que permitan dibujar cada una de las figuras. Se Puede utilizar instancias o herencia.
8. Utilizando herencia crear un programa que permita calcular la nomina de un empresa. Se debe tener en cuenta la fecha de ingreso del trabajador para pagarle una bonificación de 100.000 en el mes que cumpla años.

9. Utilizando herencia, crear un programa que permita al usuario escoger una figura unidimensional, la posición, el tamaño y si se rellena o no.
10. Utilizando herencia, hacer un programa que permita calcular la comisión de los vendedores de una empresa de confecciones. Se debe tener en cuenta que existen 10 productos en la empresa y que de acuerdo a un criterio de cantidad la comisión puede ser mayor o menor.

10. Manejo de Excepciones

10.1 Conceptos

La extensibilidad de Java puede aumentar el número y los tipos de errores que pueden ocurrir. Toda clase nueva puede agregar sus propias posibilidades de error.

El propósito del manejo de excepciones es permitir a los programas atrapar y manejar los errores en lugar de dejar que ocurran y sufrir las consecuencias. El manejo de excepciones está diseñado para manejar errores sincrónicos, como por ejemplo un intento de dividir por cero. El manejo de excepciones se usa en situaciones en las que el sistema puede recuperarse de la falla que causó la excepción. El procedimiento de recuperación se denomina manejador de excepciones. Las excepciones en Java son objetos reales, instancias de clases que heredan de la clase Throwable. Una instancia de la clase Throwable se crea cuando se lanza una excepción o en otras palabras ha sucedido un error. Throwable tiene dos subclases: Error y Exception. Las instancias Error, son errores internos en el ambiente de la unidad en tiempo de ejecución de Java (máquina virtual). Estos errores son raros y con frecuencia fatales.

La clase Exception cae en dos grupos:

- Excepciones en tiempo de ejecución: estas ocurren normalmente debido a que el código no es muy robusto, tales como: `ArrayIndexOutOfBoundsException`, `securityException` o `NullPointerException`.
- Otras Excepciones : incluye excepciones creadas por el programador para que den aviso de los casos anormales que pueden ocurrir en un programa, tales como `:EOFException` y `MalformedURLException`.

El programador encierra en un bloque Try(bloque de intento) el código que puede generar una excepción. El bloque try va seguido inmediatamente de uno o más bloques catch (bloque de atrapar). Cada bloque catch especifica el tipo de excepción que puede atrapar y contiene el manejador de excepciones. Después del último bloque catch, un bloque finally (finalmente) opcional proporciona el código que siempre se ejecuta, sea que haya ocurrido una excepción o no.

Ejemplo 1: Hacer un programa que capture un número desde teclado y lo imprima en pantalla.

```
//paquete que permite el manejo entrada/salida
import java.io.*;
public class GeneraExcepciones
{
    public static void main(String args[])
    {
//try -catch permite el manejo de excepciones y errores para programas autónomos
        try{
            //Clase que me permite crear un objeto para que capture datos desde el teclado
            BufferedReader linea = new BufferedReader(
                new InputStreamReader( System.in ) );
            System.out.println("Digite un numero:");
            // creación de un objeto de tipo String
            String lin = linea.readLine();
            System.out.println("Es número digitado fue:"+lin);
        }catch(IOException e) {
            System.out.println("Error: " + e);
        }
    }
}
```

```

    }
}
}

```

Nota : Si trata de compilar el programa sin realizar la excepción (try - catch), este generará un error de excepción I/O.

Ejemplo 2: programa que permite detectar, indicar y manejar una excepción de una división por cero.

```

import java.awt.*;
import java.applet.*;

```

```

class MisExcepciones extends ArithmeticException
{
    public MisExcepciones ()
    {
        super ("intento de dividir por Cero (0)");
    }
}

```

```

public class Excepciones extends Applet
{
    Label solicita,solicita2;
    TextField entrada, entrada2;
    int numero,numero2;
    double resultado;
    public void init()
    {
        solicita= new Label("Teclee Numerador: ");
        entrada= new TextField(10);
        solicita2= new Label("Teclee Denominador: ");
        entrada2= new TextField(10);
        add(solicita);
        add(entrada);
        add(solicita2);
        add(entrada2);
    }
}

```

```

public boolean action(Event e, Object o)
{
    if(e.target==entrada2)
    {
        numero=Integer.parseInt(entrada.getText());
        entrada.setText("");
        numero2=Integer.parseInt(entrada2.getText());
        entrada2.setText("");
        try
        {
            resultado=cociente (numero,numero2);
            showStatus(numero+" / "+numero2 + " = " + Double.toString(resultado));
        }
        catch (MisExcepciones exception)
        {

```

```

        showStatus(exception.toString());
    }
}
return true;
}
public double cociente( int numerador, int denominador)throws MisExcepciones
{
    if(denominador==0)
        throw new MisExcepciones();
    return (double) numerador/denominador;
}
}

```

Ejemplo 3: Hacer un programa que utilice excepciones para leer dos arreglos y con finally imprima el último número de cada arreglo.

```

class Final
{
    int [] numero1={12,15,10,8,-1,7};
    int [] numero2={1,5,20,8,1,13};
    public static void main(String[] arg)
    {
        Final fin=new Final();
        System.out.println("Primer arreglo");
        fin.leernumero(fin.numero1);
        System.out.println("Segundo arreglo");
        fin.leernumero(fin.numero2);
    }
}

```

```

void leernumero(int[] narreglo)
{
    int cuenta=0;
    int ultimonumero=0;
    try
    {
        while(cuenta<narreglo.length)
        {
            ultimonumero=narreglo[cuenta++];
            if(ultimonumero== -1)
                return;
        }
    }finally
    {
        System.out.println("Ultimo numero leído:"+ultimonumero);
    }
    return;
}
}

```

10.2 Ejercicios

1. Hacer un programa utilizando excepciones que permita visualizar un error al tratar de calcular la raíz de un numero negativo.
2. Hacer un programa utilizando excepciones que permita visualizar un error al existir un desbordamiento de elementos de un arreglo.
3. Hacer un programa utilizando excepciones que permita visualizar un error al existir un desbordamiento del tamaño de tipo de dato.
4. Hacer un programa utilizando excepciones que permita visualizar un error al realizar una operación matematica con caracteres y numeros.
5. Hacer un programa utilizando excepciones que permita visualizar los errores de una superclase (herencia).
6. Hacer un programa utilizando excepciones que permita visualizar los errores al crear una pila.
7. Hacer un programa utilizando excepciones que permita visualizar un error al asignarle a una variable un valor de un tipo de dato diferente.
8. Hacer un programa utilizando excepciones que permita visualizar un error al geernrar un error de desbordamiento de memoria.
9. Hacer un programa utilizando excepciones que permita visualizar un error al generarse un error de ejecución en el sistema operativo.
10. Hacer un programa utilizando excepciones que permita visualizar un error al generarse un error de encadenamiento de una excepción.

11. Hilos y Animación en Java

Incluir animación en un programa lo hace más atractivo a la vista y en Java es sumamente fácil hacerlo, sin embargo se debe conocer algo sobre hilos, el propósito es dar una introducción básica a estos conceptos sin profundizar en ellos, para centrarnos exclusivamente en la animación.

11.1. Hilos (Thread)

Son objetos o clases que nos permiten manipular el tiempo en el que los hilos se encuentran activos. Los hilos se pueden iniciar, pausar, detener, colocarlos a dormir o volverlos a despertar.

Por ejemplo, podemos crear un hilo de forma directa de la siguiente forma:

```
Thread miHilo;
```

Este es un hilo nombrado o etiquetado como "miHilo". La clase Thread indica que este objeto es un hilo, se puede inicializar mediante uno de sus constructores, en este caso se inicializará sin ningún argumento:

```
miHilo = new Thread();
```

Una vez inicializado podemos usar una de los siguientes métodos para manipularlo:

- `miHilo.start()`: Iniciar Hilo
- `miHilo.suspend()`: Suspende la ejecución de un Hilo
- `miHilo.resume()`: Reanudar un hilo que previamente ha sido suspendido
- `miHilo.sleep(long milisegundos)`: coloca a dormir un hilo una cantidad de tiempo especificada en milisegundos.
- `miHilo.stop()`: permite detener un hilo que previamente ha sido inicializado.

Pero además de estos métodos, existe un método principal llamado *run*, en el cual se deben colocar las instrucciones que se ejecutarán una y otra vez hasta que se decida detenerlo permanentemente, existen dos formas para usar la función *run*, una es heredar la clase Thread y rescribir una nueva función *run* de acuerdo a nuestras necesidades y la otra es usando la Interfaz Runnable.

11.2. Hilos con Interfaz Runnable

Lo mejor para comprender la interfaz Runnable es mostrar un ejemplo y explicarlo paso por paso:

```
import java.awt.*;
import java.applet.Applet;
public class HilosSegundos extends Applet implements Runnable {
    Thread hilo;
    int segundo;
    public void init() {
        //inicializar hilo y la variable entera segundo
        hilo = new Thread(this);
        segundo = 0;
        //comenzar ejecución del hilo
        hilo.start();
    }
}
```

```

}
// método principal de ejecución de un Hilo
public void run() {
    while (true) {
        segundo++;
        repaint();
        try { hilo.sleep(1000); }
        catch (InterruptedException e) { }
    }
}
// mostrar la variable segundos en la pantalla
public void paint(Graphics g) {
    g.drawString("Segundos: " + String.valueOf(segundo), 20, 20);
}
}

```

En la primera línea del Applet implementamos la interface Runnable mediante implements, una interface es muy similar a una clase abstracta (la que solo incluye el nombre y tipo de una función, sin cuerpo):

```

public interface java.lang.Runnable {
    public abstract void run();
}

```

Las interfaces a diferencia de las clases convencionales no contienen variables y tienen casi la misma función que las clases abstractas: obligar a los que las usan a implementar las funciones que tienen. Esto significa que obligatoriamente debemos poner un método run() en nuestra Applet. Este método realiza todo el trabajo de un hilo.

Thread tiene varios constructores, el que uso recibe como argumento un objeto de tipo Runnable, Este contiene la cláusula this porque al implementar Runnable la clase también se convierte en un objeto de tipo Runnable. Inicializamos el entero segundos en cero y finalmente arrancamos el hilo llamando a su método start().

El método run es de tipo void, lo que significa que no devuelve ningún valor, solo ejecuta lo que tiene dentro de su cuerpo. En el método tenemos un ciclo while con la condición "true", esto significa que este while nunca se va a detener puesto que solo lo hace si recibe un "false" y en este caso "true" no es una variable sino un valor booleano constante que nunca cambia.. En este while se incrementa el valor entero de segundos y llamar al método repaint(). Finalmente viene el manejo de una excepción mediante try - catch:

El try ejecuta la función: hilo.sleep(1000) y si esta se ejecuta en forma normal, la ejecución se realiza como si no existiera ni el try ni el catch, pero, si la ejecución falla, el "catch" se encarga de atrapar la excepción y de mostrarla al usuario en la pantalla de Java (mostrar el error que ocurrió). La función hilo.sleep(1000) se encarga de dormir al hilo 1000 milisegundos, es decir, un segundo, después de este tiempo se regresa al inicio del while y se comienza el ciclo una y otra vez.:

11.3. Animación

Animar es dar la sensación de movimiento a una imagen mediante diversas técnicas. Una de tantas es cambiar varios cuadros por segundo de modo que se aparente un cambio de posición, por eso la animación está ligada al tiempo.

Ejemplo 1: Hacer un programa que dibuje una línea que gire alrededor de la coordenada (150,150) del Applet. Su longitud (o radio) es de 100. (recuerde que las unidades que manejan las Applets son píxeles).

```
import java.awt.*;
import java.applet.Applet;

public class GiroLinea extends Applet implements Runnable {
    Thread hilo;
    double decima, segundo;
    int xPos;
    int yPos;
    public void init() {
        hilo = new Thread(this);
        decima_segundo = 0;
        hilo.start();
    }
    public void run() {
        while (true) {
            decima_segundo += 0.1;
            repaint();
            try { hilo.sleep(100); }
            catch (InterruptedException e) { }
        }
    }
    public double calcularPosX(double tiempo) {
        return 100*Math.cos(1*tiempo);
    }
    public double calcularPosY(double tiempo) {
        return (100*Math.sin(1*tiempo));
    }
    public void paint(Graphics g) {
        xPos = (int) calcularPosX(decima_segundo);
        yPos = (int) calcularPosY(decima_segundo);
        g.drawLine(150,150,150+xPos,150+yPos);
    }
}
```

El método run() se incrementa en intervalos de 0.1 para la variable de tiempo, el hilo también duerme 0.1 segundos en cada intervalo para mantenerlos ambos iguales (porque 0.1 segundos = 100 milisegundos), la variable decima_segundo será algo así como una variable de control que relaciona el hilo y el tiempo.

Los métodos calcularPosX y calcularPosY se encargan de recalculan en función del tiempo la posición del punto que se mueve. Utilizamos double para los cálculos porque es de mayor exactitud, la animación se logra usando la fórmula de velocidad angular:

$$\text{angulo} = \text{velocidad} \times \text{tiempo}$$

Usando una velocidad angular de 1 radián por segundo y tomando el tiempo de decima_segundo (más adelante). Después, la posición se calcula con las funciones trigonométricas $x = R \cdot \cos(\text{angulo})$, $y = R \cdot \sin(\text{angulo})$, donde R es el radio de nuestra línea, tomamos los métodos seno y coseno de la clase Math.

Finalmente desde el método paint llama a las funciones de cálculo y muestra la línea. Se puede cambiar la velocidad de la animación ajustando los valores de incremento para decima_segundo y el tiempo que duerme el hilo:

```
decima_segundo += 0.1; //si lo haces más pequeño se suaviza la animación
hilo.sleep(100);        //pero también tendrías que ajustar este valor
```

Ejemplo 2. Hacer un programa para producir un efecto de movimiento de un mensaje de Texto de arriba a abajo.

```
import java.awt.*;
import java.applet.Applet;
public class TextoAnimado extends Applet implements Runnable {
    Thread hilo;
    double tiempo;
    int x = 20, y = 20;
    int Vy; //Velocidad en y
    public void init() {
        hilo = new Thread(this);
        tiempo = 0;
        Vy = 23; //23 pixeles por segundo
        hilo.start();
    }
    public void run() {
        while (true) {
            tiempo += 0.07;
            repaint();
            try { hilo.sleep(70); }
            catch (InterruptedException e) { }
        }
        public double posY(double tiempo) {
            return y + Vy*tiempo;
        }
        public void paint(Graphics g) {
            if(posY(tiempo) > 200 && Vy > 0) {
                y = (int) posY(tiempo);
                Vy = -23;
                tiempo = 0;
            } else if ( posY(tiempo) < 20 && Vy < 0 ) {
                y = (int) posY(tiempo);
                Vy = 23;
                tiempo = 0;
            }
            setBackground(Color.white);
            g.drawString("¡Hola!, se supone que debería estar durmiendo!", x, (int)posY(tiempo));
        }
    }
}
```

Ejemplo 3: Hacer un programa que dibuje una pelota que se mueva dentro de una caja.

```
import java.awt.*;
import java.applet.Applet;
import java.util.*;
public class Animacion1b extends Applet implements Runnable
{
    Point pt = new Point(20, 20);
```

```

final int radio = 5;
final int diametro = radio * 2;
int inc = 2;
Thread correr = null;
public void init() {
    setBackground(Color.white);
    repaint();
}
public void start() {
    if (correr == null) {
        correr = new Thread(this);
        correr.start();
    }
}
public void run() {
    while (true) {
        repaint();
        try { correr.sleep(15); }
        catch (InterruptedException e) {};
    }
}
public void stop() {
    if (correr != null) {
        correr.stop();
        correr = null;
    }
}
public void paint( Graphics g ) {
    g.setColor(Color.blue);
    g.fillOval(pt.x-radio, pt.y-radio, diametro, diametro);
    pt.x += inc;
    if (pt.x > size().width - 5 - radio)
        inc = -1;
    else if (pt.x <= 3 + radio)
        inc = 1;
    g.setColor(Color.red);
    g.drawRect(2, 2, size().width-5, size().height-5);
}
}

```

Ejemplo 4: Hacer un programa que dibuje una pelota que rebote dentro de una caja (utilizar las fórmulas de tiro parabólico y $g = 150$ (recuerde que java trabaja en pixeles)).

```

import java.awt.*;
import java.applet.Applet;
class pelota {
    private int Vix, Viy;
    private int x, y;
    private double t;
    public pelota(int x, int y, int Vix, int Viy) {
        this.x = x;
        this.y = y; this.Vix = Vix;
        this.Viy = Viy;
    }
}

```

```

}
public void tiempo(double t) {
    this.t += t;
}
public int posX() {
    return (int)(x + Vix*t);
}
public int posY() { return (int)(y + Viy*t + (0.5)*(150)*t*t);
}
public int Vfx() { return (Vix);
}
public int Vfy() { return (int)(150*t + Viy);
}
public void dibujar(Graphics g) {
    g.fillOval( posX(), posY(), 20, 20); }
}
public class PruebaPelota extends Applet implements Runnable {
    Thread hilo; pelota P; int x, y, Vx, Vy;
    public void init() {
        hilo = new Thread(this);
        P = new pelota(40, 40, 50, 10);
        hilo.start();
    }
    public void run() {
        while (true) {
            P.tiempo(0.04); repaint();
            try { hilo.sleep(40); }
            catch (InterruptedException e) { }
        }
    }
    public void paint(Graphics g) {
        setBackground(Color.white);
        g.drawLine(20, 20, 20, 200);
        g.drawLine(20, 200, 200, 200);
        g.drawLine(200, 200, 200, 20);
        g.drawLine(200, 20, 20, 20);
        P.dibujar(g);
        if( P.posX() > 180 && P.Vfx() > 0 ) {
            x = P.posX();
            y = P.posY();
            Vx = -P.Vfx();
            Vy = P.Vfy();
            P = null;
            P = new pelota(x, y, Vx, Vy);
        } else if( P.posX() < 20 && P.Vfx() < 0 ) {
            x = P.posX();
            y = P.posY();
            Vx = -P.Vfx();
            Vy = P.Vfy();
            P = null;
            P = new pelota(x, y, Vx, Vy);
        } else if( P.posY() > 180 && P.Vfy() > 0 ) {
            x = P.posX();
            y = P.posY();

```

```

Vx = P.Vfx();
Vy = -P.Vfy();
P = null;
P = new pelota(x, y, Vx, Vy);
} else if( P.posY() < 20 && P.Vfy() < 0) {
x = P.posX();
y = P.posY();
Vx = P.Vfx();
Vy = -P.Vfy();
P = null;
P = new pelota(x, y, Vx, Vy);
}
}
}
}

```

Ejemplo 5: Hacer un programa que muestre una pelota moviéndose por delante y por detrás de un texto.

```

import java.awt.*;
import java.applet.Applet;
import java.lang.Thread;
public class Animacion4 extends Applet implements Runnable
{
    Font f;
    FontMetrics fm;
    int x, y, desplazamiento;
    int tamano = 20;
    Thread t;
    String st = "ANIMACION";
    public void init() {
        f = new Font("Helvetica", Font.BOLD, 30);
        fm = getFontMetrics(f);
        setBackground(Color.white);
        x = 50;
        y = 30;
        desplazamiento = 10;
        t = new Thread(this);
    }
    public void start() { t.start(); }
    public void stop() { t.stop(); }
    // requerido por Runnable
    public void run() {
        while (true) {
            x = x + desplazamiento;
            if (x + tamano > size().width)
                desplazamiento = -10;
            else if (x < 0)
                desplazamiento = 10;
            repaint();
        }
        try {
            t.sleep(100); // duerme por 100 milisegundos
        } // redibuja 10 veces por segundo
        catch (Exception e) {}
    }
}

```

```

    }
    }
    public void paint( Graphics g ) {
        int localx;
        g.setFont(f);
        // dibuja las letras en azul primero
        localx = 10;
        g.setColor(Color.blue);
        for (int i = 0; i < st.length(); i += 2) {
            g.drawString(st.substring(i, i+1), localx, y+20);
            if (i + 1 < st.length())
                localx = localx + fm.charWidth(st.charAt(i)) + fm.charWidth(st.charAt(i+1));
            else
                localx = localx + fm.charWidth(st.charAt(i));
        }
        // dibuja el circulo que se esta moviendo
        g.setColor(Color.yellow);
        g.fillOval(x, y, tamano, tamano);
        // dibuja la letras rojas que quedaran sobre el circulo
        localx = 10 + fm.charWidth(st.charAt(0));
        g.setColor(Color.red);
        for (int i = 1; i < st.length(); i += 2) {
            g.drawString(st.substring(i, i+1), localx, y+20);
            if (i + 1 < st.length())
                localx = localx + fm.charWidth(st.charAt(i)) + fm.charWidth(st.charAt(i+1));
            else
                localx = localx + fm.charWidth(st.charAt(i));
        }
    }
}

```

11.4 Ejercicios

1. Hacer un programa que permita determinar si un hilo esta activo o no.
2. Hacer un programa que permita la ejecución de hilos de alta prioridad, visualizando los hilos que se detienen para dar inicio a los hilos de alta prioridad.
3. Hacer un programa que muestre el tiempo de ejecución de 4 hilos sincronizados.
4. Hacer un programa que inactive un hilo al iniciarse la ejecución de un segundo hilo. El usuario debera defenir el lapso del tiempo para el inicio del segundo hilo.
5. Hacer un programa que haga rebotar una pelota dentro un applet. La pelota debe iniciarse con un evento mouseDown. Cuando la pelota choque con el borde del applet, debera rebotar al azar a un angulo entre 20 y 60 grados.
6. Modifique el programa anterior para que el usuario pueda dibujar 5 lineas en el applet mientras la pelota esta rebotando. Cuando la pelota choque con una linea deberá rebotar al azar a un angulo entre 20 y 60 grados.
7. Hacer un programa que simule la atención a clientes de un banco utilizando hilos.
8. Hacer un programa utilizando excepciones que permita visualizar un error al generar un error de desbordamiento de memoria.
9. Hacer un programa utilizando excepciones que permita visualizar un error al generarse un error de ejecución en el sistema operativo.
10. Hacer un programa utilizando excepciones que permita visualizar un error al generarse un error de encadenamiento de una excepción.

12. Manejo de Datos (archivos) a través de los flujos de Java

Muchos de los programas que se crean con Java necesitarán interactuar con datos del exterior, procesarlos y luego presentar un resultado de alguna forma: en pantalla, guardándolos en un archivo, enviándolos a la red, imprimirlos en papel, etc. Se usan archivos para conservar a largo plazo grandes cantidades de datos. Los datos guardados en archivos se conocen como datos persistentes. Los computadores guardan los archivos en dispositivos de almacenamiento secundario como discos magnéticos, ópticos y cintas magnéticas.

12.1. Archivos y Flujos

Java considera a los archivos como flujos secuenciales de bytes. Cada archivo termina en un marcador de fin de archivo. Cuando se abre un archivo se crea un objeto y se asocia un flujo (Stream) a dicho objeto. Cuando comenzamos a ejecutar una aplicación o un applet de Java, se crean automáticamente tres objetos: System.in, System.out, System.err.; los flujos asociados a estos objetos proporcionan canales de comunicación entre un programa y un archivo o dispositivo en particular.

- System.in: permite enviar información desde el teclado
- System.out : permite enviar datos a la pantalla
- System.err : permite enviar mensajes de error a la pantalla.

Para procesar archivos en Java debemos importar el paquete **java.io**. Este paquete contiene las clases de flujo como FileInputStream (flujo de entrada de un archivo) y FileOutputStream (para el flujo de salida de un archivo). Los archivos se abren creando objetos de estas clases que se derivan de las clases InputStream (flujo de entrada) y OutputStream (flujo de salida). Para realizar operaciones de entrada y de salida de tipos de datos primitivos, se usaran los objetos de las clases DataInputStream (flujo de entrada de datos) y DataOutputStream (flujo de salida de datos) junto con las clases de flujos de archivos.

12.1.1 Creación de un archivo de acceso secuencial

Java no obliga a los archivos a tener una estructura; por lo tanto, conceptos tales como registro no existen en los archivos de java. Esto significa que los programadores deben estructurar los archivos a modo de satisfacer las necesidades de las aplicaciones.

Ejemplo : Hacer un programa que cree un archivo secuencial que maneje un sistema de cuentas por cobrar. Cada cliente tiene: identificación, nombre de la empresa, representante legal, ciudad, teléfono y crédito máximo.

```
import java.awt.*;
import java.io.*;

public class ArchivosSecuenciales extends Frame
{
    TextField nit,nombre,empresa,ciudad,telefono,credito;
    Button entradas,salidas;
    Label rotulon,rotulonm,rotuloe,rotuloc,rotulot,rotulocr;
    DataOutputStream salida;
    public ArchivosSecuenciales()
    {
```

```

    super ("Crear un archivo de clientes");
}
public void adicionarregistros()
{
    int cuenta=0;
    Double d;
    cuenta=(new Integer(nit.getText())).intValue();
    System.out.println(cuenta);
    try
    {
        if(cuenta>0)
        {
            salida.writeInt(cuenta);
            salida.writeUTF(empresa.getText());
            salida.writeUTF(nombre.getText());
            salida.writeUTF(ciudad.getText());
            salida.writeUTF(telefono.getText());
            d=new Double(credito.getText());
            salida.writeDouble(d.doubleValue());
        }
    }
    catch(IOException e)
    {
        System.err.println("\nError al escribir en el archivo\n"+e.toString());
        System.exit(1);
    }
    nit.setText("");
    empresa.setText("");
    nombre.setText("");
    ciudad.setText("");
    telefono.setText("");
    credito.setText("");
}

public void configurar()
{
    resize(350,250);
    setLayout(new GridLayout(7,2));
    nit=new TextField(20);
    rotulon=new Label ("Identificacion:");
    rotuloe=new Label("Empresa:");
    empresa=new TextField(40);
    nombre=new TextField(40);
    rotulonm=new Label("Representante:");
    ciudad=new TextField(20);
    rotuloc=new Label("Ciudad :");
    telefono=new TextField(20);
    rotulot=new Label("Telefono :");
    credito=new TextField(20);
    rotulocr=new Label("Valor Maximo Credito:");
    entradas=new Button("Guardar");
    salidas=new Button("Terminar");
    add(rotulon);
    add(nit);

```

```

add(rotuloe);
add(empresa);
add(rotulonm);
add(nombre);
add(rotuloc);
add(ciudad);
add(rotulot);
add(telefono);
add(rotulocr);
add(credito);
add(entradas);
add(salidas);
show();
try
{
    salida=new DataOutputStream(new FileOutputStream("clientes.dat"));
}
catch(IOException e)
{
    System.err.println("\nNo se abrio bien el archivo\n"+e.toString());
    System.exit(1);
}
}

```

```

public void limpiar()
{
    if(!nit.getText().equals(""))
        adicionarregistros();
    try
    {
        salida.flush();
        salida.close();
    }
    catch(IOException e)
    {
        System.err.println("\nNo se cerro bien el archivo\n"+e.toString());
        System.exit(1);
    }
}

public boolean handleEvent(Event e)
{
    if(e.target==salidas || e.id==Event.WINDOW_DESTROY)
    {
        limpiar();
        hide();
        dispose();
        System.exit(0);
        return true;
    }
    if(e.target instanceof Button)
    {
        if(e.arg.equals("Guardar"))
        {

```

```

        adicionarregistros();
        return true;
    }
}
return super.handleEvent(e);
}
public static void main(String arg[])
{
    ArchivosSecuenciales clientes= new ArchivosSecuenciales();
    cliente.configurar();
}
}

```

12.1.2 Lectura de un archivo secuencial

Los datos se almacenan en archivos con el fin de poderlos recuperar para procesarlos cuando sea necesario. Los archivos se abren para entrada creando un objeto `FileInputStream`. el nombre del archivo se pasa como argumento al constructor `FileInputStream`.

Ejemplo 1: Hacer un programa que lea un archivo secuencial que maneje un sistema de clientes. Para cada cliente se debe mostrar la identificación, nombre de la empresa, representante legal, ciudad, teléfono y valor del crédito máximo.

```

import java.awt.*;
import java.io.*;
public class LeerArchivoSecuencial extends Frame
{
    TextField nit,empresa,nombre,ciudad,telefono,credito;
    Button entradas,salidas;
    Label rotulon,rotuloe,rotulonm,rotuloc,rotulot,rotulocr;
    boolean masregistros=true;
    DataInputStream salida;
    public LeerArchivoSecuencial()
    {
        super ("Leer un archivo de clientes");
    }
    public void adicionarregistros()
    {
        int cuenta=0;
        double d;
        String snombre, sciudad, sempresa, stelefono;
        try
        {
            cuenta=salida.readInt();
            snombre=salida.readUTF();
            sempresa=salida.readUTF();
            sciudad=salida.readUTF();
            stelefono=salida.readUTF();
            d=salida.readDouble();
            nit.setText(String.valueOf(cuenta));
            empresa.setText(String.valueOf(sempresa));

```

```

        nombre.setText(String.valueOf(snombre));
        ciudad.setText(String.valueOf(sciudad));
        telefono.setText(String.valueOf(stelefono));
        credito.setText(String.valueOf(d));
    }
    catch(IOException e)
    {
        System.err.println("\nError al leer en el archivo\n"+e.toString());
        System.exit(1);
    }
}
public void configurar()
{
    try
    {
        salida=new DataInputStream(new FileInputStream("clientes.dat"));
    }
    catch(IOException e)
    {
        System.err.println("\nNo se abrió bien el archivo\n"+e.toString());
        System.exit(1);
    }

    resize(300,150);
    setLayout(new GridLayout(7,2));
    nit=new TextField(20);
    rotulon=new Label ("Identificación :");
    nombre=new TextField(20);
    rotulonm=new Label("Empresa :");
    empresa=new TextField(20);
    rotuloe=new Label("Representante :");
    ciudad=new TextField(20);
    rotuloc=new Label("Ciudad :");
    telefono=new TextField(20);
    rotulot=new Label("Telefono :");
    credito=new TextField(20);
    rotulocr=new Label("Valor maximo de Credito :");
    entradas=new Button("Siguiete");
    salidas=new Button("Terminar");
    add(rotulon);
    add(nit);
    add(rotuloe);
    add(empresa);
    add(rotulonm);
    add(nombre);
    add(rotuloc);
    add(ciudad);
    add(rotulot);
    add(telefono);
    add(rotulocr);
    add(credito);
    add(entradas);
    add(salidas);
    show();
}

```

```

    }
    public void limpiar()
    {
        try
        {
            salida.close();
        }
        catch(IOException e)
        {
            System.err.println("\nNo se cerro bien el archivo\n"+e.toString());
            System.exit(1);
        }
    }
}
public boolean handleEvent(Event e)
{
    if(masregistros==false || e.id==Event.WINDOW_DESTROY ||e.target==salidas)
    {
        limpiar();
        hide();
        dispose();
        System.exit(0);
        return true;
    }
    return super.handleEvent(e);
}
public boolean action(Event e, Object o)
{
    if(e.target instanceof Button)
        if(e.arg.equals("Siguiente"))
            adicionarregistros();
    return true;
}
public static void main(String arg[])
{
    LeerArchivoSecuencial cuentas= new LeerArchivoSecuencial();
    cuentas.configurar();
}
}

```

Ejemplo 2: Hacer un programa que lea un archivo secuencial que maneje un sistema de clientes y el usuario pueda solicitar información sobre valor del crédito: mayores de 800000, menores de 800000 y 0.

```

import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class CreditoClientes extends Frame
{
    TextArea mostrar;
    Button salir,credito1,credito2,ceros;
    Panel botones;
    String tipo;
    File archivo;
}

```

```

DataInputStream entrada;
public CreditoClientes()
{
    super("Programa para consultar valores máximos de crédito");
}
public void leerregistro()
{
    int contar;
    String primero,ultimo;
    double d;
    try
    {
        mostrar.setText(tipo+ "\n");
        while(true)
        {
            contar=entrada.readInt();
            primero=entrada.readUTF();
            ultimo =entrada.readUTF();
            d=entrada.readDouble();
            if(mostrarsaldos(d))
                mostrar.appendText(contar + "\t" + primero+"\t"+ultimo+"\t"+d+"\n");
        }
    }
    catch (EOFException eof){}
    catch(IOException e){
        System.err.println("Error al leer archivo \n" + e.toString());
        System.exit(1);
    }
}
public boolean mostrarsaldos(double balance)
{
    if(tipo.equals("Credito menor de 800000") && balance<800000)
        return true;
    if(tipo.equals("Credito mayor de 800000") && balance>800000)
        return true;
    if(tipo.equals("Credito en Ceros") && balance==0)
        return true;
    return false;
}
public void configurar()
{
    resize(600,150);
    mostrar=new TextArea(4,40);
    botones=new Panel();
    salir=new Button("Fin");
    credito1=new Button("Credito mayor de 800000");
    credito2=new Button("Credito menor de 800000");
    ceros=new Button("Credito en Ceros");
    botones.add(credito1);
    botones.add(credito2);
    botones.add(ceros);
    botones.add(salir);
    add("Center", mostrar);
    add("South",botones);
}

```

```

        show();
    }
    public boolean handleEvent(Event e)
    {
        if(e.id==e.WINDOW_DESTROY||e.target==salir)
        {
            hide();
            dispose();
            System.exit(0);
            return true;
        }
        if(e.target instanceof Button)
        {
            tipo =e.arg.toString()    ;
            try{
                entrada=new DataInputStream(new FileInputStream("clientes.dat"));
            }
            catch(IOException el)
            {
                System.err.println("No se cerro\n"+el.toString());
                System.exit(1);
            }
            leerregistro();
            try{
                entrada.close();
            }
            catch(IOException el)
            {
                System.err.println("No se cerro\n"+el.toString());
                System.exit(1);
            }
        }
        return super.handleEvent(e);
    }
    public static void main(String args[])
    {
        CreditoClientes f = new CreditoClientes();
        f.configurar();
        f.show();
    }
}

```

12.2 Archivos de acceso aleatorio

Los archivos de acceso aleatorio son apropiados para aplicaciones de acceso instantáneo donde se necesita localizar un registro con información particular. La técnica más sencilla para utilizar archivos de acceso aleatorio es emplear registros de longitud fija, lo cual permite determinar la posición exacta de un registro.

12.2.1 Creación de un archivo de acceso aleatorio

Los objetos `RandomAccessFile` tiene todas las capacidades de los objetos `DataInputStream` y `DataOutputStream`. Cuando asocia un flujo `RandomAccessFile` a un archivo, los datos se leen o escriben a partir del punto en el archivo especificado por el apuntador de posición en el archivo(datos primitivos).

Ejemplo : Hacer un programa que almacene 10 registros de longitud fija. cada registro debe tener el identificación, empresa, representante, ciudad teléfono y valor máximo de crédito. El programa deberá actualizar, insertar y eliminar un cliente.

Paso 1: creación de la clase `Registro`, que permitirá la manipulación de los registros que contiene el archivo

```
import java.io.*;
public class Registro{
    int cuenta;
    String nombre,empresa, ciudad,telefono;
    double credito;
    public void leer(RandomAccessFile archivo) throws IOException
    {
        cuenta=archivo.readInt();
        byte b1[]=new byte[15];
        archivo.readFully(b1);
        nombre=new String(b1,0);
        byte b2[]=new byte[15];
        archivo.readFully(b2);
        empresa=new String(b2,0);
        byte b3[]=new byte[15];
        archivo.readFully(b3);
        ciudad=new String(b3,0);
        byte b4[]=new byte[15];
        archivo.readFully(b4);
        telefono=new String(b4,0);
        credito=archivo.readDouble();
    }
    public void escribir(RandomAccessFile archivo) throws IOException
    {
        archivo.writeInt(cuenta);
        byte b1[]=new byte[15];
        if(nombre!=null)
            nombre.getBytes(0,nombre.length(),b1,0);
        archivo.write(b1);
        byte b2[]=new byte[15];
        if(empresa!=null)
            empresa.getBytes(0,empresa.length(),b2,0);
        archivo.write(b2);
        byte b3[]=new byte[15];
        if(ciudad!=null)
            ciudad.getBytes(0,ciudad.length(),b3,0);
        archivo.write(b3);
        byte b4[]=new byte[15];
        if(telefono!=null)
            telefono.getBytes(0,telefono.length(),b4,0);
```

```

    archivo.write(b4);
    archivo.writeDouble(credito);
}
public int size() {
    return 42;}
}

```

Paso 2: Programa que permite crear el archivo aleatorio.

```

import java.io.*;
import java.awt.*;
public class EscribirArchivo
{
    private Registro blanco;
    RandomAccessFile archivo;
    public EscribirArchivo()
    {
        blanco=new Registro();
        try{
            archivo=new RandomAccessFile("clientes.dat","rw");
        }
        catch(IOException e)
        {
            System.err.println("No se abrió el archivo\n"+e.toString());
            System.exit(1);
        }
    }
    public void crearlo()
    {
        try
        {
            for(int i=0;i<100;i++)
                blanco.escribir(archivo);
        }
        catch(IOException e)
        {
            System.err.println(e.toString());
        }
    }
    public static void main(String arg[])
    {
        EscribirArchivo cuentas= new EscribirArchivo();
        cuentas.crearlo();
    }
}

```

Paso 3: programa que permite guardar información en un archivo de acceso aleatorio.

```

import java.io.*;
import java.awt.*;
public class EscribirArchivoAleatorio extends Frame
{
    TextField cuenta,nombre,empresa,ciudad,telefono,credito;
    Button entradas,salidas;
    Label rotulon,rotulonm,rotuloe,rotulot,rotulocr, rotuloc;
}

```

```

boolean masregistros=true;
RandomAccessFile salida;
Registro dato;
public EscribirArchivoAleatorio()
{
    super ("Escribir en archivo de acceso aleatorio");
    dato=new Registro();
    try{
        salida=new RandomAccessFile("clientes.dat","rw");
    }
    catch(IOException e)
    {
        System.err.println("No se abrió el archivo\n"+e.toString());
        System.exit(1);
    }
    configurar();
}
public void adicionarregistros()
{
    int contar=0;
    Double d;
    contar=(new Integer(cuenta.getText())).intValue();
    try
    {
        if(contar>0 && contar<=10)
        {
            dato.cuenta=contar;
            dato.nombre=nombre.getText();
            dato.empresa=empresa.getText();
            dato.ciudad=ciudad.getText();
            dato.telefono=telefono.getText();
            d=new Double(credito.getText());
            dato.credito=d.doubleValue();
            salida.seek((long)(contar-1)*dato.size());
            dato.escribir(salida);
            nombre.setText("");
            cuenta.setText("");
            empresa.setText("");
            ciudad.setText("");
            telefono.setText("");
            credito.setText("");
        }
        else
        {
            cuenta.setText("Error al escribir el valor de cuenta (1-10)");
            cuenta.selectAll();
        }
    }
    catch(IOException e)
    {
        System.err.println("\nError al escribir en el archivo\n"+e.toString());
        System.exit(1);
    }
}

```

```

public void configurar()
{
    resize(300,150);
    setLayout(new GridLayout(7,2));
    cuenta=new TextField(20);
    rotulon=new Label ("Identificación :");
    nombre=new TextField(20);
    rotulonm=new Label("Empresa :");
    empresa=new TextField(20);
    rotuloe=new Label("Representante :");
    ciudad=new TextField(20);
    rotuloc=new Label("Ciudad :");
    telefono=new TextField(20);
    rotulot=new Label("Telefono :");
    credito=new TextField(20);
    rotulocr=new Label("Valor maximo de Credito :");
    entradas=new Button("Guardar");
    salidas=new Button("Terminar");
    add(rotulon);
    add(cuenta);
    add(rotuloe);
    add(empresa);
    add(rotulonm);
    add(nombre);
    add(rotuloc);
    add(ciudad);
    add(rotulot);
    add(telefono);
    add(rotulocr);
    add(credito);
    add(entradas);
    add(salidas);
    show();
}

public void limpiar()
{
    if(!cuenta.getText().equals(""))
        adicionarregistros();
    try
    {
        salida.close();
    }
    catch(IOException e)
    {
        System.err.println("\nNo se cerro bien el archivo\n"+e.toString());
        System.exit(1);
    }
}

public boolean handleEvent(Event e)
{
    if(e.id==Event.WINDOW_DESTROY ||e.target==salidas)
    {
        limpiar();
        hide();
    }
}

```

```

        dispose();
        System.exit(0);
        return true;
    }
    if(e.target instanceof Button)
    {
        if(e.arg.equals("Guardar"))
        {
            adicionarregistros();
            return true;
        }
    }
    return super.handleEvent(e);
}
public static void main(String arg[])
{
    EscribirArchivoAleatorio cuentas= new EscribirArchivoAleatorio(); }

```

12.2.2 Leer datos de un archivo aleatorio

Se realizará un programa que permita abrir un RandomAccessFile para lectura con el modo de apertura de archivo "r".

Ejemplo: Hacer un programa que lea los registros de un archivo aleatorio llamado clientes previamente creado. El programa debe mostrar toda información del cliente.

```

import java.awt.*;
import java.io.*;
public class LeerArchivoAleatorio extends Frame
{
    TextField nit,nombre,empresa,ciudad,telefono,credito;
    Button entradas,salidas;
    Label rotulon,rotulonm,rotuloc,rotuloe,rotulocr,rotulot;
    boolean masregistros=true;
    RandomAccessFile salida;
    Registro dato;
    public LeerArchivoAleatorio()
    {
        super ("Leer un archivo de clientes");
        try{
            salida=new RandomAccessFile("clientes.dat","r");
        }
        catch(IOException e)
        {
            System.err.println("No se abrió el archivo\n"+e.toString());
            System.exit(1);
        }
        dato=new Registro();
        configurar();
    }
    public void leerregistros()
    {

```

```

try
{
    do {
        dato.leer(salida);
    }while(salida.getFilePointer()<salida.length() && dato.cuenta==0);
}
catch(IOException e)
{
    masregistros=false;
}
if(dato.cuenta!=0)
{
    nit.setText(String.valueOf(dato.cuenta));
    nombre.setText(String.valueOf(dato.nombre));
    empresa.setText(String.valueOf(dato.empresa));
    ciudad.setText(String.valueOf(dato.ciudad));
    telefono.setText(String.valueOf(dato.telefono));
    credito.setText(String.valueOf(dato.credito));
}
}
public void configurar()
{
    resize(300,150);
    setLayout(new GridLayout(7,2));
    nit=new TextField(20);
    rotulon=new Label ("Identificación:");
    nombre=new TextField(20);
    rotulonm=new Label("Representante :");
    empresa=new TextField(20);
    rotuloe=new Label("Empresa :");
    ciudad=new TextField(20);
    rotuloc =new Label("Ciudad :");
    telefono=new TextField(20);
    rotulot=new Label("Ciudad :");
    credito=new TextField(20);
    rotulocr=new Label("Valor maximo de Credito :");
    entradas=new Button("Siguiente");
    salidas=new Button("Terminar");
    add(rotulon);
    add(nit);
    add(rotulonm);
    add(nombre);
    add(rotuloe);
    add(empresa);
    add(rotuloc);
    add(ciudad);
    add(rotulot);
    add(telefono);
    add(rotulocr);
    add(credito);
    add(entradas);
    add(salidas);
    show(); }

```

```

public void limpiar()
{
    try
    {
        salida.close();
    }
    catch(IOException e)
    {
        System.err.println("\nNo se cerro bien el archivo\n"+e.toString());
        System.exit(1);
    }
}
public boolean handleEvent(Event e)
{
    if(masregistros==false || e.id==Event.WINDOW_DESTROY ||e.target==salidas)
    {
        limpiar();
        hide();
        dispose();
        System.exit(0);
        return true;
    }
    return super.handleEvent(e);
}
public boolean action(Event e, Object o)
{
    if(e.target instanceof Button)
        if(e.arg.equals("Siguiente"))
            leerregistros();
    return true;
}

public static void main(String arg[])
{
    LeerArchivoAleatorio cuentas= new LeerArchivoAleatorio();
}
}

```

12.3 Ejercicios

1. Hacer un programa que permita capturar la siguiente información de una ferretería: código, elemento, cantidad, valor de compra, valor de ventas, utilizando archivos de acceso secuencial.
2. Hacer un programa que permita realizar una venta de un producto de la ferretería, se debe capturar la siguiente información: código, elemento, cantidad, valor de ventas. SE debe descargar la cantidad vendida por cada elemento, utilizando archivos de acceso secuencial.
3. Hacer un programa que permita eliminar un registro de la ferretería, utilizando archivos de acceso secuencial.
4. Hacer un programa que permita visualizar todos los registros que contiene el archivo de la ferretería, utilizando archivos de acceso secuencial.

5. Hacer un programa que permita modificar un registro de la ferreteria.
6. Hacer un programa que permita buscar un registro especifico de la ferreteria. La búsqueda la debe realizar por codigo , utilizando archivos de acceso secuencial.
7. Hacer un programa que permita realizar una venta de un producto de la ferreteria, se debe capturar la siguiente información: codigo, elemento, cantidad, valor de ventas. Se debe descargar la cantidad vendida por cada elemento, utilizando archivos de acceso aeatorio.
8. Hacer un programa que permita eliminar un registro de la ferreteria, utilizando archivos de acceso aleatorio.
9. Hacer un programa que permita visualizar todos los registros que contiene el archivo de la ferreteria, utilizando archivos de acceso aleatorio.
10. Hacer un programa que permita buscar un registro especifico de la ferreteria. La búsqueda la debe realizar por codigo y utilizando archivo de acceso aleatorio.

13. Trabajo con redes

Java ofrece varias capacidades integradas de trabajo en red que facilitan el desarrollo de aplicaciones basadas en Internet y Web. Java habilita programas para buscar información en todo el mundo y colaborar con programas que se ejecuten en computadores internacionales, nacionales o sólo dentro de la organización, Java incluso puede permitir que applets y aplicaciones se ejecuten en un mismo computador.

Java ofrece comunicaciones basadas en *Sockets* que permiten a las aplicaciones manejar el trabajo en redes como si fuera E/S de archivos; un programa puede leer de un socket o escribir en un socket.. Java ofrece los siguientes Sockets:

- Sockets de flujo: proceso que establece una conexión con otro proceso. Mientras la conexión exista, los datos fluyen entre procesos en un flujo continuo.
- Sockets de datagrama: transmiten paquetes individuales de información.

13.1 Manipulación de URL

El protocolo http(hypertext transfer protocol) que constituye la base de la World Wide Web emplea localizadores uniformes de recursos (URL, Uniform Resource Locators) para localizar datos en la Internet. Si se conoce el URL de archivos HTML disponibles en Internet puede acceder a esos datos por medio de la manipulación de los URL. En la biblioteca de clases de Java se incluye el paquete java.net , que hace posible la comunicación en una red.

Ejemplo: Hacer un programa que manipule URL, para abrir páginas de Internet.

```
import java.awt.*;
import java.applet.*;
import java.net.*;

public class SelectordeSitios extends Applet
{
    Sitio listadesitios[];
    public void init()
    {
        listadesitios=new Sitio[2];
        listadesitios[0]=new Sitio ("Banco de Occidente",
            "http://www.bancodeoccidente.com");
        listadesitios[1]=new Sitio ("Comcel",
            "http://www.comcel.com");
        for(int i=0;i<listadesitios.length;i++)
            add(new Button(listadesitios[i].getTitle()));
    }
    public boolean action(Event e, Object o)
    {
        if(e.target instanceof Button)
        {
            String titulo;
            URL localizacion;
            for(int i=0;i<listadesitios.length;i++)
            {
                titulo=listadesitios[i].getTitle();
                localizacion=listadesitios[i].cogeLocation();
            }
        }
    }
}
```

```

        if(titulo.equals(o.toString()))
        {
            iralsitio(localizacion);
            return true;
        }
    }
}
return false;
}
public void iralsitio(URL loca)
{
    getAppletContext().showDocument(loca);
}
}
class Sitio extends Button
{
    private String titulo;
    private URL localiza;
    Sitio(String titulodelsitio, String sitiodellocalizacion)
    {
        titulo=titulodelsitio;
        try
        {
            localiza=new URL(sitiodellocalizacion);
        }
        catch(MalformedURLException e)
        {
            System.err.print("Dirección no valido:" + sitiodellocalizacion);
        }
    }
    public String getTitle()
    {
        return titulo;
    }
    public URL cogeLocation()
    {
        return localiza;
    }
}

```

13.2 Conexiones de Sockets de flujos (cliente /servidor)

Los programas que se muestran a continuación sirven para demostrar una aplicación cliente servidor. La aplicación cliente se conecta con el servidor, la aplicación servidor envía datos al cliente y muestra los datos recibidos.

La clase Servidor hereda de la clase Frame, por lo que se proporciona un constructor para establecer los componentes gráficos de la ventana.

Deberá abrir dos sesiones de interfaz de comandos para ejecutar :

- en la primera sesión:
 - o java Servidor
- en la segunda sesión:
 - o java Cliente

Ejemplo 1: Hacer un Programa servidor - cliente, donde se hace la conexión al cliente. El servidor debe enviar un mensaje al cliente de conexión recibida.

- **Programa Applet que permite crear el Servidor**

```
import java.awt.*;
import java.applet.*;
import java.net.*;
import java.io.*;
public class Servidor extends Frame
{
    TextArea ver;
    public Servidor()
    {
        super("Servidor");
        ver=new TextArea(20,5);
        add("Center",ver);
        resize(300,150);
        show();
    }
    public void runServidor()
    {
        ServerSocket servidor;
        Socket conexion;
        OutputStream salida;
        try
        {
            servidor=new ServerSocket(5000,100);
            conexion=servidor.accept();
            ver.setText("Conexion recibida...\n");
            ver.appendText(" Enviando datos...\n");
            salida=conexion.getOutputStream();
            String s=new String("Conexion establecida\n");
            for(int i=0;i<s.length();i++)
                salida.write((int)s.charAt(i));
            ver.appendText("Transmisión terminada. Cerrada socket.. \n");
            conexion.close();
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
    public boolean handleEvent(Event e)
    {
        if(e.id==Event.WINDOW_DESTROY)
        {
            hide();
            dispose();
            System.exit(0);
        }
        return super.handleEvent(e);
    }
    public static void main(String a[])
    {

```

```

Servidor s=new Servidor();
s.runServidor();
}
}

```

- **Programa Applet que permite crear el Cliente**

```

import java.awt.*;
import java.applet.*;
import java.net.*;
import java.io.*;
public class Cliente extends Frame
{
    TextArea ver;
    public Cliente()
    {
        super("Cliente");
        ver=new TextArea(20,5);
        add("Center",ver);
        resize(300,150);
        show();
    }
    public void runCliente()
    {
        Socket cliente;
        InputStream entrada;
        try
        {
            cliente=new Socket(InetAddress.getLocalHost(),5000);
            ver.appendText(" Socket creado ...\n");
            entrada=cliente.getInputStream();
            ver.appendText(" Flujo de entrada creado..\n");
            ver.appendText(" El texto del servidor..\n\t");
            char c;
            while((c=(char)entrada.read())!='\n')
                ver.appendText(String.valueOf(c));
            ver.appendText("\n");
            cliente.close();
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
    public boolean handleEvent(Event e)
    {
        if(e.id==Event.WINDOW_DESTROY)
        {
            hide();
            dispose();
            System.exit(0);
        }
        return super.handleEvent(e);
    }
    public static void main(String a[])

```

```

{
    Cliente s=new Cliente();
    s.runCliente();
} }

```

Ejemplo 2: Hacer un Programa servidor - cliente, donde se hace la conexión el cliente recibe una pregunta y le pide al usuario una respuesta. El servidor verifica la respuesta si es correcta le pregunta al cliente que si desea continuar, en caso contrario si la respuesta es incorrecta le muestre la respuesta correcta y le pregunte si desea continuar.

Nota : es importante tener en cuenta lo siguiente:

- 1) Se debe crear un archivo llamado preguntas.txt, el cual contendrá la pregunta y la respectiva respuesta. Ejemplo:
 - Cual es el presidente de Colombia?
 - Uribe

Se debe elaborar varias preguntas para que el servidor pueda escoger aleatoriamente la pregunta.

- 2) Como lo más posible es que no estemos conectados en red, podemos crear el servidor y el cliente en el mismo computador realizando los siguientes pasos:
 - a. Abrir una sesión en el sistema operacional (Interfaz de Comando)

- Digitar:


```
java ServidorUniversidad "localhost"
```

ó

```
java ServidorUniversidad "127.0.0.1"
```

el argumento"localhost" reemplaza la dirección del servidor

- b. Abrir otra sesión en el sistema operativo

- Digitar:


```
java Universidad "localhost"
```

- Programa para crear el Servidor

```

import java.io.*;
import java.net.*;
import java.util.Random;
public class ServidorUniversidad extends Thread
{
    private static final int numero_puerto=1234;
    private static final int espera_para_cliente=0;
    private static final int espera_para_respuesta=1;
    private static final int espera_para_confirmar=2;
    private String[] cuestionario;
    private String[] respuesta;
    private ServerSocket servidorsocket;
    private int numero_cuestionario;
    private int numero=0;
    private int estado=espera_para_cliente;
    private Random aleatorio=new Random();
    public ServidorUniversidad()
    {
        super("Servidor de la Distrital");
    }
}

```

```

try
{
    servidorsocket=new ServerSocket(numero_puerto);
    System.out.println("Servidor de la Distrital Ejecutandose.....");
}
catch(IOException e)
{
    System.err.println("No se puede hacer conexión...");
    System.exit(1);
}
}
public static void main(String[] argumentos)
{
    ServidorUniversidad servidor=new ServidorUniversidad();
    servidor.start();
}
public void run()
{
    Socket clientesocket=null;
    if(!iniciarpreguntas())
    {
        System.err.println("No se puede inicializar preguntas y respuestas...");
        return;
    }
    while(true)
    {
        if(servidorsocket==null)
            return;
        try
        {
            clientesocket=servidorsocket.accept();
        }
        catch(IOException e)
        {
            System.err.println("No se puede hacer conexión el cliente...");
            System.exit(1);
        }
        try{
            InputStreamReader esrespuesta=new
InputStreamReader(clientesocket.getInputStream());
            BufferedReader es= new BufferedReader(esrespuesta);
            PrintWriter un=new PrintWriter(new
BufferedOutputStream(clientesocket.getOutputStream()),false);
            String salidalinea;
            salidalinea=procesoentrada(null);
            un.println(salidalinea);
            un.flush();
            while(true)
            {
                String enlinea=es.readLine();
                if(enlinea.length()>0)
                {
                    salidalinea=procesoentrada(enlinea);
                    un.println(salidalinea);
                }
            }
        }
    }
}

```

```

        un.flush();
        if(salidalinea.equals("Adios."))
            break;
    }
}
un.close();
es.close();
clientesocket.close();
}
catch(IOException e)
{
    System.err.println("Excepcion..." + e);
    e.printStackTrace();
}
}
}
private boolean iniciarpreguntas()
{
    try{
        File entradaarchivo =new File("preguntas.txt");
        FileInputStream enflujo=new FileInputStream(entradaarchivo);
        byte[] dato=new byte[(int)entradaarchivo.length()];
        if(enflujo.read(dato)<=0){
            System.err.println("No se puede leer preguntas y respuestas");
            return false;
        }
        for(int i=0;i<dato.length;i++)
            if(dato[i]==(byte)\n')
                numero_cuestionario++;
        numero_cuestionario/=2;
        cuestionario=new String[numero_cuestionario];
        respuesta=new String[numero_cuestionario];
        int arranca=0,indice=0;
        boolean esq=true;
        for(int i=0;i<dato.length;i++)
            if(dato[i]==(byte)\n')
            {
                if(esq)
                {
                    cuestionario[indice]= new String(dato,arranca,i-arranca-1);
                    esq=false;
                }
                else
                {
                    respuesta[indice]=new String(dato,arranca,i-arranca-1);
                    esq=true;
                    indice++;
                }
            }
        arranca=i+1;
    }
}
catch(FileNotFoundException e)
{
    System.err.println("No se puede encontrar el archivo de cuestionarios");
}

```

```

        return false;
    }
    catch(IOException e)
    {
        System.err.println("Error de lectura del archivo de cuestionarios");
        return false;
    }
    return true;
}

String procesoentrada(String entradacadena)
{
    String salidacadena=null;
    switch(estado){
        case espera_para_cliente:
            salidacadena=cuestionario[numero];
            estado=espera_para_respuesta;
            break;
        case espera_para_respuesta:
            if(entradacadena.equalsIgnoreCase(respuesta[numero]))
                salidacadena="es correcto, continuar (y/n) ";
            else
                salidacadena="la respuesta correcta es: "+respuesta[numero]+" continuar(y/n):";
            estado=espera_para_confirmar;
            break;
        case espera_para_confirmar:
            if(entradacadena.equalsIgnoreCase("Y")){
                numero=Math.abs(aleatorio.nextInt())%cuestionario.length;
                salidacadena=cuestionario[numero];
                estado=espera_para_respuesta;
            }
            else {
                salidacadena="Adios.";
                estado=espera_para_cliente;
            }
            break;
    }
    return salidacadena;
}
}

```

- Programa que permite crear el Cliente

```

import java.io.*;
import java.net.*;
public class Universidad{
    private static final int numero_puerto=1234;
    public static void main(String[] argumentos)
    {
        Socket socket =null;
        InputStreamReader esrespuesta=null;
        BufferedReader entrada=null;
        PrintWriter salida=null;
        String direccion;
    }
}

```

```

if(argumentos.length!=1)
{
    System.out.println("Use la direccion ");
    return ;
}
else
    direccion=argumentos[0];
try
{
    socket=new Socket(direccion, numero_puerto);
    esrespuesta=new InputStreamReader(socket.getInputStream());
    entrada=new BufferedReader(esrespuesta);
    salida=new PrintWriter(socket.getOutputStream());
}
catch(IOException e)
{
    System.err.println("Error"+e.getMessage());
    System.exit(1);
}
try
{
    StringBuffer cadena=new StringBuffer(128);
    String entradacadena;
    int c;
    while((entradacadena=entrada.readLine())!=null)
    {
        System.out.println("Servidor : "+entradacadena);
        if(entradacadena.equals("Adios."))
            break;
        while((c=System.in.read())!='\n')
            cadena.append((char)c);
        System.out.println("Cliente : "+cadena);
        salida.println(cadena.toString());
        salida.flush();
        cadena.setLength(0);
    }
    salida.close();
    entrada.close();
    socket.close();
}
catch(IOException e)
{
    System.err.println("I/O Error"+e.toString());
}
}
}

```

13.3 Utilización de Socket de datagrama (Servidor – Cliente)

Ejemplo: Hacer un Programa servidor - cliente, donde se utilice los datagramas. El programa permitirá desde el cliente enviarle un mensaje al servidor. El servidor retornara la información que le ha sido enviada al cliente.

- **Programa que permite crear el Servidor**

```
import java.awt.*;
import java.io.*;
import java.net.*;
public class Servidor2 extends Frame
{
    TextArea ver;
    DatagramPacket enviar, recibir;
    DatagramSocket enviars, recibirs;
    public Servidor2()
    {
        super("Servidor");
        ver=new TextArea(20,10);
        add("Center",ver);
        resize(400,300);
        show();
        try{
            enviars=new DatagramSocket();
            recibirs=new DatagramSocket(5000);
        }
        catch(SocketException ex)
        {
            ex.printStackTrace();
            System.exit(1);
        }
    }
    public void esperar_paquete()
    {
        while(true) {
            try{
                byte arreglo[]=new byte[100];
                recibir=new DatagramPacket(arreglo,arreglo.length);
                recibirs.receive(recibir);
                ver.appendText("\nPaquete recibido:"+ "\n del anfitrión: "+recibir.getAddress()+
                    "\n puerto del host: "+recibir.getPort()+ "\nLongitud: "+recibir.getLength()+"\n Que
                    contiene:\n\t");
                byte dato[]=recibir.getData();
                String recibido=new String(dato,0);
                ver.appendText(recibido);
                ver.appendText("\n\nDevolviendo informacion al cliente....");
                enviar=new DatagramPacket(dato, dato.length,recibir.getAddress(),5001);
                enviars.send(enviar);
                ver.appendText("\nPaquete enviado \n");
            }
            catch(IOException exc)
            {
                ver.appendText(exc.toString()+"\n");
                exc.printStackTrace();
            }
        }
    }
}
```

```

    }
}
public boolean handleEvent(Event e)
{
    if(e.id==Event.WINDOW_DESTROY)
    {
        hide();
        dispose();
        System.exit(0);
    }
    return super.handleEvent(e);
}
public static void main(String arg[])
{
    Servidor2 s=new Servidor2();
    s.esperar_paquete();
}
}

```

- **Programa que permite crear el Cliente**

```

import java.awt.*;
import java.io.*;
import java.net.*;
public class Cliente2 extends Frame
{
    TextArea ver;
    TextField entrada;
    Panel entradap;
    Label entradal;
    DatagramPacket enviar, recibir;
    DatagramSocket enviars,recibirs;
    public Cliente2()
    {
        super("Cliente");
        entradap=new Panel();
        entradal=new Label("Teclee mensaje");
        entrada=new TextField(20);
        ver=new TextArea(20,10);
        entradap.add(entradal);
        entradap.add(entrada);
        add("North",entradap);
        add("Center",ver);
        resize(400,300);
        show();
        try{
            enviars=new DatagramSocket();
            recibirs=new DatagramSocket(5001);
        }
        catch(SocketException ex)
        {ex.printStackTrace();
        System.exit(1);
        }
    }
}

```

```

    }
}
public void esperar_paquete()
{
    while(true) {
        try{
            byte arreglo[]=new byte[100];
            recibir=new DatagramPacket(arreglo,arreglo.length);
            recibirs.receive(recibir);
            ver.appendText("\nPaquete recibido:"+ "\n del anfitrión: "+recibir.getAddress()+
            "\n puerto del anfitrión: "+recibir.getPort()+ "\nLongitud: "+recibir.getLength()+ "\n Que
            contienen:\n\t");
            byte dato[]=recibir.getData();
            String recibido=new String(dato,0);
            ver.appendText(recibido);
        } catch(IOException exc)
        {
            ver.appendText(exc.toString()+"\n");
            exc.printStackTrace();
        }
    }
}
public boolean handleEvent(Event e)
{
    if(e.id==Event.WINDOW_DESTROY)
    {
        hide();
        dispose();
        System.exit(0);
    }
    return super.handleEvent(e);
}
public boolean action(Event ev, Object o)
{
    try{
        ver.appendText("\n Enviando paquete que contiene: :"+ o.toString()+"\n");
        String s=o.toString();
        byte dato[]=new byte[100];
        s.getBytes(0,s.length(),dato,0);
        enviar=new DatagramPacket(dato,s.length(),InetAddress.getLocalHost(),5000);
        enviars.send(enviar);
        ver.appendText("paquete enviado\n");
    }
    catch(IOException exc)
    {
        ver.appendText(exc.toString()+"\n");
        exc.printStackTrace();
    }
    return true;
}
public static void main(String arg[])
{
    Cliente2 c=new Cliente2();
    c.esperar_paquete();
}

```

```
}  
}
```

13.4 Ejercicios

1. Utilizando una conexión Socket realizar un programa que permita a un cliente especificar el nombre de un archivo y hacer que el servidor envíe el contenido del archivo.
2. Utilizando una conexión Socket realizar un programa que permita a un cliente modificar un archivo enviado por el servidor y devolverlo al servidor ya modificado.
3. Escriba un programa para jugar “veintiuna”, en el que el servidor reparta los naipes a un cliente.
4. Escriba un programa para jugar “poker”, en el que el servidor reparta los naipes a un cliente.
5. Escriba un programa para jugar “triqui”, entre el servidor y un cliente.
6. Escriba un programa que permita capturar la ip de un cliente y enviarle un mensaje.
7. Escriba un programa que permita que un cliente le envíe un mensaje al servidor y este le responda.
8. Escriba un programa que permita que el servidor le envíe un archivo al cliente.
9. Escriba un programa que permita que un cliente le envíe una dirección IP al servidor y este le envíe un mensaje al dueño de esa IP.
10. Escriba un programa que permita que un cliente le envíe un mensaje a otro cliente.

14. Bases de datos con Java (JDBC)

JDBC (Java DataBase Connectivity) es un API de Java que permite al programador ejecutar instrucciones en lenguaje estándar de acceso a Bases de Datos SQL (Structured Query Language, Lenguaje estructurado de consultas). El API JDBC consiste de un conjunto de clases e interfaces que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea. En otras palabras, con el API JDBC no es necesario escribir un programa para acceder a Sybase, otro programa para acceder a Oracle, y otro programa para acceder a MySQL; con esta API, se puede crear un sólo programa en Java que sea capaz de enviar sentencias SQL a la base de datos apropiada.

Al igual que ODBC, la aplicación de Java debe tener acceso a un controlador (*driver*) JDBC adecuado. Este controlador es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real. De manera muy simple, al usar JDBC se pueden hacer tres cosas:

- Establecer la conexión a una base de datos, ya sea remota o no
- Enviar sentencias SQL a esa base de datos
- Procesar los resultados obtenidos de la base de datos.

Por medio de las siguientes clases:

Clase/Interface	Descripción
Driver	Permite conectarse a una Base de Datos: cada gestor de Base de Datos requiere un Driver distinto.
DriverManager	Permite gestionar todos los Drivers instalados en el sistema.
DriverPropertyInfo	Proporciona diversa información acerca de un Driver.
Connection	Representa una conexión con una Base de Datos. Una aplicación puede tener más de una conexión a más de una Base de Datos.
DatabaseMetadata	Proporciona información acerca de una Base de Datos, como las tablas que contiene, etc.
Statement	Permite ejecutar sentencias SQL sin parámetros.
PreparedStatement	Permite ejecutar sentencias SQL con parámetros de entrada.

CallableStatement	Permite ejecutar sentencias SQL con parámetros de entrada y salida, típicamente procedimientos almacenados.
ResultSet	Contiene las filas o registros obtenidos al ejecutar un SELECT.
ResultSetMetadata	Permite obtener información sobre un ResultSet, como el número de columnas, sus nombres, etc.

Distribuida en dos paquetes:

- java.sql, dentro de Java 2
- javax.sql JDBC Standard Extension

Los distribuidores de bases de datos suministran los controladores que implementan el API JDBC y que permiten acceder a sus propias implementaciones de bases de datos. De esta forma JDBC proporciona a los programadores de Java una interfaz de alto nivel y les evita el tener que tratar con detalles de bajo nivel para acceder a bases de datos.

14.1 Que es una Base de Datos

Es un conjunto exhaustivo no redundante de datos estructurados organizados independientemente de su utilización y su implementación en máquina, accesibles en tiempo real y compatibles con usuarios concurrentes con necesidad de información diferente y no predicable en tiempo.

14.2 Tipos de Bases de Datos (BD)

14.2.1 Relacionales

Son las que más se utilizan. Las bases de datos relacionales son un conjunto de tablas relacionadas entre sí, cada tabla esta definida por una serie de campos. Los campos forman las columnas de las tablas; definen el tipo y la variedad de sus datos. Las filas de datos se denominan registros (tuplas), cada tipo definido en un registro se le denomina atributo. Las tablas pertenecientes a una base de datos pueden relacionarse entre sí utilizando campos clave comunes entre las tablas.

14.2.2 Enfoque Orientado a Objetos

el esquema de una base de datos por objetos está representado por un conjunto de clases que definen las características y el comportamiento de los objetos que poblarán la base de datos. Con una base de datos orientada a objetos, los objetos memorizados en la base de datos contienen tanto los datos como las operaciones posibles con tales datos. En cierto

sentido, se podrá pensar en los objetos como en datos a los que se les ha puesto una inyección de inteligencia que les permite saber cómo comportarse, sin tener que apoyarse en aplicaciones externas.

14.3 Lenguaje de consulta estructurado (S.Q.L.)

Es un lenguaje de base de datos normalizado, utilizado por los diferentes motores de bases de datos para realizar determinadas operaciones sobre los datos o sobre la estructura de los mismos.

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

14.3.1 Comandos

Existen dos tipos de comandos SQL:

- DDL que permiten crear y definir nuevas bases de datos, campos e índices.
- DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DDL	
Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e índices
DROP	Empleado para eliminar tablas e índices
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.
Comandos DML	
Comando	Descripción
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados
DELETE	Utilizado para eliminar registros de una tabla de una base de datos

14.3.2 Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Cláusula	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

14.3.3 Operadores Lógicos

Operador	Uso
AND	Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

14.3.4 Operadores de Comparación

Operador	Uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor o igual que
>=	Mayor o igual que
=	Igual que
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo
In	Utilizado para especificar registros de una base de datos

14.3.5 Funciones de Agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Función	Descripción
AVG	Utilizada para calcular el promedio de los valores de un campo determinado
COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un campo especificado
MIN	Utilizada para devolver el valor más bajo de un campo especificado

14.4 Acceder a una Base de Datos

14.4.1 Establecer una Conexión

Lo primero que tenemos que hacer es establecer una conexión con el controlador de base de datos que queremos utilizar. Esto implica dos pasos: (1) cargar el driver y (2) hacer la conexión.

- **Cargar los Drivers:** Cargar el driver o drivers que queremos utilizar es muy sencillo y sólo implica una línea de código. Si, por ejemplo, queremos utilizar el puente JDBC-ODBC, se cargaría la siguiente línea de código:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

La documentación del driver nos dará el nombre de la clase a utilizar. Por ejemplo, si el nombre de la clase es **jdbc.DriverXYZ**, cargaríamos el driver con esta línea de código:

```
Class.forName("jdbc.DriverXYZ");
```

No necesitamos crear un ejemplar de un driver y registrarlo con el **DriverManager** porque la llamada a **Class.forName** lo hace automáticamente. Si hubiéramos creado nuestro propio ejemplar, crearíamos un duplicado innecesario, pero no pasaría nada. Una vez cargado el driver, es posible hacer una conexión con un controlador de base de datos.

- **Hacer la Conexión:** El segundo paso para establecer una conexión es tener el driver apropiado conectado al controlador de base de datos. La siguiente línea de código ilustra la idea general:

```
Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");
```

Este paso también es sencillo, lo más duro es saber qué suministrar para **url**. Si estamos utilizando el puente JDBC-ODBC, el JDBC URL empezará con **jdbc:odbc:**. el resto de la URL normalmente es la fuente de nuestros datos o el sistema de base de datos. Por eso, si estamos utilizando ODBC para acceder a una

fuentes de datos ODBC llamada "Mijdbc" por ejemplo, nuestro URL podría ser **jdbc:odbc:Mijdbc**. En lugar de "myLogin" pondríamos el nombre utilizado para entrar en el controlador de la base de datos; en lugar de "myPassword" pondríamos nuestra password para el controlador de la base de datos. Por eso si entramos en el controlador con el nombre "Bases" y el password "Datos," estas dos líneas de código establecerán una conexión:

```
String url = "jdbc:odbc:Mijdbc";  
Connection con = DriverManager.getConnection(url, "Bases", "Datos");
```

Si estamos utilizando un puente JDBC desarrollado por una tercera parte, la documentación nos dirá el subprotocolo a utilizar, es decir, qué colocar después de **jdbc:** en la URL. Por ejemplo, si el desarrollador ha registrado el nombre "acme" como el subprotocolo, la primera y segunda parte de la URL de JDBC serán **jdbc:acme:**. La documentación del driver también nos dará las guías para el resto de la URL del JDBC. Esta última parte de la URL suministra información para la identificación de los datos fuente.

Si uno de los drivers que hemos cargado reconoce la URL suministrada por el método **DriverManager.getConnection**, dicho driver establecerá una conexión con el controlador de base de datos especificado en la URL del JDBC. La clase **DriverManager**, como su nombre indica, maneja todos los detalles del establecimiento de la conexión detrás de la escena. A menos que estemos escribiendo un driver, posiblemente nunca utilizaremos ningún método del interface **Driver**, y el único método de **DriverManager** que realmente necesitaremos conocer es **DriverManager.getConnection**.

La conexión devuelta por el método **DriverManager.getConnection** es una conexión abierta que se puede utilizar para crear sentencias JDBC que pasen nuestras sentencias SQL al controlador de la base de datos. En el ejemplo anterior, **con** es una conexión abierta.

14.4.2 Seleccionar Tablas

Un objeto **Statement** es el que envía nuestras sentencias SQL al controlador de la base de datos. Simplemente creamos un objeto **Statement** y lo ejecutamos, suministrando el método SQL apropiado con la sentencia SQL que queremos enviar. Para una sentencia **SELECT**, el método a ejecutar es **executeQuery**. Para sentencias que crean o modifican tablas, el método a utilizar es **executeUpdate**.

Se toma un ejemplar de una conexión activa para crear un objeto **Statement**. En el siguiente ejemplo, utilizamos nuestro objeto **Connection: con** para crear el objeto **Statement: stmt**:

```
Statement stmt = con.createStatement();
```

En este momento **stmt** existe, pero no tiene ninguna sentencia SQL que pasarle al controlador de la base de datos. Necesitamos suministrarle el método que utilizaremos para ejecutar **stmt**. Por ejemplo, en el siguiente fragmento de código, suministramos **executeUpdate** con la sentencia SQL para crear una tabla:

```
stmt.executeUpdate("CREATE TABLE CLIENTES " + "(NOMBRE VARCHAR(32),  
NIT INTEGER, CIUDAD VARCHAR(20), TELEFONO VARCHAR(15), CREDITO  
INTEGER)");
```

Ejemplo : Hacer un programa que utilice el JDBC, que permite adicionar y visualizar registros, utilizando un tabla de Access.

Para realizar el programa debemos realizar los siguientes pasos:

1. Crear Tabla en Access: la estructura de la tabla DatosPersonales será:

- Nombre Texto 20
- Apellido Texto 20
- edad Texto 3
- Telefono Texto 10

2. Crear el puente ODBC::JDBC : debemos abrir el panel de control y escoger el icono "Fuente de Datos ODBC", el cual nos mostrará la ventana "Administrador de orígenes de datos ODBC", donde realizamos:

- En página DSN de Usuario pulsamos el botón "agregar".
- Seleccionamos el Driver que necesitamos en este caso "Controlador para Microsoft Access (*.mdb)".
- Aparecerá la ventana de configuración de ODBC Microsoft Access, en la opción origen de Datos escribe "Personal".
- En el botón "Seleccionar", seleccionar la base de datos "Datospersonales" y pulsar el botón "aceptar".
- En la opción "Avanzadas" puedes colocar el administrador y el password.
- Cerrar la ventana de "Administrador de orígenes de datos ODBC".

Nota: Los datos que se escriben son sensibles a las mayúsculas, ejemplo: DatosPersonales.mdb es diferente de datosPersonales.mdb

3. Realizar el código fuente de Java

*/*Este Código fue escrito como ejemplo de aplicaciones JDBC para el curso Ingeniería de Software I en la Universidad Distrital "FJC" en Colombia. Si planea utilizarlo, no dude en hacerlo. Pero si lo quiere publicar, por favor cite a los autores:*

**David E. Acosta R. dacosta@satannet.org*

**Germán D. Ballén R. slamort@yahoo.com*/*

```
import java.awt.*;
import java.sql.*; //Se invocan todas las funciones del paquete SQL JDBC
import java.awt.event.*;
```

```
class DatosPersonales extends Frame implements ActionListener
{
```

```
    Button crear, ver, insertar, cerrar;
    TextField informacion;
    Panel principal;
    Connection conexion;
```

```
DatosPersonales()
```

```
{
    super ("Datos Personales");
    setSize (200,120);
    principal=new Panel();
    crear=new Button ("Crear");
    crear.addActionListener(this);
    ver=new Button ("Ver");
    ver.addActionListener(this);
```

```

insertar=new Button ("Insertar");
insertar.addActionListener(this);
cerrar=new Button ("Cerrar");
cerrar.addActionListener(this);
informacion=new TextField(20);
principal.add(informacion);
principal.add(crear);
principal.add(insertar);
principal.add(ver);
principal.add(cerrar);
principal.setBackground(SystemColor.control);
add(principal);
setVisible(true);
try
{
    /*Cargamos el controlador*/
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch(java.lang.ClassNotFoundException e)
{
    System.err.print("ClassNotFoundException: ");
    System.err.println(e.getMessage());
    informacion.setText("No se pudo cargar el controlador JDBC-ODBC");
}
}
private void Crear_tabla()
{
    String url = "jdbc:odbc:Personal";
    String createString;
    Statement sentencia;
    createString = "create table DATOSPERSONALES " +
        "(nombre varchar(20), " +
        "apellidos varchar(20), " +
        "edad varchar(3), " +
        "telefono varchar(10))";

    //Se carga el controlador
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(java.lang.ClassNotFoundException e)
    {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }
    try
    {
        //Se carga la conexión
        conexion = DriverManager.getConnection(url,
            "jdbc", "jdbc");
        sentencia = conexion.createStatement();
        //Borramos la tabla en caso de existir
        sentencia.executeUpdate("DROP TABLE DATOSPERSONALES");
    }
}

```

```

        //Creamos nuevamente la tabla
        sentencia.executeUpdate(createString);
        informacion.setText("Tabla creada");
        conexion.close();
    }
    catch(SQLException ex)
    {
        System.err.println("SQLException: " + ex.getMessage());
    }
}
public void actionPerformed(ActionEvent e)
{
    String com=e.getActionCommand();
    if ("Crear".equals(com))
    {
        informacion.setText(" ");
        Crear_tabla();
    }
    else
    {
        if ("Insertar".equals(com))
        {
            new Insertar(this);
        }
        else
        {
            if ("Ver".equals(com))
            {
                new Ver(this);
            }
            else
            {
                dispose();System.exit(0);
            }
        }
    }
}
//Este es el método principal
public static void main(String args[])
{
    new DatosPersonales();
}
}

```

/*Implementacion de la clase insertar*/

```

class Insertar extends Dialog implements ActionListener
{
    private Connection conexion;
    private Button incluir,terminar;
    private TextField nombre,apellidos,edad,telefono;

    Insertar(Frame f)
    {
        super(f,"Insertar datos",true);
        setSize(310,160);
        nombre=new TextField(20);
        apellidos=new TextField(20);
    }
}

```

```

edad=new TextField(3);
telefono=new TextField(10);
incluir=new Button("Incluir");
incluir.addActionListener(this);
terminar=new Button("Terminar");
terminar.addActionListener(this);
Panel P_Datos=new Panel();
P_Datos.add(new Label("Nombre      : "));
P_Datos.add(nombre);

P_Datos.add(new Label("Apellidos    : "));
P_Datos.add(apellidos);
P_Datos.add(new Label("Edad      : "));
P_Datos.add(edad);
P_Datos.add(new Label("Telefono   : "));
P_Datos.add(telefono);
P_Datos.add(incluir);
P_Datos.add(terminar);
nombre.setEditable(true);
apellidos.setEditable(true);
edad.setEditable(true);
telefono.setEditable(true);
add(P_Datos);
setVisible(true);
}
private void insertar_fila()
{
    Statement sentencia;
    String url = "jdbc:odbc:Personal";
    try
    {
        conexion = DriverManager.getConnection(url,
            "jdbc", "jdbc");
        sentencia=conexion.createStatement();
        sentencia.executeUpdate("INSERT          INTO          DATOSPERSONALES
"+"VALUES(""+nombre.getText()+""+" "+" "+apellidos.getText()+""+" "+" "+edad.getText()+""+" "+" "+t
elefono.getText()+""+"");

        sentencia.executeUpdate("INSERT          INTO          DATOSPERSONALES
"+"VALUES(""+identificacion.getText()+""+" "+" "+apellidos.getText()+""+" "+" "+direccion.getText()+
""+" "+" "+telefono.getText()+""+" "+" "+ciudad.getText()+""+" "+" "+credito.getText()+""+"");
    }
    catch(SQLException e){}
}

//Por medio de este método se manejan los clicks del usuario
public void actionPerformed(ActionEvent e)
{
    String com=e.getActionCommand();
    if ("Incluir".equals(com))
    {
        insertar_fila();
        nombre.setText("");
        apellidos.setText("");
    }
}

```

```

        edad.setText("");
        telefono.setText("");
    }
    else
    {
        if(conexion!=null)
        {
            try
            {
                conexion.close();
            }
            catch(SQLException ex) {}
        }
        dispose();
    }
}
}

/*Implementacion de la clase ver*/

class Ver extends Dialog
implements ActionListener
{
    private Connection conexion;
    private ResultSet resultado;
    private Button siguiente, terminar;
    private TextField nombre, apellidos, edad, telefono;
    Ver (Frame f)
    {
        super(f,"Ver datos",true);
        setSize(310,160);
        nombre=new TextField(20);
        apellidos=new TextField(20);
        edad=new TextField(3);
        telefono=new TextField(10);
        siguiente=new Button("Siguiente");
        siguiente.addActionListener(this);
        terminar=new Button("Terminar");
        terminar.addActionListener(this);
        Panel P_Datos=new Panel();
        P_Datos.add(new Label("Nombre      : "));
        P_Datos.add(nombre);
        P_Datos.add(new Label("Apellidos   : "));
        P_Datos.add(apellidos);
        P_Datos.add(new Label("Edad       : "));
        P_Datos.add(edad);
        P_Datos.add(new Label("Telefono   : "));
        P_Datos.add(telefono);

        P_Datos.add(siguiente);
        P_Datos.add(terminar);
        add(P_Datos);
        nombre.setEditable(false);
        apellidos.setEditable(false);
    }
}

```

```

        edad.setEditable(false);
        telefono.setEditable(false);
        mostrar();
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e)
    {
        String com=e.getActionCommand();
        if ("Siguiente".equals(com))
            siguiente();
        else
        {
            if (conexion!=null)
            {
                try
                {
                    conexion.close();
                }
                catch(SQLException ex) {}
            }
            dispose();
        }
    }

    private void mostrar()
    {
        String url = "jdbc:odbc:Personal";
        Statement sentencia;
        try{
            /*Se obtiene la conexion*/
            conexion = DriverManager.getConnection(url,
                "jdbc", "jdbc");
            /*Se crea una senetencia del tipo Query*/
            sentencia=conexion.createStatement();
            //obtenemos el resultado de lanzar la sentencia
            resultado=sentencia.executeQuery("select * from DATOSPERSONALES");
            siguiente();
        }
        catch(SQLException e){}
    }

    private void ver_Datos()
    { try
        {
            nombre.setText(resultado.getString("NOMBRE"));
            apellidos.setText(resultado.getString("APELLIDOS"));
            edad.setText(resultado.getString("EDAD"));
            telefono.setText(resultado.getString("TELEFONO"));
        }
        catch(SQLException e){}
    }

    private void siguiente()
    { try

```

```

{
    //nos movemos a la siguiente fila o tupla
    if(resultado.next()) ver_Datos();
}
catch(SQLException e){}
catch(Exception ex){}
}
}

```

14.5 Ejercicios

1. Hacer un programa que permita verificar una conexión a una base de datos.
2. Hacer un programa que permita crear una tabla en una base de datos.
3. Hacer un programa que permita insertar datos una tabla de una base de datos.
4. Hacer un programa que permita consultar los datos una tabla de una base de datos.
5. Hacer un programa que permita modificar un registro de una tabla de una base de datos.
6. Hacer un programa que permita eliminar un registro de una tabla de una base de datos.
7. Hacer un programa que permita buscar un registro de una tabla de una base de datos.
8. Hacer un programa que permita visualizar los registro que cumplan una condición específica de una tabla de una base de datos.
9. Hacer un programa que permita en un campo cambiar el contenido de minusculas a mayusculas.
10. Hacer un programa que permita visualizar unicamente el registro que se quiere modificar de una tabla de una base de datos.

Bibliografía

- WU, Thomas, Introducción a la Programación Orientada a objetos con Java, Editorial McGraw Hill, 2001.
- Naughton, Patrick, Childt, Herbert, Java Manual de Referencia, Editorial McGraw Hill, 1997.
- Wang, Paul, Java con Programación orientada a Objetos y aplicaciones en la www, International Thomson Editores S.A., 2000.
- Lemay, Laura, Cadenhead, Rogers, Aprendiendo Java 2 en 21 días, Editorial Prentice Hall, 1999.
- Deitel, H.M., Como Programar en Java, Editorial Prentice Hall, 1998.
- Bobadilla Jesús, Comunicaciones y Bases de Datos con Java, Editorial Alfaomega, 2003.
- Deitel, H.M., Como Programar en Java, Editorial Prentice Hall, 1998.

Infografía

<http://www.desarrolloweb.com/articulos/497.php?manual=15>
<http://www.desarrolloweb.com/articulos/1670.php?manual=57>
http://enciclopedia.us.es/index.php/Programaci%C3%B3n_orientada_a_objetos
http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos
<http://programarenc.webcindario.com/Cplus/capitulo1.htm>
<http://www.monografias.com/trabajos/objetos/objetos.shtml>
<http://www.mysql-hispano.org/page.php?id=24>
<http://www.programatium.com/sql.htm>