

** CursoJavaPranicalBasico\_ver1\_0**

**Abril 2016**

TABLA DE CONTENIDO

[TABLA DE CONTENIDO 2](#_Toc450564578)

[JAVA 3](#_Toc450564579)

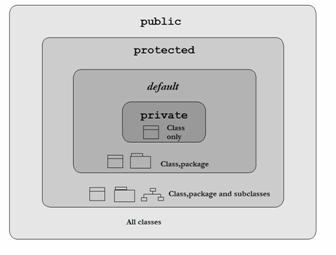
JAVA

- JDK (Java Development Kit) es el kit para desarrolladores, contiene entre otras cosas el JRE y la JVM

- JRE (Java Runtime Environment) es un conjunto de utilidades de Java contiene la JVM

- JVM (Java Virtual Machine) es la aplicacion donde corren los programas hechos en Java.

MODIFICADORES DE ACCESO JAVA: PUBLIC, PRIVATE, PROTECTED.



Modificadores de clases y métodos: STATIC, FINAL, ABSTRACT.

**STATIC:**  
La declaración de atributos de clase o métodos como estático los hace accesibles sin necesitar un instancia de la clase (uso de *new*).

**FINAL:**  
Indica que una variable, método o clase no se va a modificar.

* Aplicado a un atributo significa que es invariante (constante).
* Aplicado a un método significa que no se puede sobrescribir. Es decir la clase que hereda no puede volver a implementar el método.
* Aplicado a una clase significa que no se puede heredar.

**ABSTRACT:**

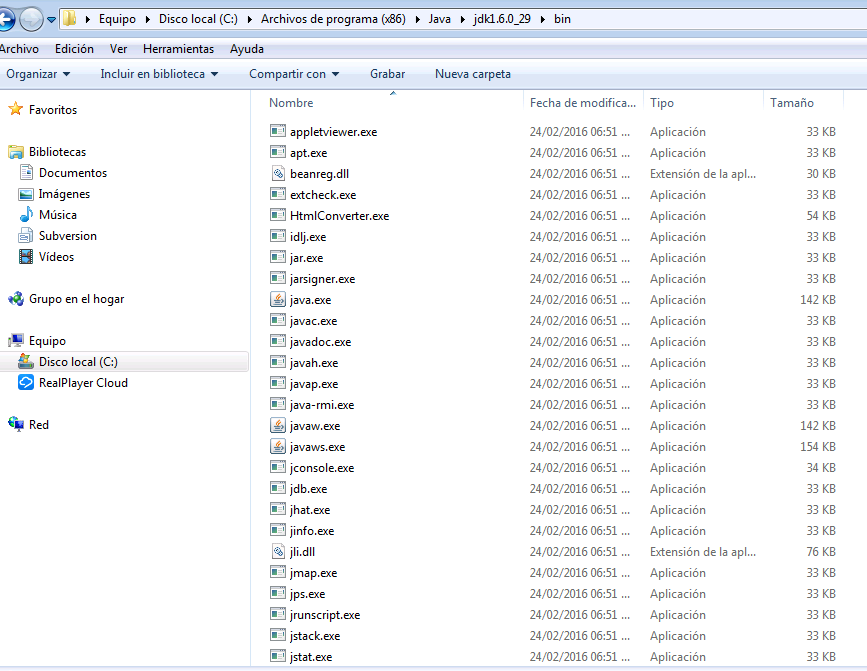
Cuando una clase es abstracta no se permite crear una instancia de la misma. Si una clase contiene un metido abstracto la misma deberá ser declarada como abstracta. Los métodos definidos como abstractos no se implementan en dicha clase simplemente se declara la firma del mismo.

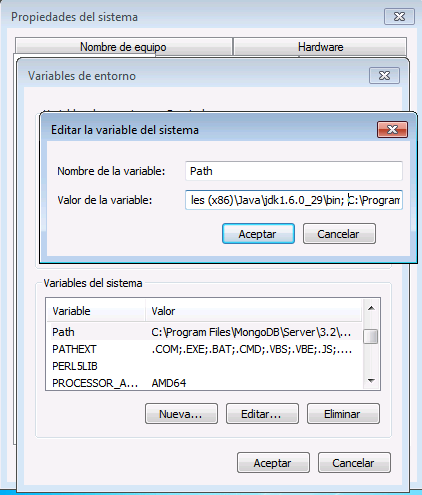
Cuando heredamos de una clase abstracta, todos los métodos definidos como abstractos en la clase padre, deben ser definidos en la clase hijo, teniendo en cuenta que la visibilidad de los mismos no puede ser menor a la definida en el método declarado como abstracto es decir no se puede restringir la visibilidad. (Ejemplo: si un método es declarado como protected en la clase abstracta, cuando implementamos el mismo en la clase hija el mismo deberá ser protected o public pero nunca podrá ser private pues estaríamos restringiendo la visibilidad).

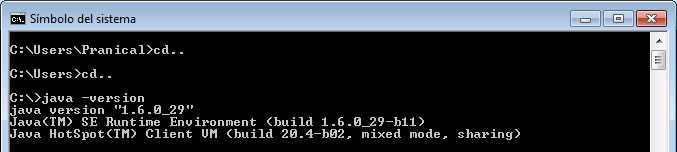
**Resumiendo:**

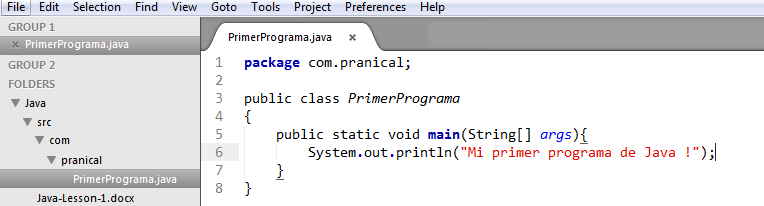
* Aplicado a un método significa que no se implementa en la clase actual.
* Aplicado a una clase significa que no se puede instanciar.

Configurar y Programa JAVA en consola – prompt DOS (windows):

**JDK**





**Código**

Archivo – Clase: PrimerPrograma.java

package com.pranical;

public class PrimerPrograma

{

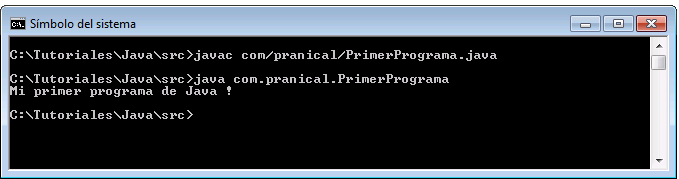
public static void main(String[] args){

System.out.println("Mi primer programa de Java !")

}

}

**Compilar y correr:**

****

Archivo – Clase: PrimerPrograma.java con error

package com.pranical;

public class PrimerPrograma

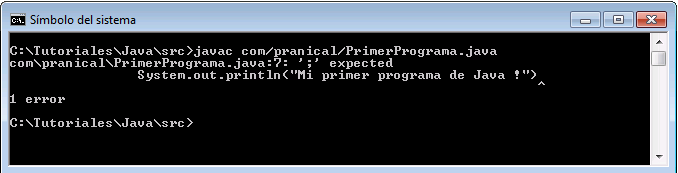
{

public static void main(String[] args){

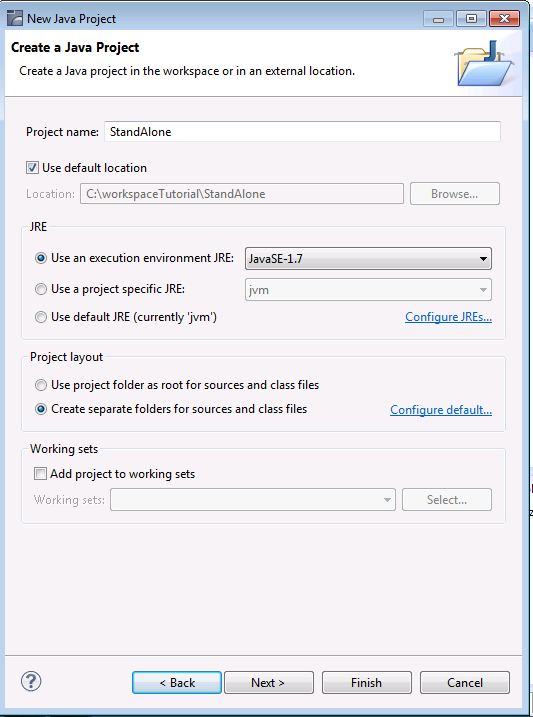
System.out.println("Mi primer programa de Java !") 🡨 Elimino el [;]

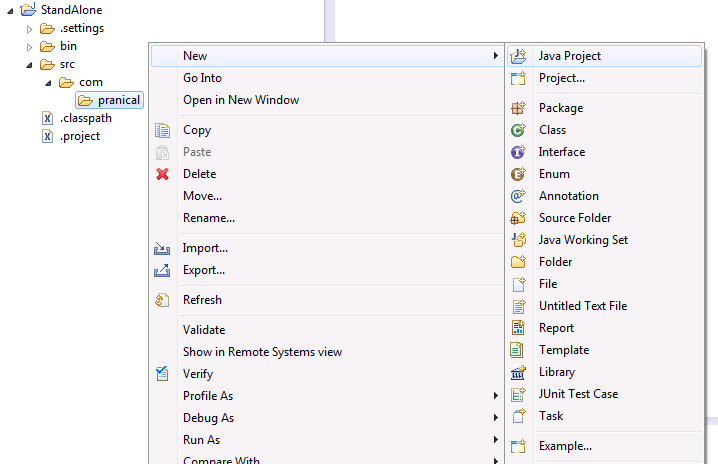
}

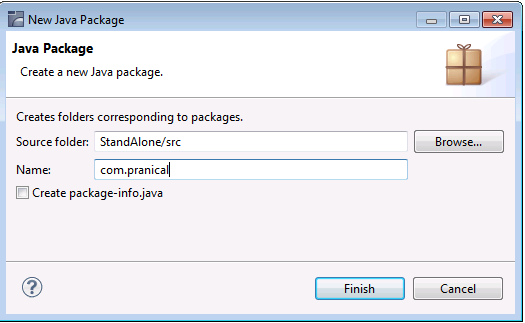
}

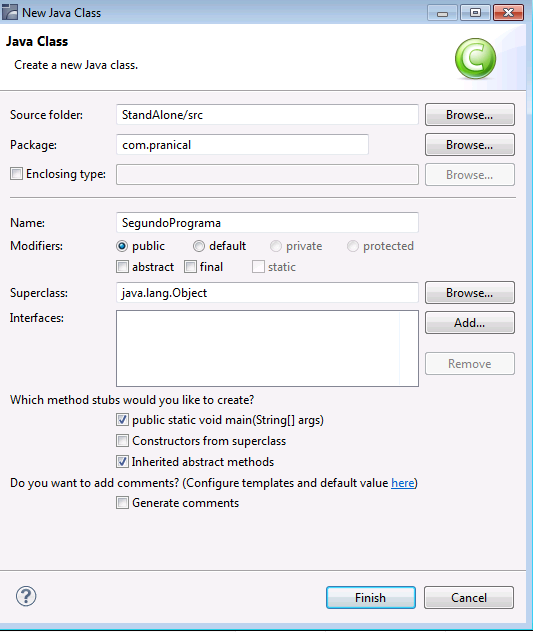
**Compilar**

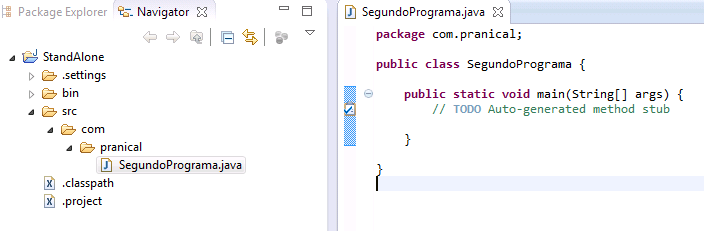
Configurar y Programa JAVA en Eclipse:

**Crear proyecto:**

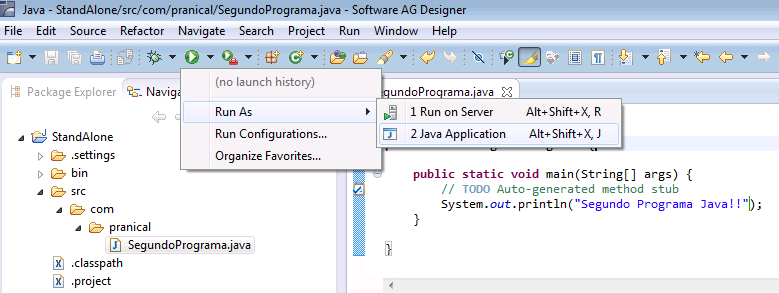
****

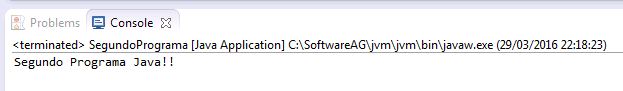
**Crear paquete**

**Crear clase**

****

**Compilar y correr (Java Application)**

****

****

Clase: SegundoPrograma.java con error

**package** com.pranical;

**public** **class** SegundoPrograma {

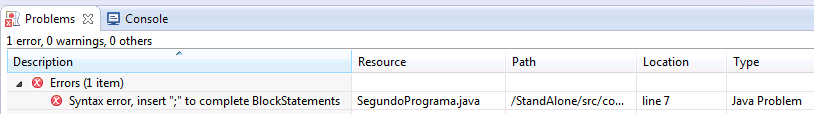
**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.*out*.println("Segundo Programa Java!!") <-- Borro el [;]

}

}

****

Clase:

La programación orientada a objetos se basa en la programación de clases; a diferencia de la programación estructurada, que está centrada en las funciones.

* Una clase es un molde del que luego se pueden crear múltiples objetos, con similares características.
* Una clase es una plantilla (molde), que define atributos (variables) y métodos (funciones)
* La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

**Al crear un objeto de una clase, se dice que se crea una instancia de la clase.**

La estructura de una clase es:

tipo class nombreClase {

atributos de la clase

métodos de la clase

main

}

**Ejercicio**

**package** com.pranical.clase;

**public** **class** MiClase {

**public** String nombre;

**public** **int** cedula;

**public** String telefono;

**public** String correo;

**public** String profesion;

**public** MiClase(String nombre, **int** cedula, String telefono, String correo, String profesion) {

**this**.nombre = nombre;

**this**.cedula = cedula;

**this**.telefono = telefono;

**this**.correo = correo;

**this**.profesion = profesion;

}

**public** **void** imprimir(MiClase miClase) {

System.*out*.println("Nombre: \t"+ miClase.nombre);

System.*out*.println("Cédula: \t"+ miClase.cedula);

System.*out*.println("Teléfono: \t"+ miClase.telefono);

System.*out*.println("Correo: \t"+ miClase.correo);

System.*out*.println("Profesión: \t"+ miClase.profesion);

}

**public** **static** **void** main(String[] ar) {

MiClase miClase =**new** MiClase("William Lopera", 9232978, "0424-1621921", "william.lopera@pranical.com", "Ingeniero");

miClase.imprimir(miClase);

}

}

**Clase - método de instancia y método de clase**

Los métodos determinan el comportamiento y la responsabilidad que tendrán las clases. Representa el uso que se les da a la clase.

 **Métodos de instancia:**

Un método de instancia trabaja sobre el estado actual de la instancia, y para cada instancia tendrán un resultado distinto, por ejemplo, el método deposita inicial permitirá a la instancia cuenta ahorro contar con saldo, mientras que la instancia cuenta corriente estará sin saldo hasta que también se le aplique el método deposito inicial.

Ejemplo 1

**package** com.pranical.clase;

**public** **class** Cuenta{

**private** **double** saldo;

**public** **void** depositoInicial(**double** cantidad)

{

saldo = cantidad;

}

**public** **double** getSaldo()

{

**return** saldo;

}

**public** **static** **void** main(String[] args){

Cuenta ahorro = **new** Cuenta();

ahorro. depositoInicial(25000.00);

Cuenta corriente = **new** Cuenta();

// -> El saldo inicial de mi cuenta de ahorro es: 25.000,00

System.*out*.println("El saldo inicial de mi cuenta de ahorro es: "+ ahorro.getSaldo());

// -> El saldo inicial de mi cuenta corriente es: 0,00

System.*out*.println("El saldo inicial de mi cuenta de corriente es: "+ corriente.getSaldo());

}

}

Ejemplo 2

**package** com.pranical.clase;

**public** **class** SaldoInicialMetodoClase{

**public** **double** saldo = 5000.00;

**public** **static** **void** main(String[] args){

// Error no se puede llamar una referenica no estatica desde un campo estatico

//System.out.println("Saldo inicial: "+ saldo);

SaldoInicialMetodoClase miSaldo = **new** SaldoInicialMetodoClase(); // Creo una instancia

System.*out*.println("Saldo inicial: "+ miSaldo.saldo); // Llamo al parametro no estatico

}

}

**Métodos de clase:**

Los métodos de clase son un comportamiento común a todas las instancias que pertenecen a la misma clase. Al ser un método de clase, no hace falta instanciar un objeto de la clase para utilizarlo. Estos métodos no hablan del estado actual de la clase, sino solo de un comportamiento genérico de la clase, de un procedimiento que solo utiliza los parámetros de entrada o las variables estáticas.

Ejemplo 1

**package** com.pranical.clase;

**public** **class** SaldoInicialMetodoInstancia1 {

**public** **double** saldo = 10000.00;

**public** **static** **void** getMetodo(){

SaldoInicialMetodoInstancia1 miSaldo = **new** SaldoInicialMetodoInstancia1(); // Creo una instancia

System.*out*.println("Saldo inicial: "+ miSaldo.saldo); // Llamo al parametro no estatico

}

**public** **static** **void** main(String[] args){

*getMetodo*();

}

}

Ejemplo 2

**package** com.pranical.clase;

**public** **class** SaldoInicialMetodoInstancia2 {

**public** **static** **double** *saldo* = 10000.00;

**public** **static** **void** main(String[] args){

System.*out*.println("Saldo inicial: "+ *saldo*); // Llamo al parametro no estatico

}

}

Ejemplo 3

**package** com.pranical.clase;

**public** **class** SaldoInicialMetodoInstancia3 {

**public** **static** **void** main(String[] args){

// Llamo al parametro estatico del objeto Cuenta

System.*out*.println("Saldo inicial: "+ CuentaBancaria.*saldo*);

System.*out*.println("Tipo De Cuenta: "+ CuentaBancaria.*getTipoCienta*());

}

}

**class** CuentaBancaria{

**public** **static** **double** *saldo* = 10000.00;

**public** String tipoCuenta = "AHORRO";

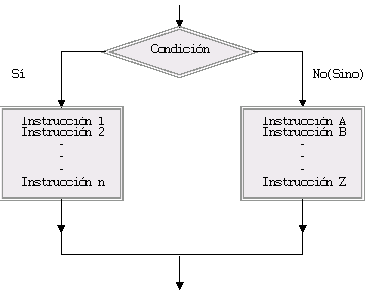
**public** **static** String getTipoCienta(){

**return** **new** CuentaBancaria().tipoCuenta;

}

}

**Estructura o esquema de decisión IF ELSE, IF ELSE IF**

****

/\* Estructura if-else \*/:

if (condición 1) {

        instrucciones

} else if(condición 2){

        instrucciones

} else{

Instrucciones

}

**package com.pranical.test;**

**import** java.util.\*;

**public** **class** TestIfElse {

**public** **static** **void** main(String[] args) {

@SuppressWarnings("resource")

Scanner entrada = **new** Scanner(System.*in*);

**int** opcion;

System.*out*.println("Menú de opciones");

System.*out*.println("-------------------------");

System.*out*.println("1. Calcular el área de un Cuadrado");

System.*out*.println("2. Calcular el área de un Triángulo");

System.*out*.println("3. Calcular el área de un Circulo");

System.*out*.println("4. Finalizar");

System.*out*.print("Elija una opción: ");

opcion = entrada.nextInt();

**if** (opcion == 1) {

System.*out*.println("Ha seleccionado calcular el área de un cuadrado...");

} **else** **if** (opcion == 2) {

System.*out*.println("Ha seleccionado calcular el área de un triángulo...");

} **else** **if** (opcion == 3) {

System.*out*.println("Ha seleccionado calcular el área de un circulo...");

} **else** {

System.*out*.println("Ha seleccionado terminar");

}

}

}

La clase **Scanner**: Facilita la lectura de datos en los programas Java (>Java 4)

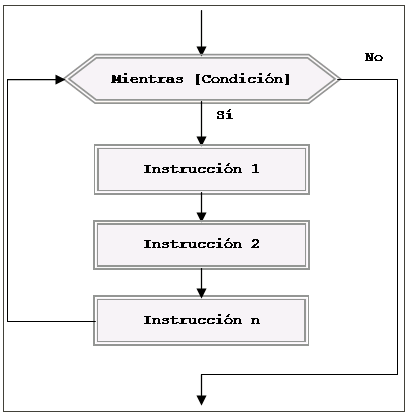
* Para leer podemos usar el método nextXxx() donde Xxx indica en tipo, por ejemplo nextInt() para leer un entero, nextDouble() para leer un double, etc.

**System.in**: Acceso al teclado

**Operador Ternario “?:”**

resultado = (condicion) ? valor1 : valor2;

**Sentencia WHILE**

****/\* Estructura while \*/: La iteración continuará hasta que su condición sea falsa.

while (condición) {

sentencia ;

}

**package com.pranical.test;**

**import** java.util.\*;

**public** **class** TestWhile {

@SuppressWarnings("resource")

**public** **static** **void** main(String[] args) {

Scanner entrada = **new** Scanner (System.*in*); // Acesso a la consola

**int** numero;

**int** calificacion=0;

**int** sumaCalificaciones=0;

**double** promedio = 0.0;

// inicialización

numero = 1;

//ciclo while con condición de terminación

**while** (numero <= 10) {

System.*out*.printf ("Escriba la calificación %d:", numero);

calificacion = entrada.nextInt(); // Espera por la entrada

sumaCalificaciones = sumaCalificaciones + calificacion;

//incremento

numero++;

}

promedio = sumaCalificaciones / (numero \* 1.0);

System.*out*.printf ("El promedio es: %5.2f \n", promedio);

}

}

La clase **Scanner**: Facilita la lectura de datos en los programas Java (Java 5)

* Para leer podemos usar el método nextXxx() donde Xxx indica en tipo, por ejemplo nextInt() para leer un entero, nextDouble() para leer un double, etc.

**System.in**: Acceso al teclado

**Bucle FOR**

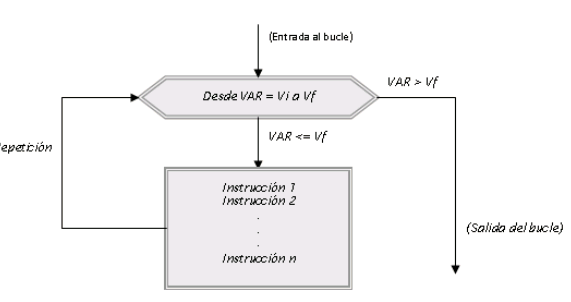
/\* Estructura for \*/:

**for** (**valor inicial** ; **condición de terminación** ; **incremento**)

{

**sentencias**;

}

****

**package com.pranical.test;**

**import** java.util.Scanner;

**public** **class** TestFor1 {

**public** **static** **void** main(String[] args) {

@SuppressWarnings("resource")

Scanner entrada = **new** Scanner (System.*in*);

**int** numeroTabla;

**int** numero;

System.*out*.print ("Cual tabla de multiplicar deseas? ");

numeroTabla = entrada.nextInt();

System.*out*.println ("Tabla de Multiplicar");

**for** (numero = 1; numero <= 10; numero+=1) {

System.*out*.printf(" %d x %d = %d \n",

numeroTabla, numero, numeroTabla \* numero);

}

}

}

La clase **Scanner**: Facilita la lectura de datos en los programas Java (>Java 4)

* Para leer podemos usar el método nextXxx() donde Xxx indica en tipo, por ejemplo nextInt() para leer un entero, nextDouble() para leer un double, etc.

**System.in**: Acceso al teclado

**Nota: La sentencia “break” rompe el ciclo.**

**El bucle for mejorado – for-each**

El bucle “for” mejorado se introdujo con la llegada de Java 5. Este se utiliza principalmente para el manejo de arreglos y listas.

for(declaracion : expresion)

{

//Bloque de código

}

* **Declaración**: La variable de bloque recién declarado, debe ser compatible con los elementos del arreglo que está accediendo. La variable estará disponible dentro del bloque "for" y su valor será el mismo que el elemento dentro del arreglo.
* **Expresión**: Esta se evalúa como el arreglo o lista que se tiene que recorrer. La expresión puede ser una variable de tipo arreglo o una llamada al método que devuelve un arreglo.

**package com.pranical.test;**

**public** **class** TestForMejorado {

**public** **static** **void** main(String args[]){

String [] empleados = {"Anabel", "Karina", "Listbeth", "Rebeca", "Yenny"};

// Ciclo FOR: Declaracion = String nombre , Expresion = empleados

**for**(String nombre : empleados) {

System.*out*.println("Empleada:\t" + nombre);

}

}

}

Ejercicio Cálculo factorial

**package** com.pranical.clase;

**import** java.util.Scanner;

**public** **class** Factorial {

**private** Scanner teclado;

**private** **int** valor;

**public** **void** inicializar() {

teclado=**new** Scanner(System.*in*);

System.*out*.print("Ingrese valor a factorizar [0 <= valor <= 25]");

valor=teclado.nextInt();

}

**public** **long** calcularFactorial() {

**if** (valor < 0 || valor > 25){

**return** -1;

} **else** {

**long** resultado = 1;

**for**(**int** i=valor; i>0; i--)

{

resultado \*= i;

System.*out*.println(i + " -- " + resultado);

}

**return** resultado;

}

}

**public** **void** imprimir(**long** resultado) {

System.*out*.println("Numero:"+valor + " su factorial es: "+ resultado);

}

**public** **static** **void** main(String[] ar) {

Factorial factorial =**new** Factorial();

factorial.inicializar();

**long** resultado = factorial.calcularFactorial();

**if** (resultado > 0)

factorial.imprimir(resultado);

**else**

System.*out*.println("No existe factorial de números negativos. Pruebe [0 <= valor <= 25]");

}

}

**Selección SWITCH**

La instrucción switch es una forma de expresión de un anidamiento múltiple de instrucciones if ... else.

/\* Estructura Switch \*/

switch (expresión) {

        case valor1:

        instrucciones;

        break;

        case valor2:

        instrucciones;

        break;

        .

        .

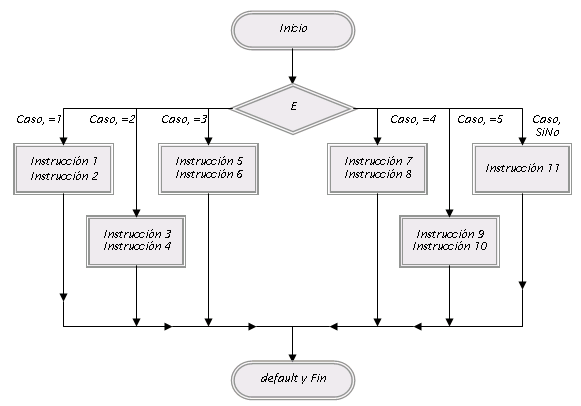
        .

        default:

        sentencias;

        break;

 }



Ejemplo 1

**package com.pranical.test;**

**import** java.util.Scanner;

**public** **class** TestSwitch1 {

@SuppressWarnings("resource")

**public** **static** **void** main(String[] args) {

Scanner entrada = **new** Scanner(System.*in*);

System.*out*.println("En cual rango esta tu edad: '");

System.*out*.println("-----------------------------");

System.*out*.println(" 0) 0 años");

System.*out*.println(" 1) De 1 a 3 años");

System.*out*.println(" 2) De 4 a 10 años");

System.*out*.println(" 3) De 11 a 14 años");

System.*out*.println(" 4) De 15 a 17 años");

System.*out*.println(" 5) Tienes 18 años");

System.*out*.println(" 6) De 19 a 30 años");

System.*out*.println(" 7) De 31 a 50 años");

System.*out*.println(" 8) De 51 a 65 años");

System.*out*.println(" 9) De 66 a 80 años");

System.*out*.println(" 10) De 81 a 120 años");

System.*out*.println(" 11) > 120");

System.*out*.print ("Elija un valor del rango: ");

**int** edad = entrada.nextInt();

**switch** (edad) {

**case** 0:

System.*out*.println ("Acaba de nacer hace poco. No ha cumplido el año");

**break**;

**case** 1:

System.*out*.println ("Eres un bebe");

**break**;

**case** 2:

System.*out*.println ("Eres niño(a)");

**break**;

**case** 3:

System.*out*.println ("Eres adolecente");

**break**;

**case** 4:

System.*out*.println ("Eres un pavo(a)");

**break**;

**case** 5:

System.*out*.println ("Está justo en la mayoría de edad");

**break**;

**case** 6:

System.*out*.println ("Eres un joven");

**break**;

**case** 7:

System.*out*.println ("Eres una persona adulta");

**break**;

**case** 8:

System.*out*.println ("Debes ser una persona madura");

**break**;

**case** 9:

System.*out*.println ("Debes ser un abuelo consentidor");

**break**;

**case** 10:

System.*out*.println ("Hace rato estas jubilado");

**break**;

**case** 11:

System.*out*.println ("Estas escondiendo de San Pedro...");

**break**;

**default**:

System.*out*.println ("No es un valor valido o Estas en el recuerdo");

**break**;

}

}

}

Ejemplo 2

**package com.pranical.test;**

**import** java.util.Scanner;

**public** **class** TestSwitch2 {

@SuppressWarnings("resource")

**public** **static** **void** main(String[] args) {

Scanner entrada = **new** Scanner(System.*in*);

**int** opcionAlimento,

opcionBebida;

**int** cantidadAlimento,

cantidadBebida;

**int** totalAlimentos=0,

totalBebidas=0,

totalPagar=0;

System.*out*.println("Loncheria 'La Sabrosita'");

System.*out*.println("------------------------");

System.*out*.println("--|Menu de alimentos|--");

System.*out*.println(" 1) Tortas.............. 400.00 Bs.");

System.*out*.println(" 2) Dulces Salados...... 250.00 Bs.");

System.*out*.println(" 3) Pan................. 250.00 Bs.");

System.*out*.println(" 4) Chocolates.......... 400.00 Bs.");

System.*out*.println("");

System.*out*.println("--|Menu de bebidas|--");

System.*out*.println(" 1) Refresco........ 200.00 Bs.");

System.*out*.println(" 2) Te o Limonada.... 200.00 Bs.");

System.*out*.println(" 3) Jugo natural..... 250.00 Bs.");

System.*out*.println(" 4) Cervezas......... 300.00 Bs.");

System.*out*.print ("Elija un alimento: ");

opcionAlimento = entrada.nextInt();

System.*out*.print (" ¿Cual es la cantidad? ");

cantidadAlimento = entrada.nextInt();

System.*out*.print ("Elija una bebida: ");

opcionBebida = entrada.nextInt();

System.*out*.print (" ¿Cual es la cantidad? ");

cantidadBebida = entrada.nextInt();

// analizar el tipo de alimento comprado

**switch** (opcionAlimento) {

**case** 1:

**case** 4:

totalAlimentos = cantidadAlimento \* 400;

**break**;

**case** 2:

**case** 3:

totalAlimentos = cantidadAlimento \* 250;

**break**;

**default**:

totalAlimentos = 0;

}

// analizar el tipo de bebida comprada

**switch** (opcionBebida) {

**case** 1:

**case** 2:

totalBebidas = cantidadBebida \* 200;

**break**;

**case** 3:

totalBebidas = cantidadBebida \* 250;

**break**;

**case** 4:

totalBebidas = cantidadAlimento \* 300;

**break**;

**default**:

totalBebidas = 0;

}

// cálculo del total a pagar

totalPagar = totalAlimentos + totalBebidas;

System.*out*.printf ("Total alimentos: %d Bs. \n", totalAlimentos);

System.*out*.printf ("Total bebidas : %d Bs. \n", totalBebidas);

System.*out*.printf ("Total a pagar : %d Bs. \n", totalPagar);

}

}

La clase **Scanner**: Facilita la lectura de datos en los programas Java (>Java 4)

* Para leer podemos usar el método nextXxx() donde Xxx indica en tipo, por ejemplo nextInt() para leer un entero, nextDouble() para leer un double, etc.

**System.in**: Acceso al teclado

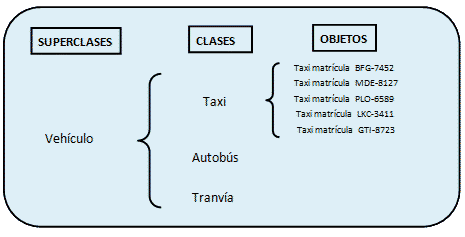
Herencia - Clases en Java

**Clase** abstracción que define un tipo de objeto especificando qué propiedades (atributos) y operaciones (métodos) disponibles va a tener.

**Objeto** entidad existente en la memoria del ordenador que tiene unas propiedades (atributos o datos sobre sí mismo almacenados por el objeto) y unas operaciones disponibles específicas (métodos).

**Herencia**

* Existen dos clases, a las que llamaremos padre (superclase o clase base) e hija (subclase o clase derivada)
* Al igual que las herencias en la vida real, la clase hija pasa a tener lo que tiene la clase padre
  + Atributos
  + Métodos
* Un objeto de la clase hija es también un objeto de la clase padre



* Se indica usando la palabra reservada **extends**
  + class Autobus extends Vehiculo
* Visibilidad
  + Los miembros privados de la superclase no son visibles desde la subclase.
  + Los miembros públicos de la superclase son visibles y siguen siendo públicos en la subclase.
* Se puede acceder a los miembros de la superclase usando la palabra reservada super.
* Si una clase se declara como final no se puede heredar de ella.
* En java, todas las clases heredan implícitamente de la clase Object.

**Ejercicio Herencia**

Clase Padre: Vehiculo.java

**package** com.pranical.herencia;

**public** **class** Vehiculo {

**public** String color;

**public** Integer motor;

**public** String placa;

**public** **int** puertas;

**public** Vehiculo() {}

**public** Vehiculo(String color, Integer motor, String placa, **int** puertas) {

**this**.color = color;

**this**.motor = motor;

**this**.placa = placa;

**this**.puertas = puertas;

}

**public** **void** imprimirColor(){

System.*out*.println("El color del vehiculo: "+ color);

}

**public** **void** imprimirMotor(){

System.*out*.println("El motor del vehiculo: "+ motor);

}

**public** **void** imprimirPlaca(){

System.*out*.println("La placa del vehiculo: "+ placa);

}

**public** **void** imprimirPuertas(){

System.*out*.println("Cantidad de puertas del vehiculo: "+ puertas);

}

**public** **void** queSoy(){

System.*out*.println("SOY UN VEHICULO");

}

}

Clase Hija: Taxi.java

**package** com.pranical.herencia;

**public** **class** Taxi **extends** Vehiculo {

**public** Double costo;

**public** Taxi(String color, Integer motor, String placa, **int** puertas, Double costo) {

**super**(color, motor, placa, puertas);

**this**.costo = costo;

}

@Override // Sobreescribir el metodo

**public** **void** imprimirColor(){

System.*out*.println("El color del Taxi: "+ color);

}

@Override

**public** **void** imprimirMotor(){

System.*out*.println("El motor del Taxi: "+ motor);

}

@Override

**public** **void** imprimirPlaca(){

System.*out*.println("La placa del Taxi: "+ placa);

}

@Override

**public** **void** imprimirPuertas(){

System.*out*.println("Cantidad de puertas del Taxi: "+ puertas);

}

**public** **void** getCosto(){

System.*out*.println("El costo del Taxi: "+ costo);

}

}

Clases Abstractas

* En ciertos casos, una clase se diseña directamente para ser extendida por un conjunto de subclases.
* En estos casos suele interesar no implementar alguno de sus métodos, pues no tiene ningún significado en la clase base.
* Es necesario declarar tanto la clase como los métodos no implementados como abstractos a través de la palabra reservada **abstract**.
* Una clase abstracta es, por tanto, aquella que tiene alguno de sus métodos definido pero no implementado.

**package** com.pranical.clase;

**public** **abstract** **class** Mascota {

**private** String nombre;

**public** **abstract** **void** getSonido();

**public** **void** setNombre(String nombre){

**this**.nombre = nombre;

}

}

Interfaces

* A veces nos interesa que todos los métodos de una clase abstracta sean abstractos.
  + Lo usamos para obligar a que todas las subclases re implementen esos métodos
* Para estos casos, Java proporciona unas clases especiales llamadas **interfaces**
* Se declaran como **interface**, no como **class**.
* Representan el concepto de **clase abstracta pura**
* Una clase declarada como interfaz no puede tener ningún método implementado

**Importante**

* **Para heredar de una interfaz se usa la palabra reservada implements**
  + - **public class MiClase implements Serializable**

**Reglas**

* Una interfaz puede heredar de otra interfaz
* Una clase (abstracta o no) puede heredar de una interfaz
* Una interfaz **NO** puede heredar de una clase

ArrayList

La clase ArrayList en Java, es una clase que permite almacenar datos en memoria de forma similar a los Arrays, con la ventaja de que el número de elementos que almacena, lo hace de forma dinámica, es decir, que no es necesario declarar su tamaño como pasa con los Arrays.

Los ArrayList nos permiten añadir, eliminar y modificar elementos (que pueden ser objetos o elementos atómicos) de forma trasparente para el programador.

Ejemplo 1: Llenar lista de valores enteros y ordenar ascendentemente y descendentemente

**package com.pranical.test;**

**import** java.util.ArrayList;

**import** java.util.Collections;

**public** **class** Lista1 {

**private** **static** ArrayList<Integer> *arrayListInt* = **new** ArrayList<Integer>();

**public** **static** **void** main(String[] args) {

Lista1 lista = **new** Lista1();

System.*out*.println("/++++++++++++++ Lista original +++++++++++++++/");

lista.llenarlista();

lista.imprimirLista();

System.*out*.println("\n/++++++++++++++ Lista ordenada +++++++++++++++/");

lista.ordernarLista();

}

**public** **void** llenarlista(){

*arrayListInt*.add(3);

*arrayListInt*.add(4);

*arrayListInt*.add(2);

*arrayListInt*.add(6);

*arrayListInt*.add(5);

*arrayListInt*.add(1);

*arrayListInt*.add(7);

}

**public** **void** ordernarLista(){

Collections.*sort*(*arrayListInt*);

imprimirLista();

}

**public** **void** imprimirLista(){

**for**(**int** i=0; i<*arrayListInt*.size(); i++){

System.*out*.println("Posicion["+i+"]: "+ *arrayListInt*.get(i));

}

}

}

Ejemplo 2: Llenar lista de objetos y ordenar ascendentemente y descendentemente

**package com.pranical.test;**

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.Iterator;

**public** **class** Lista2 {

**private** **static** ArrayList<Persona> *personas* = **new** ArrayList<Persona>();

**public** **static** **void** main(String[] args) {

Lista2 lista = **new** Lista2();

System.*out*.println("/++++++++++++++ Lista original +++++++++++++++/");

lista.llenarlista();

lista.imprimirLista();

System.*out*.println("\n/++++++++++++++ Lista Orden Ascendente+++++++++++++++/");

lista.ordernarLista();

System.*out*.println("\n/++++++++++++++ Lista Orden Descendente +++++++++++++++/");

lista.ordernarListaReversa();

}

**public** **void** llenarlista(){

*personas*.add(**new** Persona("Pepe", 28));

*personas*.add(**new** Persona("Juan", 32));

*personas*.add(**new** Persona("Paco", 40));

*personas*.add(**new** Persona("Susi", 24));

*personas*.add(**new** Persona("Lola", 20));

*personas*.add(**new** Persona("Jose", 28));

*personas*.add(**new** Persona("Dani", 24));

*personas*.add(**new** Persona("Sara", 36));

}

// Este ArrayList lo queremos ordenar (de menor a mayor edad) en función de la edad de las personas,

// por tanto tenemos que sobre escribir el método “compare” de la clase Comparator y llamar al método “sort”

// de la clase collections

// Odeno el arrayList de mayor a menor por edad y lo imprimo

**public** **void** ordernarLista(){

Collections.*sort*(*personas*, **new** Comparator<Persona>() {

@Override

**public** **int** compare(Persona p1, Persona p2) {

**return** **new** Integer(p1.getEdad()).compareTo(**new** Integer(p2.getEdad()));

}

});

imprimirLista();

}

// Odeno el arrayList de menor a mayor por edad y lo imprimo

**public** **void** ordernarListaReversa(){

Collections.*sort*(*personas*, **new** Comparator<Persona>() {

@Override

**public** **int** compare(Persona p1, Persona p2) {

// Aqui esta el truco, ahora comparamos p2 con p1 y no al reves como antes

**return** **new** Integer(p2.getEdad()).compareTo(**new** Integer(p1.getEdad()));

}

});

imprimirLista();

}

**public** **void** imprimirLista(){

Iterator<Persona> itrArrayList = *personas*.iterator();

**int** posicion = 1;

**while** (itrArrayList.hasNext()) {

System.*out*.println("Posicion(" + posicion + ") = " + itrArrayList.next().toString());

posicion++;

}

}

}

**class** Persona {

**private** String nombre;

**private** **int** edad;

**public** Persona() {

}

**public** Persona(String nombre, **int** edad) {

**this**.nombre = nombre;

**this**.edad = edad;

}

**public** String getNombre() {

**return** nombre;

}

**public** **void** setNombre(String nombre) {

**this**.nombre = nombre;

}

**public** **int** getEdad() {

**return** edad;

}

**public** **void** setEdad(**int** edad) {

**this**.edad = edad;

}

@Override

**public** String toString() {

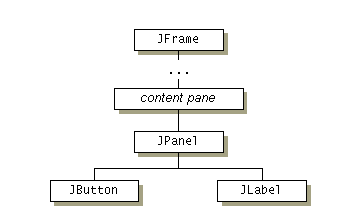
**return** **this**.getNombre() + " - " + **this**.getEdad();

}

}

Java Swing Ejemplo

**Swing:** Librería de componente básicos que permiten implementar interfaces gráficas. (javax.swing). Paquete que hace parte de la Java Foundation Classes (JFC)



**JFrame:** clase encapsula una Ventana clásica de cualquier sistema operativo con entorno gráfico (Windows, OS X, Linux etc.) – Contenedor

**package com.pranical.test;**

**import** java.awt.\*;

**import** javax.swing.\*;

@SuppressWarnings("serial")

**public** **class** TestFrame **extends** JFrame

{

**private** Container contenedor;

**public** TestFrame ()//constructor

{

contenedor=getContentPane();

contenedor.setLayout(**null**);

//Asigna un titulo a la barra de titulo

setTitle("Titulo de la Ventana");

//tamaño de la ventana

setSize(400,200);

//pone la ventana en el Centro de la pantalla

setLocationRelativeTo(**null**);

setVisible(**true**);

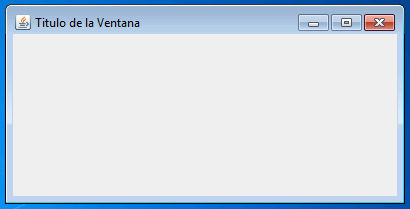
}

**public** **static** **void** main(String[] args){

**new** TestFrame ();

}

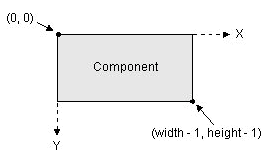
}

****

**JPanel: Este contenedor permite la creación de paneles independientes donde se almacenan otros componentes, de esta manera decidimos que elementos se alojan en que paneles y podemos usar sus propiedades para ocultar, mover o delimitar secciones...**

****

**Sistema de Coordenadas:** Área de Dibujo sobre la vista:



**ActionListener:**

Se usa para detectar y manejar eventos de acción. Dichos eventos tienen lugar cuando se produce una acción sobre un elemento del programa.

* al pulsar un botón.
* al hacer doble clic en un elemento de lista.
* al pulsar INTRO en una caja de texto.
* al elegir un menú.

Es una **interfaz** y por lo que se deben implementar todos sus métodos:

* actionPerformed(ActionEvent e).

**package com.pranical.test;**

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JOptionPane;

**public** **class** TestActionListener **implements** ActionListener{

**public** **static** **void** main(String[] args){

**new** TestActionListener ();

}

**public** TestActionListener(){

JFrame frame = **new** JFrame("Uso ActionListener Java-Swing");

JButton b=**new** JButton("Aceptar");

b.addActionListener(**this**);

frame.add(b);

frame.setSize(200,200);

frame.setVisible(**true**);

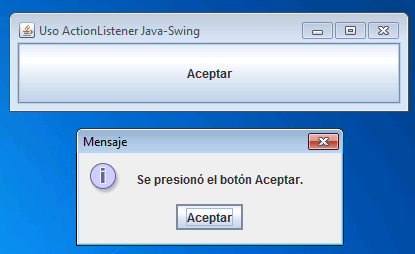
}

**public** **void** actionPerformed(ActionEvent e) {

JOptionPane.*showMessageDialog*(**null**, "Se presionó el botón Aceptar.");

}

}



**FlowLayout:**

Es aquel layout que ubica a todos los componentes en forma horizontal, en el orden que le digamos.

Primero tenemos que crear el contenedor (JFrame, JPanel, etc), y luego a través del método “setLayout()” asignarle el layout correspondiente.

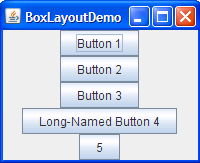
JFrame frame = new JFrame();

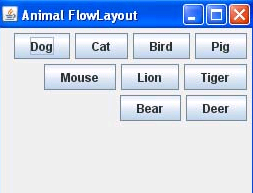
frame.setLayout(new FlowLayout());

Una vez, asignado el layout al contenedor, tendremos que agregar los componentes, plenamente ya declarados y creados.

JButton boton =  new JButton("Aceptar");

frame.add(boton);

****

****

**Borderlayout:**

BorderLayout, es un layout que ubica los componentes en cualquiera de sus 5 regiones que tiene..

Un componente podemos ubicarlo arriba, abajo, izquierda o a la derecha.

Para establecer a BorderLayout como manegador de Diseño:

JFrame frame = new JFrame();

frame.setLayout(new BorderLayout());

Luego puedo agregar los componentes

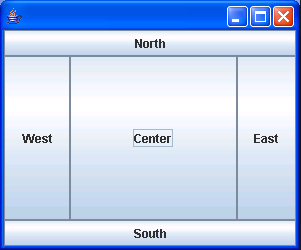
frame.add(etiqeuta, BorderLayout.CENTER);

frame.add(botonIzquierdo,BorderLayout.WEST);

frame.add(botonDerecho,BorderLayout.EAST);

frame.add(cajaTexto,BorderLayout.NORTH);

frame.add(panelInferior,BorderLayout.SOUTH);

****

EJERCICIO AVANZADO 1: DIBUJAR SOBRE UNA VISTA:

Se desea generar un JFrame al cual se debe agregar un JPanel y en este, dibujar un Rectángulo y un Círculo.

package com.pranical.grafico;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.Ellipse2D;

import java.awt.geom.Rectangle2D;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class Dibujo {

private static JFrame frame; // Ventana

public static void main(String[] args) {

frame = new JFrame();

frame.setTitle("Test");

frame.add(new JPanelTest());

frame.pack();

frame.setVisible(true);

}

}

// Clase interna

@SuppressWarnings("serial")

class JPanelTest extends JPanel {

// Tamaño del panel

@Override

public Dimension getPreferredSize() {

return new Dimension(300, 300);

}

@Override

public void paintComponent(Graphics g) {

// Contexto grafico

Graphics2D g2 = (Graphics2D) g;

// Agregar Circulo

Ellipse2D circle = new Ellipse2D.Double(getWidth() - 120, getHeight() - 120, 120, 120);

g2.draw(circle);

// Agregar linea

Rectangle2D rect = new Rectangle2D.Double(0, 0, 100, 100);

g2.draw(rect);

}

}

EJERCICIO AVANZADO 2: DIBUJAR ELEMENTOS GRAFICOS SOBRE UNA VISTA :

Se desea generar elementos gráficos que permitan dibujar sobre un JPanel un Rectángulo, un Círculo, una Línea y un Texto; utilizando los conceptos de Interfax, Clase Abstracta y herencia. Además agregar un botón para imprimir información de los elementos gráficos generados.

Crear Interfaz: ElementoGrafico.java: Define métodos requeridos para dibujar y obtener los datos de cada elemento gráfico.

**package** com.pranical.grafico.negocio;

**import** java.awt.Graphics2D;

**public** **interface** ElementoGrafico {

**public** **void** dibujar(Graphics2D g2D);

**public** String getInformacionElemento();

}

Crear Clase Abstracta: DefaultElementoGrafico.java: Implementa la interfaz “ElementoGrafico” y define el color de relleno de los componentes gráficos.

package com.pranical.grafico.negocio;

import java.awt.Color;

import java.awt.Graphics2D;

public abstract class DefaultElementoGrafico implements ElementoGrafico {

private Color color;

public abstract void dibujar(Graphics2D g2d) ;

public abstract String getInformacionElemento() ;

public Color getColor(){

return color;

}

public void setColor(Color color){

this.color = color;

}

}

Crear Clase Rectangulo.java: Elemento grafico que permite dibujar rectángulos

package com.pranical.grafico.elemento;

import java.awt.Graphics2D;

import java.awt.geom.Rectangle2D;

import com.pranical.grafico.negocio.DefaultElementoGrafico;

public class Rectangulo extends DefaultElementoGrafico{

private int ancho;

private int alto;

public Rectangulo(int ancho, int alto) {

this.ancho = ancho;

this.alto = alto;

}

public void dibujar(Graphics2D g2D) {

Rectangle2D rect = new Rectangle2D.Double(0, 0, ancho, alto);

g2D.setColor(getColor());

g2D.fill(rect);

}

@Override

public String getInformacionElemento() {

return "Rectangulo de ancho: "+ancho+" y alto: "+ alto;

}

}

Crear Clase Circulo.java: Elemento grafico que permite dibujar círculos

package com.pranical.grafico.elemento;

import java.awt.Graphics2D;

import java.awt.geom.Ellipse2D;

import com.pranical.grafico.negocio.DefaultElementoGrafico;

public class Circulo extends DefaultElementoGrafico{

private int radio;

public Circulo(int radio) {

this.radio = radio;

}

public void dibujar(Graphics2D g2D) {

int diametro = 2 \* radio;

Ellipse2D circle = new Ellipse2D.Double(100, 100, diametro, diametro);

g2D.setColor(getColor());

g2D.fill(circle);

}

@Override

public String getInformacionElemento() {

return "Circulo de radio:"+ radio;

}

}

Crear Clase Linea.java: Elemento grafico que permite dibujar líneas

package com.pranical.grafico.elemento;

import java.awt.Graphics2D;

import java.awt.geom.Line2D;

import com.pranical.grafico.negocio.DefaultElementoGrafico;

public class Linea extends DefaultElementoGrafico {

private int xStart;

private int yStart;

private int xEnd;

private int yEnd;

public Linea(int xStart, int yStart, int xEnd, int yEnd) {

this.xStart = xStart;

this.yStart = yStart;

this.xEnd = xEnd;

this.yEnd = yEnd;

}

public void dibujar(Graphics2D g2D) {

Line2D linea = new Line2D.Float(xStart, yStart, xEnd, yEnd);

g2D.setColor(getColor());

g2D.draw(linea);

}

@Override

public String getInformacionElemento() {

return "Línea de ("+xStart+","+ yStart+") a ("+xEnd+","+ yEnd+")";

}

}

Crear Clase Texto.java: Elemento grafico que permite dibujar textos

package com.pranical.grafico.elemento;

import java.awt.Graphics2D;

import com.pranical.grafico.negocio.DefaultElementoGrafico;

public class Texto extends DefaultElementoGrafico{

private String txt;

private int x;

private int y;

public Texto(String txt, int x, int y) {

this.txt = txt;

this.x = x;

this.y = y;

}

public void dibujar(Graphics2D g2D) {

g2D.setColor(getColor());

g2D.drawString(txt, x, y);

}

@Override

public String getInformacionElemento() {

return txt + " en la posición: ("+x+","+y+")";

}

}

Crear Clase JPanel.java: Contenedor donde se dibujan los elementos gráficos

package com.pranical.grafico.componente;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.util.ArrayList;

import javax.swing.JPanel;

import com.pranical.grafico.negocio.ElementoGrafico;

@SuppressWarnings("serial")

public class Panel extends JPanel {

private ArrayList<ElementoGrafico> elementos;

// Constructor

public Panel(ArrayList<ElementoGrafico> elementos) {

this.elementos = elementos;

this.setPreferredSize(new Dimension(400, 350)); // Variante de sobreescribir getPreferredSize() del JPanel

}

/\*

\* paintComponent: Pinta los elementos en el panel

\* @see javax.swing.JComponent#paintComponent(java.awt.Graphics)

\*/

public void paintComponent(Graphics g) {

Graphics2D g2D = (Graphics2D) g;

for (ElementoGrafico elemento: elementos){

Color colorActual = g2D.getColor();

elemento.dibujar(g2D);

g2D.setColor(colorActual);

}

}

/\*\*

\* getText: Consulta la informacion de cada elemento grafico y retorna sus resultados

\* @return Datos de elelmentos graficos

\*/

public String getText() {

StringBuffer sb = new StringBuffer();

for (ElementoGrafico elemento: elementos){

sb.append(elemento.getInformacionElemento()+ System.getProperty("line.separator"));

}

return sb.toString();

}

}

Crear Clase Vista.java: Contenedor – Interfaz Gráfica (Ventana)

package com.pranical.grafico;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JTextArea;

import com.pranical.grafico.componente.Panel;

import com.pranical.grafico.elemento.Circulo;

import com.pranical.grafico.elemento.Linea;

import com.pranical.grafico.elemento.Rectangulo;

import com.pranical.grafico.elemento.Texto;

import com.pranical.grafico.negocio.ElementoGrafico;

public class Vista implements ActionListener{

private JFrame frame;

private Panel panel;

private JButton boton;

private JTextArea textArea;

public static void main(String[] args){

new Vista().init();

}

public void init(){

frame = new JFrame();

frame.setSize(400,400);

frame.setLayout(new BorderLayout());

frame.setTitle("Elementos Gráficos");

// Crear Rectangulo

Rectangulo rectangulo = new Rectangulo(100, 100);

rectangulo.setColor(Color.red);

// Crear Circulo

Circulo circulo = new Circulo(50);

circulo.setColor(Color.yellow);

// Crear Linea

Linea linea = new Linea(200, 200, 400, 350);

linea.setColor(Color.green);

// Crear Texto

Texto texto = new Texto("TEXTO-ELEMENTO GRAFICO", 120, 20);

texto.setColor(Color.magenta);

ArrayList<ElementoGrafico> elementos = new ArrayList<ElementoGrafico>();

elementos.add(rectangulo);

elementos.add(circulo);

elementos.add(linea);

elementos.add(texto);

panel = new Panel(elementos);

textArea = new JTextArea(4,4);

boton = new JButton("Información");

boton.addActionListener(this);

frame.add(panel, BorderLayout.NORTH);

frame.add(boton, BorderLayout.WEST);

frame.add(textArea, BorderLayout.CENTER);

// Realizar ajuste de la vista al size definido

frame.pack();

// Mostrar la vista

frame.setLocationRelativeTo(null); // Centrar la vista

frame.setVisible(true);

frame.repaint();

}

@Override

public void actionPerformed(ActionEvent arg0) {

if (arg0.getSource()==boton) {

textArea.setText(panel.getText());

//JOptionPane.showMessageDialog(null, panel.getText());

}

}

}