@PROFESANTIAGO

JAVA avanzado

P.O.O

La programación orientada a objetos organiza el código de manera más **modular** y fácil de entender al modelar el mundo real a través de objetos interactivos que se comunican entre sí. Este enfoque promueve la **reutilización del código** y facilita el **mantenimiento del programa**.

Un **objeto** es una entidad que agrupa datos (llamados ***atributos***) y determinados ***métodos*** que operan sobre esos datos. Los principales principios de la POO incluyen la encapsulación (ocultar la implementación interna de un objeto), la herencia (permitir que los objetos compartan características comunes) y el polimorfismo (la capacidad de objetos distintos de responder a los mismos mensajes de diferentes maneras).

Todo el código debe pertenecer a una **clase** (plantilla) y luego se pueden instanciar mediante objetos.

Un **método** se caracteriza por un nombre, el tipo de dato que devuelve (si es que devuelve algún valor) y entre paréntesis los parámetros de entrada (si es que los recibe, se pasan al momento de invocar el método).

Un método que devuelve un valor como resultado es considerado una **función**, y en caso de no retornar un valor, se denomina **procedimiento.**

Ejercicio 0

```java
package ejemplos;

public class Empleado {  //Creo clase empleado, sin método main

    //Atributos de empleado

    private String ci;
    private String nombre;
    private float sueldo;

    //Métodos

    public void aumentarSueldo (double incremento) {

        sueldo+=incremento;

    }

    public String imprimirDatos() {

        return "|CI: "+this.ci +" |Nombre: "+this.nombre+" |Sueldo: "
        +this.sueldo;

    }

    // Contructores

    public Empleado () { //Sin parámetros

        this.ci="1.111.111-0";
        this.nombre="Anónimo";
        this.sueldo=100000f;

    }

    public Empleado (String ci,String nombre,float sueldo) { //Con parámetros

        this.ci=ci;
        this.nombre=nombre;
        this.sueldo=sueldo;

    }

}
```

Modificadores de acceso (indica quien tiene derecho a acceder a esos elementos).

Los atributos tienen modificador de acceso **privado** y los métodos **públicos**.

Clase empleado con sus atributos y métodos.

Con ***this*** me refiero a un atributo o método de la misma clase (diferenciándolo así del parámetro).

Constructor: Método especial que se encarga de dar el estado inicial al objeto.

Al utilizar ***new*** comienza la creación del objeto (instancia de una clase). En caso de no tener un constructor definido, JAVA inicializará el objeto con todos sus atributos en 0. (null si es un string y false en caso de boolean).

Si quiero **especificar otros valores por defecto**, debo definir un contructor (con o sin parámetros).

Creo **otra clase** que incluya un método main para ***instanciar desde allí dos objetos Empleado (utilizando además los métodos definidos para esa clase).***

Empleado.java PruebaEmpleado.java

```
1  package ejemplos;
2
3  public class PruebaEmpleado {
4
5      public static void main(String[] args) {
6
7          Empleado emp1 = new Empleado();
8          Empleado emp2 = new Empleado ("2.222.333","Santiago",455.3f);
9
10         System.out.println(emp1.imprimirDatos());
11         System.out.println(emp2.imprimirDatos());
12         emp2.aumentarSueldo(1000); //Aumento el sueldo al empleado 2
13         System.out.println(emp2.imprimirDatos());
14
15     }
16
17 }
```

Encapsulamiento

El encapsulamiento refiere a la restricción de acceso a algunos componentes del objeto.

***evita que el usuario pueda cambiar su estado de maneras imprevistas e incontroladas.**

Un atributo/método definido como public, puede ser invocado desde otra clase.

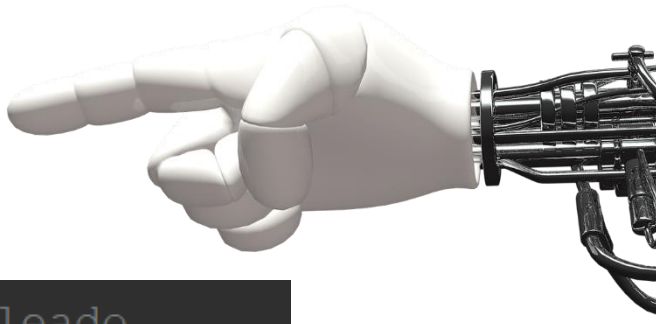
Si se define como private sólo puede utilizarse desde la clase donde se definió.

```
//Atributos de empleado

private String ci;
public String nombre;
```

```
9
10          System.out.println(emp2.nombre);
11          System.out.println(emp2.ci);
12
```

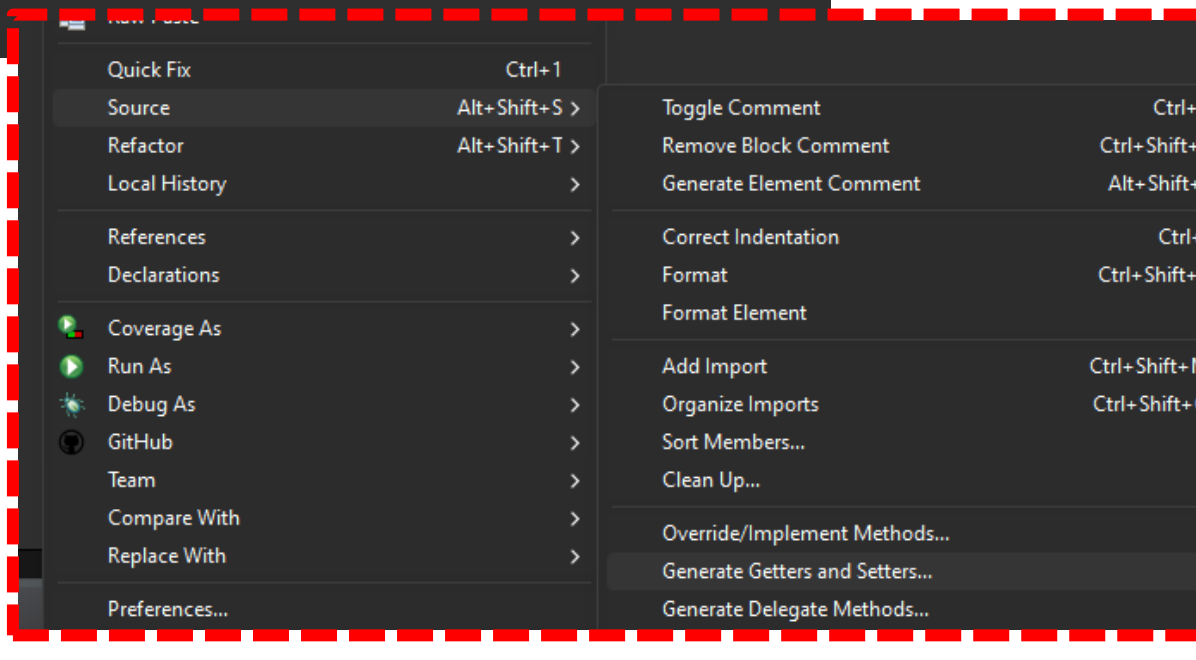
The field Empleado.ci is not visible

Lo deseable es **no definir los atributos como públicos** y obtenerlos/modificarlos mediante métodos ***getters y setters.***



```java
//Getters y Setters en clase empleado
public String obtenerCi() {
    return ci;
}
public void setearCi(String ci) {
    this.ci=ci;
}
public String obtenerNombre() {
    return nombre;
}
public void setearNombre(String nombre) {
    this.nombre=nombre;
}
```

Estos métodos pueden crearse de forma manual, aunque Eclipse es capaz de ***agregarlos automáticamente*** en el código.



@PROFESANTIAGO

Ejecución del programa

Invocación de getters y setters

Empleado.java | PruebaEmpleado.java

```
1  package ejemplos;
2  import java.util.*;
3
4  public class PruebaEmpleado {
5
6      public static void main(String[] args) {
7
8
9          Empleado emp1 = new Empleado();
10         Empleado emp2 = new Empleado ("2.222.333","Santiago",955.3f);
11
12
13         System.out.println(emp1.obtenerCi()); //Imprimo CI de emp1
14
15         System.out.println(emp1.imprimirDatos());
16         System.out.println(emp2.imprimirDatos());
17         emp2.aumentarSueldo(1000); //Aumento el sueldo al empleado 2
18         System.out.println(emp2.imprimirDatos());
19
20         System.out.println("Ingrese una CI para "+emp2.obtenerNombre()+": ");
21         Scanner sc =new Scanner (System.in);
22         emp2.setearCi(sc.nextLine());
23         System.out.println("\n"+emp2.imprimirDatos());
24
25         sc.close();
26     }
27
28 }
29
```

Console | Javadoc | Declaration | Problems

<terminated> PruebaEmpleado [Java Application] C:\Users\santi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (13 feb. 2024 11:21:59 – 11:24:31) [pid:

```
1.111.111-0
|CI: 1.111.111-0 |Nombre: Anónimo |Sueldo: 100000.0
|CI: 2.222.333 |Nombre: Santiago |Sueldo: 955.3
|CI: 2.222.333 |Nombre: Santiago |Sueldo: 1955.3
Ingrese una CI para Santiago:
4.345.222-9

|CI: 4.345.222-9 |Nombre: Santiago |Sueldo: 1955.3
```

Los atributos o métodos ***static*** no dependen de cada objeto (dependen de la clase en general).

Por ejemplo:

- El atributo nombre, es diferente para cada empleado (depende de cada instancia/objeto).
- Un atributo que cuente la cantidad de empleados, no está asociado a un objeto en particular y es considerado atributo de clase o estático.

1

```
//Atributos de empleado

private String ci;
private String nombre;
private float sueldo;
private static int contador_emp; //no depende de cada objeto
```

```
// Contructores


public Empleado () { //Sin parámetros

    this.ci="1.111.111-0";
    this.nombre="Anónimo";
    this.sueldo=100000f;
    contador_emp++; // aumento el contador cada vez que creo un empleado

}

public Empleado (String ci,String nombre,float sueldo) { //Con parámetros

    this.ci=ci;
    this.nombre=nombre;
    this.sueldo=sueldo;
    contador_emp++;

}
```

2

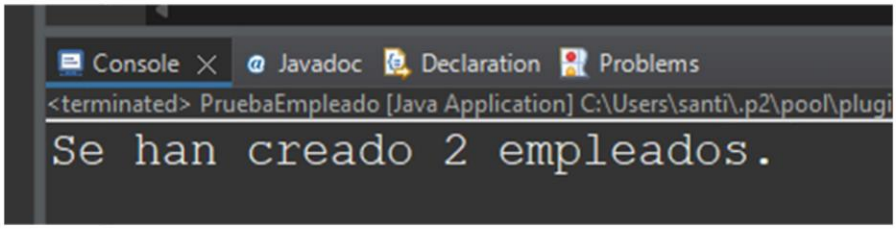
```
public static int cantidadEmpleados() { //getter del atributo estático

    return contador_emp;

}
```

3

@PROFESANTIAGO

```java
package ejemplos;
import java.util.*;

public class PruebaEmpleado {

    public static void main(String[] args) {


        Empleado emp1 = new Empleado();
        Empleado emp2 = new Empleado ("2.222.333","Santiago",955.3f);
        System.out.println("Se han creado "+Empleado.cantidadEmpleados()+" empleados.");
        // el método static cantidadEmpleados se invoca a partir del nombre de la clase
```

El método **static** ***se invoca a partir del nombre de la clase*** y no de un objeto en particular.



Ejercicio 1

Crear una clase cuenta, para modelar **cuentas bancarias**, que tengan el **titular** y la **cantidad de dinero** como atributos.

Crear constructor sin parámetros y con titular como parámetro.

Metodos: **ingresarCantidad, retirarCantidad.**

//No se permite ingresar cantidades negativas y la cantidad total nunca será menor a 0.

En el programa principal, instanciar 2 cuentas y probar funcionalidad.

Ejercicio 2

Crear una clase libro con los siguientes atributos: isbn, título, autor y cantidad de páginas.

Crear getters/setters y el método toString para mostrar la información por pantalla.

En el programa principal crear un array de 4 libros.

Mostrar cada libro por pantalla y cual de ellos tiene más páginas.

@PROFESANTIAGO

Ejercicio 1

Cuenta.java

```java
package bancario;

public class Cuenta {

    // Atributos ////////////////////////////////////////

    private String titular;
    private float cantidad;


    // Constructores ////////////////////////////////////

    public Cuenta(){ //Constructor sin parámetros

    }

    public Cuenta (String titular) { // Constructor con titular como parámetro
        this.titular=titular;

    }

    // Getters y Setters ////////////////////////////////

    public String getTitular() {
        return titular;
    }
    public void setTitular(String titular) {
        this.titular = titular;
    }
    public float getCantidad() {
        return cantidad;
    }
    public void setCantidad(float cantidad) {
        this.cantidad = cantidad;
    }


    // Métodos ////////////////////////////////////////////
```

```java
    // Métodos ////////////////////////////////////////////

    public void ingresarCantidad (float cantidad) {

        if (cantidad>0) {
            this.cantidad+=cantidad;
        }
    }


    public void retirarCantidad (float cantidad) {

        if (cantidad >0) { // No se puede retirar valores negativos

            this.cantidad-=cantidad;

            if (this.cantidad<0) { // El saldo nunca es menor a 0
            this.cantidad=0;
            }

        }

    }


}
```

@PROFESANTIAGO

PruebaCuenta.java

Cuenta.java | PruebaCuenta.java

```java
package bancario;

public class PruebaCuenta {

    public static void main(String[] args) {

        Cuenta cuenta1 = new Cuenta ();
        Cuenta cuenta2 = new Cuenta ("Santiago");

        cuenta1.setTitular("Estefani");
        cuenta1.setCantidad(1500f);
        System.out.println(cuenta1.getTitular());
        System.out.println("$"+cuenta1.getCantidad());
        System.out.println(cuenta2.getTitular());
        System.out.println("$"+cuenta2.getCantidad()+" (cantidad inicial)");
        cuenta2.ingresarCantidad(-300f);
        System.out.println("$"+cuenta2.getCantidad());
        cuenta2.ingresarCantidad(400f);
        System.out.println("$"+cuenta2.getCantidad());
        cuenta2.retirarCantidad(500f);
        System.out.println("$"+cuenta2.getCantidad());


    }

}
```

Console | Javadoc | Declaration | Problems

<terminated> PruebaCuenta [Java Application] C:\Users\santi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (13 feb. 2024 14:40:35 – 14:40:35) [pid:

```
Estefani
$1500.0
Santiago
$0.0 (cantidad inicial)
$0.0
$400.0
$0.0
```

@PROFESANTIAGO

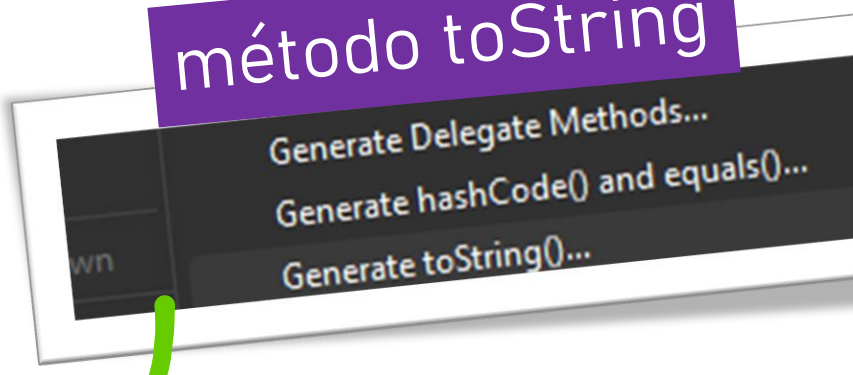
Ejercicio 2

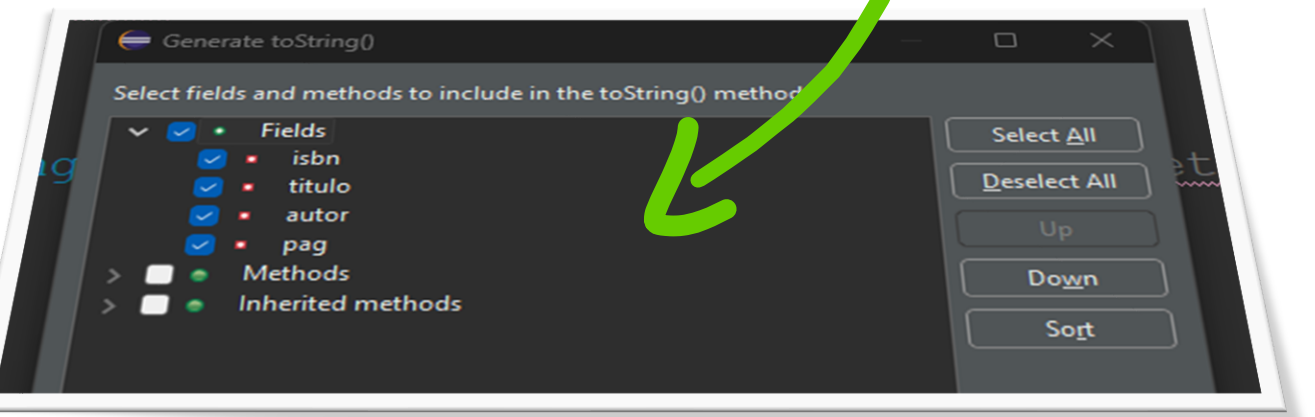
Clase Libro

*Libro.java *PruebaLibros.java

```java
package libreria;

public class Libro {

    // Atributos
    private String isbn, titulo, autor;
    private int pag;


    // Constructores

    public Libro() { //Sin parámetros

    }

    public Libro(String isbn,String titulo,String autor,int pag) { //Con parámetros
        this.isbn=isbn;
        this.titulo=titulo;
        this.autor=autor;
        this.pag=pag;

    }

    // Métodos


    @Override
    public String toString() {
        return "El libro con ISBN " + isbn + " escrito por " + autor + " tiene " + pag + " páginas.";
    }

```

método toString





@PROFESANTIAGO

getters y setters de Libro

```java
// Getters y Setters
public String getIsbn() {
    return isbn;
}
public void setIsbn(String isbn) {
    this.isbn = isbn;
}
public String getTitulo() {
    return titulo;
}
public void setTitulo(String titulo) {
    this.titulo = titulo;
}
public String getAutor() {
    return autor;
}
public void setAutor(String autor) {
    this.autor = autor;
}
public int getPag() {
    return pag;
}
public void setPag(int pag) {
    this.pag = pag;
}
```

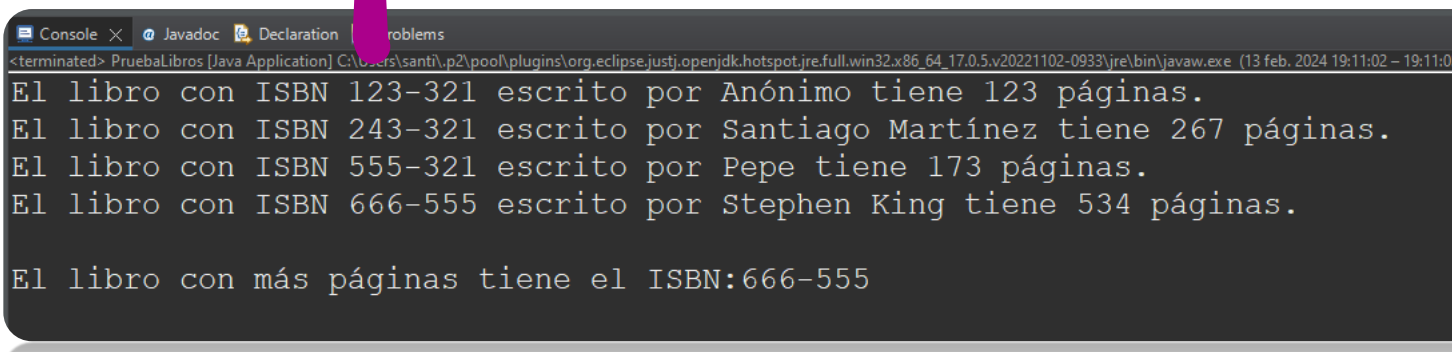
Clase PruebaLibros

*PruebaLibros.java

```java
package libreria;

public class PruebaLibros {

    public static void main(String[] args) {

        Libro l1 = new Libro ("123-321","libro1","Anónimo",123); //Creo 4 libros manualmente.
        Libro l2 = new Libro ("243-321","Soporte de Equipos Informáticos","Santiago Martínez",267); //Creo 4 libros manualmente.
        Libro l3 = new Libro ("555-321","Redes desde cero","Pepe",173);
        Libro l4 = new Libro ("666-555","El misterio de Salem's Lot","Stephen King",534);
        Libro mayor = new Libro();


        Libro [] libros = new Libro [4]; // Un arreglo de 4 objetos de tipo Libro.

        // Cargo los libros en el arreglo manualmente.

        libros[0] = l1;
        libros[1] = l2;
        libros[2] = l3;
        libros[3] = l4;


        for (Libro l:libros) { //alternativa para recorrer una coleccion de objetos (forEach)

            System.out.println(l.toString());
            if (l.getPag()>mayor.getPag()) {

                mayor=l;

            }

        }
        System.out.println("\nEl libro con más páginas tiene el ISBN:"
                +mayor.getIsbn());
    }

}
```

@PROFESANTIAGO

Ejecución del programa.



Composición

//Permite reutilizar código

Capacidad de una clase de tener atributos que son objetos de otra clase.

PruebaEmpleado.java

```
package ejemplos;
//Composición -> Los empleados tendrán como atributo, un objeto
//de tipo dirección.

public class Direccion {

    //Atributos de dirección
    private String calle;
    private int num;

    //Constructores

    public Direccion() { //Sin parámetros

    }

    public Direccion (String calle,int num) { //Con parámetros

        this.calle=calle;
        this.num=num;
    }
```

Creo una clase Direccion con sus atributos, constructores y métodos

Ejercicio 0

(continuación)

```java
    //toString

    public String toString() {
        return "Direccion [calle=" + calle + ", num=" + num + "]";
    }

    // Getters y Setters
    public String getCalle() {
        return calle;
    }

    public void setCalle(String calle) {
        this.calle = calle;
    }
    public int getNum() {
        return num;
    }
    public void setNum(int num) {
        this.num = num;
    }
}
```

```java
public class Empleado { //Creo clase empleado, sin método main


    //Atributos de empleado

    private String ci;
    private String nombre;
    private float sueldo;
    //Cada empleado tiene un objeto Direccion como atributo
    private Direccion dir;
```

Agrego como atributo de Empleado un objeto de tipo Direccion

```java
public Empleado () { //Sin parámetros

    ci="1.111.111-0";
    nombre="Anónimo";
    sueldo=100000f;
    //Creo un objeto direccion con los valores por defecto
    dir = new Direccion();

}

public Empleado (String ci,String nombre,float sueldo,Direccion dir) { //Con parámetros

    this.ci=ci;
    this.nombre=nombre;
    this.sueldo=sueldo;
    this.dir=dir;

}
```

Modifico los constructores de Empleado y agrego getter y setter para el nuevo atributo.

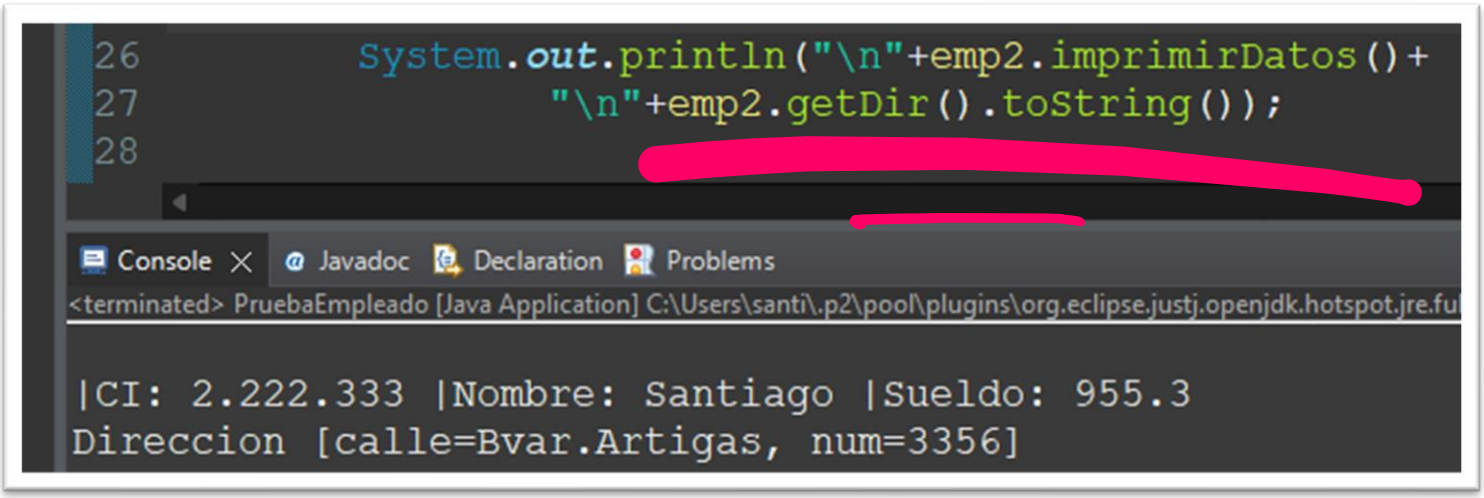
```java
    return nombre;
}

public void SetearNombre(String nombre) {

    this.nombre=nombre;
}

//Agrego getters y setters para dir
public Direccion getDir() {
    return dir;
}

public void setDir(Direccion dir) {
    this.dir = dir;
}
```

En el programa principal creo una instancia de Direccion y la paso como parámetro a la hora de crear un empleado

```java
package ejemplos;
import java.util.*;

public class PruebaEmpleado {

    public static void main(String[] args) {

        Direccion d = new Direccion("Bvar.Artigas",3356);

        Empleado emp1 = new Empleado();
        Empleado emp2 = new Empleado ("2.222.333","Santiago",955.3f,d);
```

Con emp2.getDir() obtengo el objeto de tipo Direccion asociado a emp2

Luego invoco al metodo toString() definido en la clase Direccion, el cuál retorna un String con los valores de sus atributos, y puedo verlos por pantalla con System.out.println()



Ejercicio 3

Crear una clase Fecha con atributos día, mes y año. Crear un método toString para que devuelva los atributos en formato dd/mm/yyyy.

Crear clase Persona con nombre, ci y fecha de nacimiento (de tipo Fecha).

En el programa principal inicializar un arreglo de 3 personas y mostrar sus fechas de nacimiento por pantalla.

@PROFESANTIAGO

Ejercicio 3

Clase Fecha

```java
package composicion;

public class Fecha {

	//Atributos
	private int dia;
	private int mes;
	private int anio;


	//Constructor con parametros
	public Fecha(int dia,int mes,int anio) {
		this.dia=dia;
		this.mes=mes;
		this.anio=anio;
	}

	//toString con formato dd/mm/yyyy

	public String toString() {

		String mostrar="";

		if(dia<10) {
			mostrar="0"+dia;
		}
		else {
			mostrar=mostrar+dia;
		}
		if(mes<10) {

			mostrar=mostrar+"/0"+mes;
		}
		else {
			mostrar=mostrar+"/"+mes;
		}
		return mostrar+"/"+anio;
	}
```

```java
	// Getters & Setters

	public int getDia() {
		return dia;
	}
	public void setDia(int dia) {
		this.dia = dia;
	}
	public int getMes() {
		return mes;
	}
	public void setMes(int mes) {
		this.mes = mes;
	}
	public int getAnio() {
		return anio;
	}
	public void setAnio(int anio) {
		this.anio = anio;
	}

}
```

```java
package composicion;

public class Persona {

    //Atributos
    private String nombre;
    private String ci;
    private Fecha nac; //La fecha de nacimiento es una instancia de la clase Fecha


    //Constructor con parametros
    public Persona(String nombre,String ci,Fecha nac){

        this.nombre=nombre;
        this.ci=ci;
        this.nac=nac;
    }

    //toString

    public String toString() {
        return "Persona> [nombre=" + nombre + ", ci=" + ci + ", nac=" + nac.toString() + "]";
    }
```

Clase Persona

```java
    // Getters & Setters

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getCi() {
        return ci;
    }

    public void setCi(String ci) {
        this.ci = ci;
    }

    public Fecha getNac() {
        return nac;
    }

    public void setNac(Fecha nac) {
        this.nac = nac;
    }

}
```

@PROFESANTIAGO

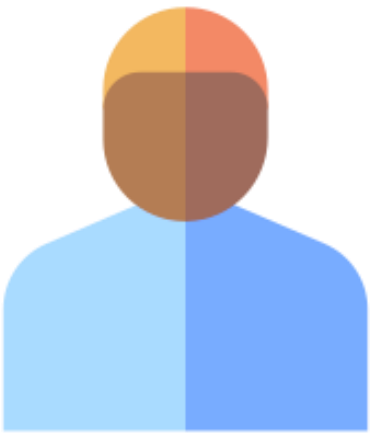
```java
package composicion;

public class Principal {

    public static void main(String[] args) {

        //Inicializo objetos Fecha y Personas
        Fecha fecha1 = new Fecha(1,2,2003);
        Fecha fecha2 = new Fecha(29,10,2023);
        Fecha fecha3 = new Fecha(15,6,1991);
        Persona per1 = new Persona("Santiago","2.455.666-1",fecha1);
        Persona per2 = new Persona("Berta","1.055.555-8",fecha2);
        Persona per3 = new Persona("Pepe","4.355.766-1",fecha3);

        //Creo un arreglo de personas y lo cargo manualmente

        Persona[] personas = new Persona[3];
        personas[0]=per1;
        personas[1]=per2;
        personas[2]=per3;

        //Recorro el arreglo mostrando la informacion de cada

        for(Persona p:personas) {

            System.out.println(p.toString());
        }
    }
}
```

main y ejecución

Console | Javadoc | Declaration | Problems

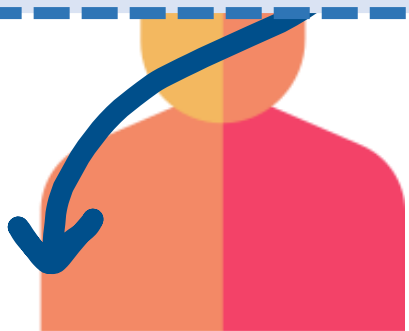
<terminated> Principal [Java Application] C:\Users\santi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.

```
Persona> [nombre=Santiago, ci=2.455.666-1, nac=01/02/2003]
Persona> [nombre=Berta, ci=1.055.555-8, nac=29/10/2023]
Persona> [nombre=Pepe, ci=4.355.766-1, nac=15/06/1991]
```



Herencia

Implica la existencia de una ***superclase*** (clase padre) que hereda sus atributos y métodos a *clases hijas*, las cuales además pueden implementar sus propios atributos y métodos.



@PROFESANTIAGO

Desarrollador es una subclase de Empleado

```java
package ejemplos;
//Herencia
public class Desarrollador extends Empleado {

    private String lenguaje; //Atributo propio de la clase hija

    //Constructor

    //Para el constructor necesito además los atributos de la superclase
    public Desarrollador (String ci,String nombre,float sueldo,Direccion dir, String lenguaje)
    {
        super(ci,nombre,sueldo,dir);//invoca al constructor de la clase superior
        this.lenguaje=lenguaje; // lenguaje es atributo propio de la clase Desarrollador

    }
```

Con **super** hago referencia a los métodos de la clase padre.

```java
    // Getters y setters de los atributos propios
    public String getLenguaje() {
        return lenguaje;
    }

    public void setLenguaje(String lenguaje) {
        this.lenguaje = lenguaje;
    }

    // toString propio, utilizando el imprimirDatos de la clase padre
    public String toString() {
        return "Desarrollador [lenguaje= " + lenguaje +" "+ super.imprimirDatos() + "]";
    }

    //Implemento un aumento de sueldo diferente para el Desarrollador (%5 extra)
    //modificando el metodo aumentarSueldo de la clase Empleado
    public void aumentarSueldo(float sueldo) {
        super.aumentarSueldo((super.getSueldo()*0.05f)+sueldo);

    }

}
```

Programa principal

```java
public static void main(String[] args) {

    Direccion d = new Direccion("Bvar.Artigas",3356);

    Empleado emp1 = new Empleado();
    Empleado emp2 = new Empleado ("2.222.333","Santiago",955.3f,d);

    Desarrollador des = new Desarrollador ("5.422.733","Roberto",1500,d,"JAVA");

    System.out.println(des.toString());
    des.aumentarSueldo(100);//invoco método de la clase hija
    System.out.println(des.toString());
    System.out.println("No desarrollador "+emp2.imprimirDatos());
    emp2.aumentarSueldo(100);//invoco método de la superclase
    System.out.println("No desarrollador "+emp2.imprimirDatos());
```

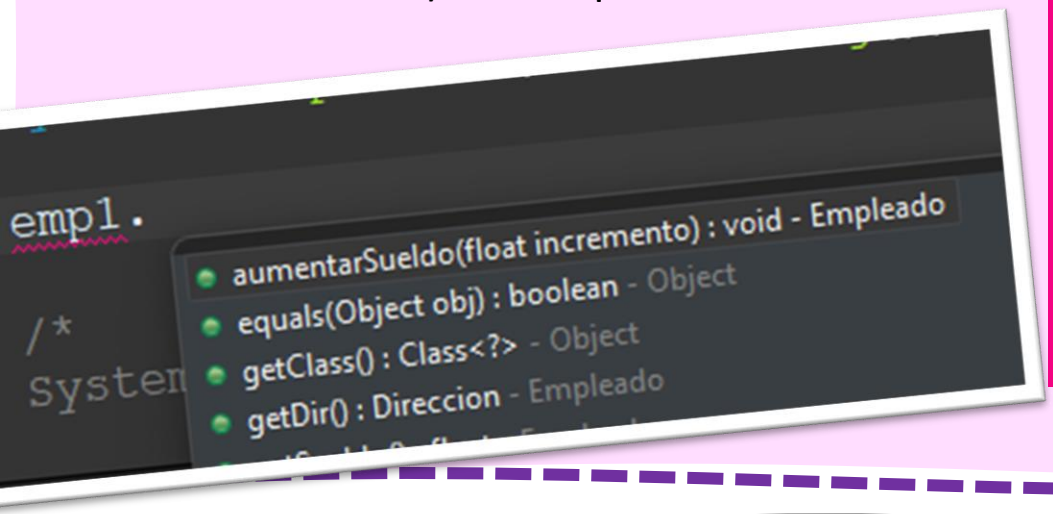
Ejecución

```
Desarrollador [lenguaje= JAVA |CI: 5.422.733 |Nombre: Roberto |Sueldo: 1500.0
Desarrollador [lenguaje= JAVA |CI: 5.422.733 |Nombre: Roberto |Sueldo: 1675.0
No desarrollador |CI: 2.222.333 |Nombre: Santiago |Sueldo: 955.3
No desarrollador |CI: 2.222.333 |Nombre: Santiago |Sueldo: 1055.3
```

clase Object

La clase Object es la ***superclase de todas las clases de Java*** (aunque no se indique de forma explícita).

Object tiene **métodos ya definidos que pueden utilizarse (y sobreescribirse) desde cualquier otra clase.**



getClass() Devuelve la clase a la que pertenece el objeto

equals() Devuelve si son el mismo objeto.

toString() Por defecto muestra el nombre de la clase instanciada y la dirección de memoria. Suele sobreescribirse para mostrar información útil del objeto.

Ejercicio 4

Crear clase **Vehículo** los atributos con matrícula y año, y el método **impuestoBase** : se calcula 0,34*año.

Existen **2 tipos de vehículos**: El **Eléctrico** tiene como propiedad el **precio** y el de **Combustión** la **cilindrada**.

Los vehículos **eléctricos tienen un impuesto del 8% de su precio más el impuesto base**, mientras que los vehículos de **combustión, tributan el triple de su cilindrada (en centímetros cúbicos) más el impuesto base.**

Crear una clase principal donde se instancien 2 vehículos y muestre el impuesto pagado por c/u.

@PROFESANTIAGO

Ejercicio 4

Clase Vehiculo

```java
package automotora;

public class Vehiculo {


        //Atributos clase padre
        private String matricula;
        private int anio;

        // Constructores

        public Vehiculo () {

        }

        public Vehiculo (String matricula, int anio) {

            this.matricula=matricula;
            this.anio=anio;

        }

        // impuestoBase


        public float impuestoBase() {

            return anio*0.34f;

        }

        //toString

        public String toString() {
            return "Vehiculo [matricula=" + matricula + ", año=" + anio + "]";
        }
```

```java
    // Getters y Setters

    public String getMatricula() {
        return matricula;
    }


    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    public int getAnio() {
        return anio;
    }

    public void setAnio(int anio) {
        this.anio = anio;
    }

}
```

Clase Combustion

```java
package automotora;

public class Combustion extends Vehiculo {


    //Atributos propios de la subclase

    private int cilindrada;


    public Combustion (String matricula, int anio, int cilindrada) {
        super(matricula,anio); // invoco el constructor de Vehiculos
        this.cilindrada=cilindrada;
    }


    //Sobreescribo el metodo de la superclase
    public float impuestoBase() {

        return super.impuestoBase()+3*cilindrada;

    }

    //toString
    public String toString() {
        return super.toString()+  " - Combustion [cilindrada=" + cilindrada + "]";
    }

    //get y set precio
    public float getCilindrada() {
        return cilindrada;
    }

    public void setCilindrada(int cilindrada) {
        this.cilindrada = cilindrada;
    }


}
```

@PROFESANTIAGO

```java
package automotora;

public class Electrico extends Vehiculo {

    // Atributos de la clase hija
    private float precio;

    // Constructor haciendo referencia al constructor y atributos de la super clase

    public Electrico ( String matricula, int anio, float precio) {
        super(matricula,anio); // invoco el constructor de Vehiculos
        this.precio=precio;
    }

    //Sobreescribo el metodo de la superclase
    public float impuestoBase() {

        return super.impuestoBase()+0.08f*precio;

    }

    // Sobreescribo el toString de la superclase
    public String toString() {
        return super.toString()+" - Electrico [precio= $" + precio + "]";
    }

    //get y set precio
    public float getPrecio() {
        return precio;
    }

    public void setPrecio(float precio) {
        this.precio = precio;
    }

}
```

Clase Electrico

Principal y ejecución

```java
package automotora;

public class Principal {

    public static void main(String[] args) {

        Electrico el = new Electrico ("AAA",2020,9000); //Instancio un auto eléctrico
        Combustion com= new Combustion ("BBB",1999,1600); //Instancio un auto a combustión

        System.out.println(com.toString());
        System.out.println("Total de impuestos: $"+com.impuestoBase()+"\n");
        System.out.println(el.toString());
        System.out.println("Total de impuestos: $"+el.impuestoBase());

    }

}
```

Console | Javadoc | Declaration | Problems

<terminated> Principal (1) [Java Application] C:\Users\santi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.

```
Vehiculo [matricula=BBB, año=1999] - Combustion [cilindrada=1600]
Total de impuestos: $5479.66

Vehiculo [matricula=AAA, año=2020] - Electrico [precio= $9000.0]
Total de impuestos: $1406.8
```

Clases abstractas

Las clases abstractas no se pueden instanciar (no se pueden crear objetos de este tipo).

Sirven para ser definir toda la ***información común que será heredada por las clases hijas.***

Ej: Figura geométrica.

La clase abstracta tiene atributos comunes, constructor y métodos con implementación, pero tiene al menos un **método abstracto** (***método que siempre será sobreescrito por las clases hijas***).

```
Public abstract class Figura {


Private String nombre;


Public abstract double area(); //La implementación se realiza en la clase de cada figura concreta(rectángulo, triángulo ,etc.)
}
```

Al definir un método como abstracto, **debe** ser implementado por **todas** las clases hijas.

Polimorfismo

- Un objeto puede comportarse de manera diferente dependiendo del contexto.
- Se puede utilizar un objeto de una subclase, siempre que el programa espere un objeto de la superclase
- Permite llamar a métodos de igual nombre pero que pertenecen a clases distintas.
- Las variables de objeto son polimórficas.

@PROFESANTIAGO

Clase abstracta Figura

```java
package claseAbstracta;

public abstract class Figura {

    //Atributo que herdedarán las clases hijas
    String nombre ;


    public Figura(String nombre) { //Constructor con parámetros
        this.nombre=nombre;

    }

    // Metodos que heredarán las clases hijas

    public String toString() {
        return "Figura [nombre=" + nombre + "]";
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /*el método area no puede implementarse en Figura
      ya que depende del tipo de figura que sea
      El metodo se define como abstracto y cada subclase tendrá el suyo propio
    */

    public abstract float area(); // método abstracto


}
```

Clase Triangulo

```java
1  package claseAbstracta;
2
3  public class Triangulo extends Figura {
4
5      private fl
6      private fl
```

Add unimplemented methods
Create new JUnit test case for 'Triangulo.java'
Make type 'Triangulo' abstract

1 method to implement:
- claseAbstracta.Figura.area()

Al crear una clase que hereda de una clase abstracta, es necesario implementar todos los métodos abstractos

```
    private float base;
    private float altura;

    // getters y setters
    public float getBase() {
        return base;
    }

    public void setBase(float base) {
        this.base = base;
    }

    public float getAltura() {
        return altura;
    }


    public void setAltura(float altura) {
        this.altura = altura;
    }


    //implemento el metodo abstracto
    public float area() {

        return (base*altura)/2;
    }


}
```

Clase Cuadrado

```
package claseAbstracta;

public class  Cuadrado extends Figura {

    private float lado; //Atributo propio

    public Cuadrado(String nombre,float lado) { //Constructor invocando al de la superclase
        super(nombre);
        this.lado=lado;
    }

    //Getters y Setters
    public float getLado() {
        return lado;
    }

    public void setLado(float lado) {
        this.lado = lado;
    }

    //Implemento el metodo abstracto area
    public float area() {
```

Principal y ejecución

```java
package claseAbstracta;
public class Principal_Polimorfismo {

    public static void main(String[] args) {

        Triangulo t =new Triangulo ("Triangulo A",8f,12f);
        Cuadrado c = new Cuadrado ("Cuadrado A",5.4f);

        //Cada uno implemente el metodo area de manera distinta
        System.out.println("El area de "+t.getNombre()+" es: "+t.area()+" m2");
        System.out.println("El area de "+c.getNombre()+" es: "+c.area()+" m2");

        ////////////// POLIMORFISMO ////////////////////

        System.out.println("\n --- Polimorfismo ---\n");
        Figura [] figuras = new Figura[4]; //arreglo de clase figura

        //Cargo el arreglo con diferentes tipos de figura (utilizando las clases hijas)

        // Puedo definirlo  como superclase y luego instanciar la subclase
        Figura f1=new Triangulo ("Triangulo B",6f,10f);
        Triangulo f2=new Triangulo ("Triangulo C",2f,6f);
        Cuadrado f3=new Cuadrado ("Cuadrado B",6f);
        Figura f4=new Cuadrado ("Cuadrado C",100f);

        figuras[0]=f1;
        figuras[1]= f2;
        figuras[2]= f3;
        figuras[3]= f4;

        for(Figura f:figuras) {

            System.out.println("El area de "+f.getNombre()+" es: "+f.area()+" m2");
        }
    }
```

Console × Javadoc Declaration Problems

<terminated> Principal_Polimorfismo [Java Application] C:\Users\santi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\

```
El area de Triangulo A es: 48.0 m2
El area de Cuadrado A es: 29.160002 m2

 --- Polimorfismo ---

El area de Triangulo B es: 30.0 m2
El area de Triangulo C es: 6.0 m2
El area de Cuadrado B es: 36.0 m2
El area de Cuadrado C es: 10000.0 m2
```

Ejercicio 4

Crear **clase persona** (nombre, apellido, edad y si está casado o no).

Constructor con todos los parámetros y sin ellos (inicializa los string como vacíos y casado en falso)

Clase **Legislador que hereda de persona**, tiene como atributo el partido político y número de despacho. También dos constructores y un método

Método abstracto **getCamara()** devuelve un String.

Crear las clases **Diputado y Senador**, que heredan de legislador.

Cada una implementa el método abstracto heredado devolviendo una cadena que indica en qué cámara trabaja.

Crear una clase de prueba, y probar la funcionalidad del programa haciendo uso de polimorfismo.

Clase Persona (superclase)

```java
package polimorfia;

public class Persona {

    private String nombre;
    private String apellido;
    private int edad;
    private boolean casado;


    //Constructores

    public Persona() {
        nombre ="";
        apellido="";
        casado =false;
    }

    public Persona (String nombre, String apellido, int edad,boolean casado) {

        this.nombre=nombre;
        this.apellido=apellido;
        this.edad=edad;
        this.casado=casado;

    }
```

@PROFESANTIAGO

```
    //toString
    public String toString() {
        return " [nombre=" + nombre + ", apellido=" + apellido +
                ", edad=" + edad + ", casado=" + casado + "]";
    }

//////////// getters y setters/////////////

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public int getEdad() {
        return edad;
    }
```

```
    public boolean isCasado() {
        return casado;
    }

    public void setCasado(boolean casado) {
        this.casado = casado;
    }

}
```

```
public abstract class Legislador extends Persona {

    private String partido;
    private String despacho;


    // constructores
    public Legislador() {
        super();
        partido="";
        despacho="";
    }

    public Legislador (String nombre,String apellido, int edad,
        boolean casado, String partido,String despacho) {
        super(nombre,apellido,edad,casado);
        this.partido=partido;
        this.despacho=despacho;
    }
```

Clase Legislador

(subclase abstracta)

```
    //// metodo abstracto //////
    public abstract String getCamara();


    ///// getters y setters //////

    public String getPartido() {
        return partido;
    }

    public void setPartido(String partido) {
        this.partido = partido;
    }

    public String getDespacho() {
        return despacho;
    }

    public void setDespacho(String despacho) {
        this.despacho = despacho;
    }

}
```

```
public class Diputado extends Legislador {

    public Diputado() {

    }

    public Diputado (String nombre,String apellido, int edad,
        boolean casado, String partido,String despacho) {

        super(nombre,apellido,edad,casado,partido,despacho);

    }
    //Implementación del método abstracto.
    public String getCamara() {

        return "Cámara de Representantes";

    }
}
```

Clase Diputado

Clase Senador

```java
public class Senador extends Legislador {

    public Senador() {

    }

    public Senador (String nombre,String apellido, int edad, boolean casado,
            String partido,String despacho) {
        super(nombre,apellido,edad,casado,partido,despacho);
    }

    //Implementación del método abstracto.
    public String getCamara() {

        return "Cámara de Senadores";
    }
}
```

Diputado y Senador no tienen getters, setters ni toString ya que no tienen atributos propios. Heredan atributos y métodos de la clase padre.

Principal y ejecución.

```java
public class Principal {

    public static void main(String[] args) {

        //Puedo definirlos de tipo Legislador e instanciar
        // un Diputado/Senador
        Diputado p1 = new Diputado();
        Senador p2 = new Senador();
        Legislador p3 = new Senador();
        Legislador p4 = new Diputado();

        Legislador[] legisladores =new Legislador[4];

        legisladores[0]=p1;
        legisladores[1]=p2;
        legisladores[2]=p3;
        legisladores[3]=p4;

        // puedo utilizar: for (Legislador l: legisladores)
        for (int i=0; i<legisladores.length;i++) {

        System.out.println(legisladores[i].getCamara());

        }
    }
}
```

```
Console  Javadoc  Declaration
<terminated> Principal (2) [Java Application] C:\User
Cámara de Representantes
Cámara de Senadores
Cámara de Senadores
Cámara de Representantes
```

La MV de Java en tiempo de ejecución determina si es un diputado o un senador e invoca la implementación del método getCamara que corresponda.

@PROFESANTIAGO

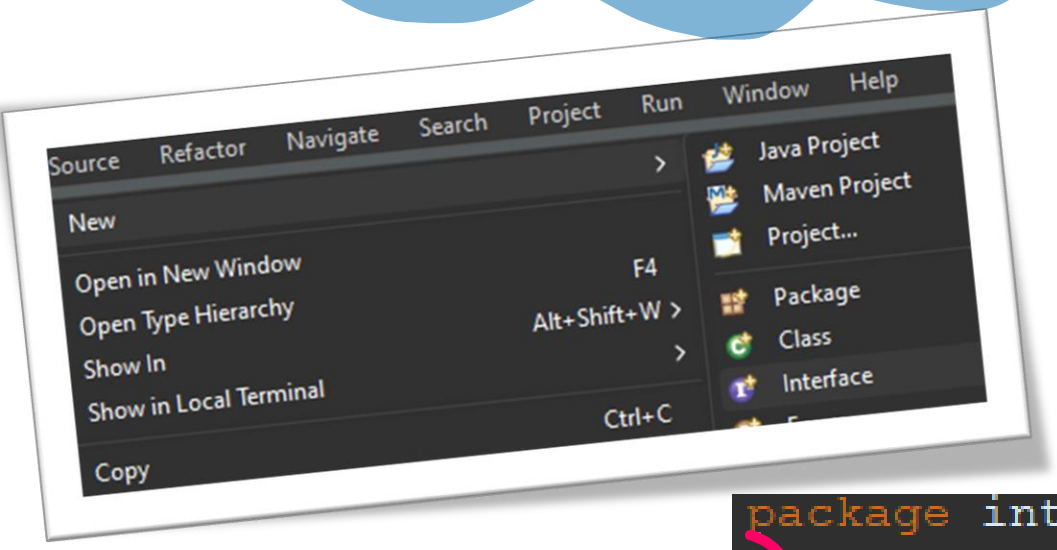
Interfaces

Se definen métodos (únicamente declarados).

Las clases que implementen la interfaz deberán implementar estos métodos.

Similar a clase abstracta, pero con diferencias:

- Las interfaces no implementan métodos propios.
- Una clase puede implementar varias interfaces, pero sólo puede heredar de una clase abstracta.



Creo interfaz y una clase que la implemente

```java
package interfaces;

public interface Acciones {

    public String cantar();
    public String caminar();
}
```

```java
public class Persona implements Acciones {
```

Add unimplemented methods
Create new JUnit test case for 'Persona.java'
Make type 'Persona' abstract

2 methods to implement:
- interfaces.Acciones.cantar()
- interfaces.Acciones.caminar()

@PROFESANTIAGO

```java
package interfaces;

public class Persona implements Acciones {

    @Override
    public String cantar() {

        return "la la la la";
    }

    @Override
    public String caminar() {

        return "Puedo caminar erguido";
    }

}
```

```java
package interfaces;

public class Pajaro implements Acciones{

    @Override
    public String cantar() {

        return "pio pio pio";
    }

    @Override
    public String caminar() {

        return "Prefiero volar";
    }

}
```

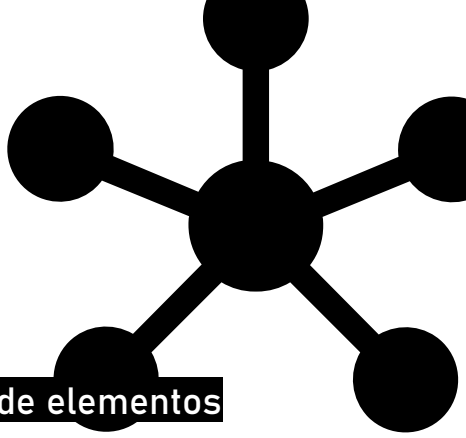
Persona y Pajaro implementan la interfaz acciones.

Deben entonces implementar todos los métodos allí declarados (cada clase puede hacer su propia implementación).

```java
package interfaces;

public class Principal {

    public static void main(String[] args) {

        Persona per =new Persona();
        Pajaro ave = new Pajaro();

        System.out.println(per.caminar());
        System.out.println(per.cantar());
        System.out.println(ave.caminar());
        System.out.println(ave.cantar());

    }
}
```

```
Console  Javadoc  Declaration  Problems
<terminated> Principal (3) [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe
Puedo caminar erguido
la la la la
Prefiero volar
pio pio pio
```

@PROFESANTIAGO



Colecciones

Una colección es un grupo de objetos (elementos).

Similares a los arreglos, pero dinámicos (la cantidad de elementos puede variar).

En Java, las colecciones se implementan mediante la interfaz Collection, donde se definen todos los métodos comunes (añadir, eliminar, obtener tamaño, etc.)

Hay otras interfaces que extienden Collection y aportan otras funcionalidades, definiendo varios tipos de colecciones.

A su vez hay varias clases que implementan estas interfaces.

//Set (Interfaz)

No tiene elementos repetidos

Clases que implementan Set:

HashSet

Elementos no se almacenan en un orden específico, es la que suele tener mejor rendimiento (TAD diccionario).

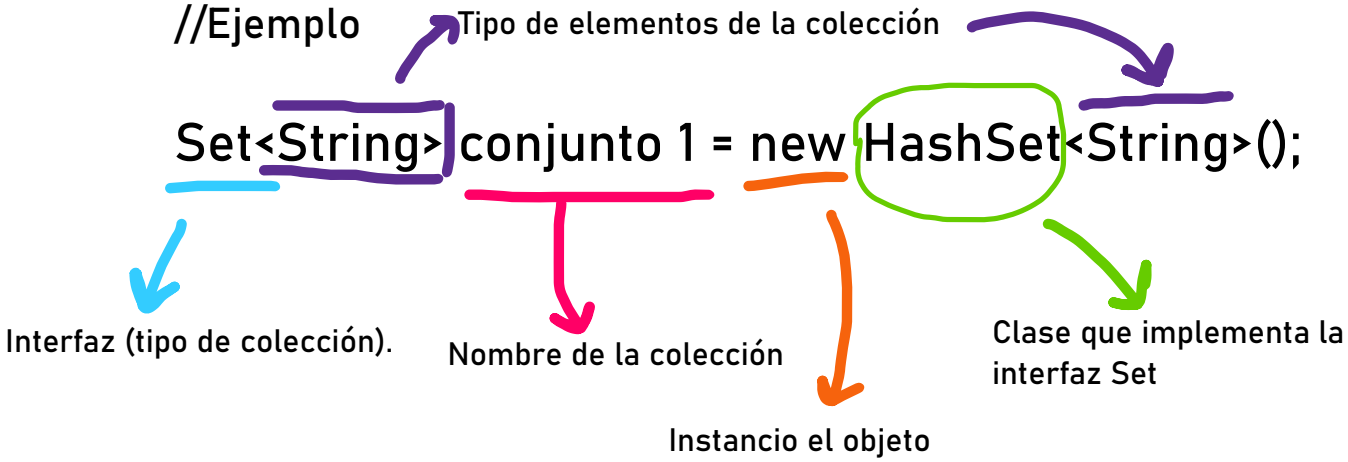
TreeSet

Se ordenan en función de sus valores.

LinkedHashSet

Se ordenan en función del orden de inserción.

//Ejemplo



//List (Interfaz)

Admite elementos duplicados y se accede de manera posicional.

Clases que implementan List:

ArrayList

Arreglo dinámico que aumenta su tamaño. Lento para eliminar/agregar elemento en medio o al inicio.

LinkedList

Lista doblemente enlazada, suele tener mejor rendimiento. Utiliza punteros/referencias a los nodos siguiente y anterior.

```
List<Libro> conjunto 2 = new ArrayList<Libro>();
```

//Map (Interfaz)

Asocia claves a valores.

No puede tener claves duplicadas y cada clave está asociada a un único valor.

HashMap

TreeMap

LinkedHashMap

```
Map<String,Libro> conjunto3 = new HashMap <String,Libro>();
```

@PROFESANTIAGO

Set

```java
package colecciones;
import java.util.*;


public class Colecciones_Main {

    public static void main(String[] args) {

        Set<String> conjuntoA = new HashSet<String>();
        conjuntoA.
    }

}
```

```
add(String e) : boolean - Set
addAll(Collection<? extends String> c) : boolean - Set
clear() : void - Set
contains(Object o) : boolean - Set
containsAll(Collection<?> c) : boolean - Set
equals(Object o) : boolean - Set
forEach(Consumer<? super String> action) : void - Iterable
```

List

```java
//Coleccion tipo List implementada como ArrayLis
//Lista de libros
List <Libro> conjuntoL = new ArrayList<Libro>();
Libro Libro1 = new Libro();
conjuntoL.add
```

```
d position in this list
rently at that position (if any)
adds one to their indices).

element is to be inserted

add(Libro e) : boolean - List
add(int index, Libro element) : void - List
addAll(Collection<? extends Libro> c) : boolean - List
addAll(int index, Collection<? extends Libro> c) : boolean - List
addFirst(Libro e) : void - List
```

Map

```java
//Map
//Colección tipo Map implementada como HashMap

//Cada elemento se almacena con una clave asociada
Map <String,Libro> conjuntoMap = new HashMap<String,Libro>();
conjuntoMap.put("Clave 1", Libro1);
conjuntoMap.get("Clave 1");
```

Cada tipo de implementación tiene sus métodos asociados.

add agrega un elemento a la colección. (si es un de tipo Map se utiliza el método put).

get devuelvle el elemento que se encuentra en la posición pasada por parámetros.

indexOf devuelve la posición en la que se encuentra por primera vez un elemento pasado por parámetros

remove elimina el elemento pasándole el índice o el elemento

contains devuelve un booleano indicando si existe o no el elemento (o clave en caso de ser Map).

Ejemplos

```java
package colecciones;
import java.util.*;

public class Colecciones_Main {

    public static void main(String[] args) {

        //Set

        //Colección tipo Set implementado como HashSet
        Set<String> conjuntoA = new HashSet<String>();
        conjuntoA.add("Elemento 1");
        conjuntoA.add("Elemento 2");

        System.out.println("Recorro Set \n");
        //recorro el Set

        for (String s: conjuntoA) {
            System.out.println(s);
        }
```

```java
//List

//Colección tipo List implementada como ArrayList
// Lista de libros
List <Libro> conjuntoL = new ArrayList<Libro>();
Libro Libro1 = new Libro("AAA-BBB","La llamada de Chtulhu","HP Lovecraft",100);
Libro Libro2 = new Libro();
Libro Libro3 = new Libro("000-ABA","Redes desde cero","Santiago Martínez",187);
conjuntoL.add(Libro1);//Agrega al final
conjuntoL.add(0,Libro2);//Agrega en posición concreta
conjuntoL.add(Libro3);//Agrega al final

System.out.println("\nRecorro ArrayList \n");

// Recorro ArrayList

for (Libro l:conjuntoL) {

    System.out.println(l.toString());

}
```

```java
        //Map
        //Colección tipo Map implementada como HashMap

        //Cada elemento se almacena con una clave asociada
        Map <String,Libro> conjuntoMap = new HashMap<String,Libro>();
        conjuntoMap.put("Clave 1", Libro1);
        conjuntoMap.put("Clave 2", Libro2);
        conjuntoMap.put("Clave 3", Libro3);


        System.out.println("\nRecorro Map \n");

        for (String clave: conjuntoMap.keySet()) {

            System.out.println(conjuntoMap.get(clave));


        }

    }

}
```

Ejecución

```
Recorro Set

Elemento 1
Elemento 2

Recorro ArrayList

El libro con ISBN null escrito por null tiene 0 páginas.
El libro con ISBN AAA-BBB escrito por HP Lovecraft tiene 100 páginas.
El libro con ISBN 000-ABA escrito por Santiago Martínez tiene 187 páginas.

Recorro Map

El libro con ISBN 000-ABA escrito por Santiago Martínez tiene 187 páginas.
El libro con ISBN AAA-BBB escrito por HP Lovecraft tiene 100 páginas.
El libro con ISBN null escrito por null tiene 0 páginas.
```

@PROFESANTIAGO

Agregar a una colección 10 enteros aleatorios del 1 al 100 sin repetir.

```java
public class ColecciónEnteros {

    public static void main(String[] args) {

        Set<Integer>aleatorios=new TreeSet<Integer>();
        //Set<Integer>aleatorios=new HashSet<Integer>();
        while (aleatorios.size()<10) {
            int numero = (int)(Math.random()*100+1);
            aleatorios.add(numero);
        }

        for (Integer i:aleatorios) {
            System.out.println(i);
        }
    }
}
```

Console × Declaration

<terminated> ColecciónEnteros [Ja

```
27
36
41
45
52
72
77
84
87
93
```

```java
public class ColecciónEnteros {

    public static void main(String[] args) {

        //Set<Integer>aleatorios=new TreeSet<Integer>();
        Set<Integer>aleatorios=new HashSet<Integer>();
        while (aleatorios.size()<10) {
            int numero = (int)(Math.random()*100+1);
            aleatorios.add(numero);
        }

        for (Integer i:aleatorios) {
            System.out.println(i);
        }
    }
}
```

Console × Declaration Javadoc Problems

```
53
86
38
70
71
24
89
41
76
79
```

TreeSet agrega en orden numérico y Hashset no

@PROFESANTIAGO

Ejercicio 5

Utilizo una colección tipo Map, utilizo como clave el nombre de la asignatura y la calificación como valor a almacenar

Escribir un programa que solicite a un alumno las asignaturas que está cursando y su correspondiente calificación.

Cuando deje de introducir asignaturas el programa deberá mostrar un mensaje con el promedio final.

```
package colecciones;
import java.util.*;


public class Map_Calificaciones {

    public static void main(String[] args) {

        Map<String,Integer> notas= new HashMap<String,Integer>();

        int opc=4;
        Scanner sc=new Scanner(System.in);

        while (opc!=2) {

            System.out.println("1.Ingresar Nota\n2.Ver Promedio");


            opc=sc.nextInt();
            sc.nextLine();

            if(opc==1) {

                System.out.println("Ingrese asignatura: ");
                String asignatura=sc.nextLine();
                System.out.println("Ingrese nota: ");
                int nota=sc.nextInt();
                notas.put(asignatura, nota);

            }//fin if
```

@PROFESANTIAGO

Otra forma de recorrer un Hashmap

```
for(Map.Entry<String,Integer> entry: notas.entrySet()) {
    total+=entry.getValue();
    promedio=(float)total/notas.size();

}
```

```
            }//fin if

        }//fin while
        sc.close();
        int total=0;
        float promedio=0f;

        for(String asig: notas.keySet()) { //Utilizo las claves para recorrer elementos
            total+=notas.get(asig);
            promedio=(float)total/notas.size();

        }
        System.out.println("Promedio final ("+notas.size()+" asignaturas): " +promedio);
    }

}
```

Recordar que la clase HashMap no ingresa claves duplicadas.

Console | Declaration | Javadoc | Problem

<terminated> Map_Calificaciones [Java Application] C:\Pro

```
1.Ingresar Nota
2.Ver Promedio
1
Ingrese asignatura:
Matemáticas
Ingrese nota:
8
1.Ingresar Nota
2.Ver Promedio
1
Ingrese asignatura:
MAtemáticas
Ingrese nota:
9
1.Ingresar Nota
2.Ver Promedio
2
Promedio final (2 asignaturas): 8.5
```

```
1.Ingresar Nota
2.Ver Promedio
1
Ingrese asignatura:
Programación
Ingrese nota:
10
1.Ingresar Nota
2.Ver Promedio
1
Ingrese asignatura:
Programación
Ingrese nota:
9
1.Ingresar Nota
2.Ver Promedio
2
Promedio final (1 asignaturas): 9.0
```

@PROFESANTIAGO

Ejercicio 6

Implementar una clase para gestión de bebidas. En un almacén se guardan un conjunto de bebidas. Las bebidas pueden ser agua mineral o refrescos. *De todas las bebidas necesita almacenarse un identificador, capacidad, marca y precio. Si es agua mineral, interesa el origen y si es refresco su sabor.*

Operaciones del almacén:

AgregarBebida -> Siempre que el ID no esté repetido

EliminarBebida -> A partir del ID

TotalDeUnaMarca -> Precio total de una marca

PrecioTotal -> Precio de todas las bebidas.

MostrarInformacion -> Mostrar información de todas las bebidas.

Integración con MySQL

Es necesario descargar el JDBC driver de MySQL

Java Database Connectivity

MySQL Community Downloads

Connector/J

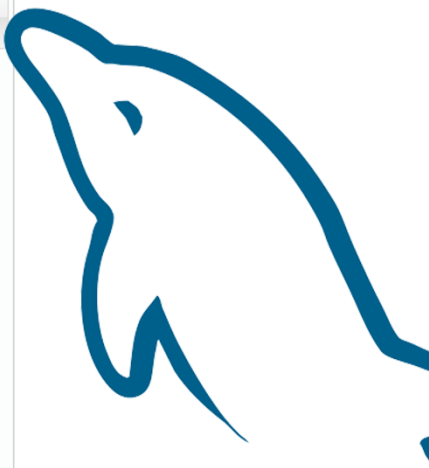
General Availability (GA) Releases | Archives

Connector/J 8.3.0

Select Operating System:

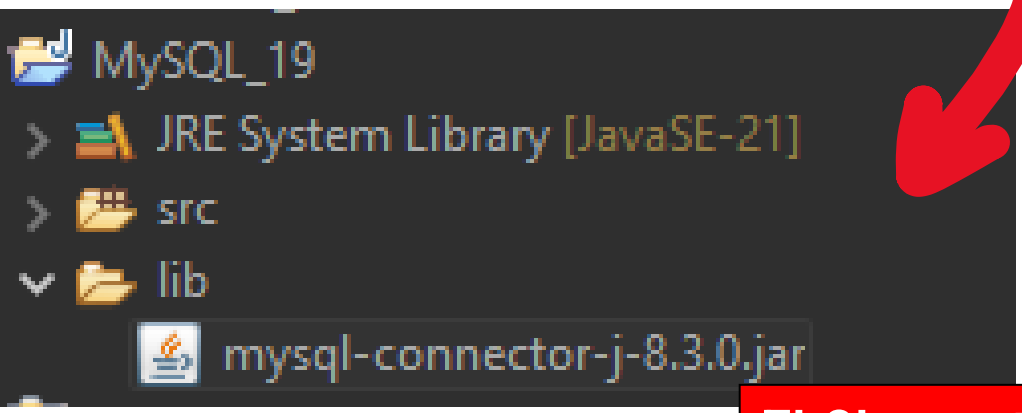
Platform Independent

Platform Independent (Architecture Independent), Compressed TAR Archive (mysql-connector-j-8.3.0.tar.gz)	8.3.0	4.1M	Download
	MD5: d29a0a1f3920e62be749cde1f61e3e1e | Signature		
Platform Independent (Architecture Independent), ZIP Archive (mysql-connector-j-8.3.0.zip)	8.3.0	4.8M	Download
	MD5: 8b4e005c4371adb851bf070c4365fa30 | Signature		

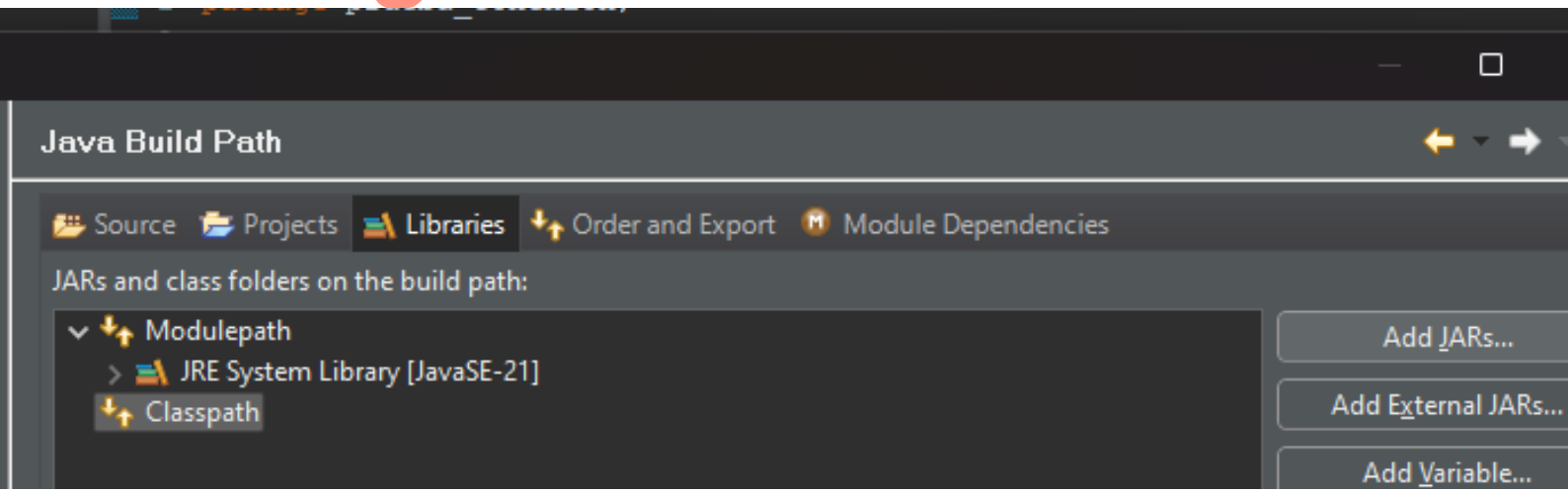


Una vez descargado el controlador, podremos conectarnos con el sistema gestor de base de datos.

Primero creo una carpeta en el proyecto para importar el .jar y posteriormente lo agrego al classpath



El Classpath indica a la Máquina Virtual de Java dónde buscar paquetes y clases definidas por el usuario a la hora de ejecutar programas.



```java
package prueba_conexion;
import java.sql.Connection;
import java.sql.DriverManager;

public class ConnectorMySQL {

//////////Atributos

     // Librería de MySQL
    //public String driver = "com.mysql.cj.jdbc.Driver"; // NO ES NECESARIO


    public String database = "Prueba";
    public String hostname = "192.168.1.111";
    public String port = "3306";
    public String url = "jdbc:mysql://" + hostname + ":" + port + "/" + database;
    public String username = "root";
    public String password = "bd64";

    Connection conn = null;

////////Constructor

    public ConnectorMySQL() {

        try {

            //Class.forName(driver);
            //YA NO ES NECESARIO REGISTRAR CLASE DRIVER

            conn = DriverManager.getConnection(url, username, password);

            if (conn==null) {
                System.out.println("Error de conexión");
            }
            else {
                System.out.println("Conexión exitosa");
            }

        }
        catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

}
```

Es necesario **crear una clase para establecer la conexión.**

Cada vez que necesite insertar/consultar datos debo crear un objeto de esta clase.

Creo además una clase para implementar las operaciones sobre la base de datos.

```java
///////Métodos getConnection y desconectar

    public Connection getConnection() {

        return conn;
    }

    public void desconectar() {

        conn=null;

    }
}
```

```java
package prueba_conexion;

import java.sql.Statement;
import java.util.Scanner;

public class OperacionesBD {

    public void insertar() {

        Scanner sc = new Scanner (System.in);
        System.out.println("Ingrese CI: ");
        String ci=sc.nextLine();
        System.out.println("Ingrese Nombre: ");
        String nom=sc.nextLine();
        System.out.println("Ingrese Apellido: ");
        String ape=sc.nextLine();
        System.out.println("Ingrese Grupo: ");
        String gru=sc.nextLine();
        sc.close();

        ConnectorMySQL con= new ConnectorMySQL();//Abrir conexión

        String sql="INSERT INTO Alumnos VALUES('"+ci+"', '"+nom+"', '"+ape+"', '"+gru+"')";
```

Creo una clase que implemente los métodos de interacción con la BD

```java
        try {

            Statement st= con.getConnection().createStatement();
            st.executeUpdate(sql);
            System.out.println("Se ha ingresado el alumno.");

        }

        catch(Exception ex) {

            System.out.println(ex.getMessage());

        }

        con.desconectar();

    }

}
```

Utilizo el objeto Statement para ejecutar una sentencia SQL y, opcionalmente, obtener el conjunto de resultados (ResultSet) generado por ella.

@PROFESANTIAGO

Main y ejecución.

```java
package prueba_conexion;


public class prueba_conexion {

    public static void main(String[] args) {

        OperacionesBD bd = new OperacionesBD();

        bd.insertar();
    }
}
```

Console | Declaration | Javado

<terminated> prueba_conexion [Java Applica

```
Ingrese CI:
4567834-0
Ingrese Nombre:
Santiago
Ingrese Apellido:
Martínez
Ingrese Grupo:
3eroBA
Conexión exitosa
Se ha ingresado el alumno.
```

Proxmox\Prueba\Alumnos\ - HeidiSQL Portable 12.5.0.6677

Archivo Editar Buscar Consulta Herramientas Ir a Ayuda

Filtro de bases de d | Filtro de tablas | Host: 192.168.1.111 | Base de datos: Prueba | Tabla: Alum

- Proxmox
  - Prueba 16,0 KiB
    - Alumnos 16,0 KiB
  - information_schema
  - mysql
  - performance_schema

Prueba.Alumnos: 1 filas en total (aproximadamente)

CI	Nombre	Apellido	Grupo
4567834-0	Santiago	Martínez	3eroBA

SQL -> INSERT, DELETE, UPDATE

- Creo objeto que establece la conexión con la BD.
- Defino un String con la sentencia SQL.
- En bloque try catch, creo objeto de tipo ***Statement*** para ejecutar la sentencia SQL.

Para una sentencia ***SELECT*** es diferente, ya que va a devolver un conjunto de valores (Se almacenan en un objeto de tipo **ResultSet**) -> *clase que se encuentra en el package java.sql*

```
public static void seleccionarAlumnos() { //Al ser estatico no se invoca desde un objeto

    ConnectorMySQL con= new ConnectorMySQL();
    String sql ="SELECT * FROM Alumnos";
    try {
        Statement st = con.getConnection().createStatement();
        ResultSet rs =st.executeQuery(sql);

        while (rs.next()) { //Recorro los registros almacenados en el objeto resultset
            String ci = rs.getString("CI");
            String nom = rs.getString("Nombre"); //atributos en la tabla
            String ape = rs.getString("Apellido");
            String gr = rs.getString("Grupo");
            System.out.println("CI: "+ci+" Nombre: "+nom+" "+ape+" Grupo: "+gr);
        }

    }
    catch(Exception ex){
        System.out.println(ex.getMessage());
    }


}
```

@PROFESANTIAGO

Main y ejecución.

ConnectorMySQL.java | prueba_conexion.java | *OperacionesBD.java

```java
package prueba_conexion;


public class prueba_conexion {

    public static void main(String[] args) {

        //OperacionesBD bd = new OperacionesBD();

        //OperacionesBD.insertar(); //insertar es un método estático
        OperacionesBD.seleccionarAlumnos();
    }
}
```

Console | Declaration | Javadoc | Problems

<terminated> prueba_conexion [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (19 mar. 2024 12:19:31 – 12:19:32) [pid: 3112]

```
Conexión exitosa
CI: 12345 Nombre: pepe prueba Grupo: 23
CI: 4567834-0 Nombre: Santiago Martínez Grupo: 3eroBA
CI: 654345-4 Nombre: Marta Almada Grupo: 1eroC
```

Unnamed\Prueba\Alumnos\ - HeidiSQL Portable 11.3.0.6295

Archivo Editar Buscar Consulta Herramientas Ir a Ayuda

Filtro de bases de d | Filtro de tablas | Host: 192.168.1.111 | Base de datos: Prueba | Tabla: Alumnos | Datos | Consulta*

- Unnamed
  - Prueba 16,0 KiB
    - Alumnos 16,0 KiB
  - information_schema
  - mysql
  - performance_schema
  - sys

Siguientes | Mostrar todo | Ordenación (1) | Columna

CI	Nombre	Apellido	Grupo
456783...	Santiago	Martínez	3eroBA
654345-4	Marta	Almada	1eroC
12345	pepe	prueba	23

@PROFESANTIAGO

```
mysql> select * from autos;
+----------+-----------+--------+---------+------+--------+
| Matricula | Marca     | Modelo | Color   | Anio | Precio |
+----------+-----------+--------+---------+------+--------+
| 1123-ABC | Mercedes  | GLK    | Rojo    | 2014 |  30000 |
| 1123-ABF | Kia       | Avella | Blanco  | 2000 |   5100 |
| 3268-BVN | Chevrolet | Corsa  | Negro   | 2002 |   6000 |
| 5068-ZZZ | Volkswagen| Gol    | Gris    | 2008 |  13500 |
| 8015-BBB | Renault   | Kwid   | Naranja | 2022 |   8350 |
+----------+-----------+--------+---------+------+--------+
5 rows in set (0.00 sec)
```

Ejercicio 7

Desarrollar un programa que muestre los **vehículos disponibles en una automotora**, los cuales se encuentran registrados en una base de datos.

Inicialmente muestra el primer registro y luego permite recorrerlas mediante las **opciones anterior y siguiente**

@PROFESANTIAGO

Ejercicio 7

```java
package autos;

import java.sql.Connection;

public class ConnectorMySQL {

//////////Atributos
    public String database = "Automotora";
    public String hostname = "192.168.1.111";
    public String port = "3306";
    public String url = "jdbc:mysql://" + hostname + ":" + port + "/" + database;
    public String username = "root";
    public String password = "bd64";

    Connection conn = null;

////////Constructor

    public ConnectorMySQL() {

        try {

            conn = DriverManager.getConnection(url, username, password);

            if (conn==null) {
                System.out.println("Error de conexión");
            }
            else {
                //System.out.println("Conexión exitosa\n");
            }

        }
        catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

    }
///////Métodos getConnection y desconectar

    public Connection getConnection() {

        return conn;
    }

    public void desconectar() {

        conn=null;

    }
}
```

Clase para establecer la conexión

@PROFESANTIAGO

```java
public class Auto {
//Creo una clase autos para poder almacenar los registros de la base de datos
    private String matricula;
    private String marca;
    private String modelo;
    private String color;
    private int anio;
    private int precio;

// constructor con parametros
    public Auto(String matricula, String marca, String modelo, String color, int anio, int precio) {

        this.matricula = matricula;
        this.marca = marca;
        this.modelo = modelo;
        this.color = color;
        this.anio = anio;
        this.precio = precio;
    }

// Getters y Setters
    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String marca) {
        this.marca = marca;
    }

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }
```

Clase Auto

(mismos atributos que en la base de datos)

```java
    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public int getAnio() {
        return anio;
    }

    public void setAnio(int anio) {
        this.anio = anio;
    }

    public int getPrecio() {
        return precio;
    }

    public void setPrecio(int precio) {
        this.precio = precio;
    }

//toString
    public String toString() {
        return "Auto [matricula=" + matricula + ", marca=" + marca + ", modelo=" + modelo + ", color=" + color
                + ", anio=" + anio + ", precio=" + precio + "]";
    }

}
```

```java
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class main {

    private static int indice=0;
    private static List <Auto> autos=new ArrayList<Auto>(); //Creo una lista de objetos tipo auto
    //Programa principal
    public static void main(String[] args) {

        int opc=0;
        Scanner sc = new Scanner (System.in);
        cargarAuto(); //Métodos estáticos definidos en la clase main
        mostrarAuto();

        while (opc!=3) {
            mostrarMenu();
            opc =sc.nextInt();

            switch(opc) {

                case 1:
                    if (indice==autos.size()-1) {
                        indice=0;
                    }
                    else {
                        indice++;
                    }
                    mostrarAuto();
                    break;
                case 2:
                    if (indice<=0) {
                        indice=autos.size()-1;
                    }
                    else {
                        indice--;
                    }
                    mostrarAuto();
                    break;
                case 3:
                        break;
                default: System.out.println("Opción inválida");
            }
        }
        sc.close();
    }
```

Clase Main

```
        sc.close();
    }


    //Metodos

    //

    public static void cargarAuto() { //carga los registros en una lista

        ConnectorMySQL con = new ConnectorMySQL(); //inicio conexion cin BD
        String sql = "SELECT * FROM autos"; //Creo un string con la consulta
        //creo objeto de tipo Statement
        try {
            Statement st = con.getConnection().createStatement();
            ResultSet rs = st.executeQuery(sql);
            //Recorro resultset
            while (rs.next()) {
                String matricula=rs.getString("Matricula");
                String marca=rs.getString("Marca");
                String modelo=rs.getString("Modelo");
                String color=rs.getString("Color");
                int anio=rs.getInt("Anio");
                int precio=rs.getInt(6); //Podemos obtener el atributo mediante su numero de columna
                //instancio un auto, utilizo el constuctor con los valores obtenidos desde
                //el resultset y agrego el auto a la lista de autos
                Auto a = new Auto(matricula,marca,modelo,color,anio,precio);
                autos.add(a);
            }
        }
        catch (Exception ex){
            System.out.println(ex.getMessage());
        }
        con.desconectar();

    }

    public static void mostrarAuto() {

        System.out.println(autos.get(indice).toString()); //muestra un elemento concreto de la lista
    }

    public static void mostrarMenu() {
        System.out.println("\nElija una opcion:\n");
        System.out.println("1.Siguiente");
        System.out.println("2.Anterior");
        System.out.println("3.Salir");

    }
```

@PROFESANTIAGO

Console | Declaration | Javadoc | Problems

main [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (20 mar. 2024 11:16:06) [pid: 3024]

```
Auto [matricula=1123-ABC, marca=Mercedes, modelo=GLK, color=Rojo, anio=2014, precio=30000]

Elija una opcion:

1.Siguiente
2.Anterior
3.Salir
1
Auto [matricula=1123-ABF, marca=Kia, modelo=Avella, color=Blanco, anio=2000, precio=5100]

Elija una opcion:

1.Siguiente
2.Anterior
3.Salir
2
Auto [matricula=1123-ABC, marca=Mercedes, modelo=GLK, color=Rojo, anio=2014, precio=30000]

Elija una opcion:

1.Siguiente
2.Anterior
3.Salir
2
Auto [matricula=8015-BBB, marca=Renault, modelo=Kwid, color=Naranja, anio=2022, precio=8350]

Elija una opcion:

1.Siguiente
2.Anterior
3.Salir
```

ejecución

Unnamed\Automotora\autos\ - HeidiSQL Portable 11.3.0.6295

Archivo Editar Buscar Consulta Herramientas Ir a Ayuda

Filtro de bases de d | Filtro de tablas

Host: 192.168.1.111 | Base de datos: Automotora | Tabla: autos | Datos | Consulta*

- Unnamed
  - Automotora 16,0 KiB
    - autos 16,0 KiB
  - Prueba
  - information_schema
  - mysql
  - performance_schema
  - sys

Automotora.autos: 5 filas en

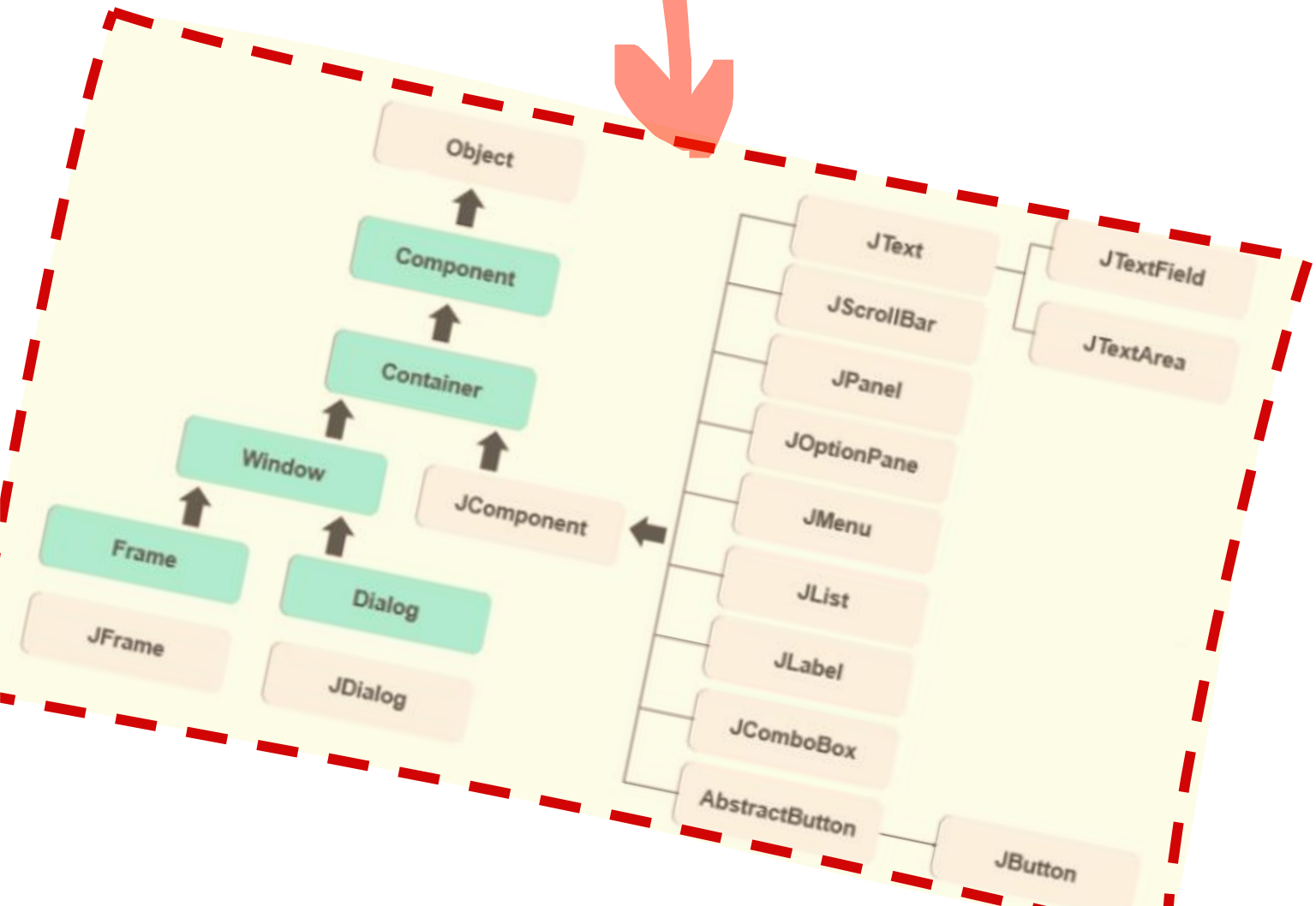
Siguientes | Mostrar todo | Ordenación | Columnas

Matricula	Marca	Modelo	Color	Anio	Precio
1123-ABC	Mercedes	GLK	Rojo	2.014	30.000
1123-ABF	Kia	Avella	Blanco	2.000	5.100
3268-BVN	Chevrolet	Corsa	Negro	2.002	6.000
5068-ZZZ	Volkswagen	Gol	Gris	2.008	13.500
8015-BBB	Renault	Kwid	Naranja	2.022	8.350

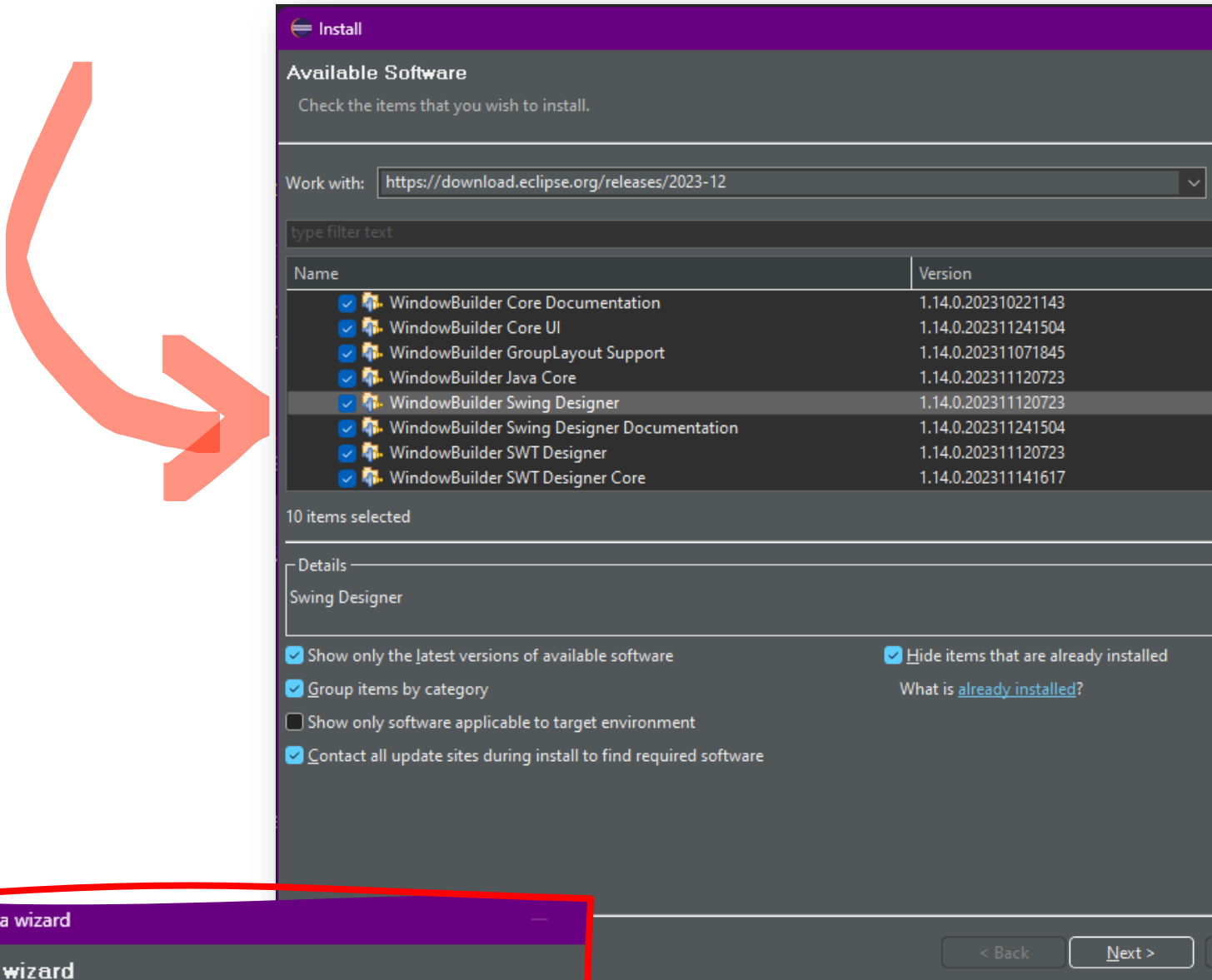
Java SWING

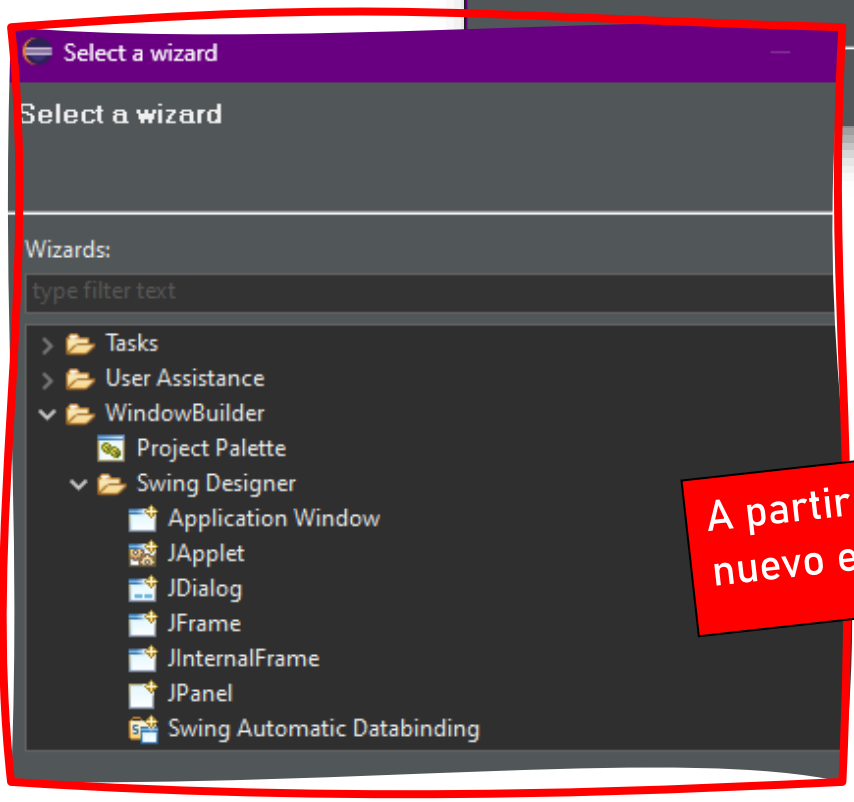
Java Swing es un conjunto de bibliotecas y componentes gráficos que se utilizan para crear interfaces de usuario (UI) en aplicaciones de escritorio en Java. Fue desarrollado como parte de la plataforma Java Standard Edition (Java SE) y proporciona una forma de construir interfaces de usuario interactivas y visualmente atractivas.

Gestionaremos eventos aplicados a determinados objetos, clases e interfaces que son proporcionados por la biblioteca gráfica Swing



Desde el menú help podemos descargar e instalar varios plugins para eclipse.





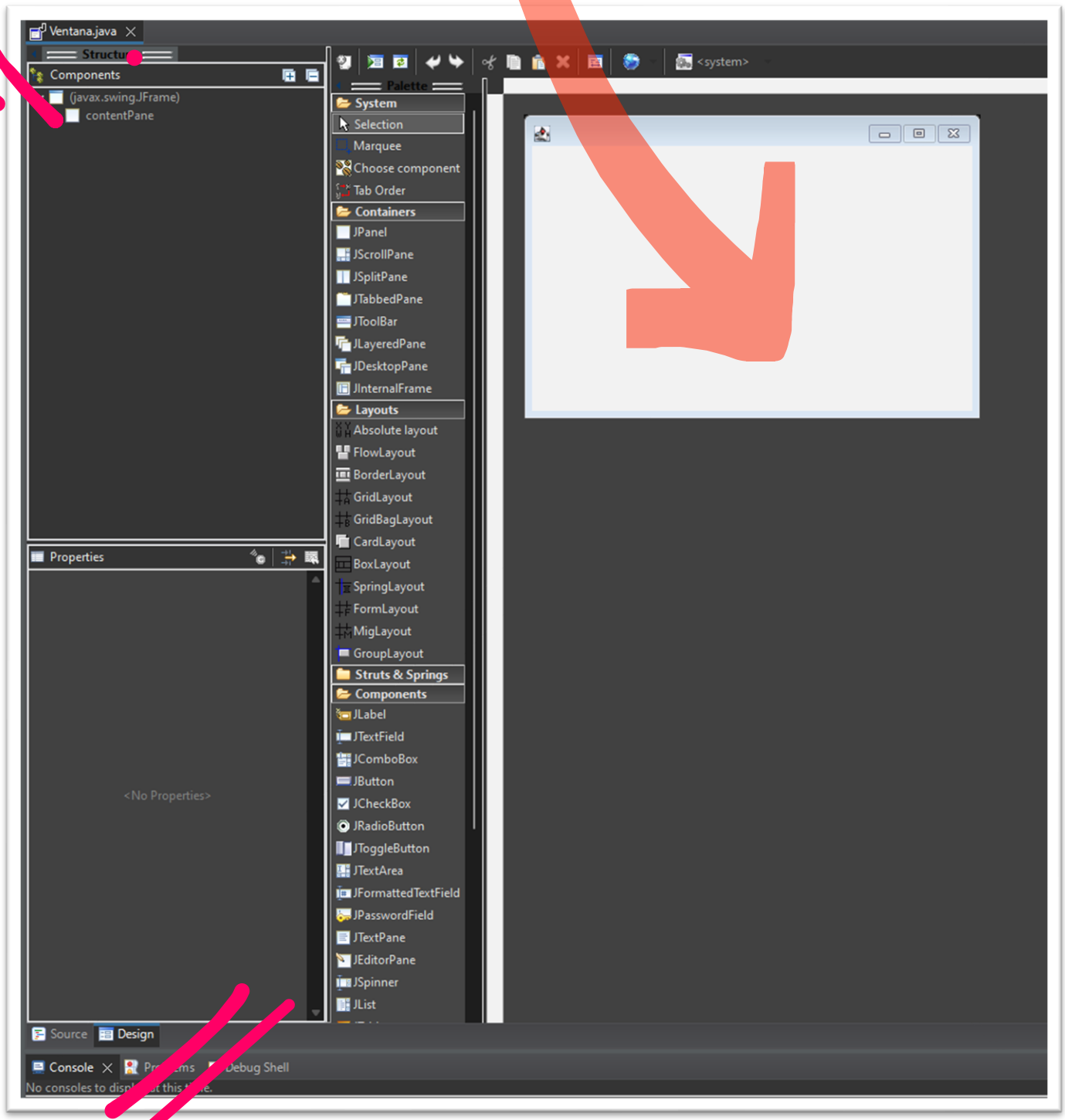
A partir de ahora podremos agregar un nuevo elemeto de Swing a nuestro proyecto

Agrego un JFrame de nombre Ventana

Ventana.java

```java
package prueba_SWING_21;

import java.awt.EventQueue;

public class Ventana extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Ventana frame = new Ventana();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Ventana() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        setContentPane(contentPane);
    }

}
```

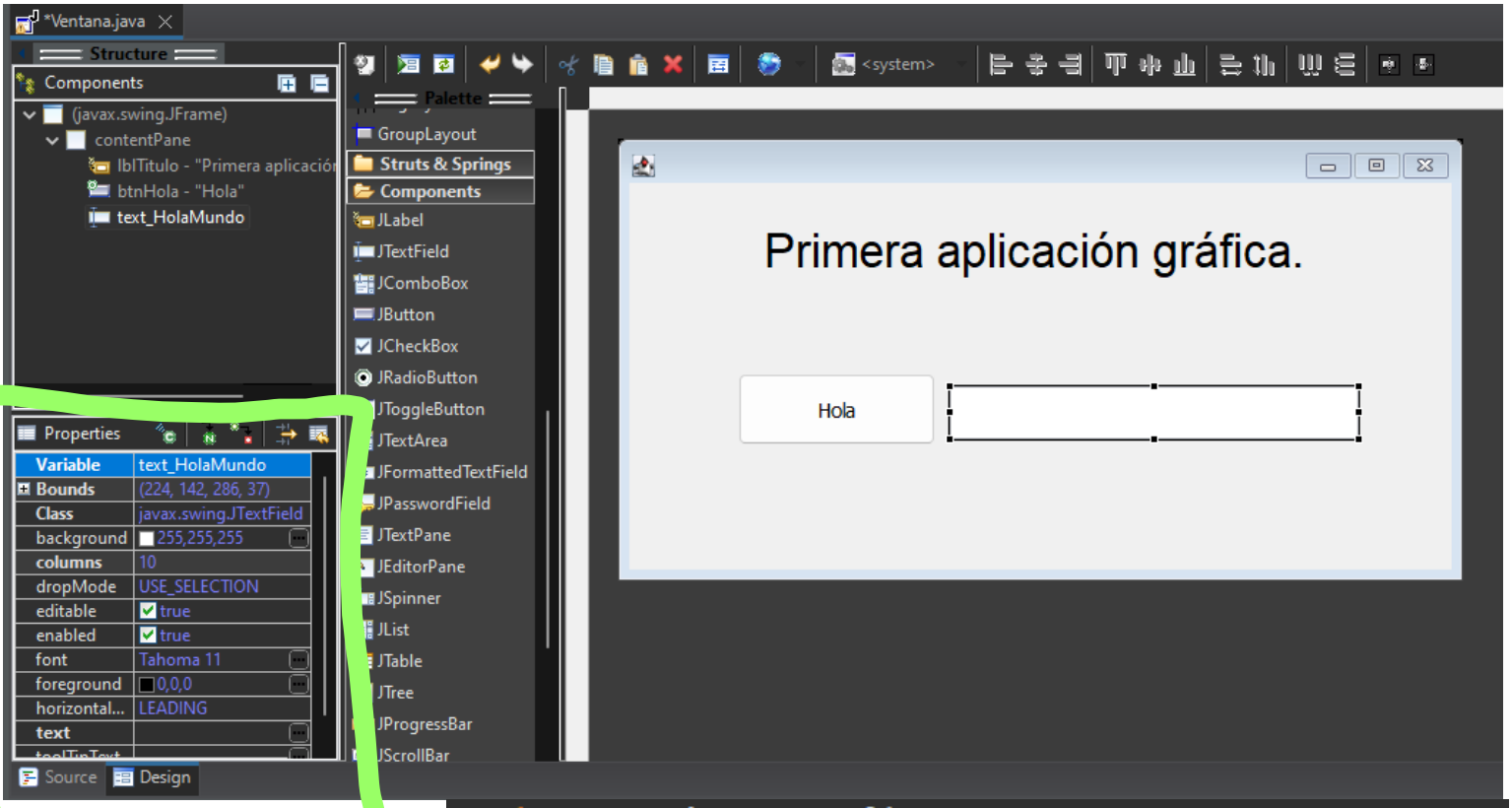
JFrame es una subclase que extiende de Frame



Al crear una clase Ventana, que extiende de JFrame.

Se creará automáticamente un método main donde se creará un objeto de ese tipo, y se seteará como visible.

En el constructor del objeto, creará además un objeto JPanel para agregar nuevos elementos al frame (***cualquier componente visual en swing se inserta dentro de un panel***).

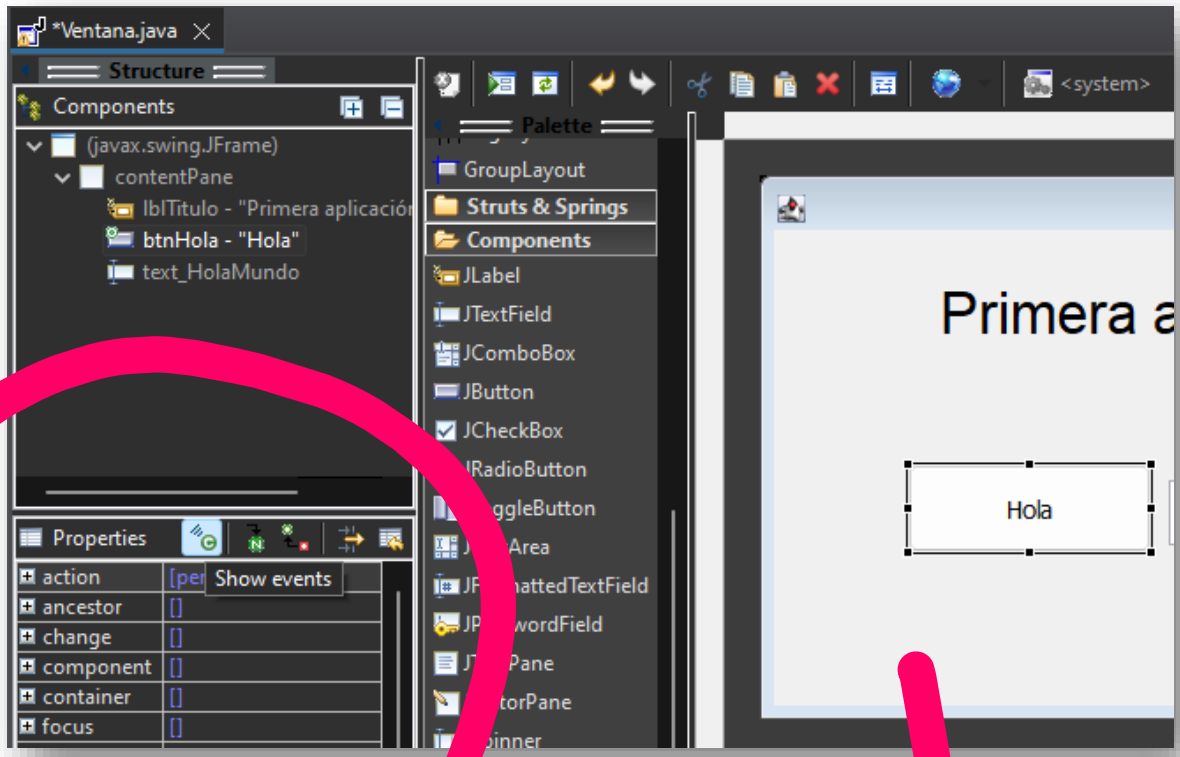



```java
package prueba_SWING_21;

import java.awt.EventQueue;

public class Ventana extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JTextField text_HolaMundo;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Ventana frame = new Ventana();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
```

Siempre que agrego un componente se inicializa en el constructor del frame (y se agregan atributos nuevos a la clase).

```
     * Create the frame.
     */
    public Ventana() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 591, 309);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblTitulo = new JLabel("Primera aplicación gráfica.");
        lblTitulo.setBackground(SystemColor.activeCaption);
        lblTitulo.setHorizontalAlignment(SwingConstants.CENTER);
        lblTitulo.setFont(new Font("Dialog", Font.PLAIN, 33));
        lblTitulo.setBounds(80, 11, 406, 71);
        contentPane.add(lblTitulo);

        JButton btnHola = new JButton("Hola");
        btnHola.setFont(new Font("Tahoma", Font.PLAIN, 14));
        btnHola.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
            }
        });
        btnHola.setBounds(76, 133, 138, 50);
        contentPane.add(btnHola);

        text_HolaMundo = new JTextField();
        text_HolaMundo.setBounds(224, 142, 286, 37);
```

rce Design



@PROFESANTIAGO

Se pueden ver los eventos asociados a cada componente y agregar un listener. El listener tendrá definidos varios métodos que el componente invocará cuando necesite notificar un evento.

mouseClicked() es un método de MouseListener,

actionPerformed() es un método de ActionListener.

Ambos pueden servir para registrar el click en un botón, aunque actionPerformed también notificará en caso de ENTER/SPACE (si el foco está puesto en el botón).

Ejercicio 8

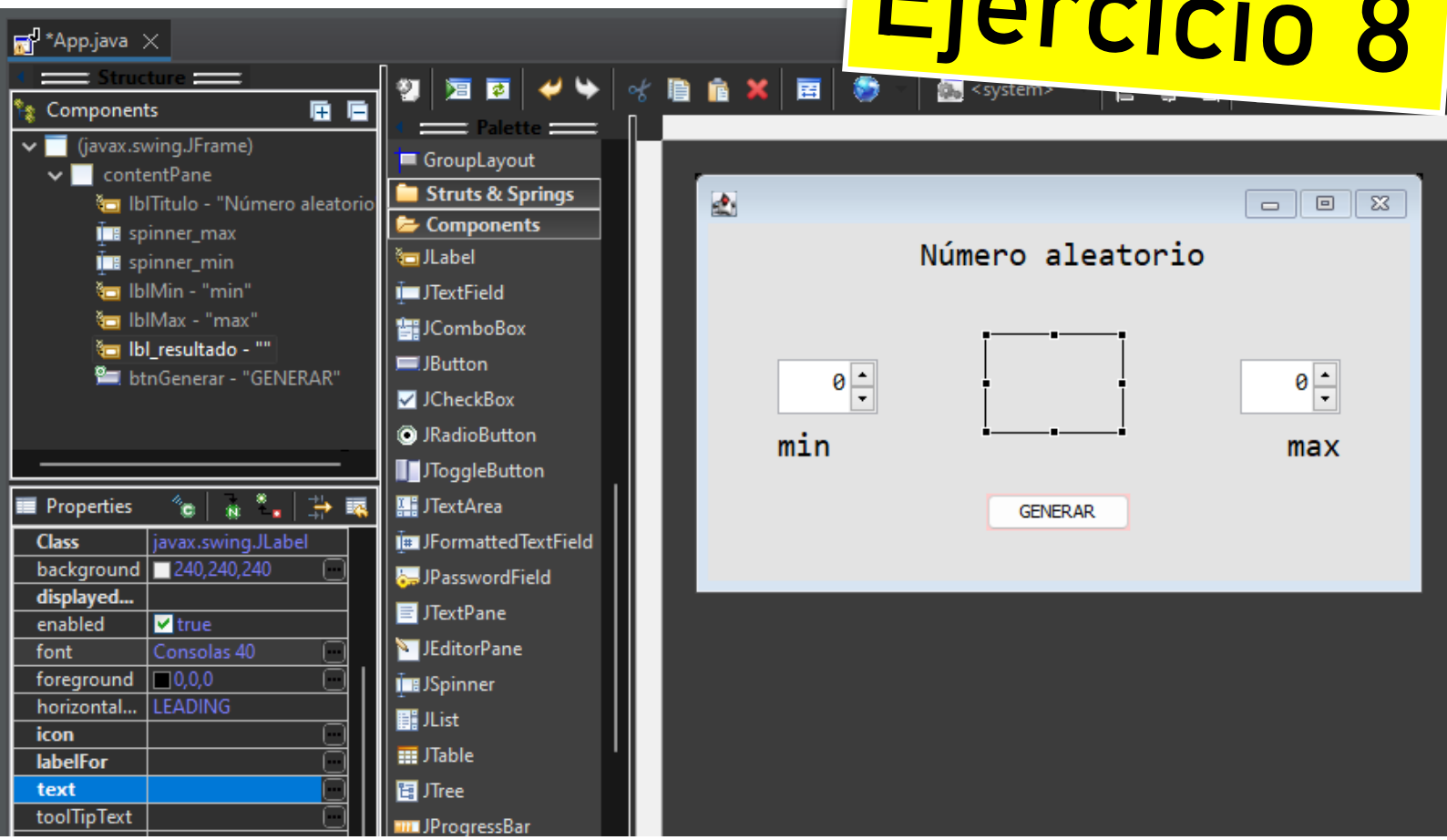
Aplicación gráfica que genere un entero aleatorio entre 2 enteros dados.

Utilizar 2 spinners para seleccionar el rango y un botón para generar el resultado.

```
(int)(Math.random()*(X-Y+1)+Y;
```

@PROFESANTIAGO

Ejercicio 8




```java
        JButton btnGenerar = new JButton("GENERAR");
        btnGenerar.setBackground(new Color(255, 204, 204));
        btnGenerar.setForeground(SystemColor.textText);
        btnGenerar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { //al hacer click en generar
            int min = (int)spinner_min.getValue();
            int max = (int)spinner_max.getValue();
            int aleatorio = (int)(Math.random()*(max-min+1)+min);// entero aleatorio entre 2 valores
            lbl_resultado.setText(String.valueOf(aleatorio));
            }
        });
        btnGenerar.setBounds(172, 167, 89, 23);
        contentPane.add(btnGenerar);
    }
}
```