

TABLA DE CONTENIDO

TABLA DE CONTENIDO	1
INTRODUCCIÓN	4
UNIDADES DE APRENDIZAJE.....	6
ENCAPSULAMIENTO	7
Programación orientada a objetos.....	7
Niveles de acceso	7
Diseño de clases	8
Constructores	9
Métodos	9
Palabra reservada this.....	10
Creación de objetos.....	10
ENCAPSULAMIENTO	17
Clase administradora.....	17
Arreglo de objetos.....	17
Clase ArrayList	17
ENCAPSULAMIENTO	27
Clase Administradora	27
Lista de objetos	27
Clase LinkedList	27
HERENCIA	38
Herencia	38
Jerarquía de Clases.....	38
Extends	39
Super	39

HERENCIA	46
Herencia con archivos de texto.....	46
Sobrecarga de métodos	50
POLIMORFISMO	54
Polimorfismo	54
Casting.....	55
Clases Abstractas.....	55
POLIMORFISMO	60
Colección de objetos polimórficos	60
Instanceof.....	60
POLIMORFISMO	64
Persistencia con archivos de texto	64
INTRODUCCIÓN A BASE DE DATOS.....	68
Acceso a Base de Datos.....	68
Creación de una Base de Datos.....	68
Conexión a una Base de Datos	73
INTRODUCCIÓN A BASE DE DATOS.....	78
Mantenimiento y consultas a una base de datos	78
GUIA DE LABORATORIO 1.....	88
GUIA DE LABORATORIO 2.....	95
GUIA DE LABORATORIO 3.....	103
GUIA DE LABORATORIO 5.....	113
GUIA DE LABORATORIO 6.....	120
GUIA DE LABORATORIO 9.....	126
GUIA DE LABORATORIO 10.....	131
GUIA DE LABORATORIO 12.....	136

GUIA DE LABORATORIO 13.....	140
GUIA DE LABORATORIO 14.....	150
BIBLIOGRAFIA	159
INDICE.....	160

INTRODUCCIÓN

Este modulo está dirigido a los estudiantes de Ingeniería de Sistemas e Informática en la Universidad de Ciencias y Humanidades que cursan el tercer ciclo de estudios. Tiene como objetivo servir como material de estudio principal en el desarrollo del curso de programación I. Su contenido separa la parte teórica de la parte práctica como guías de laboratorio.

El curso forma parte del área de formación especializada, es de carácter teórico práctico y se orienta a capacitar al estudiante para desarrollar sistemas de información empresarial en plataforma desktop utilizando el enfoque de programación orientado a objetos. El contenido está organizado en los siguientes temas generales: Encapsulamiento, Herencia, Polimorfismo, Introducción al uso de bases de datos.

El semestre académico en nuestra Universidad tiene 17 semanas que incluyen evaluaciones y exposiciones de trabajos de investigación.

En la primera semana se diseñan clases con niveles de acceso adecuados y se crean objetos desde una interface gráfica de usuario. Se desarrolla la guía de laboratorio nro. 1

En la segunda semana se diseñan clases administradoras utilizando la clase ArrayList y se utilizan objetos desde una interface gráfica de usuario. Se desarrolla la guía de laboratorio nro. 2

En la tercera semana se diseñan clases administradoras utilizando la clase LinkedList y se utilizan objetos desde una interface gráfica de usuario. Se desarrolla la guía de laboratorio nro. 3

En la cuarta semana hay evaluación como práctica calificada

En la quinta semana se aplica el mecanismo de herencia construyendo jerarquías de clases. Se desarrolla la guía de laboratorio nro. 5

En la sexta semana se aplica el mecanismo de herencia utilizando archivos de texto para la persistencia de los datos. Se desarrolla la guía de laboratorio nro. 6

En la séptima semana hay evaluación como examen parcial

En la octava semana hay exposición de trabajos de investigación

En la novena semana se aplica el mecanismo de polimorfismo con clases abstractas. Se desarrolla la guía de laboratorio nro. 9

En la decima semana se construyen colecciones de objetos polimórficos. Se desarrolla la guía de laboratorio nro. 10

En la decimo primera semana hay evaluación como práctica calificada

En la decimo segunda semana se utiliza archivos de texto para la persistencia de los datos de una colección de objetos polimórficos. Se desarrolla la guía de laboratorio nro. 12

En la decimo tercera semana se crea una base de datos con una tabla utilizando mysql workbench y se establece la conexión desde netbeans. Se desarrolla la guía de laboratorio nro. 13

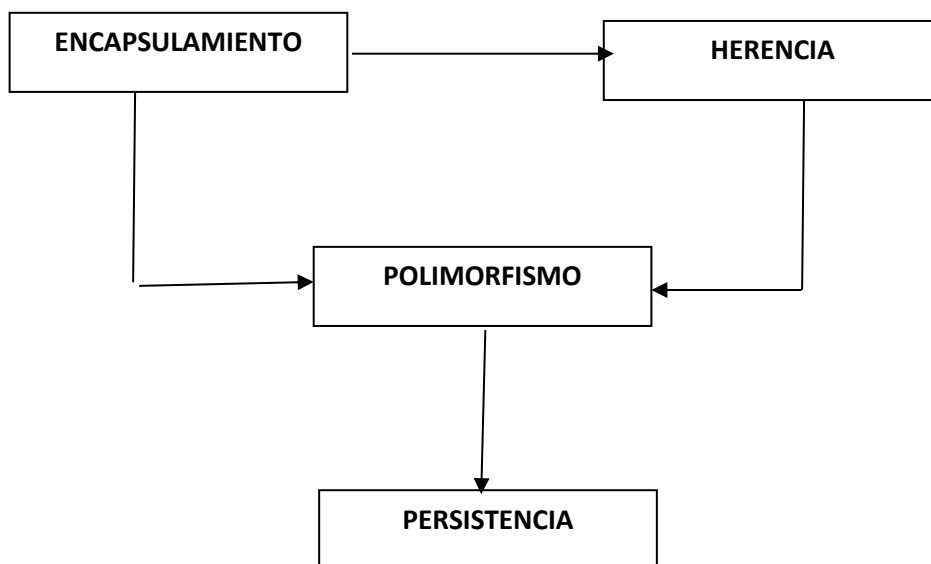
En la decimo cuarta semana se desarrollan métodos para el mantenimiento y consulta de información en la base de datos conectada. Se desarrolla la guía de laboratorio nro. 14

En la decimo quinta semana hay evaluación como examen final

En la decimo sexta semana hay exposiciones de trabajos de investigación

En la decimo séptima semana hay evaluación como examen sustitutorio y entrega de notas.

UNIDADES DE APRENDIZAJE



ENCAPSULAMIENTO

Programación orientada a objetos, niveles de acceso, diseño de clases y creación de objetos.

CAPACIDAD DE PROCESO:

Diseña clases con niveles de acceso adecuados y crea objetos desde una Interface gráfica de usuario (GUI).

Programación orientada a objetos

La programación orientada a objetos es un nuevo enfoque de programación ya aceptado universalmente por la comunidad de programadores. Esta aceptación se sustenta en los múltiples beneficios que trae y los resume en sus siguientes características principales:

Abstracción, a través de la cual definimos y establecemos el contenido de una clase.

Encapsulamiento, a través de la cual establecemos los niveles de acceso a los componentes de una clase. El acceso a los atributos debe ser privado/protegido.

Herencia, a través de la cual se puede extender y/o agregar y/o modificar y/o reutilizar la funcionalidad de una clase existente, llamada padre, en otra clase llamada hija para evitar la redundancia en el código ya desarrollado. Esta característica resume la frase “aprovecha lo que ya está hecho”.

Polimorfismo, a través de la cual se puede programar lo genérico de algún proceso sin saber el detalle de cómo se va a llevar a cabo. Por ejemplo, podemos programar la secuencia genérica de un proceso de cálculo sin saber aún la forma del cálculo que se hará. Esta característica se orienta a la construcción de plantillas de uso general para que, a través de la herencia esas plantillas se puedan particularizar.

Niveles de acceso

Existen 3 niveles de acceso a los componentes de una clase: **private**, **protected**, **public**. El acceso **private** permite la implementación de la característica del

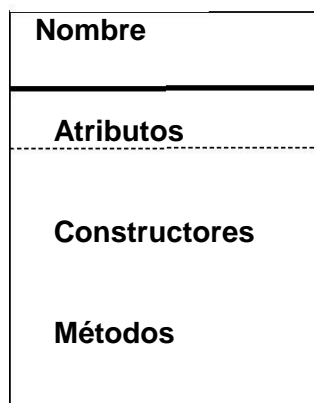
encapsulamiento, el acceso **protected** facilita la implementación de la característica de la herencia y el acceso **public** permite el acceso a la funcionalidad de la clase desde cualquier objeto. Los atributos de una clase deben ser de acceso **private** o **protected**, nunca **public**. Un constructor siempre debe ser de acceso **public**. Los métodos pueden ser de cualquier nivel de acceso, dependiendo de la apertura a la funcionalidad de la clase.

En la siguiente tabla se muestra la posibilidad de acceso a los componentes de una clase dependiendo de su nivel establecido.

Nivel de acceso	Desde la clase	Desde el objeto	Desde la clase hija
Private	SI	NO	NO
Protected	SI	NO	SI
public	SI	SI	SI

Diseño de clases

Una clase, en la práctica es un nuevo tipo de dato particularizado que se compone de: nombre, atributos, constructores y métodos. Los atributos representan los datos y/o características de la clase, los constructores permiten crear objetos de la clase y los métodos representan la funcionalidad de la clase que será utilizada por los objetos.



El nombre de una clase debe empezar con letra mayúscula. Los atributos deben ser de acceso **private**/protegido y los métodos pueden ser de acceso **publico** y/o protegido y/o **private**.

Sintaxis general:

```
public class Nombre{
```

```
    // atributos privados o protegidos
```

```
    // constructores públicos
```

```
    // métodos públicos y/o protegidos y/o privados
```

```
}
```

Constructores

Un constructor es un componente de una clase que tiene el mismo nombre de la clase, de acceso público, a través del cual se puede inicializar los atributos de un objeto creado. Cuando una clase no tiene constructor explícito, se considera un constructor implícito por defecto, sin parámetros y con un bloque de código vacío.

Una clase puede tener varios constructores. Sin embargo, deben diferenciarse por la cantidad y/o tipo de parámetros (sobrecarga). Cuando se crea un objeto, se invoca automáticamente al constructor que corresponda a los parámetros dados. Si no encuentra un constructor adecuado, se produce un error.

Métodos

Los métodos de una clase expresan la funcionalidad con la que dispone una clase para el accionar de sus objetos. Por lo general son de acceso público.

Los métodos necesarios para el acceso de los atributos son conocidos como **get/set** y los demás métodos son conocidos como métodos adicionales que permiten realizar procesos de cálculo, búsqueda, clasificación, ordenamiento, etc.

Los métodos **get** son aquellos que permiten obtener el valor de los atributos. Los métodos **set** permiten modificar el valor de los atributos.

Estos métodos se han estandarizado, por eso se les conoce como los métodos **get/set**.

Incluso, los entornos de programación modernos, como **NetBeans**, generan la escritura del código correspondiente.

Palabra reservada **this**

La palabra reservada **this** se utiliza para diferenciar el nombre del atributo con el nombre del parámetro cuando éstos coinciden.

Creación de objetos

Un objeto, en la práctica es una variable cuyo tipo de dato es una clase. Cuando el objeto es creado, obtiene una copia de los atributos de la clase para uso exclusivo de dicho objeto. Por ésta razón, todos los objetos de una clase tendrán los mismos atributos pero cada uno con valores diferentes.

Sintaxis general:

```
// declara 2 objetos
```

```
Clase objeto1, objeto2;
```

```
// crea el objeto1, invoca al constructor sin parámetros
```

```
objeto1 = new Clase();
```

```
// crea el objeto2, invoca al constructor con parámetros
```

```
objeto2 = new Clase(valor1, valor2, ...);
```

Ejemplo 1

Diseña una clase, de nombre **Producto** con los siguientes atributos privados: código (cadena), descripción (cadena), precio (real) y los métodos públicos get/set. Considere un constructor implícito.

```
public class Producto {
```

```
    // atributos privados
```

```
    private String codigo, descripcion;
```

```
    private double precio;
```

```
    // métodos get
```

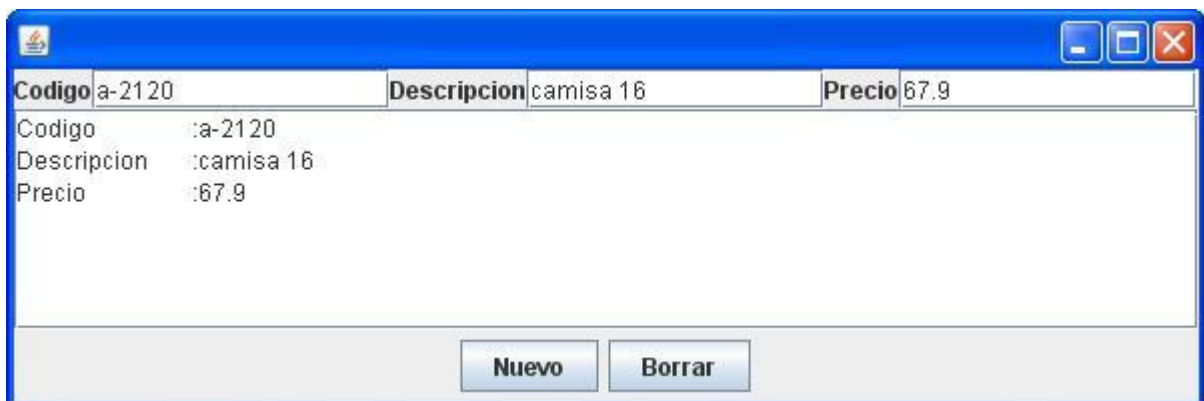
```

public String getCodigo(){return codigo;}
public String getDescripcion(){return descripcion;}
public double getPrecio(){ return precio;}

// métodos set
public void setCodigo(String c){
    codigo =c;
}
public void setDescripcion(String d){
    descripcion=d;
}
public void setPrecio(double p){
    precio=p;
}
}

```

Considere el siguiente diseño de una GUI y programe la acción del botón Nuevo



```

private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt) {
    // crea un objeto nuevo de la clase Producto
    Producto nuevo = new Producto();

    // lee los valores de la GUI y los coloca como atributos del objeto
    nuevo.setCodigo(leeCodigo()); nuevo.setDescripcion(leeDescripcion());
    nuevo.setPrecio(leePrecio());

    // muestra la información del nuevo objeto
}

```

```
        lista(nuevo);
    }

    // métodos que leen los datos de la GUI
    private String leeCodigo(){return txtCodigo.getText();}
    private String leeDescripcion(){return txtDescripcion.getText();}
    private double leePrecio(){return Double.parseDouble(txtPrecio.getText());}

    // método que muestra la información de un objeto de la clase Producto
    private void lista(Producto p){
        imprime("Codigo\t:"+p.getCodigo());
        imprime("Descripcion\t:"+p.getDescripcion());
        imprime("Precio\t:"+p.getPrecio());
    }

    // método que muestra una cadena en el objeto de salida de la GUI
    private void imprime(String s){
        txtSalida.append(s+"\n");
    }
}
```

Ejemplo 2

Diseñe una clase de nombre **Persona** con los siguientes atributos privados: nombres (cadena), apellidos (cadena), edad(entero), peso(real) y los métodos get/set. Considere un constructor explícito con parámetros.

```
public class Persona {
    // atributos privados
    private String nombres, apellidos;
    private int edad;
    private double peso;

    // constructor explícito con parámetros
    public Persona(String nombres, String apellidos, int edad, double peso){
        this.nombres = nombres;
        this.apellidos=apellidos;
    }
}
```

```

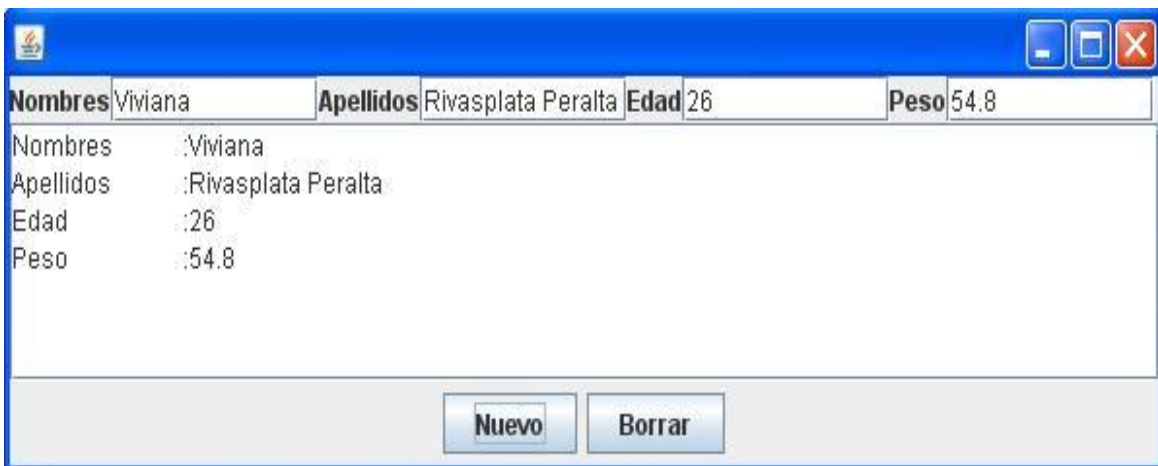
        this.edad = edad;
        this.peso = peso;
    }

    // métodos get
    public String getNombres(){return nombres;}
    public String getApellidos(){return apellidos;}
    public int getEdad(){return edad;}
    public double getPeso(){ return peso;}

    // métodos set
    public void setNombres(String nombres){
        this.nombres = nombres;
    }
    public void setApellidos(String apellidos){
        this.apellidos = apellidos;
    }
    public void setEdad(int edad){
        this.edad = edad;
    }
    public void setPeso(double peso){
        this.peso = peso;
    }
}

```

Considere el siguiente diseño de una GUI y programe la acción del botón Nuevo



Nombres	Apellidos	Edad	Peso
Viviana	Rivasplata Peralta	26	54.8

Nombres : Viviana
 Apellidos : Rivasplata Peralta
 Edad : 26
 Peso : 54.8

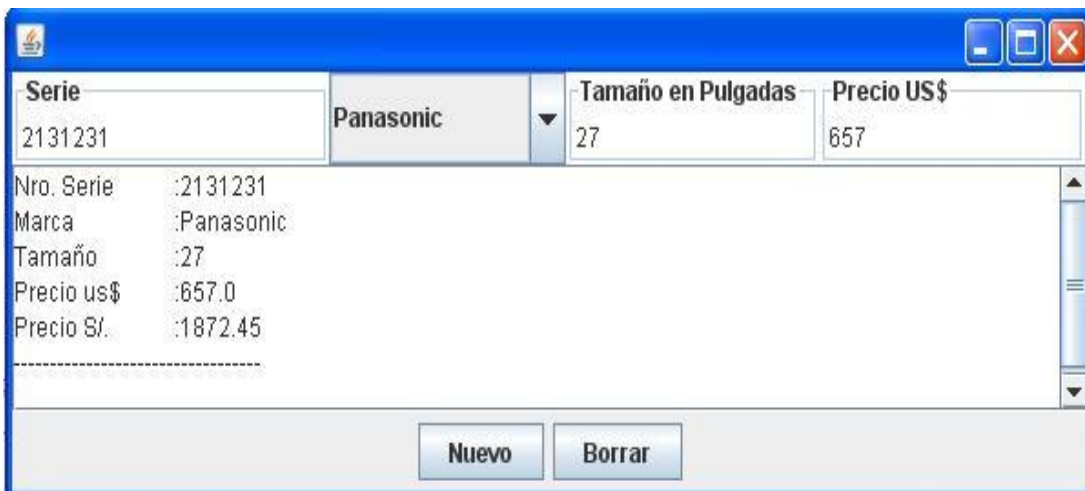
```
private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt) {  
    // crea un objeto nuevo de la clase Persona  
    Persona nuevo = new Persona(leeNombres(), leeApellidos(),  
                                leeEdad(), leePeso());  
    // muestra la información del nuevo objeto  
    lista(nuevo);  
}  
  
// métodos que leen los datos de la GUI  
private String leeNombres(){return txtNombres.getText();}  
private String leeApellidos(){return txtApellidos.getText();}  
private int leeEdad(){return Integer.parseInt(txtEdad.getText());}  
private double leePeso(){return Double.parseDouble(txtPeso.getText());}  
  
// método que muestra la información de un objeto de la clase Producto  
private void lista(Persona p){  
    imprime("Nombres\t:"+p.getNombres());  
    imprime("Apellidos\t:"+p.getApellidos());  
    imprime("Edad\t:"+p.getEdad());  
    imprime("Peso\t:"+p.getPeso());  
}  
  
// método que muestra una cadena en el objeto de salida de la GUI  
private void imprime(String s){  
    txtSalida.append(s+"\n");  
}
```

- Observe la utilidad y el funcionamiento del constructor explícito
- Una clase siempre va a considerar un constructor implícito?
- Puede una clase considerar varios constructores?
- Observe cuando se usa la palabra reservada this
- Observe cómo se genera el código para los métodos get/set

Ejemplo 3

Diseñe una clase de nombre **TV** con los siguientes atributos privados: serie (cadena), marca (entero), tamaño en pulgadas(entero), precio (real) y con los métodos get/set, con un método adicional que devuelve el precio en soles dado el tipo de cambio como parámetro, y con un método adicional que retorne el nombre de la marca. Considere las siguientes marcas: Sony, LG, Samsung, Panasonic, otro. Considere un constructor implícito.

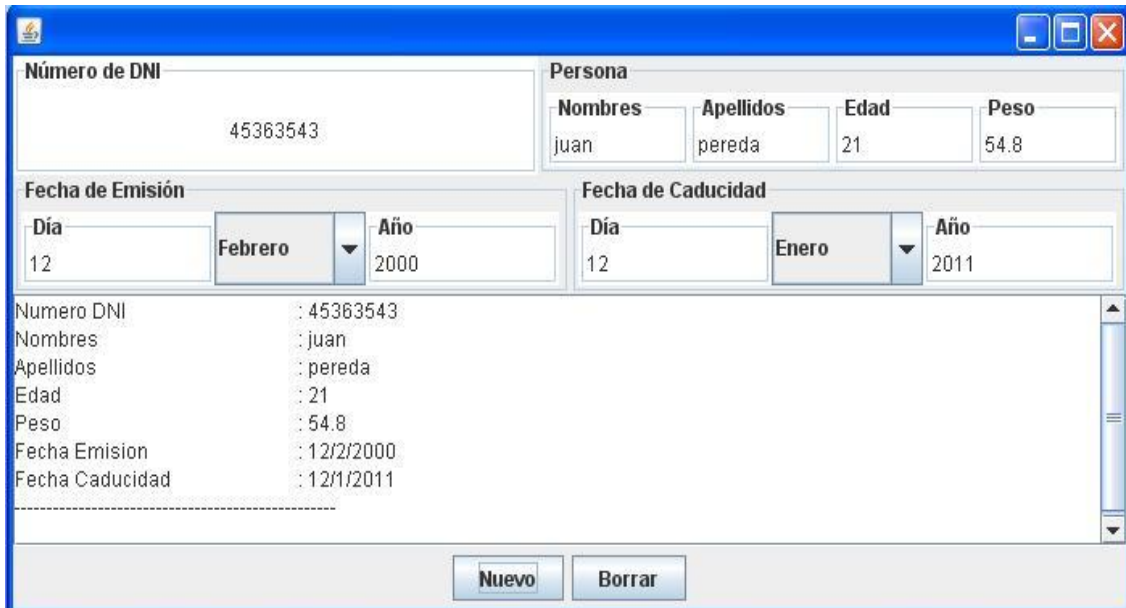
Considere el siguiente diseño de una GUI y programe la acción del botón Nuevo



Ejemplo 4

Diseñe una clase de nombre **Fecha** con los siguientes atributos privados: dia(entero), mes(entero), año(entero) y con los métodos get/set. Considere un constructor explícito. Diseñe otra clase de nombre **DNI** con los siguientes atributos privados: numero(cadena), dueño(Persona), fecha de emisión (Fecha), fecha de caducidad(Fecha) y con los métodos get/set. Considere un constructor explícito.

Considere el siguiente diseño de una GUI y programe la acción del botón Nuevo.



Número de DNI		Persona			
45363543		Nombres	Apellidos	Edad	Peso
		juan	pereda	21	54.8
Fecha de Emisión			Fecha de Caducidad		
Día	Febrero	Año	Día	Enero	Año
12		2000	12		2011
Numero DNI : 45363543 Nombres : juan Apellidos : pereda Edad : 21 Peso : 54.8 Fecha Emision : 12/2/2000 Fecha Caducidad : 12/1/2011					
<div>Nuevo</div> <div>Borrar</div>					

Ejemplo 5

Diseñe una clase de nombre **Punto** con los siguientes atributos privados: x (entero), y (entero), con un constructor explícito y con los métodos get/set.

Diseñe una clase de nombre **Triangulo** con los siguientes atributos privados: a(**Punto**), b(**Punto**), c(**Punto**), con un constructor implícito y con los métodos get/set además de los métodos de cálculo para los lados del triángulo, para el perímetro del triángulo y para el área del triángulo.

Considere el diseño de una GUI para programar la acción del botón nuevo.

ENCAPSULAMIENTO

Clases administradoras con arreglo de objetos, clase ArrayList

CAPACIDAD DE PROCESO:

- Diseña clases administradoras utilizando la clase ArrayList
- Desarrolla métodos de procesos específicos con información del arreglo de objetos
- Desarrolla aplicaciones, con interface gráfica de usuarios, con objetos pertenecientes a clases administradoras

Clase administradora

Una clase administradora es aquella que tiene todo lo necesario para administrar información de cualquier tipo, considerando los atributos y funcionalidad necesarios para ello.

Arreglo de objetos

Un arreglo de objetos es aquel cuyos elementos que componen el arreglo son objetos. Es decir, cada casillero del arreglo es un objeto. Sin embargo, la posición de los casilleros siguen siendo enteros consecutivos a partir de 0.

Clase ArrayList

La clase **ArrayList** es una clase que administra un arreglo de objetos de cualquier tipo. No tiene restricciones de capacidad. Su tamaño se ajusta en forma dinámica utilizando eficientemente la memoria del computador.

- Tiene un Constructor por defecto que define un tamaño inicial de 10. Va creciendo de 10 en 10.
- Tiene varios constructores explícitos sin parámetros y con parámetros donde recibe el tamaño inicial del arreglo que usted quiera establecer.
- Los elementos dentro de un **ArrayList** son **Objetos** de cualquier tipo. Para un

uso adecuado se recomienda particularizarlo al objeto que se quiera administrar.

- La clase **ArrayList** forma parte del paquete **java.util**

Métodos de uso frecuente:

Método	Interpretación
int size()	Retorna el número de objetos guardados.
void add (Object obj)	Añade un objeto al final del arreglo, incrementando su tamaño actual en 1.
Object get (int index)	Devuelve el objeto almacenado a la posición index en el arreglo. index tiene que ser un entero entre 0 y size()-1 .
void set (int index, Object obj)	reemplaza el objeto de la posición index por el objeto obj.
Object remove (int index)	Elimina el objeto a la posición index (index entre 0 y size()-1). Devuelve el objeto eliminado. Los objetos después de este objeto se trasladan a una posición anterior. El tamaño del ArrayList disminuye en 1.
int indexOf (Object obj)	Busca el objeto obj dentro del arreglo, si lo encuentra devuelve la posición donde lo ha encontrado. Si no lo encuentra, devuelve -1.

Ejemplo 1:

Diseña una clase administradora para un arreglo de objetos de tipo Producto considerando la siguiente GUI:



Diseñe la clase **Producto** con constructor explícito y genere los métodos get-set:

```
public class Producto {  
    // atributos privados  
    private String codigo, descripcion;  
    private double precio;  
  
    // constructor  
    public Producto(String codigo, String descripcion, double precio){  
        this.codigo=codigo;  
        this.descripcion=descripcion;  
        this.precio=precio;  
    }  
    // métodos get-set  
}
```

Diseñe la clase **ArregloProductos**, que tenga como atributo privado un objeto de la clase **ArrayList** donde se guardarán los objetos y un constructor explícito sin parámetros.

```
public class ArregloProductos {  
    // atributos  
    // objeto ArrayList particularizado para objetos de tipo Producto  
    private ArrayList <Producto> a ;  
  
    // constructor explícito  
    public ArregloProductos(){  
        a = new ArrayList<Producto>();  
    }  
    // métodos de administración  
    public int tamaño(){ return a.size(); }  
    public void agrega(Producto p){  
        a.add(p);  
    }  
}
```

```
public Producto obtiene(int i){ // retorna un producto de la posición i
    return a.get(i);
}
```

```
public void actualiza(Producto p, int i){
    // reemplaza un producto
    a.set(i, p);
}

// retorna la posición de un producto según su código.
public int busca(String codigo){
    for(int i=0; i<tamaño(); i++){
        if(obtiene(i).getCodigo().equals(codigo))
            return i;
    }
    return -1; // no lo encontró
}
```

```
public void elimina(int p){// elimina el producto de la posición p
    a.remove(p);
}
```

```
public double mayorPrecio(){// retorna el mayor precio
    double m=a.get(0).getPrecio();
    for(Producto p: a){ // for each: por cada producto p en a
        if(p.getPrecio() > m)
            m =p.getPrecio();
    }
    return m;
}
```

```
public double menorPrecio(){// retorna el menor precio
    double m=a.get(0).getPrecio();
    for(Producto p: a){ // for each: por cada producto p en a
```

```
        if(p.getPrecio() < m)
            m =p.getPrecio();
    }
    return m;
}

public double precioPromedio(){// retorna el precio promedio
    double suma=0;
    for(Producto p: a){ // for each: por cada producto p en a
        suma += p.getPrecio();
    }
    return suma / tamaño();
}

} // fin de la clase ArregloProductos
```

Ahora en la clase de la GUI considere como atributo un objeto de tipo ArregloProductos y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo producto evitando que se repita el código
- El botón Busca ubica un producto según el código ingresado y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos del producto ubicado con el botón Busca.
- El botón Elimina saca del arreglo el producto según el código ingresado.
- El botón Lista muestra una relación de todos los productos guardados en el arreglo.
- El botón Reporte muestra los precios mayor, menor y promedio.
- El botón Borrar limpia todo el contenido del GUI.

Ejemplo 2:

Diseñe una clase administradora con soporte para un arreglo de objetos tipo Persona considerando la siguiente GUI:



Diseñe la clase **Persona** con un constructor explícito:

```
public class Persona {
    // atributos privados
    private String dni, nombres, apellidos;
    private int edad;
    private double peso;
    // constructor explícito
    public Persona(String dni, String nombres, String apellidos,
                    int edad, double peso){
        this.dni=dni;
        this.nombres=nombres;
        this.apellidos=apellidos;
        this.edad=edad;
        this.peso=peso;
    }
    // métodos get-set
}
```

Diseñe la clase administradora **ArregloPersonas**, que tenga como atributo privado un objeto de la clase **ArrayList** particularizado para la clase **Persona**, donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- a) Un método que retorne el peso mayor de todas las personas.
- b) Un método que retorne el peso menor de todas las personas.

- c) Un método que retorne el peso promedio de todas las personas.
- d) Un método que retorne la cantidad de personas cuyo peso se encuentra en un rango dado como parámetros
- e) Un método que retorne la cantidad de personas mayores de edad.
- f) Un método que retorne la cantidad de personas menores de edad.

En la clase de la GUI considere como atributo un objeto de tipo `ArregloPersonas` y programe la acción de los botones de acuerdo a lo siguiente:

- El botón **Nuevo** permite registrar una nueva persona evitando que se repita el dni
- El botón **Busca** ubica una persona según el dni ingresado y muestra sus datos donde corresponda.
- El botón **Modifica** reemplaza los datos de la persona ubicada con el botón **Busca**.
- El botón **Elimina** saca del arreglo la persona según el dni ingresado.
- El botón **Lista** muestra una relación de todas las personas guardadas en el arreglo.
- El botón **Reporte** muestra el resultado de todos los métodos adicionales.
- El botón **Borrar** limpia todo el contenido del GUI.

Ejemplo 3

Diseñe una clase administradora con soporte para un arreglo de objetos tipo `TV`. Considere una GUI adecuada para los datos de la clase `TV` que se indica a continuación.

Diseñe la clase `TV` con los siguientes atributos: `serie` (cadena), `marca` (entero), `tamaño` en pulgadas(entero), `precio` en dólares (real). Considere un constructor explícito y sus métodos `get/set`. Considere un método adicional que devuelve el precio en soles dado el tipo de cambio como parámetro, y un método adicional que retorne el nombre de la marca. Considere las siguientes marcas: Sony, LG, Samsung, Panasonic, otro.

Diseñela clase administradora **`ArregloTV`**, que tenga como atributo privado un objeto de la clase **`ArrayList`** particularizado para la clase `TV` donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- a) Un método que retorne la cantidad de televisores que pertenecen a una marca específica dada como parámetro.
- b) Un método que retorne el precio promedio de los televisores de una marca específica dada como parámetro.
- c) Un método que modifique el precio de todos los televisores incrementándolo en un porcentaje dado como parámetro.

En la clase de la GUI considere como atributo un objeto de tipo ArregloTV y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo televisor evitando que se repita la serie.
- El botón Busca ubica un televisor según la serie ingresada y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos del televisor ubicado con el botón Busca.
- El botón Elimina saca del arreglo el televisor según la serie ingresada.
- El botón Lista muestra una relación de todos los televisores guardados en el arreglo.
- El botón Reporte muestra el resultado de todos los métodos adicionales.
- El botón AumentaPrecio aplica el resultado del método c) y muestra el resultado del botón Lista.
- El botón Borrar limpia todo el contenido del GUI.

Ejemplo 4

Modifique el contenido de la clase **ArregloProductos**, para que considere los siguientes métodos adicionales:

- a) Incremente el precio de todos los productos en 15%.
- b) Incremente el precio de los productos cuyo precio actual sea inferior a un valor dado como parámetro.

- c) Disminuya el precio de los productos cuyo precio actual esté en un rango dado como parámetro.
- d) Retorne la cantidad de productos cuyo precio sea inferior al precio promedio.
- e) Retorne en un arreglo los productos cuya descripción empiece con una letra dada como parámetro.

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para llamar a los métodos adicionales desarrollados.

Ejemplo 5

Modifique el contenido de la clase **ArregloPersonas**, para que considere los siguientes métodos adicionales:

- a) Incremente el peso de todas las personas en 5%
- b) Incremente el peso de las personas cuyo peso actual sea inferior a un valor dado como parámetro.
- c) Disminuya el peso de las personas cuyo peso actual esté en un rango dado como parámetro.
- d) Retorne la cantidad de personas menores de edad
- e) Retorne en un arreglo las personas cuya edad sea superior a un valor dado como parámetro.

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para llamar a los métodos adicionales desarrollados.

Ejemplo 6

Modifique el contenido de la clase **ArregloTV**, que considere los siguientes métodos adicionales:

- a) Incremente el precio de todos los televisores en 8%

- b) Incremente el precio de los televisores de una marca dada como parámetro en un porcentaje también dado como parámetro
- c) Disminuya el precio de los televisores cuyo tamaño actual esté en un rango dado como parámetro
- d) Retorne la cantidad de televisores de una marca y tamaño dado como parámetros.
- e) Retorne en un arreglo los televisores de una marca dada como parámetro.

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para llamar a los métodos adicionales desarrollados.

ENCAPSULAMIENTO

Clases administradoras con listas de objetos, clase LinkedList

CAPACIDAD DE PROCESO:

- Diseña clases administradoras utilizando la clase LinkedList.
- Desarrolla aplicaciones, con interface gráfica de usuarios, con objetos pertenecientes a clases administradoras

Clase Administradora

Una clase administradora es aquella que tiene todo lo necesario para administrar información de cualquier tipo, considerando los atributos y funcionalidad necesarios para ello. Administrar información implica registrar un nuevo elemento, buscarlo, obtenerlo, modificarlo, eliminarlo y saber cuántos elementos tiene.

Lista de objetos

Una lista de objetos es aquella cuyos elementos que la componen son objetos que están vinculados a través de un enlace. A diferencia de los arreglos, una lista no tiene casilleros ni se dimensiona, van creciendo o decreciendo de uno en uno.

Clase LinkedList

La clase **LinkedList** es una clase que administra una lista de objetos de cualquier tipo. Sin embargo, para un uso adecuado se recomienda particularizarlo al objeto que se quiera administrar. Utiliza eficientemente la memoria del computador ajustado estrictamente a la cantidad de objetos que lo contenga.

Dispone de métodos para obtener, remover, insertar objetos al principio, al final, después de algún objeto, antes de algún objeto, etc.

Cuando una lista sólo utiliza **addFirst** para agregar objetos, se le conoce como lista tipo **pila**. Cuando sólo utiliza **addLast** se le conoce como lista tipo **cola** y cuando utiliza indistintamente **addFirst**, **addLast**, **add** se le conoce como lista general.

Métodos de uso frecuente:

Método	Interpretación
void addFirst (Object info)	agrega un nuevo objeto info al inicio de la lista.
void addLast (Object info)	agrega un nuevo objeto info al final de la lista.
void add (int posición, Object info)	agrega un nuevo objeto en la posición entera que se indique.
Object get (int posición)	retorna el objeto de la posición indicada.
Object removeFirst ()	elimina el objeto del inicio de la lista.
Object removeLast ()	elimina el objeto del final de la lista.
Object remove (int posición)	elimina el objeto de la posición indicada.
Object remove (Object info)	elimina el objeto que se indica.
int size ()	retorna la cantidad de objetos de la lista.
void clear ()	elimina todos los objetos de la lista.
int indexOf (Object info)	retorna la posición del objeto que se indica.

Ejemplo 1:

Diseña una clase administradora para una lista tipo pila de objetos de tipo Producto considerando la siguiente GUI:



Diseña la clase **Producto** con constructor explícito y genere los métodos get-set:

```

public class Producto {
    // atributos privados
    
```

```
private String codigo, descripcion;
private double precio;
// constructor
public Producto(String codigo, String descripcion, double precio){
    this.codigo=codigo;
    this.descripcion=descripcion;
    this.precio=precio;
}
// métodos get-set
}
```

Diseñe la clase **ListaPilaProductos**, que tenga como atributo privado un objeto de la clase **LinkedList** donde se guardarán los objetos y un constructor explícito sin parámetros.

```
public class ListaPilaProductos {
    // atributos
    // objeto LinkedList particularizado para objetos de tipo Producto
    private LinkedList <Producto> pila ;

    // constructor explícito
    public ListaProductos(){
        pila = new LinkedList<Producto>();
    }

    // métodos de administración
    public int tamaño(){ return pila.size(); }
    public void agrega(Producto p){
        pila.addFirst(p);
    }
    public Producto obtiene(int i){ // retorna un producto de la posición i
        return pila.get(i);
    }
    // retorna la posición de un producto según su código.
    public int busca(String codigo){
```

```
        for(int i=0; i<tamaño(); i++){
            if(obtiene(i).getCodigo().equals(codigo))
                return i;
        }
        return -1; // no lo encontró
    }

    public void elimina(){// elimina el primer producto de la pila
        pila.removeFirst();
    }

    public double mayorPrecio(){// retorna el mayor precio
        double m=pila.get(0).getPrecio();
        for(Producto p: pila){ // for each: por cada producto p en pila
            if(p.getPrecio() > m)
                m =p.getPrecio();
        }
        return m;
    }

    public double menorPrecio(){// retorna el menor precio
        double m=pila.get(0).getPrecio();
        for(Producto p: pila){ // for each: por cada producto p en pila
            if(p.getPrecio() < m)
                m =p.getPrecio();
        }
        return m;
    }

    public double precioPromedio(){// retorna el precio promedio
        double suma=0;
        for(Producto p: pila){ // for each: por cada producto p en pila
            suma += p.getPrecio();
        }
        return suma / tamaño();
    }
} // fin de la clase ListaPilaProductos
```

Ahora en la clase de la GUI considere como atributo un objeto de tipo ListaPilaProductos y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo producto al principio de la lista, evitando que se repita el código
- El botón Busca ubica un producto según el código ingresado y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos del producto ubicado con el botón Busca.
- El botón Elimina saca de la lista el primer producto.
- El botón Lista muestra una relación de todos los productos guardados en la lista.
- El botón Reporte muestra los precios mayor, menor y promedio.
- El botón Borrar limpia todo el contenido del GUI.

Ejemplo 2:

Diseñe una clase administradora con soporte para una lista tipo cola de objetos tipo Persona considerando la siguiente GUI:



Diseñe la clase **Persona** con un constructor explícito:

```

public class Persona {
    // atributos privados
    private String dni, nombres, apellidos;
    private int edad;
    private double peso;
    // constructor explícito
    public Persona(String dni, String nombres, String apellidos,
                    int edad, double peso){
        this.dni=dni;
    }
}
    
```

```
        this.nombres=nombres;
        this.apellidos=apellidos;
        this.edad=edad;
        this.peso=peso;
    }
    // métodos get-set
}
```

Diseñe la clase administradora **ListaColaPersonas**, que tenga como atributo privado un objeto de la clase **LinkedList** particularizado para la clase **Persona**, donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- Un método que retorne el peso mayor de todas las personas.
- Un método que retorne el peso menor de todas las personas.
- Un método que retorne el peso promedio de todas las personas.
- Un método que retorne la cantidad de personas cuyo peso se encuentra en un rango dado como parámetros
- Un método que retorne la cantidad de personas mayores de edad.
- Un método que retorne la cantidad de personas menores de edad.

En la clase de la GUI considere como atributo un objeto de tipo **ListaColaPersonas** y programe la acción de los botones de acuerdo a lo siguiente:

- El botón **Nuevo** permite registrar una nueva persona al final de la lista, evitando que se repita el dni
- El botón **Busca** ubica una persona según el dni ingresado y muestra sus datos donde corresponda.
- El botón **Modifica** reemplaza los datos de la persona ubicada con el botón **Busca**.
- El botón **Elimina** saca de la cola la primera persona.
- El botón **Lista** muestra una relación de todas las personas guardadas en la lista.
- El botón **Reporte** muestra el resultado de todos los métodos adicionales.
- El botón **Borrar** limpia todo el contenido del GUI.

Ejemplo 3

Diseñe la clase **Empleado** con los siguientes atributos: código(cadena), nombre(cadena), sueldo(real).

Diseñe la clase **ListaDobleEmpleados** con los siguientes atributos: un objeto lista de tipo **LinkedList**. También considere los siguientes métodos de administración: agregaAlInicio, agregaAlFinal(), elimina(), busca(), obtiene(), tamaño().

```
public class ListaDobleEmpleados{
    private LinkedList<Empleado> lista;

    public ListaDobleEmpleados(){
        lista = new LinkedList<Empleado>();
    }

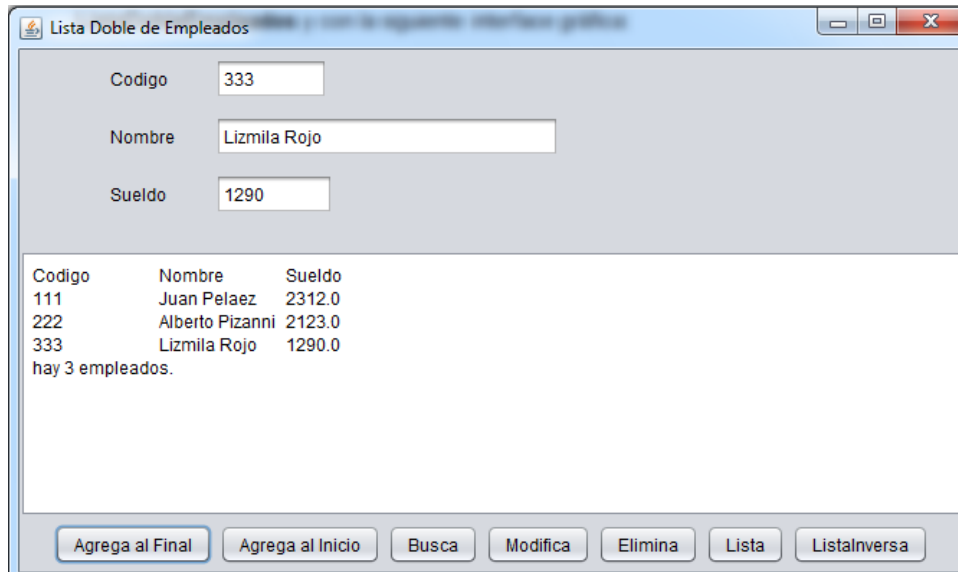
    // métodos de administración
    public int tamaño(){ return lista.size(); }
    public Empleado obtiene(int i){ return lista.get(i);}

    public void agregaAlFinal(Empleado e){
        lista.addLast(e);
    }
    public int busca(String codigo){
        for(Empleado e: lista){
            if(e.getCodigo().equalsIgnoreCase(codigo))
                return lista.indexOf(e);
        }
        return -1;
    }
    public void elimina(int n){
        lista.remove(n);
    }
    public void elimina(Empleado e){
        lista.remove(e);
    }

    public LinkedList<Empleado> getList() {
        return lista;
    }

    public void setLista(LinkedList<Empleado> lista) {
        this.lista = lista;
    }
}
```

Diseñe la clase **PanelEmpleados** con el siguiente atributo: un objeto **Ide** de tipo **ListaDobleEmpleados** y con la siguiente interface gráfica:



Codigo	Nombre	Sueldo
111	Juan Pelaez	2312.0
222	Alberto Pizanni	2123.0
333	Lizmila Rojo	1290.0

hay 3 empleados.

```

public class PanelPrincipal extends javax.swing.JPanel {
    protected ListaDobleEmpleados lde;
    /** Creates new form PanelPrincipal */
    public PanelPrincipal() {
        initComponents();
        lde = new ListaDobleEmpleados();
    }
}
    
```

Programe la acción de los botones.

```

private void btnAgregaFinalActionPerformed(java.awt.event.ActionEvent evt) {
    int p = lde.busca(leeCodigo());
    if(p!=-1){
        Empleado n = new Empleado(leeCodigo(), leeNombre(), leeSueldo());
        lde.agregaAlFinal(n);
        lista();
    }else
        mensaje("Codigo repetido!");
}
    
```

```

private void btnEliminaActionPerformed(java.awt.event.ActionEvent evt) {
    int p = lde.busca(leeCodigo());
    if(p!=-1)
        mensaje("Codigo no registrado!");
    else{
        lde.elimina(p);
        lista();
    }
}
    
```

```

public void lista() {
    txtSalida.setText("Codigo\tNombre\t\tSueldo\n");
    for (Empleado aux: lde.getList()) {
        imprime(aux.getCodigo()+"\t"+
                aux.getNombre()+"\t\t"+
                aux.getSueldo());
    }
    imprime("Hay "+lde.getNodos()+" empleados.");
}

public void imprime(String s) {
    txtSalida.append(s+"\n");
}

public void mensaje(String s) {
    JOptionPane.showMessageDialog(this, s);
}

private void btnListaInversaActionPerformed(java....){
    for (int i=lde.tamaño()-1 ; i>=0; i--){
        Empleado aux = lde.obtiene(i);
        Imprime(aux.getCodigo()+"\t" +
                aux.getNombre()+"\t\t" +
                aux.getSueldo() );
    }
    Imprime("Hay " + lde.tamaño() + " empleados. ");
}

```

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejemplo 4

Diseñe la clase TV con los siguientes atributos: serie (cadena), marca (entero), tamaño en pulgadas (entero), precio en dólares (real). Considere un constructor explícito y sus métodos get/set. Considere un método adicional que devuelve el precio en soles dado el tipo de cambio como parámetro, y un método adicional que retorne el nombre de la marca. Considere las siguientes marcas: Sony, LG, Samsung, Panasonic, otro.

Diseñela clase administradora **ListaPilaTV**, que tenga como atributo privado un objeto de la clase **LinkedList** particularizado para la clase TV donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- a) Un método que retorne la cantidad de televisores que pertenecen a una marca específica dada como parámetro.
- b) Un método que retorne el precio promedio de los televisores de una marca específica dada como parámetro.
- c) Un método que modifique el precio de todos los televisores incrementándolo en un porcentaje dado como parámetro.

En la clase de la GUI considere como atributo un objeto de tipo ListaPilaTV y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo televisor evitando que se repita la serie.
- El botón Busca ubica un televisor según la serie ingresada y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos del televisor ubicado con el botón Busca.
- El botón Elimina saca al primer televisor de la lista.
- El botón Lista muestra una relación de todos los televisores guardados en la lista.
- El botón Reporte muestra el resultado de todos los métodos adicionales.
- El botón AumentaPrecio aplica el resultado del método c) y muestra el resultado del botón Lista.
- El botón Borrar limpia todo el contenido del GUI.

EVALUACIÓN

HERENCIA

Herencia. Jerarquía de Clases, extends, super

CAPACIDAD EN PROCESO:

- Aplica el mecanismo de herencia construyendo una jerarquía de clases.
- Desarrolla métodos de procesos específicos con información de una lista dinámica de objetos.

Herencia

La Herencia es una de las características más importantes y beneficiosas para el enfoque de programación orientado a objetos. En resumen se trata de “utilizar lo que ya está hecho” para agregarle y/o cambiarle funcionalidad, nunca quitarle. Esto en la vida real tiene mucho sentido. Por ejemplo, Existiendo el televisor en blanco y negro (clase Padre) fabricaron el televisor a color (clase Hija) aprovechando todo lo bueno del televisor en blanco y negro ya implementado como cambio de canales, administración del volumen, mecanismo de encendido/apagado. Al televisor a color le agregaron más funcionalidad como: soportar audio, video, cable. Más herencia: Existiendo el televisor a color (clase Padre) fabricaron el televisor digital (clase Hija) al que le cambiaron y agregaron funcionalidad para el soporte de la tecnología digital. Más herencia: Existiendo el televisor digital (clase Padre) fabricaron el televisor Smart (clase Hija) al que le cambiaron y agregaron funcionalidad para el soporte de internet.

En programación sucede lo mismo, por ejemplo: Existiendo una clase Fecha (clase Padre) cuya funcionalidad soporta un formato de presentación DD/MM/AA, se puede construir una nueva clase FechaM (clase Hija) cuya funcionalidad soporte más formatos de presentación como DD/NOMBRE DEL MES/AAAA, AA/MM/DD, etc.

Jerarquía de Clases

La jerarquía de clases se va construyendo con la aplicación del concepto de herencia donde una clase padre extiende una clase hija y ésta a su vez extiende otra clase hija convirtiéndose en clase padre y así sucesivamente.

Como parte de la terminología de aplicación de herencia, en programación orientada a objetos, a la clase ya existente se le conoce como clase Padre ó clase Base ó clase Ascendiente o Super clase; en cambio, a la nueva clase que hereda se le conoce como clase Hija ó clase Derivada ó clase Descendiente o Subclase.

Cuando uno va a desarrollar por primera vez una clase y piensa que con el tiempo se podría utilizar como clase Padre, entonces el nivel de acceso a sus atributos deja de ser `prívate` y se debe considerar como `protected` ya que sigue permitiendo el encapsulamiento pero además dispone su acceso directo de todas sus clases descendientes. En cambio, cuando la clase que se va a heredar ya existe con un nivel de acceso `prívate` para sus atributos, entonces la clase hija tiene que utilizar la funcionalidad pública de acceso a sus atributos.

Extends

Es una palabra reservada que se utiliza para implementar el concepto de herencia. Significa **extensión** de una clase existente hacia una nueva clase aprovechando todo su contenido.

Super

Es una palabra reservada que se utiliza para referirse al constructor de la clase padre desde el constructor de la clase hija o también para referirse a algún método de la clase padre que también exista con el mismo nombre en la clase hija.

Si la clase Padre no tiene un constructor explícito, entonces la clase Hija no está obligada a tenerlo. Sin embargo, cuando la clase Padre sí tiene un constructor explícito, entonces la clase Hija sí está obligada a tener su constructor desde donde debe invocar al constructor de la clase Padre utilizando la palabra reservada **super**.

Ejemplo 1

Considere la existencia de la clase **TV** desarrollada anteriormente y diseñe una nueva clase hija de nombre **TVH**, aplicando herencia, considerando los siguientes atributos protegidos adicionales: `origen(entero)`, `tecnología(entero)`. Considere para el campo `origen`: nacional, americano, japonés, coreano, chino, otro. Considere para el campo `tecnología`: Tradicional, LCD, Plasma, Digital. Considere métodos adicionales para que

retornen el nombre del origen y el nombre de la tecnología.

La nueva clase hija debe ser utilizada para el proceso de la siguiente GUI:



Serie	Marca	Tamaño	Pulgadas	Precio US\$	Precio S/	Origen	Tecnología
423432	LG	27		1567.5	4467.375	Koreano	LCD

Datos de la clase Padre-----
 Nro. Serie :423432
 Marca :LG
 Tamaño :27
 Precio us\$:1567.5
 Precio S/ :4467.375
 Datos de la clase Hija-----
 Origen :Koreano
 Tecnología :LCD

Nuevo Borrar

Diseño de la clase hija, aplicando herencia:

public class TVH extends TV {

// atributos protegidos

protected int origen, tecnologia;

// constructor explícito

public TVH(String serie, int marca, int tamaño, double precio, int
origen, int tecnologia){

// invocación al constructor de la clase Padre

super(serie, marca, tamaño, precio);

// inicializa sus propios atributos

this.origen = origen;

this.tecnologia=tecnologia;

}

// métodos get-set

public int getOrigen() {

return origen;


```
}  
public void setOrigen(int origen) {  
    this.origen = origen;  
}  
public int getTecnologia() {  
    return tecnologia;  
}  
public void setTecnologia(int tecnologia) {  
    this.tecnologia = tecnologia;  
}
```

// metodos adicionales

```
public String nombreOrigen(){  
    switch(origen){  
        case 1: return "Nacional";  
        case 2: return "Americano";  
        case 3: return "Japonés";  
        case 4: return "Koreano";  
        case 5: return "Chino";  
        default: return "Otro";  
    }  
}  
  
public String nombreTecnologia(){  
    switch(tecnologia){  
        case 1: return "Tradicional";  
        case 2: return "LCD";  
        case 3: return "Plasma";  
        default: return "Digital";  
    }  
}
```

}// fin de la clase hija

- La nueva clase hija tendrá acceso a todo lo **public** y **protected** de la clase padre. Sin embargo, no tendrá acceso a ningún elemento **private**.

- La nueva clase hija tiene sus atributos con nivel de acceso `protected` porque estamos previniendo que ésta sea heredada por otras clases.
- Observe y analice la aplicación de Herencia: **extends**, **super**

Programamos la acción del botón **Nuevo**:

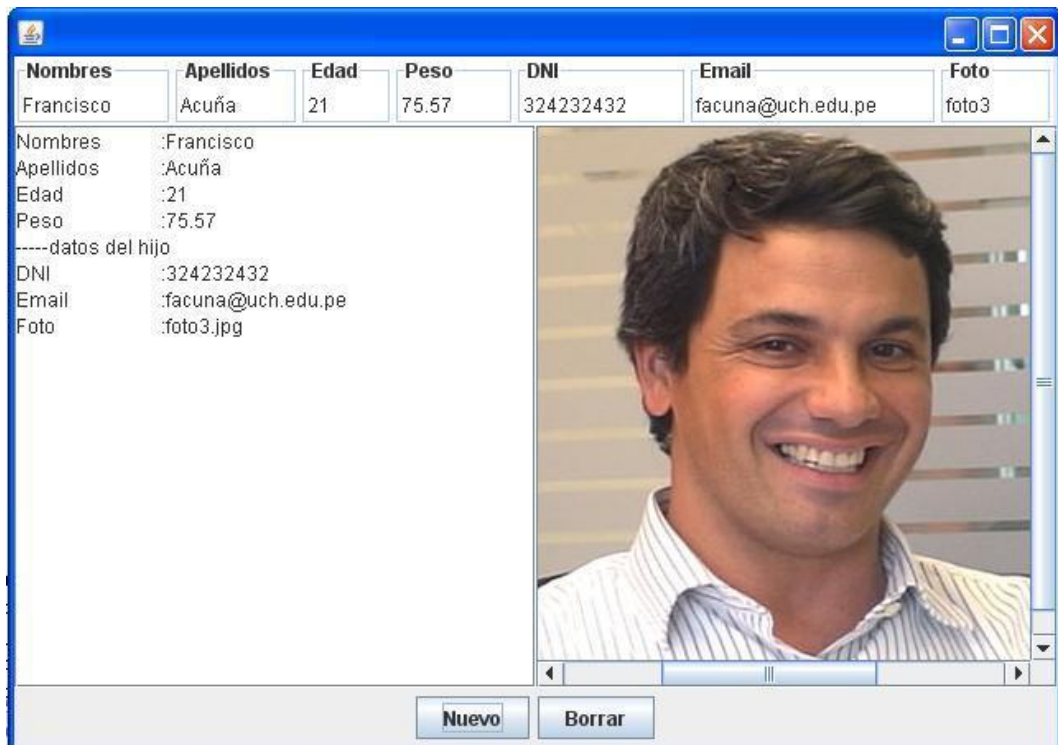
```
private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt) {  
    TVH nuevo = new TVH(leeSerie(), leeMarca(), leeTamaño(), leePrecio(),  
                        leeOrigen(), leeTecnologia());  
    lista(nuevo);  
}  
private void lista(TVH t){  
    imprime("Datos de la clase Padre-----");  
    imprime("Nro. Serie\t:" + t.getSerie());  
    imprime("Marca\t:" + t.nombreMarca());  
    imprime("Tamaño\t:" + t.getTamaño());  
    imprime("Precio us$\t:" + t.getPrecio());  
    imprime("Precio S/.\t:" + t.precioSoles(2.85));  
    imprime("Datos de la clase Hija-----");  
    imprime("Origen\t:" + t.nombreOrigen());  
    imprime("Tecnologia\t:" + t.nombreTecnologia());  
    imprime("-----");  
}
```

- Observe e identifique los datos que se utilizan al momento de crear el objeto.
- Observe y verifique la compatibilidad entre la invocación del método `lista()` y su desarrollo.
- Repase el desarrollo de los métodos de lectura de los datos de la GUI y también el método `imprime()`.

Ejemplo 2

Considere la existencia de una clase base de nombre **Persona** ya desarrollada anteriormente y diseñe una nueva clase hija de nombre **PersonaH**, aplicando herencia, con los siguientes atributos protegidos adicionales: `dni` (cadena), `email`(cadena), `foto`(cadena).

La nueva clase hija debe ser utilizada para el proceso de la siguiente GUI:



Diseño de la clase hija aplicando herencia:

```
public class PersonaH extends Persona {
```

```
    // atributos protegidos
```

```
    protected String dni, email, foto;
```

```
    // constructor
```

```
    public PersonaH(String nombres, String apellidos, int edad, double peso,  
                    String dni, String email, String foto){
```

```
        super(nombres,apellidos,edad,peso);
```

```
        this.dni=dni;
```

```
        this.email =email;
```

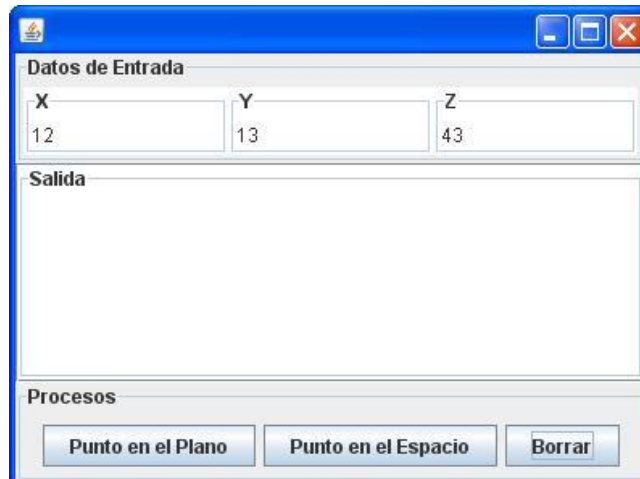
```
        this.foto =foto;
```

```
    }
```

- Complete los métodos get-set.
- Observe y analice la aplicación de Herencia.
- Programe la acción del botón **Nuevo**.

Ejemplo 3

Diseñe una clase padre de nombre **PuntoP** cuyos atributos sean las coordenadas **x,y** de su ubicación en el plano. Considere un constructor con parámetros, la funcionalidad de acceso a sus atributos y un método adicional que retorne la distancia desde su ubicación hasta el punto de origen. Luego, diseñe una clase hija de nombre **PuntoE** que **herede** a la clase **PuntoP** y considere un atributo adicional para la coordenada **z** de su ubicación en el espacio. Considere un constructor con parámetros, la funcionalidad de acceso a su atributo y un método adicional que retorne la distancia desde su ubicación hasta el punto de origen. Considere una clase de GUI donde utilice objetos de ambas clases para mostrar su información.



Ejemplo 4

Diseñe la clase **ListaPilaTVH**, que administre una lista tipo pila de objetos tipo TVH y que considere los siguientes métodos adicionales:

- Incrementa el precio de todos los televisores en 8%
- Incrementa el precio de los televisores de una marca dada como parámetro en un porcentaje también dado como parámetro
- Disminuya el precio de los televisores cuyo tamaño actual esté en un rango dado como parámetro
- Retorne la cantidad de televisores de una marca y tamaño dado como parámetros.

- e) Retorne en un arreglo los televisores de una marca dada como parámetro.

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para incorporar a los métodos adicionales desarrollados

Ejemplo 5

Diseñe la clase **ListaColaPersonasH**, que administre una lista tipo cola de objetos tipo **PersonaH** y que considere los siguientes métodos adicionales:

- a) Incremente el peso de todas las personas en 5%
- b) Incremente el peso de las personas cuyo peso actual sea inferior a un valor dado como parámetro.
- c) Disminuya el peso de las personas cuyo peso actual esté en un rango dado como parámetro.
- d) Retorne la cantidad de personas menores de edad
- e) Retorne en un arreglo las personas cuya edad sea superior a un valor dado como parámetro.

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para incorporar a los métodos adicionales desarrollados.

HERENCIA

Herencia con archivos de texto, sobrecarga de métodos

CAPACIDAD EN PROCESO:

- Aplica el mecanismo de herencia utilizando archivos de texto como medio de almacenamiento.
- Aplica sobrecarga de métodos en una jerarquía de clases.

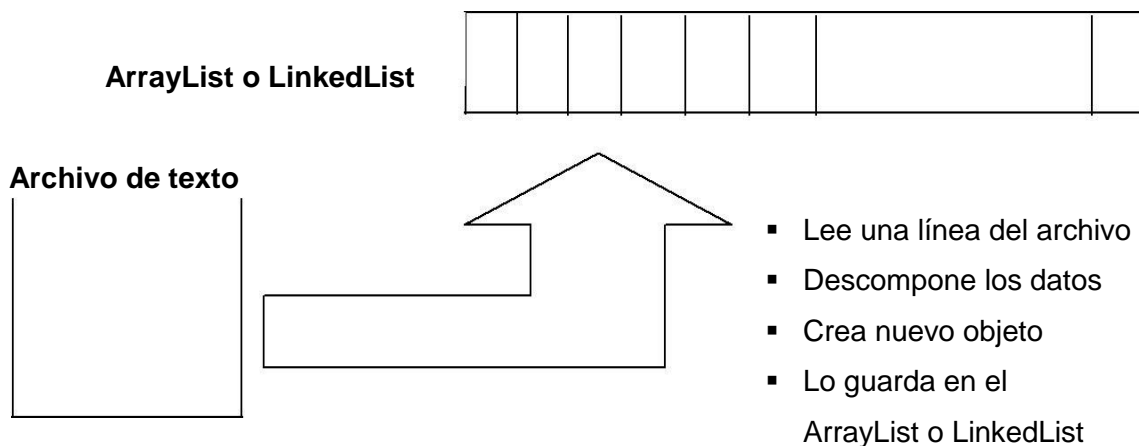
Herencia con archivos de texto

Hasta ahora, las clases **ArrayList** y **LinkedList** nos ha dado 2 ventajas inobjtables: tamaño ilimitado y funcionalidad que facilitan la administración de colecciones de objetos con código simplificado. Sin embargo, todo lo que se guarda en los objetos **ArrayList** y **LinkedList** permanece en memoria sólo mientras se esté ejecutando el programa, si lo ejecutamos nuevamente, tendremos que ingresar todos los datos nuevamente!.

Afortunadamente éste problema se resuelve utilizando archivos de texto para guardar permanentemente los datos en el disco.

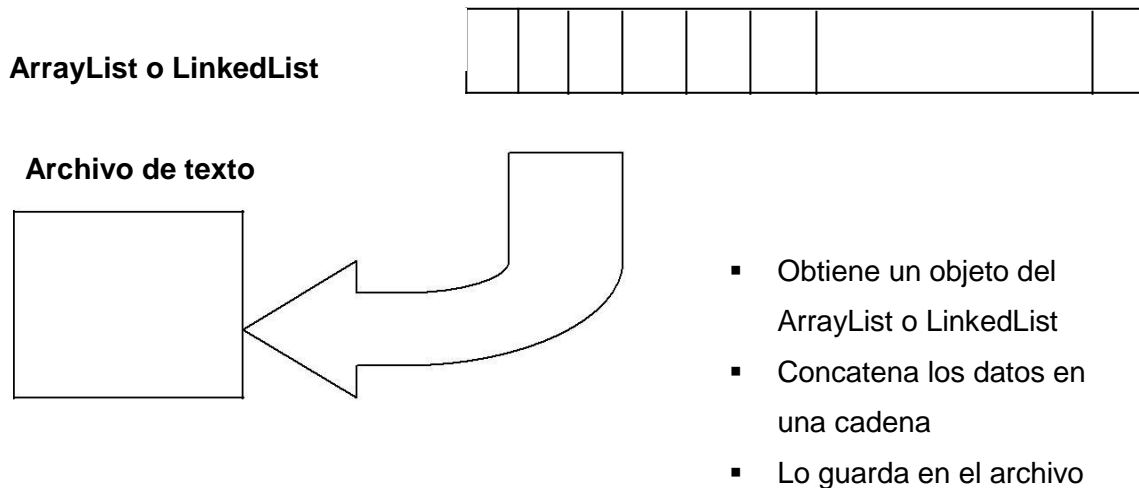
Vamos a tomar como clase padre a una clase administradora, heredamos toda su funcionalidad y a la clase hija le agregamos la funcionalidad que permita grabar en el archivo y leer desde el archivo.

Proceso gráfico de cómo se realiza la **lectura** de los datos dese el archivo de texto hacia la lista.



Para leer los datos del archivo hacia el objeto **ArrayList** o **LinkedList** utilizaremos las siguientes clases: **FileReader**, **BufferedReader**, **StringTokenizer**.

Proceso gráfico de cómo se realiza la **grabación** de los datos desde la lista hacia el archivo de texto.



Para grabar los datos en el archivo desde el objeto **ArrayList** o **LinkedList** utilizaremos las siguientes clases: **FileWriter**, **PrintWriter**.

Ejemplo 1

Considere la existencia de una clase administradora de nombre **ListaPilaTVH** desarrollada anteriormente. Aplique herencia y desarrolle la clase hija **ArchivoTVH** donde se consideren métodos para leer y para grabar en un archivo de texto.

```

public class ArchivoTVH extends ListaPilaTVH {
    // atributos protegidos
    protected String nombre;

    // constructor
    public ArchivoTVH(String nombre){
        super();
        this.nombre=nombre;
    }
}
    
```

```
        lee();

    }

    // métodos que operan un archivo de texto
    public void lee(){
        try{
            FileReader fr = new FileReader(nombre);
            BufferedReader br = new BufferedReader(fr);
            String linea=br.readLine();
            while(linea != null){
                StringTokenizer st = new StringTokenizer(linea,"/");
                String serie=st.nextToken();
                int marca=Integer.parseInt(st.nextToken());
                int tamaño=Integer.parseInt(st.nextToken());
                double precio=Double.parseDouble(st.nextToken());
                int origen=Integer.parseInt(st.nextToken());
                int tecnologia=Integer.parseInt(st.nextToken());
                agrega(serie,marca,tamaño,precio,origen,tecnologia);
                linea= br.readLine();
            }
            br.close();
        }catch(Exception ex){ }
    }

    public void graba(){
        try{
            FileWriter fw = new FileWriter(nombre);
            PrintWriter pw= new PrintWriter(fw);
            for(TVH t : pila){
                pw.println(t.getSerie()+"/"+
                    t.getMarca()+"/"+
                    t.getTamaño()+"/"+
                    t.getPrecio()+"/"+
                    t.getOrigen()+"/"+
```



```

        t.getTecnologia());
    }
    pw.close();
} catch (Exception ex) {}
}
} // fin de la clase
    
```

Considere el mismo diseño de GUI



Serie	Seleccione Marca	Pulgadas	Precio US\$	Origen	Tecnología
12112	Sony	12	2112.0	Americano	Tradicional
2332	Samsung	43	2343.0	Japonés	LCD

Nuevo Lista Consulta Elimina Reporte Borrar

- Declare y cree un objeto de la clase ArchivoTVH en la clase de la GUI
- La grabación de datos debe hacerse en cada botón que lo requiera.
- Programe la acción de los botones.

Ejemplo 2

Considere la existencia de una clase administradora de nombre **ListaColaPersonasH** desarrollada anteriormente. Aplique herencia y desarrolle la clase hija **ArchivoPersonasH** donde se consideren métodos para leer y para grabar en un archivo de texto.

Considere el mismo diseño de GUI:

Nombres	Apellidos	Edad	Peso	DNI	Email	Foto

--	--	--	--	--	--	--

Nuevo	Lista	Consulta	Elimina	Reporte	Borrar
-------	-------	----------	---------	---------	--------

- Declare y cree un objeto de la clase **ArchivoTVH** en la clase de la GUI
- La grabación de datos debe hacerse en cada botón que lo requiera.
- Programe la acción de los botones.

Sobrecarga de métodos

El concepto de sobrecarga va orientado a que un método puede repetirse con el mismo nombre pero con la condición de que se diferencien por los parámetros: en número y/o en cantidad.

Por ejemplo, si existiera un método que suma dos números enteros que recibe como parámetros, tendría la siguiente declaración:

```
public int suma(int a, int b)
```

Ahora, si quisiéramos sumar dos números reales, no tenemos porqué cambiar el nombre al método, porque su tarea va a ser la misma: sumar. Entonces aplicamos sobrecarga para declararlo así:

```
public double suma(double a, double b)
```

Nuevamente, si ahora quisiéramos sumar 3 números enteros, aplicamos sobrecarga así:

```
public int suma(int a, int b, int c)
```

Este concepto de sobrecarga también es aplicable en los **constructores** de las clases.

La ventaja de éste concepto de sobrecarga es que flexibiliza la funcionalidad de las clases, ofreciendo varias alternativas de uso de una misma actividad: los métodos y los constructores.

Por ejemplo, en una clase administradora se puede considerar métodos **sobrecargados** para agregar objetos de diferente manera:

```
public void agrega(TVH t){  
    a.add(t);  
}  
public void agrega(String serie, int marca, int tamaño, double precio,  
                    int origen, int tecnologia){  
    a.add(new TVH(serie, marca,tamaño,precio,origen,tecnologia));  
}
```

Asimismo, se puede considerar métodos sobrecargados para eliminar objetos de diferente manera:

```
public void elimina(int i){  
    a.remove(i);  
}  
public void elimina(String serie){  
    if(busca(serie)!=null)  
        a.remove(busca(serie));  
}  
public void elimina(TVH t){  
    a.remove(t);  
}
```

- Incorpore estos métodos sobrecargados en las clases administradoras.

EVALUACION

EXPOSICIÓN DE TRABAJOS DE INVESTIGACIÓN**EVALUACION INTEGRAL**

POLIMORFISMO

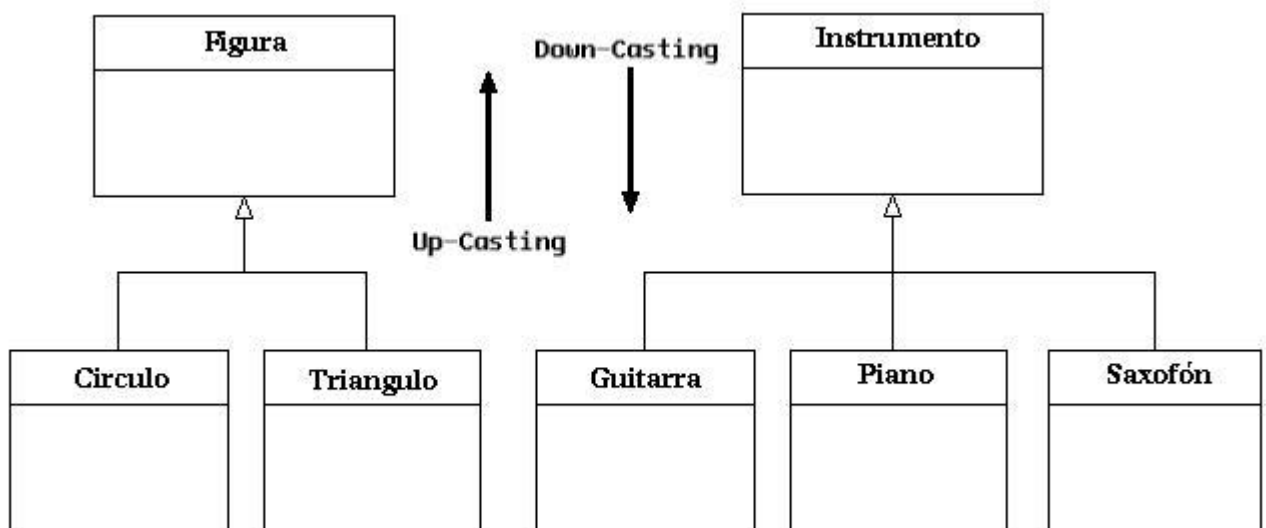
Polimorfismo, custing, up-casting, down-custing, clases Abstractas,

CAPACIDAD EN PROCESO:

Aplica polimorfismo diseñando clases abstractas y sus clases derivadas correspondientes.

Polimorfismo

Es una de las características fundamentales para cualquier lenguaje orientado a Objetos, La derivación de su significado: *Poli* = Multiple, *morfismo*= Formas, implica que Objetos de la misma clase pueden tomar diversas formas de comportamiento. Por ejemplo, el comportamiento de una figura puede ser el de un círculo o el de un triángulo dado que ambas son figuras. Igualmente, el comportamiento de un instrumento puede ser el de una guitarra o el de un piano o el de un saxofón dado que todos ellos son instrumentos.



El poder manipular un Objeto como si éste fuera de un tipo genérico otorga mayor flexibilidad al momento de programar con Objetos. Observe el siguiente fragmento de código:

```
Figura a = new Circulo();
Figura b = new Triangulo();
```

Los objetos a y b siendo de la misma clase Figura, se comportan como clases diferentes: Circulo y Triángulo.

Casting

El término "Casting" viene de la palabra "Cast" que significa Molde, por lo que el termino literal es Hacer un Molde. En Polimorfismo se lleva a cabo este proceso de "Casting" en dos sentidos: hacia arriba (up-custing) o hacia abajo (down-custing). Cuando la relación es up-custing el proceso es implícito y no requiere aplicar el molde. Por ejemplo, un círculo es una figura y un triangulo también es una figura por tanto no requieren molde. Sin embargo cuando la relación es down-custing el proceso requiere aplicar un molde que lo represente. Por ejemplo, una figura requiere el molde de un circulo para utilizar su propia funcionalidad específica. Asimismo, una figura requiere el molde de un triangulo para utilizar su propia funcionalidad.

Para aplicar un molde solamente se debe anteponer al objeto genérico la clase específica que se quiere utilizar como molde. Por ejemplo, observe el siguiente código:

```
public void lista (Figura f){
    if (f instanceof Circulo)
        // aplica molde de circulo para acceder al valor de su radio
        imprime ("radio = "+ ((Circulo)(f)).getRadio() );
    else
        // aplica molde de triangulo para acceder al valor de su base
        imprime ("radio = "+ ((Triangulo)(f)).getBase() );
}
```

Clases Abstractas

Son clases Padre donde se consideran métodos que aún no se conoce su desarrollo. Recién se conocerá en alguna clase Hija. A estos métodos se les conoce como métodos abstractos y utilizan la palabra reservada **abstract** para su identificación.

Cuando una Clase Padre contiene por lo menos un método abstracto se le conoce como *Clase Abstracta* y también debe estar precedida por la palabra reservada **abstract**.

Estas clases pueden tener además métodos no abstractos que invoquen a métodos abstractos si lo necesitan para definir un comportamiento. Sin embargo, no se puede instanciar objetos invocando al constructor de la clase base abstracta.

Las clases Hija deben desarrollar todos los métodos abstractos de la clase Padre. Además, la clase Hija puede redefinir cualquier método de la clase Padre, con el mismo nombre. Incluso pueden utilizar la palabra reservada **super** para referirse a cualquier método del mismo nombre que esté definido en la clase Padre.

En el funcionamiento del polimorfismo se puede identificar el concepto de “enlace tardío” que consiste en que recién se conocerá el método a ejecutar en el momento de la ejecución, más no en la compilación.

Ejemplo 1

Diseñe una clase abstracta de nombre **Figura** que tenga un atributo protegido para el nombre de la figura, métodos get-set, métodos abstractos para el `area()` y el `perimetro()` y un método no abstracto que retorne la información correspondiente al nombre, area y perímetro de cualquier figura.

Aplique polimorfismo y diseñe una clase hija de nombre **Circulo** que tenga un atributo para el radio, constructor, métodos get-set y desarrollo de los métodos abstractos de la clase padre. Luego diseñe una clase hija de nombre **Cuadrado** que tenga un atributo para el lado, constructor, métodos get-set, y desarrollo de los métodos abstractos de la clase padre.

Finalmente, diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente figura.

Diseño de la clase abstracta **Figura**:

```
public abstract class Figura{  
    protected String nombre;  
    public Figura(String nombre){  
        this.nombre=nombre;  
    }  
}
```



```
// métodos abstractos
public abstract double area();
public abstract double perimetro();

// método no abstracto
public String info(){
    return "Figura\t: "+nombre+"\n" + "Area\t: "+ area()+"\n"+
        "Perímetro\t: "+perimetro();
}
}
```

Diseño de la clase **Circulo**:

```
public class Circulo extends Figura{
    protected double radio;
    public Circulo(double radio){
        super("Circulo");
        this.radio = radio;
    }
    // desarrollo de los métodos abstractos
    public double area(){ return Math.PI * radio * radio;}
    public double perimetro(){ return 2 * Math.PI * radio;}
}
```

Diseño de la clase **Cuadrado**:

```
public class Cuadrado extends Figura{
    protected double lado;
    public Cuadrado(double lado){
        super("Cuadrado");
        this.lado = lado;
    }
    // desarrollo de los métodos abstractos
    public double area(){ return lado * lado;}
    public double perimetro(){ return 4*lado;}
}
```

```
Programación del botón Nuevo Circulo:{  
    Figura a = new Circulo(leeRadio());  
    lista(a);  
}
```

```
Programación del botón Nuevo Cuadrado:{  
    Figura b = new Cuadrado(leeLado());  
    lista(b);  
}
```

```
public void lista(Figura f){  
    imprime(f.info());  
}
```

Observe el parámetro del método lista() y compruebe la compatibilidad entre la invocación del método y su desarrollo.

Ejemplo 2

Diseñe una clase abstracta de nombre **Empleado** que tenga atributos protegidos para el nombre, apellidos, dni, métodos get-set, métodos abstractos para ingresos(), bonificaciones(), descuentos(), un método no abstracto que retorne el sueldoNeto() y otro método no abstracto que retorne la información correspondiente al nombre, apellidos, dni, ingresos, bonificaciones, descuentos, sueldo neto de cualquier empleado.

Aplique polimorfismo y diseñe una clase hija de nombre **EmpleadoVendedor** que tenga atributos para monto vendido y porcentaje de comisión, constructor, métodos get-set, desarrollo de métodos abstractos. Luego, diseñe una clase hija de nombre **EmpleadoPermanente** que tenga atributos para sueldo basico y afiliación (afp ó snp), constructor, métodos get-set, desarrollo de métodos abstractos.

Finalmente, diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente tipo de empleado.

Ejemplo 3:

Diseñe la clase abstracta **Celular** con los siguientes atributos: número, dueño, precio, marca (1=samsung, 2=htc, 3=lg, 4=otro). Considere un método abstracto para el cálculo del impuesto. Considere un método no abstracto para que retorne el número, dueño, nombre de marca y monto de impuestos. Asuma los métodos get-set, no los desarrolle.

Aplique polimorfismo y diseñe la clase hija **CelularSmart** con el siguiente atributo adicional: país de origen. Para el cálculo del monto de su impuesto considere lo siguiente: si es de marca samsung aplique 25%, si es htc aplique 20%, si es lg aplique 15% y si es otro aplique 10% sobre el precio. Luego Diseñe la clase hija **Celular4G** con el siguiente atributo adicional: operador(1=movistar, 2=claro, 3=entel, 4=otro). Para el cálculo del monto de su impuesto considere siguiente: si el operador es movistar aplique 10%, , si es claro aplique 9%, si es entel aplique 7% y si es otro aplique 5% sobre el precio.

Finalmente, diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente tipo de celular.

Ejemplo 4:

Diseñe una clase abstracta de nombre **Vehiculo** con los siguientes atributos: placa, marca, precio. Con el siguiente método no abstracto: info() que retorna, en una cadena los siguientes datos tabulados: placa, marca, precio, impuesto. Considere que el monto del impuesto depende del precio y del tipo de vehículo que sea.

Aplique polimorfismo y diseñe una clase hija de nombre **Automovil** cuyo monto de su impuesto es el 13% de su precio si éste supera los 10,000 dólares y 9% en caso contrario. Diseñe otra clase hija de nombre **Camion** cuyo monto de su impuesto es el 21% de su precio si éste supera los 20,000 dólares y 7% en caso contrario.

Finalmente, diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente tipo de vehiculo.

POLIMORFISMO

Coleccion de Objetos Polimórficos, instanceof

CAPACIDAD EN PROCESO:

Construye un arreglo de objetos polimórficos diferenciando los objetos particulares.

Colección de objetos polimórficos

Estas colecciones son clases administradoras que se caracterizan porque soportan cualquier objeto de las clases descendientes de una jerarquía de clases, teniendo como clase **Padre** a una clase **abstracta**.

Instanceof

Es una palabra reservada que permite identificar a que clase hija pertenece un objeto cuyo tipo de dato es una clase abstracta.

Ejemplo 1

Considere la existencia de la clase abstracta de nombre **Figura**, las clases hijas **Cuadrado** y **Circulo**.

Diseñe una clase administradora de nombre **ColeccionFiguras** que permita la administración de objetos de tipo Cuadrado y/o Circulo a la vez, utilizando un objeto de la clase **ArrayList**.

```
public class ColeccionFiguras{
    protected ArrayList <Figura> coleccion;

    public ArregloFiguras() {
        coleccion = new ArrayList<Figura>();
    }
    // métodos de administración
    public void agrega(Figura f){
        coleccion.add(f);
    }
}
```

```
    }  
    public Figura obtiene(int i){ return coleccion.get(i); }  
    public int tamaño(){ return coleccion.size();}  
    public void elimina(int i){ coleccion.remove(i); }  
    //... complete métodos adicionales  
}
```

Diseñe una clase de interfaz de nombre **PanelPrincipal** donde cree un objeto de nombre **cf** de la clase **ColeccionFiguras**, como atributo de la clase.

Programación del botón Nuevo Circulo:{

```
    Figura a = new Circulo(leeRadio());  
    lista(a);  
    cf.agrega(a); // agrega un objeto Circulo a la colección  
}
```

Programación del botón Nuevo Cuadrado:{

```
    Figura b = new Cuadrado(leeLado());  
    lista(b);  
    cf.agrega(b); // agrega un objeto Cuadrado a la coleccion  
}  
public void lista(Figura f){  
    imprime(f.info());  
}
```

Programación del botón Lista:{

```
    for(int i=0; i<cf.tamaño(); i++){  
        Figura f=cf.obtiene(i);  
        if (f instanceof Circulo)  
            imprime("Circulo: "+f.info());  
        else  
            imprime("Cuadrado: "+f.info());  
    }  
}
```

Ejemplo 2

Considere la existencia de la clase abstracta de nombre **Empleado**, las clases hijas **EmpleadoVendedor** y **EmpleadoPermanente**.

Diseñe una clase administradora de nombre **ColeccionEmpleados** que permita la administración de objetos de tipo **EmpleadoVendedor** y/o **EmpleadoPermanente** a la vez, utilizando un objeto de la clase **ArrayList**.

Diseñe una clase de interfaz de nombre **PanelPrincipal** donde cree un objeto de la clase **ColeccionEmpleados**, como atributo de la clase.

Ejemplo 3

Considere la existencia de la clase abstracta de nombre **Celular**, las clases hijas **CelularSmart** y **Celular4G**.

Diseñe una clase administradora de nombre **ColeccionCelulares** que permita la administración de objetos de tipo **CelularSmart** y/o **Celular4G** a la vez, utilizando un objeto de la clase **LinkedList**.

Diseñe una clase de interfaz de nombre **PanelPrincipal** donde cree un objeto de la clase **ColeccionCelulares**, como atributo de la clase.

Ejemplo 4:

Considere la existencia de la clase abstracta de nombre **Vehiculo**, las clases hijas **Automovil** y **Camion**.

Diseñe una clase administradora de nombre **ColeccionVehiculos** que permita la administración de objetos de tipo **Automovil** y/o **Camion** a la vez, utilizando un objeto de la clase **LinkedList**.

Diseñe una clase de interfaz de nombre **PanelPrincipal** donde cree un objeto de la clase **ColeccionVehiculos**, como atributo de la clase.

EVALUACIÓN

POLIMORFISMO

Persistencia con archivos de texto

CAPACIDAD EN PROCESO:

Utiliza archivos de texto para la persistencia de información de los arreglos polimórficos

Persistencia con archivos de texto

La persistencia de datos se refiere a la forma de conservar los datos en el disco. Hay dos maneras de hacerlo: en archivos de texto y en bases de datos.

En esta parte utilizaremos archivos de texto para guardar los datos de los objetos polimórficos que se encuentren en las colecciones.

Ejemplo 1

Considere la existencia de la clase **ColeccionFiguras** desarrollada anteriormente y aplique herencia desarrollando la clase **ArchivoFiguras** que permita conservar la información de las figuras en un archivo de texto.

```
public class ArchivoFiguras extends ColeccionFiguras{
```

```
    protected String nombre;
```

```
    public ArchivoFiguras(String nombre){
```

```
        this.nombre=nombre;
```

```
        lee();
```

```
    }
```

```
    public void graba(){
```

```
        try{
```

```
            FileWriter fw = new FileWriter(nombre);
```

```
            PrintWriter pw = new FilePrinter(fw);
```

```
            for(Figura f : coleccion){
```

```
                if(f instanceof Circulo)
```

```
                    pw.println("1/"+
```



```
                ((Circulo)(f)).getRadio() ); // molde
            else
                pw.println("2/"+
                    ((Cuadrado)(f)).getLado() ); // molde
        }
        pw.close();
    }catch(Exception ex){
    }
    public void lee(){
    try{
        FileReader fr = new FileReader(nombre);
        BufferedReader br = new BufferedReader(fr);
        String linea = br.readLine();
        while(linea!=null){
            StringTokenizer st =new StringTokenizer(linea,"/");
            int tipo=Integer.parseInt(st.nextToken());
            if(tipo==1){ // circulo
                double radio=Double.parseDouble(st.nextToken());
                Figura a=new Circulo(radio);
                agrega(a);
            }else{// cuadrado
                double lado=Double.parseDouble(st.nextToken());
                Figura b=new Cuadrado(lado);
                agrega(b);
            }
            linea=br.readLine();
        }
        br.close();
    }catch(Exception ex){}
    }
}
```

Utilizando la interface que se diseñó para administrar una colección de figuras polimórficas, en un nuevo proyecto haga las modificaciones necesarias para utilizar la clase **ArchivoFiguras** en lugar de la clase **ColeccionFiguras** y programe la acción de los botones, así:

```
// atributo de la clase interfaz
```

```
protected ArchivoFiguras af = new ArchivoFiguras();
```

Programación del botón Nuevo Circulo:{

```
    Figura a = new Circulo(leeRadio());  
    lista(a);  
    af.agrega(a); // agrega un objeto Circulo  
    af.graba(); // graba la información en el archivo de texto  
}
```

Programación del botón Nuevo Cuadrado:{

```
    Figura b = new Cuadrado(leeLado());  
    lista(b);  
    af.agrega(b); // agrega un objeto Cuadrado  
    af.graba(); // graba la información en el archivo de texto  
}
```

Ejemplo 2

Considere la existencia de la clase **ColeccionEmpleados** desarrollada anteriormente y aplique herencia desarrollando la clase **ArchivoEmpleados** que permita conservar la información de los empleados en un archivo de texto.

Utilizando la interface que se diseñó para administrar una colección de empleados polimórficos, en un nuevo proyecto haga las modificaciones necesarias para utilizar la clase **ArchivoEmpleados** en lugar de la clase **ColeccionEmpleados** y programe la acción de los botones.

Ejemplo 3

Considere la existencia de la clase **ColeccionCelulares** desarrollada anteriormente y aplique herencia desarrollando la clase **ArchivoCelulares** que permita conservar la

información de los celulares en un archivo de texto.

Utilizando la interface que se diseñó para administrar una colección de celulares polimórficos, en un nuevo proyecto haga las modificaciones necesarias para utilizar la clase **ArchivoCelulares** en lugar de la clase `ColeccionCelularess` y programe la acción de los botones.

Ejemplo 4

Considere la existencia de la clase **ColeccionVehiculos** desarrollada anteriormente y aplique herencia desarrollando la clase **ArchivoVehiculos** que permita conservar la información de los vehiculos en un archivo de texto.

Utilizando la interface que se diseñó para administrar una colección de vehiculos polimórficos, en un nuevo proyecto haga las modificaciones necesarias para utilizar la clase **ArchivoVehiculos** en lugar de la clase `ColeccionVehiculos` y programe la acción de los botones.

INTRODUCCIÓN A BASE DE DATOS

Acceso a base de datos, creación, conexión

CAPACIDAD EN PROCESO:

- Crea una base de datos con una tabla
- Establece una conexión a la base de datos desde su aplicación

Acceso a Base de Datos

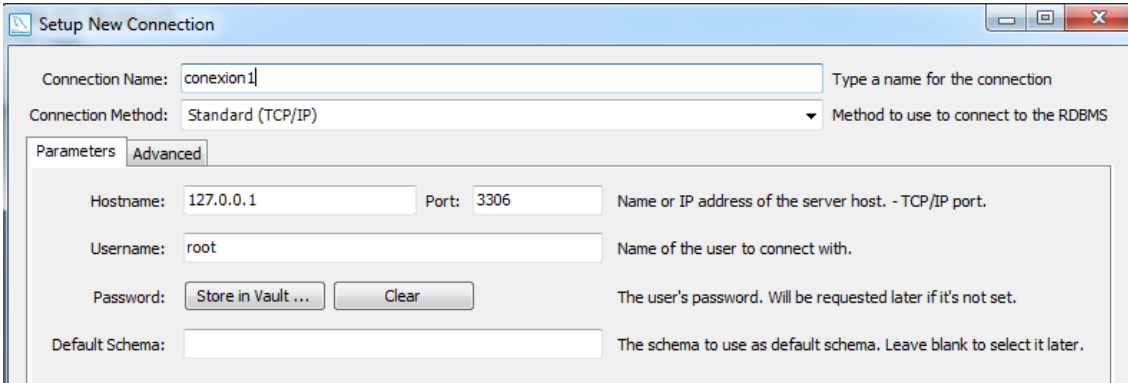
El acceso a una base de datos es siempre un punto infaltable a considerar por nuestras aplicaciones para conservar los datos en el disco. Para esto, en Java tenemos una API muy sencilla que es **Java Database Connectivity** conocido por la abreviación **JDBC**.

Cuando instalamos **JDK** (Java Development Kit) automáticamente se instala esta API (**JDBC**) y con ella en forma muy sencilla podemos hacer consultas, mantenimientos y todo tipo de transacciones a una base de datos.

Para desarrollar nuestras primeras aplicaciones nosotros usaremos la forma de conexión a base de datos con el driver de protocolo de java puro para mysql que está disponible en internet para su descarga gratuita (mysql-connector-java-5.1.37) y la librería Mysql JDBC driver incorporado en el entorno de NetBeans.

Creación de una Base de Datos

Vamos a crear nuestra base de datos en el gestor de base de datos MySQL WorkBench. Abrimos Workbeanch, elegimos **New Conexion** para llegar a la siguiente ventana:



Setup New Connection

Connection Name: Type a name for the connection

Connection Method: Method to use to connect to the RDBMS

Parameters ☒ Advanced

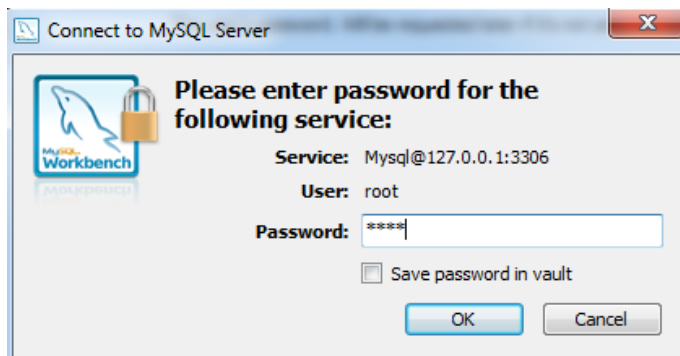
Hostname: Port: Name or IP address of the server host. - TCP/IP port.

Username: Name of the user to connect with.

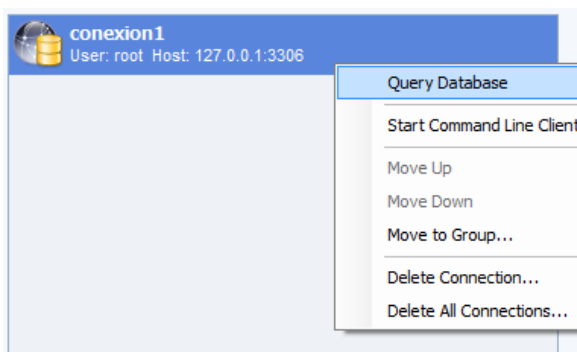
Password: The user's password. Will be requested later if it's not set.

Default Schema: The schema to use as default schema. Leave blank to select it later.

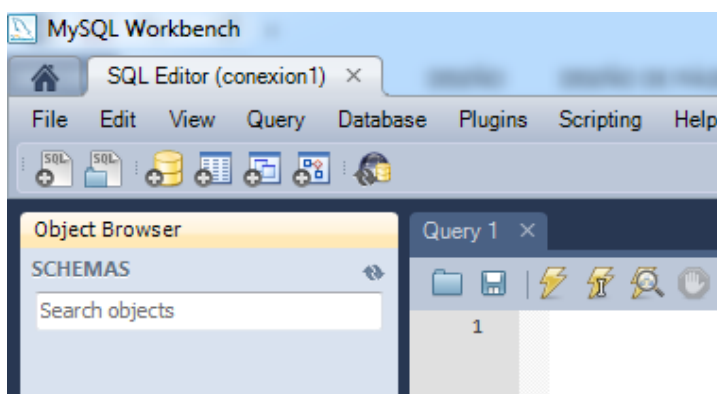
Escribimos el nombre de la nueva conexión (conexion1) y damos clic en el botón Test connection para llegar a la siguiente ventana:



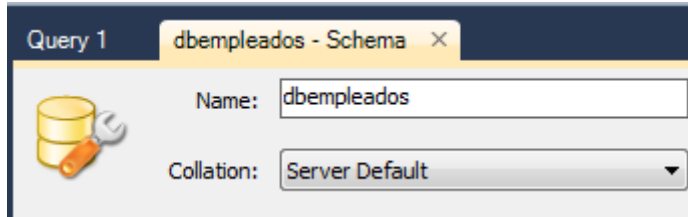
Damos el password (root) y presionamos el botón OK para llegar a la ventana de conexión con éxito y luego damos OK nuevamente para que quede establecida nuestra nueva conexión.



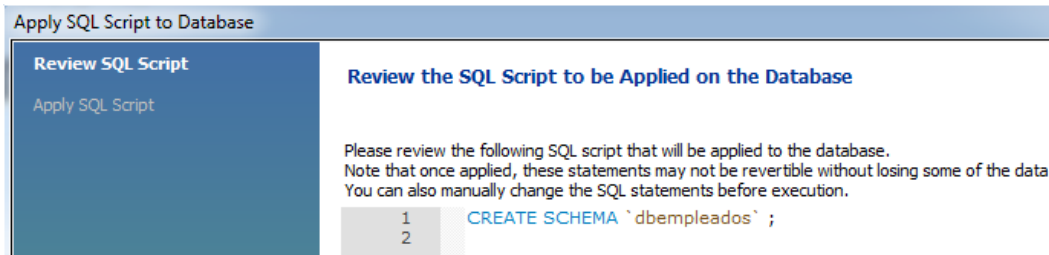
Damos clic derecho en la nueva conexión creada y elegimos Query Database para llegar a la siguiente ventana:



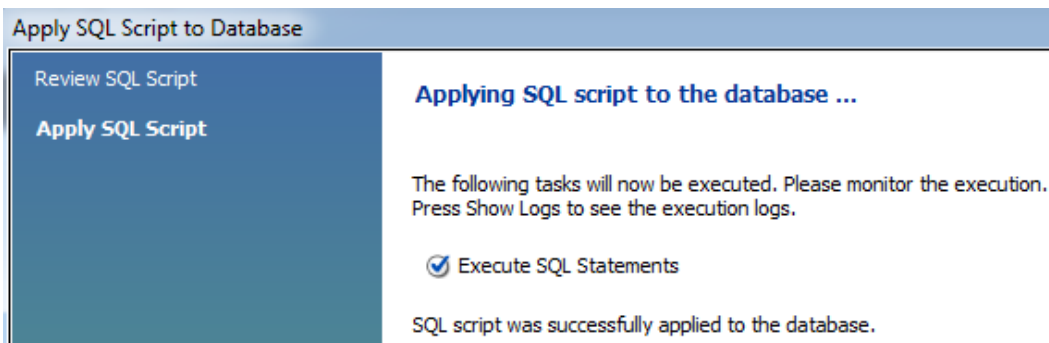
Damos clic en el ícono para crear un nuevo esquema



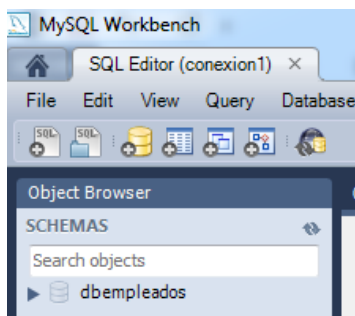
Escribimos el nombre del esquema (dbempleados) y damos clic en el botón Apply para llegar a la siguiente ventana:



Damos clic en el botón Apply nuevamente para llegar a la siguiente ventana:



Damos clic en el botón Finish para llegar a la siguiente ventana:



Donde comprobamos que se ha creado un nuevo esquema (dbempleados) donde daremos clic derecho y elegimos set as default schema.

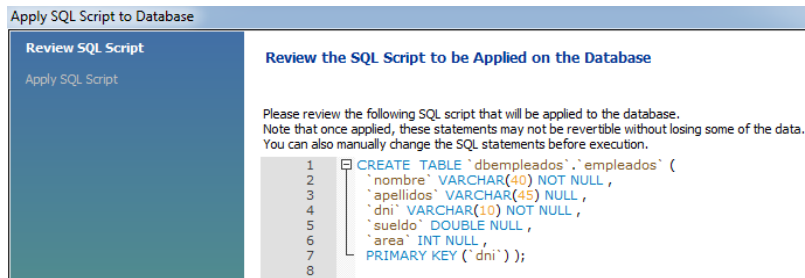
Luego damos clic en el ícono para crear una nueva tabla donde escribimos el nombre y llenamos los campos (column name) conforme a la siguiente ventana:

Query 1 dbempleados - Schema empleados - Table x

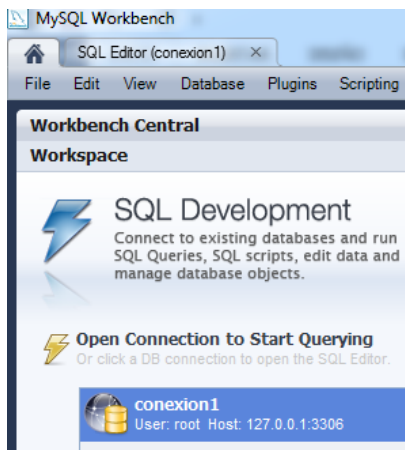
Table Name: Schema: **dbempleados**

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
nombre	VARCHAR(40)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
apellidos	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
dni	VARCHAR(10)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
sueldo	DOUBLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
area	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

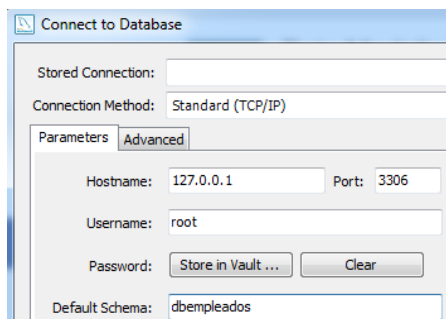
Damos clic en el botón Apply para llegar a la siguiente ventana:



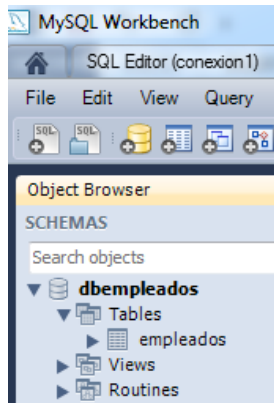
Damos clic en el botón Apply y luego en Finish. Vamos a la ventana de inicio de Workbench donde comprobamos que se ha creado una nueva conexión.



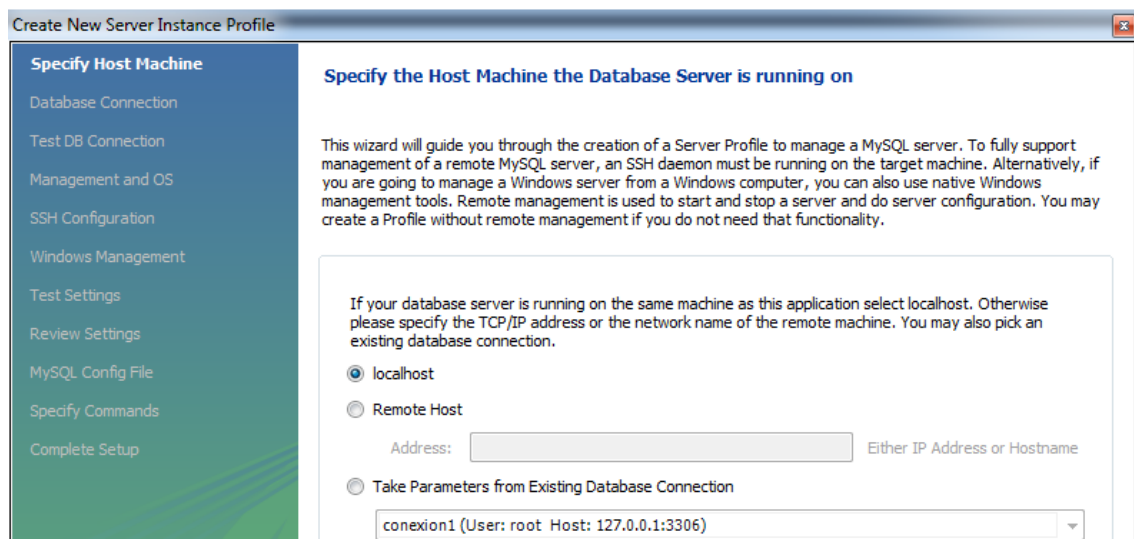
Damos clic en Open Connection to Start Querying para llegar a la siguiente ventana donde escribimos el nombre del esquema (dbempleados):



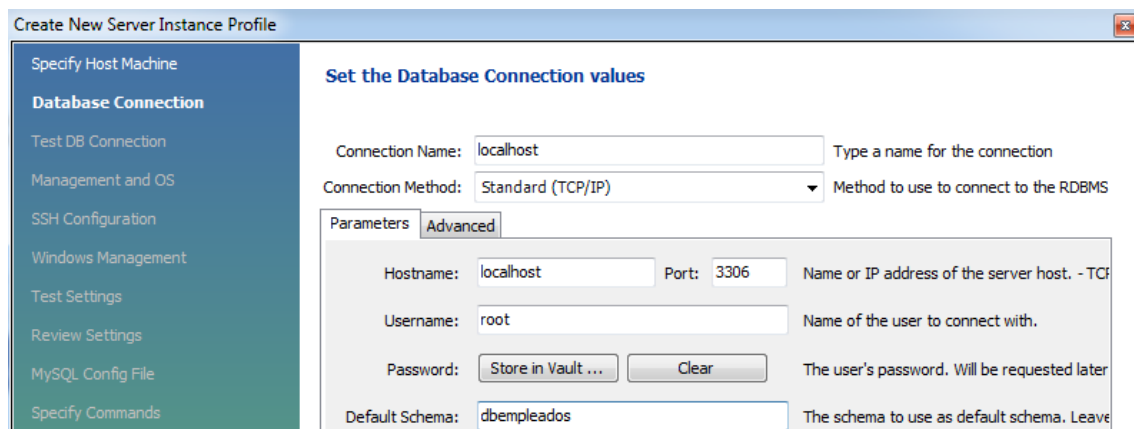
Damos clic en el botón OK, ingresamos el password para llegar a la siguiente ventana:



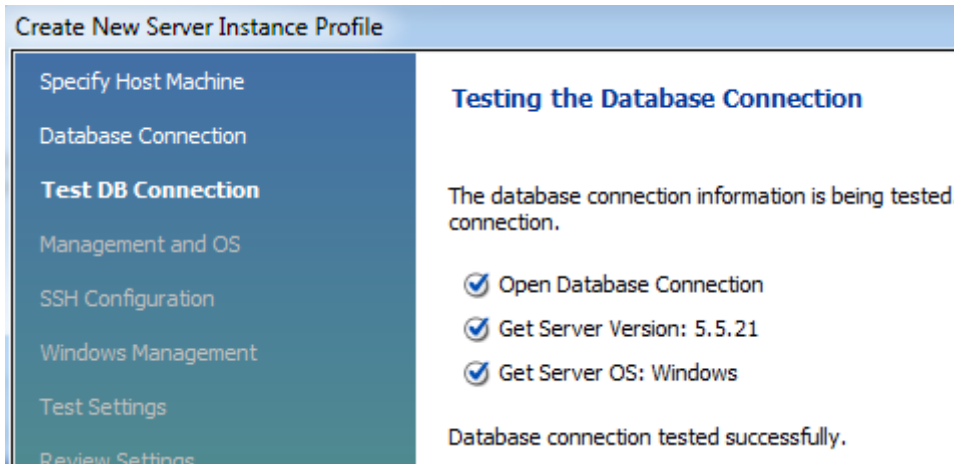
Regresamos al inicio de Workbench para crear una nueva instancia de servidor:



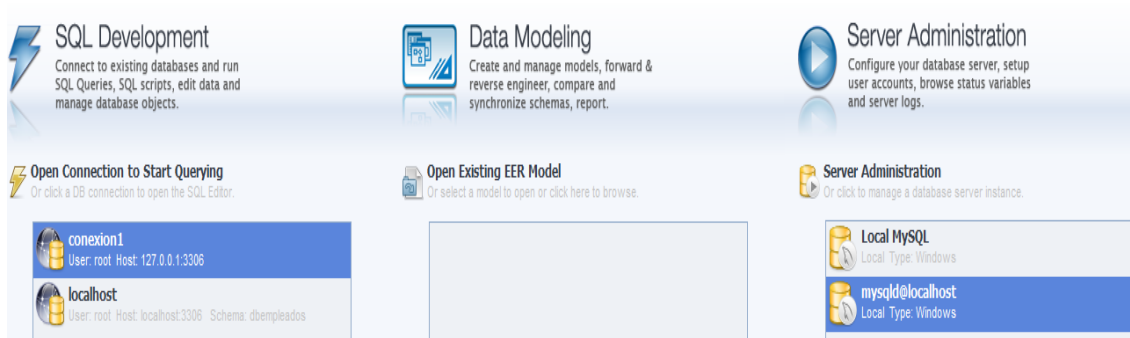
Damos clic en el botón Next y en la siguiente ventana escribimos el nombre del esquema “dbempleados”



Damos clic en el botón Next para llegar a la siguiente ventana:

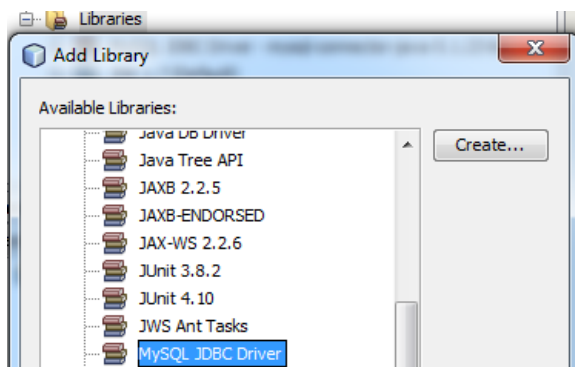


Damos clic en el botón Next hasta finalizar. Vamos al inicio del Workbench para verificar que se ha creado la conexión y la instancia del servidor:



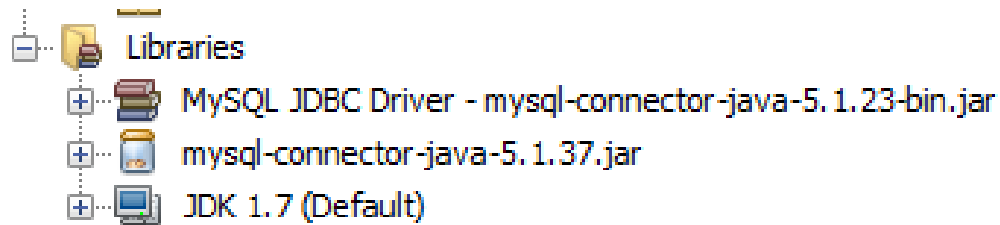
Conexión a una Base de Datos

Entramos a NetBeans, creamos un proyecto para incorporar el driver descargado y la librería en la carpeta librerías de nuestra aplicación: Clic derecho en Libraries de nuestra aplicación y elegimos Library y seleccionamos MySQL JDBC Driver.

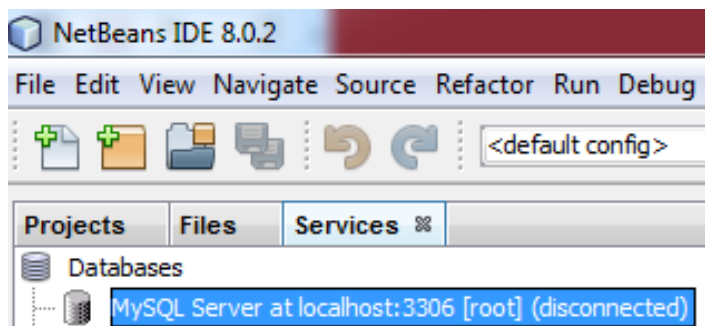


Luego de aceptar, verifique su incorporación en las librerías de nuestra aplicación:

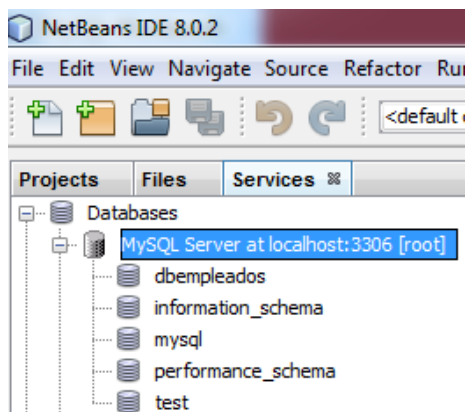
Ahora vamos a incorporar el driver conector que hemos descargado de internet. Nuevamente clic derecho en la carpeta Libraries de nuestra aplicación, elegimos add JAR/folder para que finalmente quede así la carpeta Libraries de nuestra aplicación:



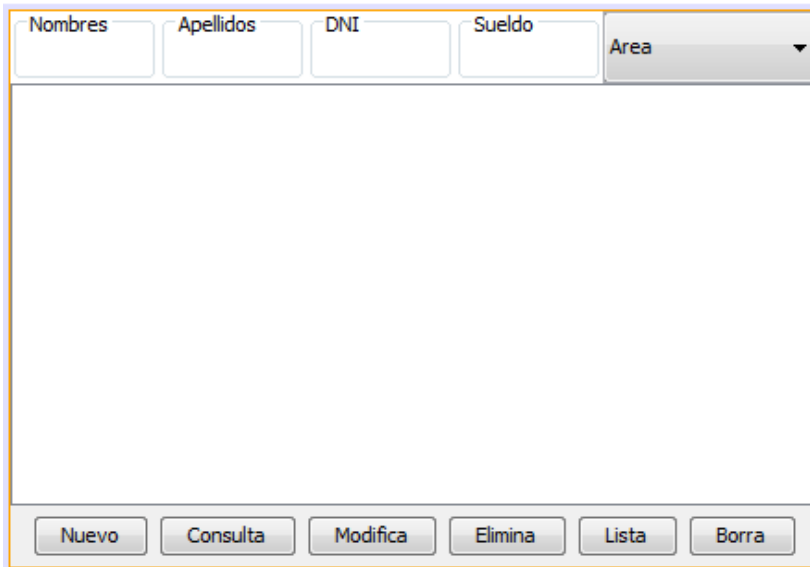
Ahora vamos a la ficha Services para conectar el servidor localhost que inicialmente está desconectado:



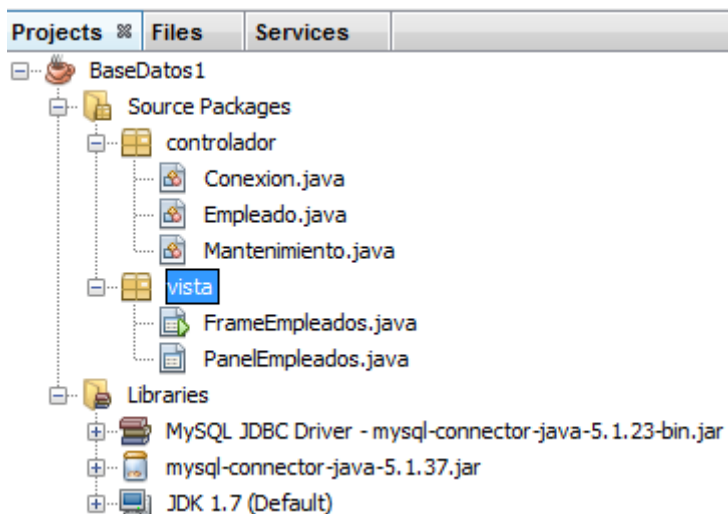
Para conectarlo, le damos clic derecho y elegimos Connect para que quede así: (Observe que nuestra base de datos creada en Workbench (dbempleados) ya es reconocida):



Ahora, vamos a crear una aplicación con un formulario para hacer el mantenimiento de la tabla de empleados considerando el siguiente diseño:



En la ficha Projects de nuestra aplicación creamos un paquete de nombre **controlador** donde diseñamos las clases Empleado, Conexión y Mantenimiento. Luego crearemos el paquete **vista** donde diseñamos un PanelEmpleados para el formulario y un FrameEmpleados para ejecutar el panel, de la siguiente manera:



Las clases Empleado y Mantenimiento lo desarrollaremos en la siguiente sesión. Ahora sólo probaremos la conexión con el diseño de la siguiente clase Conexión:

```
package controlador;
import java.sql.*; // acceso a Connection, Statement y ResultSet
public class Conexion {
    // atributos de uso compartido
    protected static String servidor="jdbc:mysql://localhost/dbempleados";
    protected static String usuario="root";
    protected static String password="root";
    protected static String driver="com.mysql.jdbc.Driver";
    protected static Connection conexion;

    // constructor que realiza la conexión a la base de datos
    public Conexion(){
        try{
            Class.forName(driver);
            conexion=DriverManager.getConnection(servidor,usuario,password);
            System.out.println("conexion exitosa!");
        }catch(ClassNotFoundException | SQLException e){
            System.out.println("Conexion fallida: "+e.getMessage());
        }
    }

    // método que retorna el resultado de la conexión realizada
    public Connection getConexion(){
        return conexion;
    }
}
```

En el constructor del PanelEmpleados escribimos lo siguiente:

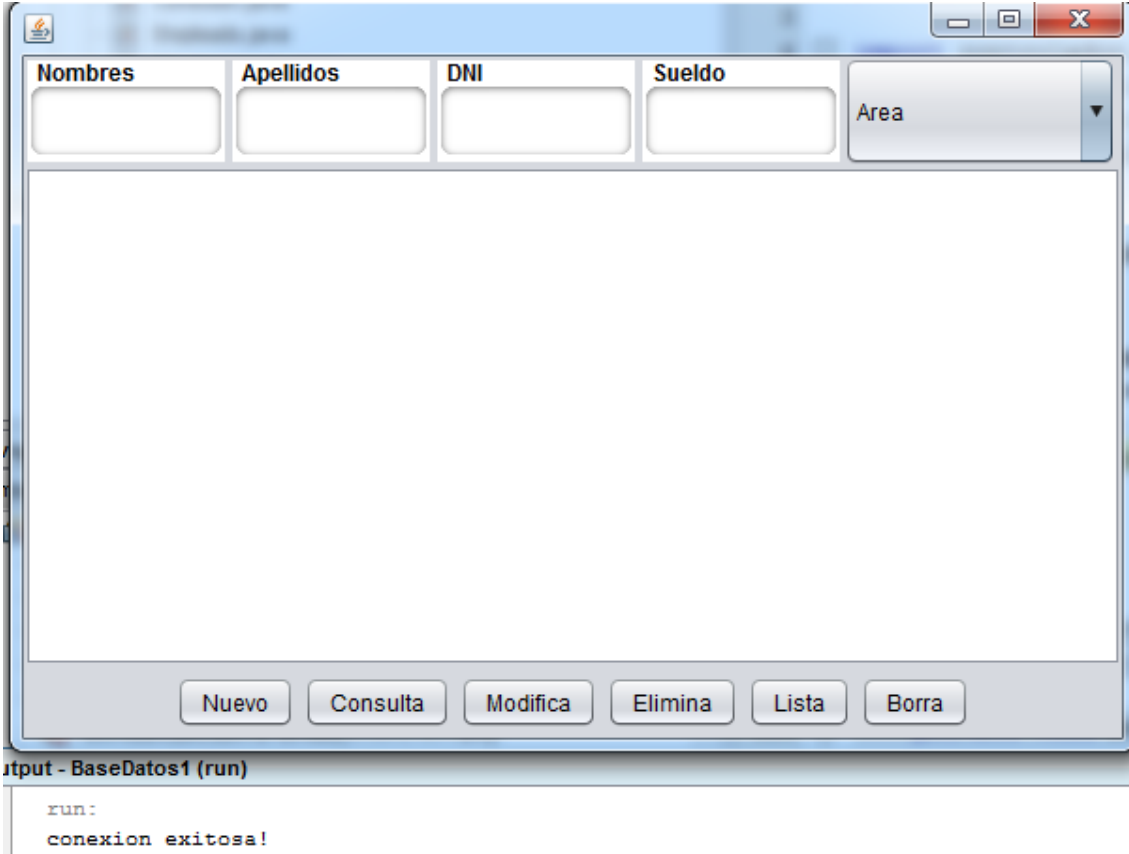
```
package vista;

import controlador.Conexion;

public class PanelEmpleados extends javax.swing.JPanel {

    public PanelEmpleados() {
        initComponents();
        // conecta a la base de datos
        Conexion con = new Conexion();
    }
}
```

Ejecutamos nuestra aplicación:



The image shows a Java Swing window titled "BaseDatos1" with a standard Windows-style title bar. The window contains a form with five input fields at the top: "Nombres", "Apellidos", "DNI", "Sueldo", and "Area". Below these fields is a large, empty rectangular area, likely for displaying data. At the bottom of the window, there are six buttons: "Nuevo", "Consulta", "Modifica", "Elimina", "Lista", and "Borra". Below the Swing window, a terminal window titled "Output - BaseDatos1 (run)" is visible, showing the output of a Java program. The terminal displays the text "run:" followed by "conexion exitosa!" on the next line.

Nombres	Apellidos	DNI	Sueldo	Area
---------	-----------	-----	--------	------

Nuevo Consulta Modifica Elimina Lista Borra

```
Output - BaseDatos1 (run)
run:
conexion exitosa!
```

Observe el mensaje en la salida de consola: "conexión exitosa! "

INTRODUCCIÓN A BASE DE DATOS

Mantenimiento y consultas a una base de datos

CAPACIDAD EN PROCESO:

Desarrolla métodos para el mantenimiento y consulta de información en una base de datos.

Mantenimiento y consultas a una base de datos

Luego de haber logrado una conexión exitosa, ahora desarrollamos la clase

Mantenimiento con el siguiente atributo y constructor:

```
package controlador;

import java.sql.ResultSet;
import java.sql.Connection;
import java.sql.Statement;

public class Mantenimiento {
    protected Conexion conecta;

    public Mantenimiento() {
        conecta = null;
    }
}
```

Luego desarrollamos el método para registrar un nuevo empleado en la base de datos:

```
public void nuevo(Empleado e){
    String sql;

    sql = "insert into empleados (nombre,apellidos,dni,sueldo,area) "
        + "values ('"+e.getNombre()+"', '"+
            e.getApellidos()+"', '"+
            e.getDni()+"', '"+
            e.getSueldo()+"', '"+
            e.getArea()+"')";

    try{
        conecta = new Conexion();
        Connection con = conecta.getConnection();
        Statement st;
        st=con.createStatement();
        st.executeUpdate(sql);
        con.close();
        st.close();
    }catch(Exception ex){
        System.out.println("Error en el ingreso nuevo: "+ex.getMessage());
    }
}
```

Ahora desarrollamos el método para hacer una modificación en un empleado:

```
public void modifica(Empleado e){
    String sql;

    sql = "update empleados set nombre = '"+e.getNombre()+
        "',apellidos = '"+e.getApellidos()+
        "',dni = '"+e.getDni()+
        "',sueldo = '"+e.getSueldo()+
        "',area = '"+e.getArea()+
        "'where dni = '"+e.getDni()+"'";

    try{
        conecta = new Conexion();
        Connection con = conecta.getConnection();
        Statement st;
        st=con.createStatement();
        st.executeUpdate(sql);
        con.close();
        st.close();
    }catch(Exception ex){
        System.out.println("Error en la modificacion: "+ex.getMessage());
    }
}
```

Ahora desarrollamos el método para hacer una eliminación de un empleado:

```
public void elimina(String dni){
    String sql;

    sql = "delete from empleados where dni = '"+dni+"'";

    try{
        conecta = new Conexion();
        Connection con = conecta.getConnection();
        Statement st;
        st=con.createStatement();
        st.executeUpdate(sql);
        con.close();
        st.close();
    }catch(Exception ex){
        System.out.println("Error en la eliminacion: "+ex.getMessage());
    }
}
```

Ahora desarrollamos el método para hacer una consulta de un empleado:

```
public Empleado consulta(String dni){
    String sql;
    sql = "select * from empleados where dni = '"+dni+"'";
    try{
        conecta = new Conexion();
        Connection con = conecta.getConnection();
        Statement st;
        st=con.createStatement();
        ResultSet rs= st.executeQuery(sql);
        if (rs.next()){ // lo encontré
            return new Empleado(rs.getString(1),
                                rs.getString(2),
                                rs.getString(3),
                                rs.getDouble(4),
                                rs.getInt(5));
        }
        con.close(); st.close();
    }catch(Exception ex){
        System.out.println("Error en la eliminacion: "+ex.getMessage());
    }
    return null;
}
```

Finalmente, desarrollamos el método que retorna la información de toda la tabla de empleados:


```
public String lista(){
    String sql, resultado="";
    sql = "select * from empleados";
    try{
        conecta = new Conexion();
        Connection con = conecta.getConnection();
        Statement st;
        ResultSet rs;
        st=con.createStatement();
        rs = st.executeQuery(sql);
        while (rs.next()){
            resultado += (rs.getString(1) +"\t"+
                           rs.getString(2)+"\t"+
                           rs.getString(3)+"\t"+
                           rs.getDouble(4)+"\t"+
                           rs.getInt(5)+"\n");
        }
        con.close(); st.close();
    }catch(Exception ex){
        System.out.println("Error en la eliminacion: "+ex.getMessage());
    }
    return resultado;
}
```

Ahora vamos a programar los botones del PanelEmpleados:

```
private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Mantenimiento man = new Mantenimiento();
    Empleado e = man.consulta(leeDNI());
    if (e!=null)
        mensaje("DNI repetido!");
    else{
        Empleado nuevo = new Empleado(leeNombre(),
                                         leeApellidos(),
                                         leeDNI(),
                                         leeSueldo(),
                                         leeArea());
        man.nuevo(nuevo);
        btnLista.doClick();
    }
}
```

```
private void btnConsultaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Mantenimiento man = new Mantenimiento();

    Empleado e = man.consulta(leeDNI());
    if (e==null)
        mensaje("DNI no registrado!");
    else{
        txtNombres.setText(e.getNombre());
        txtApellidos.setText(e.getApellidos());
        txtDNI.setText(e.getDni());
        txtSueldo.setText(e.getSueldo()+"");
        cboArea.setSelectedIndex(e.getArea());
    }
}
```

```
private void btnModificaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Mantenimiento man = new Mantenimiento();
    Empleado actual = man.consulta(leeDNI());
    if(actual == null)
        mensaje("DNI no registrado!");
    else{
        actual = new Empleado(leeNombre(),
                               leeApellidos(),
                               leeDNI(),
                               leeSueldo(),
                               leeArea());
        man.modifica(actual);
        btnLista.doClick();
    }
}
```

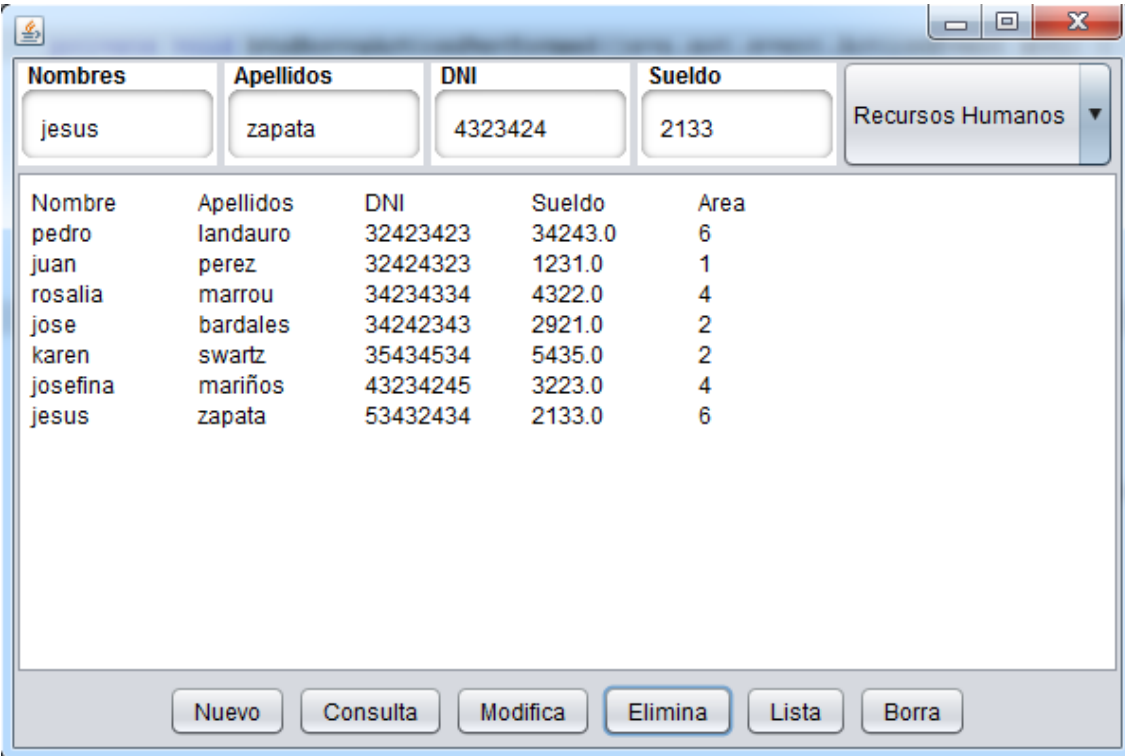
```
private void btnEliminaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Mantenimiento man = new Mantenimiento();
    Empleado actual = man.consulta(leeDNI());
    if(actual == null)
        mensaje("DNI no registrado!");
    else{
        man.elimina(leeDNI());
        btnLista.doClick();
    }
}
```

```
private void btnListaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Mantenimiento man = new Mantenimiento();
    txtSalida.setText("Nombre\tApellidos\tDNI\tSueldo\tArea\n");
    imprime(man.lista());
}
}
```

Métodos complementarios:

```
private String leeNombre(){ return txtNombres.getText();}
private String leeApellidos() { return txtApellidos.getText();}
private String leeDNI(){ return txtDNI.getText(); }
private double leeSueldo(){ return Double.parseDouble(txtSueldo.getText());}
private int leeArea(){ return cboArea.getSelectedIndex(); }
private void imprime(String s){
    txtSalida.append(s+"\n");
}
private void mensaje(String s){
    JOptionPane.showMessageDialog(this, s);
}
}
```

Ejecute su aplicación:



Nombres	Apellidos	DNI	Sueldo	Area
jesus	zapata	4323424	2133	
pedro	landauro	32423423	34243.0	6
juan	perez	32424323	1231.0	1
rosalia	marrou	34234334	4322.0	4
jose	bardales	34242343	2921.0	2
karen	swartz	35434534	5435.0	2
josefina	mariños	43234245	3223.0	4
jesus	zapata	53432434	2133.0	6

La información ya está registrada en la base de datos, puede cerrar su aplicación y volver a ejecutarlo para comprobarlo.

Añada un comboBox de reportes para mostrar la siguiente información:

- Empleados que pertenecen a un área seleccionada
- Empleados con un sueldo inferior al sueldo promedio
- Empleados en orden alfabetico por apellidos

EVALUACIÓN

EXPOSICION DE PROYECTO

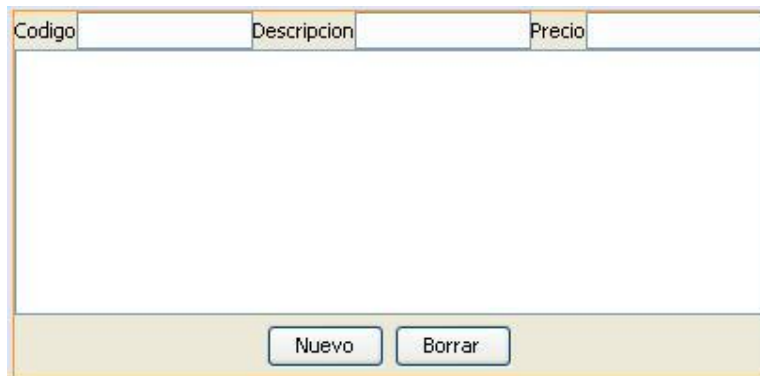
EVALUACION INTEGRAL

GUIA DE LABORATORIO 1

Encapsulamiento, diseño de clases y objetos

Ejercicio 1

Cree un proyecto nuevo de nombre **P01E01**. Cree un paquete nuevo de nombre **p01e01**. Diseñe una clase de nombre **Producto** con los siguientes atributos privados: código (cadena), descripción (cadena), precio (real) y con los métodos públicos get/set. Diseñe la GUI en un Panel de nombre **PanelPrincipal** para el ingreso de los datos con el siguiente diseño:



Cada vez que hace clic en el botón **Nuevo** debe crear un objeto de la clase **Producto** y mostrar la información de sus atributos. Cada vez que hace clic en el botón **Borrar** debe borrar la información del área de texto, de las cajas de texto y enfocar el ingreso en la caja de texto del código.

Damos doble clic en el botón **Nuevo** para programar su acción:

```

private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt) {
    Producto nuevo = new Producto();
    nuevo.setCodigo(leeCodigo());
    nuevo.setDescripcion(leeDescripcion());
    nuevo.setPrecio(leePrecio());
    lista(nuevo);
}

private String leeCodigo(){return txtCodigo.getText();}
private String leeDescripcion(){return txtDescripcion.getText();}
    
```



```
private double leePrecio(){return Double.parseDouble(txtPrecio.getText());}
private void lista(Producto p){
    imprime("Codigo\t:"+p.getCodigo());
    imprime("Descripcion\t:"+p.getDescripcion());
    imprime("Precio\t:"+p.getPrecio());
}
private void imprime(String s){
    txtSalida.append(s+"\n");
}
```

Regresamos al diseño y hacemos doble clic en el botón **Borrar** para programar su acción:

```
private void btnBorrarActionPerformed(java.awt.event.ActionEvent evt) {
    txtSalida.setText("");
    txtCodigo.setText("");
    txtDescripcion.setText("");
    txtPrecio.setText("");
    txtCodigo.requestFocus();
}
```

Desarrolle la clase de aplicación de nombre **Principal** en un **JFrame**. Coloque distribución **BorderLayout**, vaya a la ficha **Source** y escriba lo que está en negrita:

```
public class Principal extends javax.swing.JFrame {
    public Principal() {
        initComponents();
        add(new PanelPrincipal());
        setSize(600,200);
    }
}
```

Ejecute su aplicación.

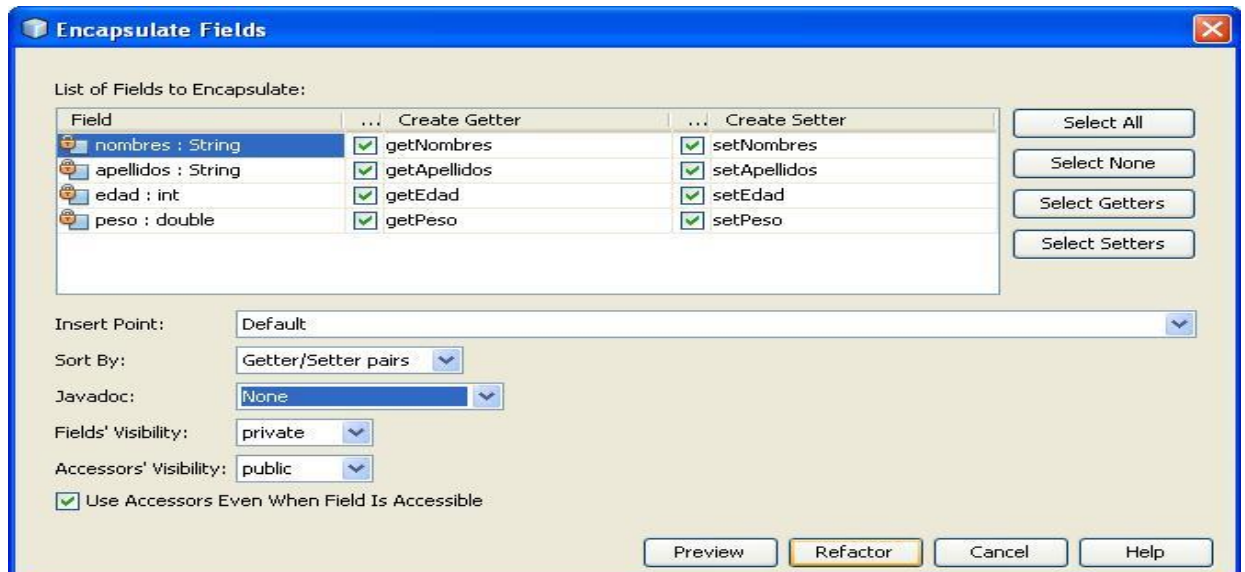
Ejercicio 2

Cree un proyecto nuevo de nombre **P01E02**. Cree un paquete nuevo de nombre **p01e02**. Diseñe una clase de nombre **Persona** con los siguientes atributos privados: nombres (cadena), apellidos (cadena), edad(entero), peso(real); con un constructor explícito y con los métodos get/set.

Clic derecho en el paquete creado, elegimos **New Java Class** y escribimos el nombre **Persona** y escribimos lo siguiente:

```
public class Persona {  
    // atributos privados  
    private String nombres, apellidos;  
    private int edad;  
    private double peso;  
  
    // constructor  
    public Persona(String nombres, String apellidos, int edad, double peso){  
        this.nombres=nombres;  
        this.apellidos=apellidos;  
        this.edad=edad;  
        this.peso=peso;  
    }  
}
```

Ahora, vamos a utilizar NetBeans para que genere, automáticamente, los métodos get/set. Clic derecho en la clase creada, elegimos **Refactor**, **Encapsulate Fields** y nos aparece la siguiente ventana donde seleccionamos todo (**Select All**).

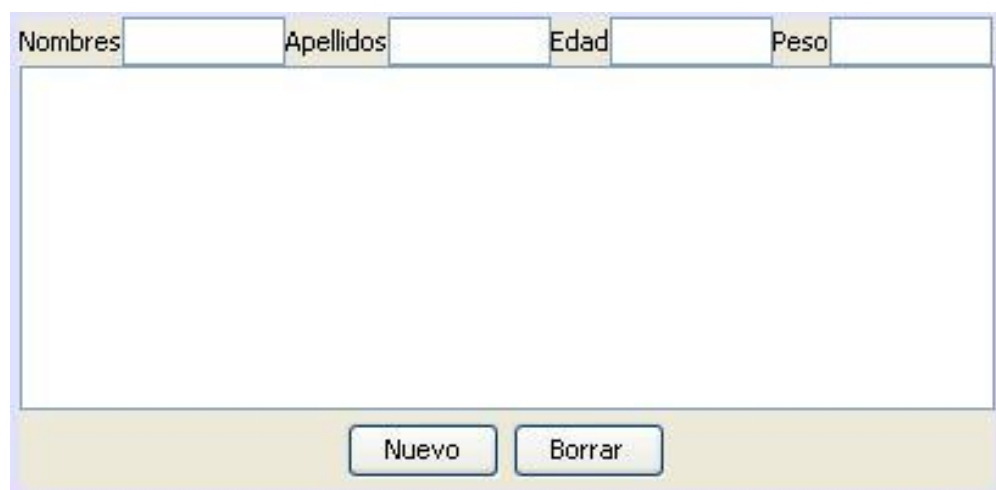


En **Javadoc** elegimos **none** para evitar los comentarios que genera NetBeans y por último hacemos clic en el botón **Refactor**. Ver los métodos get-set generados

Observe y analice el desarrollo del **constructor** y el uso de la palabra reservada **this**.

También puede generar el código del constructor y los métodos get-set a través de la opción **Insert code...** de netbeans.

Diseña la GUI en un Panel de nombre **PanelPrincipal** para el ingreso de los datos con el siguiente diseño:



Cada vez que hace clic en el botón **Nuevo** debe crear un objeto de la clase **Persona** y

mostrar la información de sus atributos. Cada vez que hace clic en el botón **Borrar** debe borrar la información del área de texto, de las cajas de texto y enfocar el ingreso en la caja de texto de nombres.

Desarrolle la clase de aplicación de nombre **Principal** en un **JFrame**

Ejecute su aplicación.

Nombres	Apellidos	Edad	Peso
Viviana	Rivasplata Peralta	26	54.8

Nuevo Borrar

Ejercicio 3

Cree un proyecto nuevo de nombre **P01E03**. Cree un paquete nuevo de nombre **p01e03**. Diseñe una clase de nombre **TV** con los siguientes atributos privados: serie (cadena), marca (entero), tamaño en pulgadas(entero), precio (real) y con los métodos get/set, con un método adicional que devuelve el precio en soles dado el tipo de cambio como parámetro, y con un método adicional que retorne el nombre de la marca. Considere las siguientes marcas: Sony, LG, Samsung, Panasonic, otro. Diseñe la GUI en un Panel de nombre **PanelPrincipal** para el ingreso de los datos con el siguiente diseño:

Serie Seleccione Marca Tamaño en Pulgadas Precio US\$

Nuevo Borrar

Cada vez que hace clic en el botón **Nuevo** debe crear un objeto de la clase **TV** y mostrar la información de sus atributos además de su precio en soles considerando un tipo de cambio actual. Cada vez que hace clic en el botón **Borrar** debe borrar la información del área de texto, de las cajas de texto y enfocar el ingreso en la caja de texto de serie.

Doble clic en el botón **Nuevo** para programar su acción:

```
private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt) {  
    TV nuevo = new TV();  
    nuevo.setSerie(leeSerie());  
    nuevo.setMarca(leeMarca());  
    nuevo.setTamaño(leeTamaño());  
    nuevo.setPrecio(leePrecio());  
    lista(nuevo);  
}  
private String leeSerie(){return txtSerie.getText();}  
private int leeMarca(){return cboMarca.getSelectedIndex();}  
private int leeTamaño(){return Integer.parseInt(txtTamaño.getText());}  
private double leePrecio(){return Double.parseDouble(txtPrecio.getText());}  
private void lista(TV t){  
    imprime("Nro. Serie\t:" + t.getSerie());  
    imprime("Marca\t:" + t.nombreMarca());  
    imprime("Tamaño\t:" + t.getTamaño());  
    imprime("Precio us$\t:" + t.getPrecio());  
    imprime("Precio S/.\t:" + t.precioSoles(2.85));  
    imprime("-----");  
}  
private void imprime(String s){  
    txtSalida.append(s + "\n");  
}  
}
```

Regrese al diseño y doble clic en el botón **Borrar** para programar su acción:

```
private void btnBorrarActionPerformed(java.awt.event.ActionEvent evt) {  
    txtSalida.setText("");  
    txtSerie.setText("");  
    cboMarca.setSelectedIndex(0);  
    txtTamaño.setText("");  
    txtPrecio.setText("");  
    txtSerie.requestFocus();  
}
```

Desarrolle la clase de aplicación de nombre **Principal** en un JFrame.

Ejecute su aplicación.

Ejercicio 4

Cree un proyecto nuevo de nombre **P01E04**. Cree un paquete nuevo de nombre **p01e04**. Diseñe una clase de nombre **Fecha** con los siguientes atributos privados: dia(entero), mes(entero), año(entero) y con los métodos get/set. Diseñe una clase de nombre **DNI** con los siguientes atributos privados: numero(cadena), dueño(Persona), fecha de emisión (Fecha), fecha de caducidad(Fecha) y con los métodos get/set.

Diseñe la GUI en un Panel de nombre **PanelPrincipal** para el ingreso de los datos de un DNI. Cada vez que hace clic en el botón Nuevo debe crear un objeto de la clase **DNI** y mostrar la información de sus atributos.

Desarrolle la clase de aplicación de nombre **Principal** en un JFrame.

Ejecute su aplicación.

SEMANA 02

GUIA DE LABORATORIO 2

Clases administradoras de objetos con ArrayList

Ejercicio 1:

Cree un proyecto nuevo de nombre P02E01. Cree un paquete nuevo de nombre p02e01. Diseñe una clase administradora para un arreglo de objetos de tipo Producto considerando la siguiente GUI:



Diseñe la clase **Producto** con constructor explícito y genere los métodos get-set:

```
public class Producto {  
    // atributos privados  
    private String codigo, descripcion;  
    private double precio;  
  
    // constructor  
    public Producto(String codigo, String descripcion, double precio){  
        this.codigo=codigo;  
        this.descripcion=descripcion;  
        this.precio=precio;  
    }  
    // métodos get-set  
}
```

Diseñe la clase **ArregloProductos**, que tenga como atributo privado un objeto de la clase **ArrayList** donde se guardarán los objetos y un constructor explícito sin parámetros.

```
public class ArregloProductos {  
    // atributos  
    // objeto ArrayList particularizado para objetos de tipo Producto  
    private ArrayList <Producto> a ;  
  
    // constructor explícito  
    public ArregloProductos(){  
        a = new ArrayList<Producto>();  
    }  
    // métodos de administración  
    public int tamaño(){ return a.size(); }  
    public void agrega(Producto p){  
        a.add(p);  
    }  
    public Producto obtiene(int i){ // retorna un producto de la posición i  
        return a.get(i);  
    }  
    public void actualiza(Producto p, int i){  
        // reemplaza un producto  
        a.set(i, p);  
    }  
    // retorna la posición de un producto según su código.  
    public int busca(String codigo){  
        for(int i=0; i<tamaño(); i++){  
            if(obtiene(i).getCodigo().equals(codigo))  
                return i;  
        }  
        return -1; // no lo encontró  
    }  
    public void elimina(int p){// elimina el producto de la posición p
```



```
        a.remove(p);
    }
    public double mayorPrecio(){// retorna el mayor precio
        double m=a.get(0).getPrecio();
        for(Producto p: a){ // for each: por cada producto p en a
            if(p.getPrecio() > m)
                m =p.getPrecio();
        }
        return m;
    }
    public double menorPrecio(){// retorna el menor precio
        double m=a.get(0).getPrecio();
        for(Producto p: a){ // for each: por cada producto p en a
            if(p.getPrecio() < m)
                m =p.getPrecio();
        }
        return m;
    }
    public double precioPromedio(){// retorna el precio promedio
        double suma=0;
        for(Producto p: a){ // for each: por cada producto p en a
            suma += p.getPrecio();
        }
        return suma / tamaño();
    }
}
// fin de la clase ArregloProductos
```

Diseñe la GUI en un Panel de nombre **PanelPrincipal** con un atributo de tipo ArregloProductos y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo producto evitando que se repita el código
- El botón Busca ubica un producto según el código ingresado y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos del producto ubicado con el botón Busca.

- El botón Elimina saca del arreglo el producto según el código ingresado.
- El botón Lista muestra una relación de todos los productos guardados en el arreglo.
- El botón Reporte muestra los precios mayor, menor y promedio.
- El botón Borrar limpia todo el contenido del GUI.

Desarrolle la clase de aplicación de nombre **Principal** en un JFrame. Ejecute su aplicación.

Ejercicio 2:

Cree un proyecto nuevo de nombre P02E02. Cree un paquete nuevo de nombre p02e02. Diseñe una clase administradora para un arreglo de objetos de tipo Persona considerando la siguiente GUI:



Diseñe la clase **Persona** con un constructor explícito:

```

public class Persona {
    // atributos privados
    private String dni, nombres, apellidos;
    private int edad;
    private double peso;

    // constructor explícito
    public Persona(String dni, String nombres, String apellidos, int edad,
        double peso){
    
```

```
        this.dni=dni;
        this.nombres=nombres;
        this.apellidos=apellidos;
        this.edad=edad;
        this.peso=peso;
    }
    // métodos get-set
}
```

Diseñe la clase administradora **ArregloPersonas**, que tenga como atributo privado un objeto de la clase **ArrayList** particularizado para la clase **Persona**, donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- Un método que retorne el peso mayor de todas las personas.
- Un método que retorne el peso menor de todas las personas.
- Un método que retorne el peso promedio de todas las personas.
- Un método que retorne la cantidad de personas cuyo peso se encuentra en un rango dado como parámetros
- Un método que retorne la cantidad de personas mayores de edad.
- Un método que retorne la cantidad de personas menores de edad.

Diseñe la GUI en un Panel de nombre **PanelPrincipal** con un atributo de tipo **ArregloPersonas** y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar una nueva persona evitando que se repita el dni
- El botón Busca ubica una persona según el dni ingresado y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos de la persona ubicada con el botón Busca.
- El botón Elimina saca del arreglo la persona según el dni ingresado.
- El botón Lista muestra una relación de todas las personas guardadas en el arreglo.
- El botón Reporte muestra el resultado de todos los métodos adicionales.
- El botón Borrar limpia todo el contenido del GUI.

Desarrolle la clase de aplicación de nombre **Principal** en un JFrame. Ejecute su aplicación.

Ejercicio 3

Cree un proyecto nuevo de nombre P02E03. Cree un paquete nuevo de nombre p02e03. Diseñe una clase administradora para un arreglo de objetos de tipo TV. Considere una GUI adecuada para los datos de la clase TV.

Diseñe la clase TV con los siguientes atributos: serie (cadena), marca (entero), tamaño en pulgadas (entero), precio en dólares (real). Considere un constructor explícito y sus métodos get/set. Considere un método adicional que devuelve el precio en soles dado el tipo de cambio como parámetro, y un método adicional que retorne el nombre de la marca. Considere las siguientes marcas: Sony, LG, Samsung, Panasonic, otro.

Diseñela clase administradora **ArregloTV**, que tenga como atributo privado un objeto de la clase **ArrayList** particularizado para la clase TV donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- a) Un método que retorne la cantidad de televisores que pertenecen a una marca específica dada como parámetro.
- b) Un método que retorne el precio promedio de los televisores de una marca específica dada como parámetro.
- c) Un método que modifique el precio de todos los televisores incrementándolo en un porcentaje dado como parámetro.

Diseñe la GUI en un Panel de nombre **PanelPrincipal** con un atributo de tipo ArregloTV y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo televisor evitando que se repita la serie.
- El botón Busca ubica un televisor según la serie ingresada y muestra sus datos

donde corresponda.

- El botón **Modifica** reemplaza los datos del televisor ubicado con el botón **Busca**.
- El botón **Elimina** saca del arreglo el televisor según la serie ingresada.
- El botón **Lista** muestra una relación de todos los televisores guardados en el arreglo.
- El botón **Reporte** muestra el resultado de todos los métodos adicionales.
- El botón **AumentaPrecio** aplica el resultado del método c) y muestra el resultado del botón **Lista**.
- El botón **Borrar** limpia todo el contenido del GUI.

Desarrolle la clase de aplicación de nombre **Principal** en un JFrame. Ejecute su aplicación.

Ejercicio 4

Modifique el contenido de la clase **ArregloProductos**, para que considere los siguientes métodos adicionales:

- a) Incremente el precio de todos los productos en 15%.
- b) Incremente el precio de los productos cuyo precio actual sea inferior a un valor dado como parámetro.
- c) Disminuya el precio de los productos cuyo precio actual esté en un rango dado como parámetro.
- d) Retorne la cantidad de productos cuyo precio sea inferior al precio promedio.
- e) Retorne en un arreglo los productos cuya descripción empiece con una letra dada como parámetro.

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para llamar a los métodos adicionales desarrollados.

Ejercicio 5

Modifique el contenido de la clase **ArregloPersonas**, para que considere los siguientes métodos adicionales:

- a) Incremente el peso de todas las personas en 5%
- b) Incremente el peso de las personas cuyo peso actual sea inferior a un valor dado como parámetro.
- c) Disminuya el peso de las personas cuyo peso actual esté en un rango dado como parámetro.
- d) Retorne la cantidad de personas menores de edad
- e) Retorne en un arreglo las personas cuya edad sea superior a un valor dado como parámetro.

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para llamar a los métodos adicionales desarrollados.

Ejercicio 6

Modifique el contenido de la clase **ArregloTV**, que considere los siguientes métodos adicionales:

- a) Incremente el precio de todos los televisores en 8%
- b) Incremente el precio de los televisores de una marca dada como parámetro en un porcentaje también dado como parámetro
- c) Disminuya el precio de los televisores cuyo tamaño actual esté en un rango dado como parámetro
- d) Retorne la cantidad de televisores de una marca y tamaño dado como parámetros.
- e) Retorne en un arreglo los televisores de una marca dada como parámetro.

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para llamar a los métodos adicionales desarrollados.

GUIA DE LABORATORIO 3

Clases administradoras de objetos con LinkedList

Ejercicio 1:

Cree un proyecto nuevo de nombre P03E01. Cree un paquete nuevo de nombre p03e01. Diseñe una clase administradora para una lista tipo pila de objetos de tipo Producto considerando la siguiente GUI:



Diseñe la clase **Producto** con constructor explícito y genere los métodos get-set:

```
public class Producto {
    // atributos privados
    private String codigo, descripcion;
    private double precio;
    // constructor
    public Producto(String codigo, String descripcion, double precio){
        this.codigo=codigo;
        this.descripcion=descripcion;
        this.precio=precio;
    }
    // métodos get-set
}
```

Diseñe la clase **ListaPilaProductos**, que tenga como atributo privado un objeto de la clase **LinkedList** donde se guardarán los objetos y un constructor explícito sin

parámetros.

```
public class ListaPilaProductos {  
    // atributos  
    // objeto LinkedList particularizado para objetos de tipo Producto  
    private LinkedList <Producto> pila ;  
  
    // constructor explícito  
    public ListaProductos(){  
        pila = new LinkedList<Producto>();  
    }  
    // métodos de administración  
    public int tamaño(){ return pila.size(); }  
    public void agrega(Producto p){  
        pila.addFirst(p);  
    }  
    public Producto obtiene(int i){ // retorna un producto de la posición i  
        return pila.get(i);  
    }  
    // retorna la posición de un producto según su código.  
    public int busca(String codigo){  
        for(int i=0; i<tamaño(); i++){  
            if(obtiene(i).getCodigo().equals(codigo))  
                return i;  
        }  
        return -1; // no lo encontró  
    }  
    public void elimina(){// elimina el primer producto de la pila  
        pila.removeFirst();  
    }  
    public double mayorPrecio(){// retorna el mayor precio  
        double m=pila.get(0).getPrecio();  
        for(Producto p: pila){ // for each: por cada producto p en pila  
            if(p.getPrecio() > m)
```



```
        m =p.getPrecio();
    }
    return m;
}

public double menorPrecio(){// retorna el menor precio
    double m=pila.get(0).getPrecio();
    for(Producto p: pila){ // for each: por cada producto p en pila
        if(p.getPrecio() < m)
            m =p.getPrecio();
    }
    return m;
}

public double precioPromedio(){// retorna el precio promedio
    double suma=0;
    for(Producto p: pila){ // for each: por cada producto p en pila
        suma += p.getPrecio();
    }
    return suma / tamaño();
}

} // fin de la clase ListaPilaProductos
```

Diseñe la GUI en un Panel de nombre **PanelPrincipal** con un atributo de tipo ListaPilaProductos y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo producto al principio de la lista, evitando que se repita el código
- El botón Busca ubica un producto según el código ingresado y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos del producto ubicado con el botón Busca.
- El botón Elimina saca de la lista el primer producto.
- El botón Lista muestra una relación de todos los productos guardados en la lista.
- El botón Reporte muestra los precios mayor, menor y promedio.
- El botón Borrar limpia todo el contenido del GUI.

Desarrolle la clase de aplicación de nombre **Principal** en un JFrame. Ejecute su

aplicación.

Ejercicio 2:

Cree un proyecto nuevo de nombre P03E02. Cree un paquete nuevo de nombre p03e02. Diseñe una clase administradora para una lista tipo cola de objetos tipo Persona considerando la siguiente GUI:



Diseñe la clase **Persona** con un constructor explícito:

```

public class Persona {
    // atributos privados
    private String dni, nombres, apellidos;
    private int edad;
    private double peso;
    // constructor explícito
    public Persona(String dni, String nombres, String apellidos, int edad,
double peso){
        this.dni=dni;
        this.nombres=nombres;
        this.apellidos=apellidos;
        this.edad=edad;
        this.peso=peso;
    }
    // métodos get-set
}
    
```

Diseñe la clase administradora **ListaColaPersonas**, que tenga como atributo privado

un objeto de la clase **LinkedList** particularizado para la clase **Persona**, donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- a) Un método que retorne el peso mayor de todas las personas.
- b) Un método que retorne el peso menor de todas las personas.
- c) Un método que retorne el peso promedio de todas las personas.
- d) Un método que retorne la cantidad de personas cuyo peso se encuentra en un rango dado como parámetros
- e) Un método que retorne la cantidad de personas mayores de edad.
- f) Un método que retorne la cantidad de personas menores de edad.

Diseñe la GUI en un Panel de nombre **PanelPrincipal** con un atributo de tipo **ListaColaPersonas** y programe la acción de los botones de acuerdo a lo siguiente:

- El botón **Nuevo** permite registrar una nueva persona al final de la lista, evitando que se repita el dni
- El botón **Busca** ubica una persona según el dni ingresado y muestra sus datos donde corresponda.
- El botón **Modifica** reemplaza los datos de la persona ubicada con el botón **Busca**.
- El botón **Elimina** saca de la cola la primera persona.
- El botón **Lista** muestra una relación de todas las personas guardadas en la lista.
- El botón **Reporte** muestra el resultado de todos los métodos adicionales.
- El botón **Borrar** limpia todo el contenido del GUI.

Desarrolle la clase de aplicación de nombre **Principal** en un **JFrame**. Ejecute su aplicación.

Ejercicio 3

Cree un proyecto nuevo de nombre **P03E03**. Cree un paquete nuevo de nombre **p03e03**. Diseñe la clase **Empleado** con los siguientes atributos: código(cadena), nombre(cadena), sueldo(real).

Diseñe la clase **ListaDobleEmpleados** con los siguientes atributos: un objeto lista de tipo **LinkedList**. También considere los siguientes métodos de administración: **agregaAlInicio**, **agregaAlFinal()**, **elimina()**, **busca()**, **obtiene()**, **tamaño()**.

```
public class ListaDobleEmpleados{
    private LinkedList<Empleado> lista;

    public ListaDobleEmpleados(){
        lista = new LinkedList<Empleado>();
    }

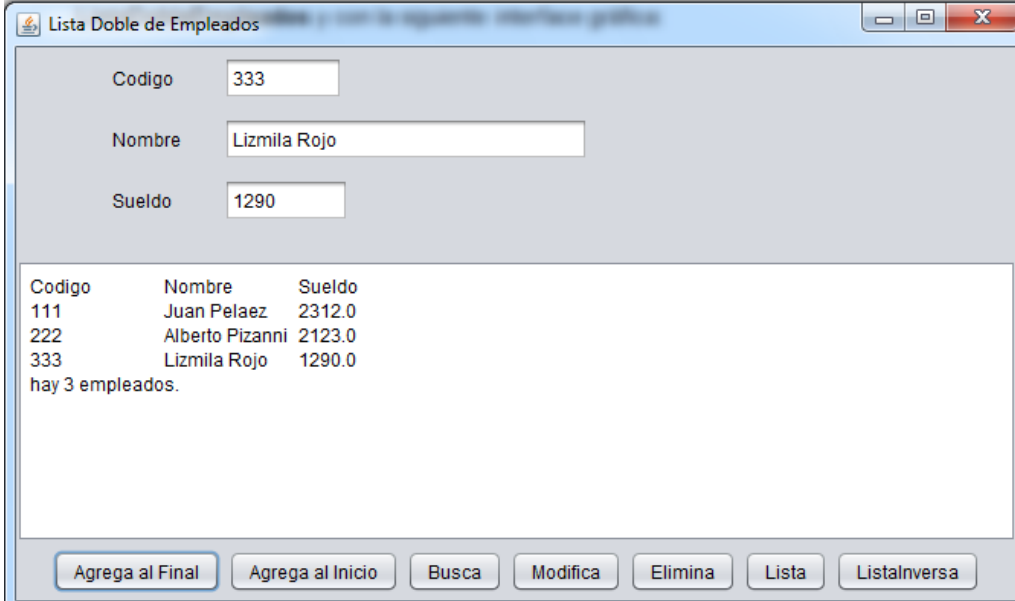
    // métodos de administración
    public int tamaño(){ return lista.size(); }
    public Empleado obtiene(int i){ return lista.get(i);}

    public void agregaAlFinal(Empleado e){
        lista.addLast(e);
    }
    public int busca(String codigo){
        for(Empleado e: lista){
            if(e.getCodigo().equalsIgnoreCase(codigo))
                return lista.indexOf(e);
        }
        return -1;
    }
    public void elimina(int n){
        lista.remove(n);
    }
    public void elimina(Empleado e){
        lista.remove(e);
    }

    public LinkedList<Empleado> getLista() {
        return lista;
    }

    public void setLista(LinkedList<Empleado> lista) {
        this.lista = lista;
    }
}
```

Diseñe la clase **PanelEmpleados** con el siguiente atributo: un objeto **Id** de tipo **ListaDobleEmpleados** y con la siguiente interface gráfica:



Codigo	Nombre	Sueldo
111	Juan Pelaez	2312.0
222	Alberto Pizanni	2123.0
333	Lizmila Rojo	1290.0

hay 3 empleados.

```

public class PanelPrincipal extends javax.swing.JPanel {
    protected ListaDobleEmpleados lde;
    /** Creates new form PanelPrincipal */
    public PanelPrincipal() {
        initComponents();
        lde = new ListaDobleEmpleados();
    }
}
    
```

Programe la acción de los botones.

```

private void btnAgregaFinalActionPerformed(java.awt.event.ActionEvent evt) {
    int p = lde.busca(leeCodigo());
    if(p!=-1){
        Empleado n = new Empleado(leeCodigo(), leeNombre(), leeSueldo());
        lde.agregaAlFinal(n);
        lista();
    }else{
        mensaje("Codigo repetido!");
    }
}

private void btnEliminaActionPerformed(java.awt.event.ActionEvent evt) {
    int p = lde.busca(leeCodigo());
    if(p!=-1){
        mensaje("Codigo no registrado!");
    }else{
        lde.elimina(p);
        lista();
    }
}
    
```

```

public void lista() {
    txtSalida.setText("Codigo\tNombre\t\tSueldo\n");
    for(Empleado aux:lde.getList()) {
        imprime(aux.getCodigo()+"\t"+
            aux.getNombre()+"\t\t"+
            aux.getSueldo());
    }
    imprime("Hay "+lde.getNodos()+" empleados.");
}

public void imprime(String s) {
    txtSalida.append(s+"\n");
}

public void mensaje(String s) {
    JOptionPane.showMessageDialog(this,s);
}

private void btnListaInversaActionPerformed(java....){
    for (int i=lde.tamaño()-1 ; i>=0; i--){
        Empleado aux = lde.obtiene(i);
        Imprime(aux.getCodigo()+"\t" +
            aux.getNombre()+"\t\t" +
            aux.getSueldo() );
    }
    Imprime("Hay " + lde.tamaño() + " empleados. ");
}

```

Desarrolle la clase de aplicación de nombre **Principal** en un JFrame. Ejecute su aplicación.

Ejercicio 4

Cree un proyecto nuevo de nombre P03E04. Cree un paquete nuevo de nombre p03e04. Diseñe la clase TV con los siguientes atributos: serie (cadena), marca (entero), tamaño en pulgadas (entero), precio en dólares (real). Considere un constructor explícito y sus métodos get/set. Considere un método adicional que devuelve el precio en soles dado el tipo de cambio como parámetro, y un método adicional que retorne el nombre de la marca. Considere las siguientes marcas: Sony, LG, Samsung, Panasonic, otro.

Diseñela clase administradora **ListaPilaTV**, que tenga como atributo privado un objeto de la clase **LinkedList** particularizado para la clase TV donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- a) Un método que retorne la cantidad de televisores que pertenecen a una marca específica dada como parámetro.
- b) Un método que retorne el precio promedio de los televisores de una marca específica dada como parámetro.
- c) Un método que modifique el precio de todos los televisores incrementándolo en un porcentaje dado como parámetro.

Diseñe la GUI Principal donde considere como atributo un objeto de tipo ListaPilaTV y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo televisor evitando que se repita la serie.
- El botón Busca ubica un televisor según la serie ingresada y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos del televisor ubicado con el botón Busca.
- El botón Elimina saca al primer televisor de la lista.
- El botón Lista muestra una relación de todos los televisores guardados en la lista.
- El botón Reporte muestra el resultado de todos los métodos adicionales.
- El botón AumentaPrecio aplica el resultado del método c) y muestra el resultado del botón Lista.
- El botón Borrar limpia todo el contenido del GUI.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

EVALUACION

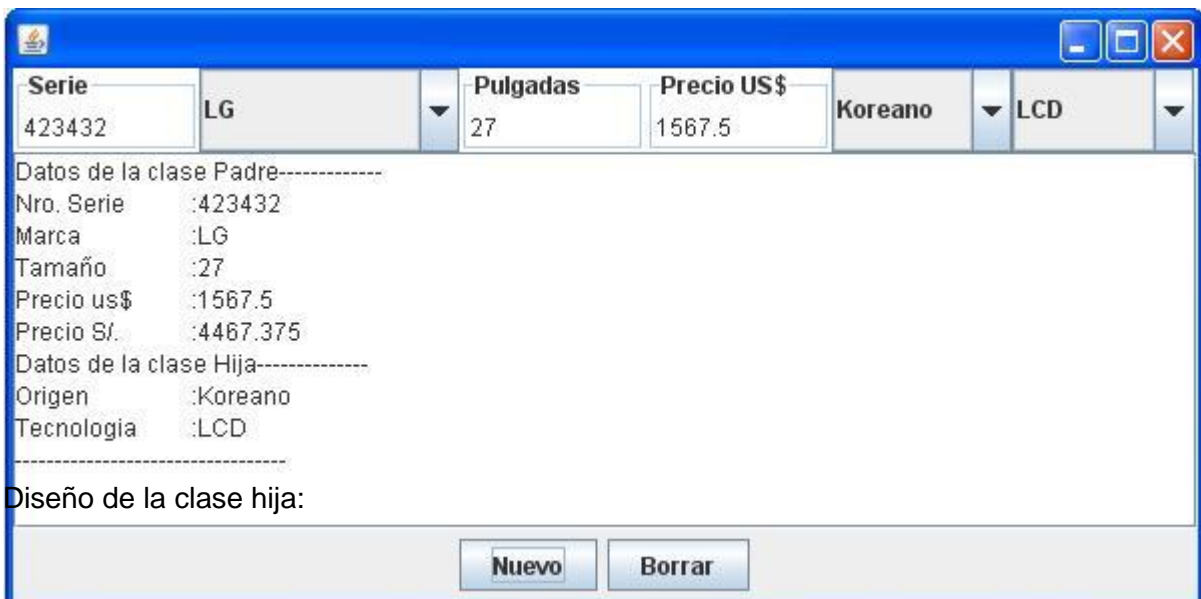
GUIA DE LABORATORIO 5

Herencia

Ejercicio 1

Cree un proyecto nuevo de nombre P05E01. Cree un paquete nuevo de nombre p05e01. Considere la existencia de la clase **TV** desarrollada anteriormente y diseñe una nueva clase hija de nombre **TVH**, aplicando herencia, considerando los siguientes atributos protegidos adicionales: origen(entero), tecnología(entero). Considere para el campo origen: nacional, americano, japonés, coreano, chino, otro. Considere para el campo tecnología: Tradicional, LCD, Plasma, Digital. Considere métodos adicionales para que retornen el nombre del origen y el nombre de la tecnología.

La nueva clase hija debe ser utilizada para el proceso de la siguiente GUI:



The screenshot shows a Java Swing window with a blue title bar. It contains a table with the following data:

Serie	LG	Pulgadas	Precio US\$	Koreano	LCD
423432		27	1567.5		

Below the table, there are two sections of text:

Datos de la clase Padre-----
 Nro. Serie :423432
 Marca :LG
 Tamaño :27
 Precio us\$:1567.5
 Precio S/. :4467.375

Datos de la clase Hija-----
 Origen :Koreano
 Tecnologia :LCD

At the bottom of the window, there are two buttons: 'Nuevo' and 'Borrar'.

```
public class TVH extends TV {
```

```
    // atributos protegidos
```

```
    protected int origen, tecnologia;
```

```
    // constructor explícito
```

```
public TVH(String serie, int marca, int tamaño, double precio, int
    origen, int tecnologia){
    // invocación al constructor de la clase Padre
    super(serie, marca, tamaño, precio);

    // inicializa sus propios atributos
    this.origen = origen;
    this.tecnologia=tecnologia;
}

// métodos get-set
public int getOrigen() {
    return origen;
}
public void setOrigen(int origen) {
    this.origen = origen;
}
public int getTecnologia() {
    return tecnologia;
}
public void setTecnologia(int tecnologia) {
    this.tecnologia = tecnologia;
}

// metodos adicionales
public String nombreOrigen(){
    switch(origen){
        case 1: return "Nacional";
        case 2: return "Americano";
        case 3: return "Japonés";
        case 4: return "Koreano";
        case 5: return "Chino";
        default: return "Otro";
    }
}
```

```
public String nombreTecnologia(){
    switch(tecnologia){
        case 1: return "Tradicional";
        case 2: return "LCD";
        case 3: return "Plasma";
        default: return "Digital";
    }
}
```

}// fin de la clase hija

Diseñe la GUI Principal donde considere como atributo un objeto de tipo TVH y programe la acción de los botones de acuerdo a lo siguiente:

```
private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt) {
    TVH nuevo = new TVH(leeSerie(),leeMarca(),leeTamaño(),leePrecio(),
                        leeOrigen(), leeTecnologia());

    lista(nuevo);
}

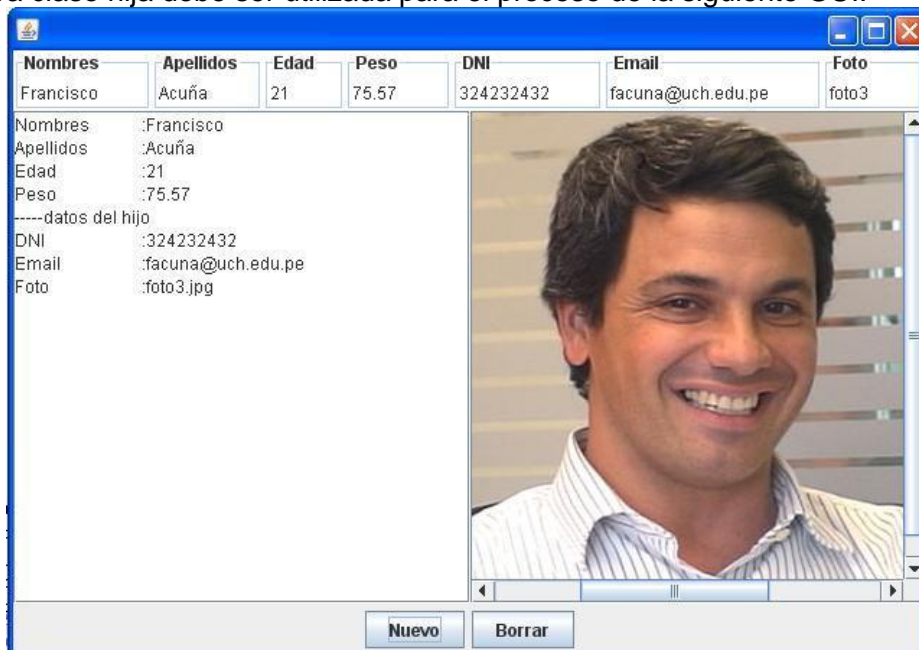
private void lista(TVH t){
    imprime("Datos de la clase Padre-----");
    imprime("Nro. Serie\t:"+t.getSerie());
    imprime("Marca\t:"+t.nombreMarca());
    imprime("Tamaño\t:"+t.getTamaño());
    imprime("Precio us$\t:"+t.getPrecio());
    imprime("Precio S/.\t:"+t.precioSoles(2.85));
    imprime("Datos de la clase Hija-----");
    imprime("Origen\t:"+t.nombreOrigen());
    imprime("Tecnologia\t:"+t.nombreTecnologia());
    imprime("-----");
}
}
```

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 2

Cree un proyecto nuevo de nombre P05E02. Cree un paquete nuevo de nombre p05e02. Considere la existencia de una clase base de nombre **Persona** ya desarrollada anteriormente y diseñe una nueva clase hija de nombre **PersonaH**, aplicando herencia, con los siguientes atributos protegidos adicionales: dni (cadena), email(cadena), foto(cadena).

La nueva clase hija debe ser utilizada para el proceso de la siguiente GUI:



Diseño de la clase hija:

```
public class PersonaH extends Persona {
    // atributos protegidos
    protected String dni, email, foto;

    // constructor
    public PersonaH(String nombres, String apellidos, int edad, double peso,
                    String dni, String email, String foto){
        super(nombres,apellidos,edad,peso);
        this.dni=dni;
        this.email =email;
        this.foto =foto;
    }
}
```

Diseñe la GUI Principal donde considere como atributo un objeto de tipo PersonaH y programe la acción de los botones.

Doble clic en el botón **Nuevo** para programar su acción.

```
private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt) {  
    PersonaH nuevo = new PersonaH(leeNombres(), leeApellidos(), leeEdad(),  
                                    leePeso(), leeDNI(), leeEmail(), leeFoto());  
    lista(nuevo);  
}  
  
private void lista(PersonaH p){  
    imprime("Nombres\t:" + p.getNombres());  
    imprime("Apellidos\t:" + p.getApellidos());  
    imprime("Edad\t:" + p.getEdad());  
    imprime("Peso\t:" + p.getPeso());  
    imprime("-----datos del hijo");  
    imprime("DNI\t:" + p.getDni());  
    imprime("Email\t:" + p.getEmail());  
    imprime("Foto\t:" + p.getFoto());  
    lblImagen.setIcon(new  
        ImageIcon(getClass().getResource("/fotos/" + p.getFoto())));  
}
```

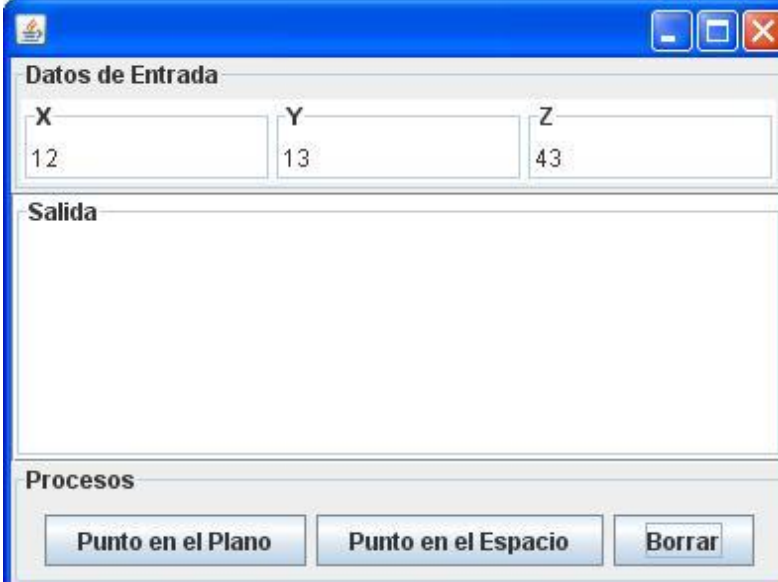
Doble clic en el botón **Borrar** para programar su acción.

```
private void btnBorrarActionPerformed(java.awt.event.ActionEvent evt) {  
    txtSalida.setText("");  
    txtNombres.setText("");  
    txtApellidos.setText("");  
    txtEdad.setText("");  
    txtPeso.setText("");  
    txtNombres.requestFocus();  
    lblImagen.setIcon(null);  
}
```

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 3

Cree un proyecto nuevo de nombre P05E03. Cree un paquete nuevo de nombre p05e03. Diseñe una clase padre de nombre **PuntoP** cuyos atributos sean las coordenadas **x,y** de su ubicación en el plano. Considere un constructor con parámetros, la funcionalidad de acceso a sus atributos y un método adicional que retorne la distancia desde su ubicación hasta el punto de origen. Luego, diseñe una clase hija de nombre **PuntoE** que **herede** a la clase **PuntoP** y considere un atributo adicional para la coordenada **z** de su ubicación en el espacio. Considere un constructor con parámetros, la funcionalidad de acceso a su atributo y un método adicional que retorne la distancia desde su ubicación hasta el punto de origen. Considere una clase de GUI donde utilice objetos de ambas clases para mostrar su información.



Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 4

Cree un proyecto nuevo de nombre P05E04. Cree un paquete nuevo de nombre p05e04. Diseñe la clase **ListaPilaTVH**, que administre una lista tipo pila de objetos tipo TVH y que considere los siguientes métodos adicionales:

- Incrementa el precio de todos los televisores en 8%
- Incrementa el precio de los televisores de una marca dada como parámetro en un porcentaje también dado como parámetro

- c) Disminuya el precio de los televisores cuyo tamaño actual esté en un rango dado como parámetro
- d) Retorne la cantidad de televisores de una marca y tamaño dado como parámetros.
- e) Retorne en un arreglo los televisores de una marca dada como parámetro.

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para incorporar a los métodos adicionales desarrollados.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 5

Cree un proyecto nuevo de nombre P05E05. Cree un paquete nuevo de nombre p05e05. Diseñe la clase **ListaColaPersonasH**, que administre una lista tipo cola de objetos tipo **PersonaH** y que considere los siguientes métodos adicionales:

- a) Incremente el peso de todas las personas en 5%
- b) Incremente el peso de las personas cuyo peso actual sea inferior a un valor dado como parámetro.
- c) Disminuya el peso de las personas cuyo peso actual esté en un rango dado como parámetro.
- d) Retorne la cantidad de personas menores de edad
- e) Retorne en un arreglo las personas cuya edad sea superior a un valor dado como parámetro.

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para incorporar a los métodos adicionales desarrollados.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

GUIA DE LABORATORIO 6

Herencia con archivos de texto

Ejercicio 1

Cree un proyecto nuevo de nombre P06E01. Cree un paquete nuevo de nombre p06e01. Considere la existencia de una clase administradora de nombre **ListaPilaTVH** desarrollada anteriormente. Aplique herencia y desarrolle la clase hija **ArchivoTVH** donde se consideren métodos para leer y para grabar en un archivo de texto.

```
public class ArchivoTVH extends ListaPilaTVH {  
    // atributos protegidos  
    protected String nombre;  
  
    // constructor  
    public ArchivoTVH(String nombre){  
        super();  
        this.nombre=nombre;  
        lee();  
    }  
  
    // métodos que operan un archivo de texto  
    public void lee(){  
        try{  
            FileReader fr = new FileReader(nombre);  
            BufferedReader br = new BufferedReader(fr);  
            String linea=br.readLine();  
            while(linea != null){  
                StringTokenizer st = new StringTokenizer(linea,"/");  
                String serie=st.nextToken();  
                int marca=Integer.parseInt(st.nextToken());
```



```
        int tamaño=Integer.parseInt(st.nextToken());
        double precio=Double.parseDouble(st.nextToken());
        int origen=Integer.parseInt(st.nextToken());
        int tecnologia=Integer.parseInt(st.nextToken());
        agrega(serie,marca,tamaño,precio,origen,tecnologia);
        linea= br.readLine();
    }
    br.close();
} catch(Exception ex){ }
}

public void graba(){
    try{
        FileWriter fw = new FileWriter(nombre);
        PrintWriter pw= new PrintWriter(fw);
        for(TVH t : pila){
            pw.println(t.getSerie()+"/"+
                        t.getMarca()+"/"+
                        t.getTamaño()+"/"+
                        t.getPrecio()+"/"+
                        t.getOrigen()+"/"+
                        t.getTecnologia());
        }
        pw.close();
    } catch(Exception ex){ }
}

} // fin de la clase
```

Considere el mismo diseño de GUI e implemente, lo necesario para la programación de los botones.



Serie	Seleccione Marca	Pulgadas	Precio US\$	Origen	Tecnología
12112	Sony	12	2112.0	Americano	Tradicional
2332	Samsung	43	2343.0	Japonés	LCD

Nuevo Lista Consulta Elimina Reporte Borrar

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 2

Cree un proyecto nuevo de nombre P06E02. Cree un paquete nuevo de nombre p06e02. Considere la existencia de una clase administradora de nombre **ListaColaPersonasH** desarrollada anteriormente. Aplique herencia y desarrolle la clase hija **ArchivoPersonasH** donde se consideren métodos para leer y para grabar en un archivo de texto.

Considere el mismo diseño de GUI



Nombres Apellidos Edad Peso DNI Email Foto

Nuevo Lista Consulta Elimina Reporte Borrar

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para la programación de los botones.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

- Incorpore métodos sobrecargados en las clases administradoras.

EVALUACION

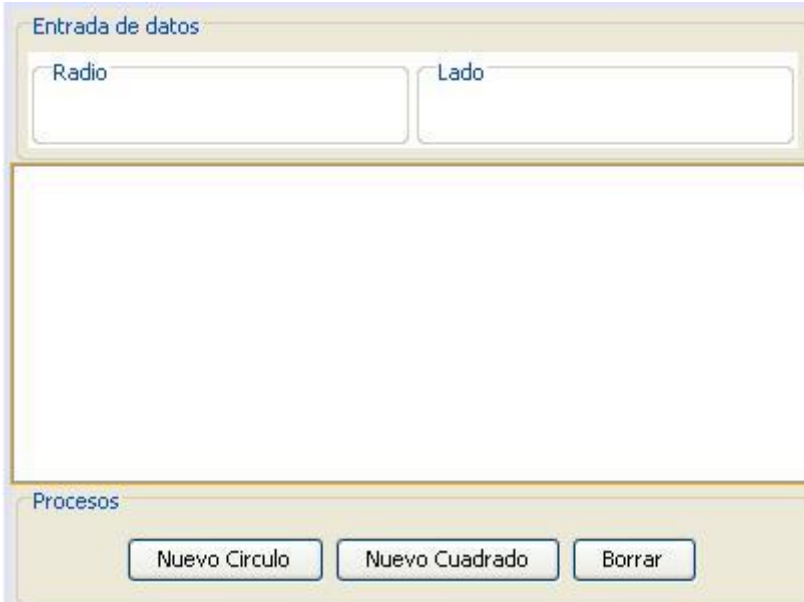
EVALUACION INTEGRAL

GUIA DE LABORATORIO 9**Polimorfismo****Ejercicio 1**

Cree un proyecto nuevo de nombre P09E01. Cree un paquete nuevo de nombre p09e01. Diseñe una clase abstracta de nombre **Figura** que tenga un atributo protegido para el nombre de la figura, métodos get-set, métodos abstractos para el `area()` y el `perimetro()` y un método no abstracto que retorne la información correspondiente al nombre, area y perímetro de cualquier figura.

Aplique polimorfismo y diseñe una clase hija de nombre **Circulo** que tenga un atributo para el radio, constructor, métodos get-set y desarrollo de los métodos abstractos de la clase padre. Luego diseñe una clase hija de nombre **Cuadrado** que tenga un atributo para el lado, constructor, métodos get-set, y desarrollo de los métodos abstractos de la clase padre.

Finalmente, diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente figura.



Diseño de la clase abstracta **Figura**:

```
public abstract class Figura{
    protected String nombre;
    public Figura(String nombre){
        this.nombre=nombre;
    }
    // métodos abstractos
    public abstract double area();
    public abstract double perimetro();

    // método no abstracto
    public String info(){
        return "Figura\t: "+nombre+"\n" + "Area\t: "+ area()+"\n"+
            "Perímetro\t: "+perimetro();
    }
}
```

Diseño de la clase extendida Circulo:

```
public class Circulo extends Figura{
    protected double radio;
    public Circulo(double radio){
        super("Circulo");
        this.radio = radio;
    }
    // desarrollo de los métodos abstractos
    public double area(){ return Math.PI * radio * radio;}
    public double perimetro(){ return 2 * Math.PI * radio;}
}
```

Diseño de la clase extendida Cuadrado:

```
public class Cuadrado extends Figura{
    protected double lado;
    public Cuadrado(double lado){
        super("Cuadrado");
        this.lado = lado;
    }
}
```

```
    }  
    // desarrollo de los métodos abstractos  
    public double area(){ return lado * lado;}  
    public double perimetro(){ return 4*lado;}  
}
```

Implemente, en la interfaz gráfica de usuario (GUI), lo necesario para la programación de los botones.

```
Programación del botón Nuevo Circulo:{  
    Figura a = new Circulo(leeRadio());  
    lista(a);  
}
```

```
Programación del botón Nuevo Cuadrado:{  
    Figura b = new Cuadrado(leeLado());  
    lista(b);  
}
```

```
public void lista(Figura f){  
    imprime(f.info());  
}
```

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 2

Cree un proyecto nuevo de nombre P09E02. Cree un paquete nuevo de nombre p09e02. Diseñe una clase abstracta de nombre **Empleado** que tenga atributos protegidos para el nombre, apellidos, dni, métodos get-set, métodos abstractos para ingresos(), bonificaciones(), descuentos(), un método no abstracto que retorne el sueldoNeto() y otro método no abstracto que retorne la información correspondiente al nombre, apellidos, dni, ingresos, bonificaciones, descuentos, sueldo neto de cualquier empleado.

Aplique polimorfismo y diseñe una clase hija de nombre **EmpleadoVendedor** que tenga

atributos para monto vendido y porcentaje de comisión, constructor, métodos get-set, desarrollo de métodos abstractos. Luego, diseñe una clase hija de nombre **EmpleadoPermanente** que tenga atributos para sueldo basico y afiliación (afp ó snp), constructor, métodos get-set, desarrollo de métodos abstractos.

Finalmente, diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente tipo de empleado.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 3:

Cree un proyecto nuevo de nombre P09E03. Cree un paquete nuevo de nombre p09e03. Diseñe la clase abstracta **Celular** con los siguientes atributos: número, dueño, precio, marca (1=samsung, 2=htc, 3=lg, 4=otro). Considere un método abstracto para el cálculo del impuesto. Considere un método no abstracto para que retorne el número, dueño, nombre de marca y monto de impuestos. Asuma los métodos get-set, no los desarrolle.

Aplique polimorfismo y diseñe la clase hija **CelularSmart** con el siguiente atributo adicional: país de origen. Para el cálculo del monto de su impuesto considere lo siguiente: si es de marca samsung aplique 25%, si es htc aplique 20%, si es lg aplique 15% y si es otro aplique 10% sobre el precio. Luego Diseñe la clase hija **Celular4G** con el siguiente atributo adicional: operador(1=movistar, 2=claro, 3=entel, 4=otro). Para el cálculo del monto de su impuesto considere siguiente: si el operador es movistar aplique 10%, , si es claro aplique 9%, si es entel aplique 7% y si es otro aplique 5% sobre el precio.

Finalmente, diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente tipo de celular.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 4:

Cree un proyecto nuevo de nombre P09E04. Cree un paquete nuevo de nombre p09e04. Diseñe una clase abstracta de nombre **Vehiculo** con los siguientes atributos: placa, marca, precio. Con el siguiente método no abstracto: info() que retorna, en una cadena los siguientes datos tabulados: placa, marca, precio, impuesto. Considere que el monto del impuesto depende del precio y del tipo de vehículo que sea.

Aplique polimorfismo y diseñe una clase hija de nombre **Automovil** cuyo monto de su impuesto es el 13% de su precio si éste supera los 10,000 dólares y 9% en caso contrario. Diseñe otra clase hija de nombre **Camion** cuyo monto de su impuesto es el 21% de su precio si éste supera los 20,000 dólares y 7% en caso contrario.

Finalmente, diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente tipo de vehiculo.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

GUIA DE LABORATORIO 10**Polimorfismo, colecciones polimorficas****Ejercicio 1**

Cree un proyecto nuevo de nombre P10E01. Cree un paquete nuevo de nombre p10e01. Considere la existencia de la clase abstracta de nombre **Figura**, las clases hijas **Cuadrado** y **Circulo**.

Diseñe una clase administradora de nombre **ColeccionFiguras** que permita la administración de objetos de tipo Cuadrado y/o Circulo a la vez, utilizando un objeto de la clase **ArrayList**.

```
public class ColeccionFiguras{
    protected ArrayList <Figura> coleccion;

    public ArregloFiguras() {
        coleccion = new ArrayList<Figura>();
    }
    // métodos de administración
    public void agrega(Figura f){
        coleccion.add(f);
    }
    public Figura obtiene(int i){
        return coleccion.get(i);
    }

    public int tamaño(){ return coleccion.size();}

    public void elimina(int i){
        coleccion.remove(i);
    }
    //... complete métodos adicionales
}
```

```
}
```

Diseñe una clase de interfaz de nombre **PanelPrincipal** donde cree un objeto de la clase **ColeccionFiguras**, como atributo de la clase y programe la acción de los botones:

```
protected ColeccionFiguras cf = new ColeccionFiguras();
```

Programación del botón Nuevo Circulo:{

```
    Figura a = new Circulo(leeRadio());  
    lista(a);  
    cf.agrega(a); // agrega un objeto Circulo a la colección
```

```
}
```

Programación del botón Nuevo Cuadrado:{

```
    Figura b = new Cuadrado(leeLado());  
    lista(b);  
    cf.agrega(b); // agrega un objeto Cuadrado a la coleccion
```

```
}
```

```
public void lista(Figura f){  
    imprime(f.info());
```

```
}
```

Programación del botón Lista:{

```
    for(int i=0; i<cf.tamaño(); i++){  
        Figura f=cf.obtiene(i);  
        if (f instanceof Circulo)  
            imprime("Circulo: "+f.info());  
        else  
            imprime("Cuadrado: "+f.info());
```

```
    }
```

```
}
```

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 2

Cree un proyecto nuevo de nombre P10E02. Cree un paquete nuevo de nombre p10e02. Considere la existencia de la clase abstracta de nombre **Empleado**, las clases hijas **EmpleadoVendedor** y **EmpleadoPermanente**.

Diseñe una clase administradora de nombre **ColeccionEmpleados** que permita la administración de objetos de tipo **EmpleadoVendedor** y/o **EmpleadoPermanente** a la vez, utilizando un objeto de la clase **ArrayList**.

Diseñe una clase de interfaz de nombre **PanelPrincipal** donde cree un objeto de la clase **ColeccionEmpleados**, como atributo de la clase y programe la acción de los botones.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 3

Cree un proyecto nuevo de nombre P10E03. Cree un paquete nuevo de nombre p10e03. Considere la existencia de la clase abstracta de nombre **Celular**, las clases hijas **CelularSmart** y **Celular4G**.

Diseñe una clase administradora de nombre **ColeccionCelulares** que permita la administración de objetos de tipo **CelularSmart** y/o **Celular4G** a la vez, utilizando un objeto de la clase **LinkedList**.

Diseñe una clase de interfaz de nombre **PanelPrincipal** donde cree un objeto de la clase **ColeccionCelulares**, como atributo de la clase y programe la acción de los botone.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 4:

Cree un proyecto nuevo de nombre P10E04. Cree un paquete nuevo de nombre p10e04. Considere la existencia de la clase abstracta de nombre **Vehiculo**, las clases hijas **Automovil** y **Camion**.

Diseñe una clase administradora de nombre **ColeccionVehiculos** que permita la administración de objetos de tipo **Automovil** y/o **Camion** a la vez, utilizando un objeto de la clase **LinkedList**.

Diseñe una clase de interfaz de nombre **PanelPrincipal** donde cree un objeto de la clase **ColeccionVehiculos**, como atributo de la clase y programe la acción de los botones.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

EVALUACION

GUIA DE LABORATORIO 12**Polimorfismo y persistencia con archivos de texto****Ejercicio 1**

Cree un proyecto nuevo de nombre P12E01. Cree un paquete nuevo de nombre p12e01. Considere la existencia de la clase **ColeccionFiguras** desarrollada anteriormente y aplique herencia desarrollando la clase **ArchivoFiguras** que permita conservar la información de las figuras en un archivo de texto.

```
public class ArchivoFiguras extends ColeccionFiguras{
    protected String nombre;

    public ArchivoFiguras(String nombre){
        this.nombre=nombre;
        lee();
    }
    public void graba(){
        try{
            FileWriter fw = new FileWriter(nombre);
            PrintWriter pw = new FilePrinter(fw);
            for(Figura f : coleccion){
                if(f instanceof Circulo)
                    pw.println("1/"+
                                ((Circulo)(f)).getRadio() ); // molde
                else
                    pw.println("2/"+
                                ((Cuadrado)(f)).getLado() ); // molde
            }
            pw.close();
        }catch(Exception ex){
        }
    }
    public void lee(){
```



```
try{
    FileReader fr = new FileReader(nombre);
    BufferedReader br = new BufferedReader(fr);
    String linea = br.readLine();
    while(linea!=null){
        StringTokenizer st =new StringTokenizer(linea,"/");
        int tipo=Integer.parseInt(st.nextToken());
        if(tipo==1){ // circulo
            double radio=Double.parseDouble(st.nextToken());
            Figura a=new Circulo(radio);
            agrega(a);
        }else{// cuadrado

            double lado=Double.parseDouble(st.nextToken());
            Figura b=new Cuadrado(lado);
            agrega(b);
        }
        linea=br.readLine();
    }
    br.close();
}catch(Exception ex){}

}
```

Utilizando la interface que se diseñó para administrar una colección de figuras polimórficas, en un nuevo proyecto haga las modificaciones necesarias para utilizar la clase **ArchivoFiguras** en lugar de la clase ColeccionFiguras y programe la acción de los botones, así:

```
// atributo de la clase interfaz
protected ArchivoFiguras af = new ArchivoFiguras();
```

Programación del botón Nuevo Circulo:{

```
Figura a = new Circulo(leeRadio());  
lista(a);  
af.agrega(a); // agrega un objeto Circulo  
af.graba(); // graba la información en el archivo de texto  
}
```

Programación del botón Nuevo Cuadrado:{

```
Figura b = new Cuadrado(leeLado());  
lista(b);  
af.agrega(b); // agrega un objeto Cuadrado  
af.graba(); // graba la información en el archivo de texto  
}
```

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 2

Cree un proyecto nuevo de nombre P12E02. Cree un paquete nuevo de nombre p12e02. Considere la existencia de la clase **ColeccionEmpleados** desarrollada anteriormente y aplique herencia desarrollando la clase **ArchivoEmpleados** que permita conservar la información de los empleados en un archivo de texto.

Utilizando la interface que se diseñó para administrar una colección de empleados polimórficos, en un nuevo proyecto haga las modificaciones necesarias para utilizar la clase **ArchivoEmpleados** en lugar de la clase **ColeccionEmpleados** y programe la acción de los botones.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 3

Cree un proyecto nuevo de nombre P12E03. Cree un paquete nuevo de nombre p12e03. Considere la existencia de la clase **ColeccionCelulares** desarrollada anteriormente y aplique herencia desarrollando la clase **ArchivoCelulares** que permita conservar la información de los celulares en un archivo de texto.

Utilizando la interface que se diseñó para administrar una colección de celulares polimórficos, en un nuevo proyecto haga las modificaciones necesarias para utilizar la clase **ArchivoCelulares** en lugar de la clase ColeccionCelularess y programe la acción de los botones.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejercicio 4

Cree un proyecto nuevo de nombre P12E04. Cree un paquete nuevo de nombre p12e04. Considere la existencia de la clase **ColeccionVehiculos** desarrollada anteriormente y aplique herencia desarrollando la clase **ArchivoVehiculos** que permita conservar la información de los vehiculos en un archivo de texto.

Utilizando la interface que se diseñó para administrar una colección de vehiculos polimórficos, en un nuevo proyecto haga las modificaciones necesarias para utilizar la clase **ArchivoVehiculos** en lugar de la clase ColeccionVehiculos y programe la acción de los botones.

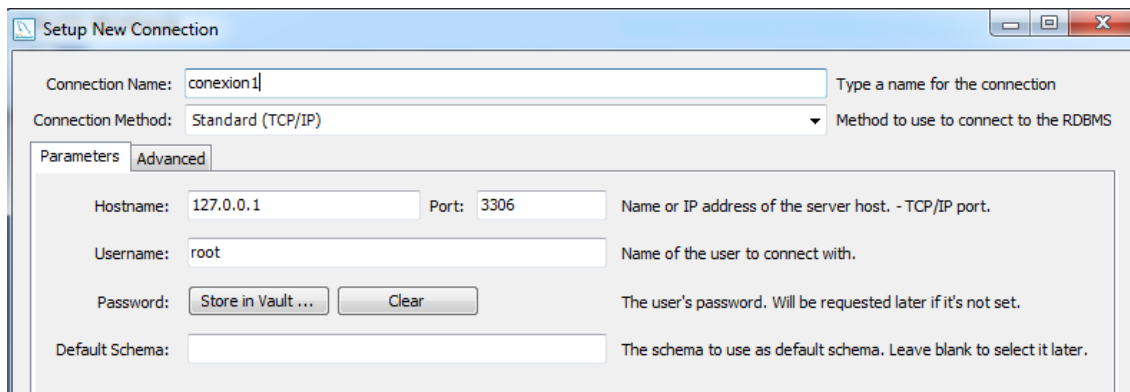
Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

GUIA DE LABORATORIO 13

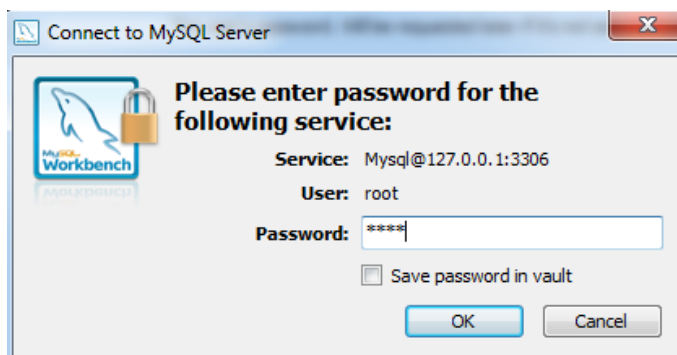
Introducción a bases de datos

Creación de una Base de Datos

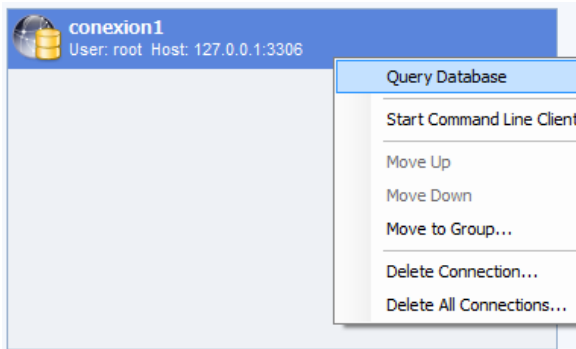
Vamos a crear nuestra base de datos en el gestor de base de datos MySQL WorkBench. Abrimos Workbeanch, elegimos **New Conexion** para llegar a la siguiente ventana:



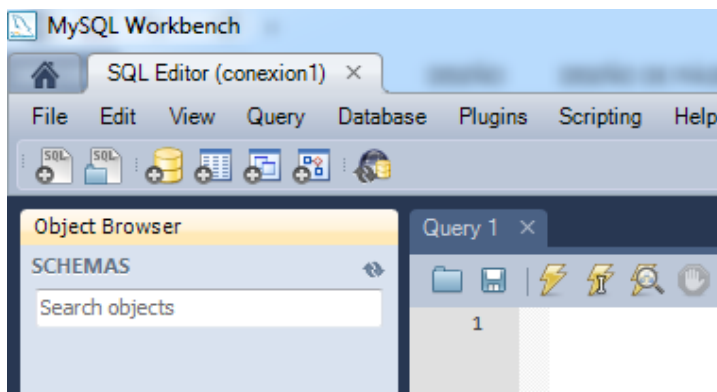
Escribimos el nombre de la nueva conexión (conexion1) y damos clic en el botón Test connection para llegar a la siguiente ventana:



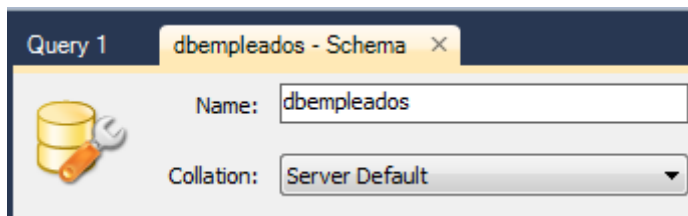
Damos el password (root) y presionamos el botón OK para llegar a la ventana de conexión con éxito y luego damos OK nuevamente para que quede establecida nuestra nueva conexión.



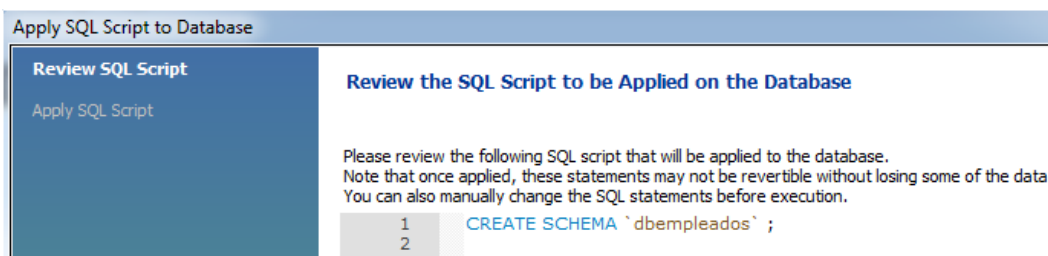
Damos clic derecho en la nueva conexión creada y elegimos Query Database para llegar a la siguiente ventana:



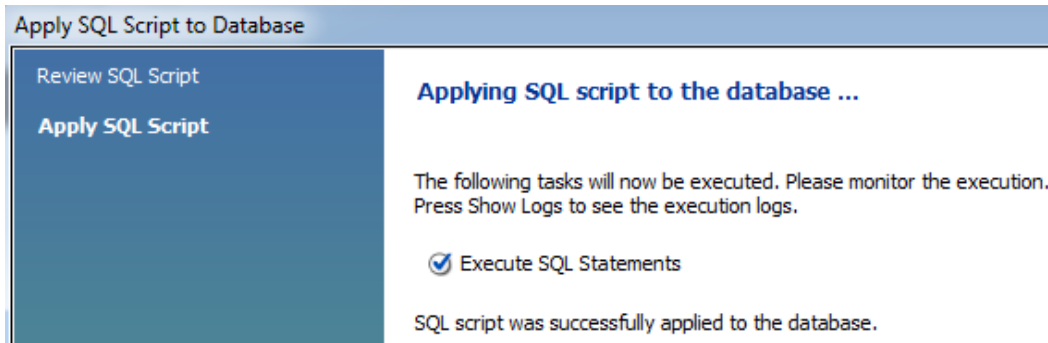
Damos clic en el ícono para crear un nuevo esquema



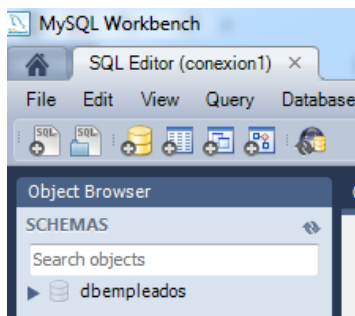
Escribimos el nombre del esquema (dbempleados) y damos clic en el botón Apply para llegar a la siguiente ventana:



Damos clic en el botón Apply nuevamente para llegar a la siguiente ventana:

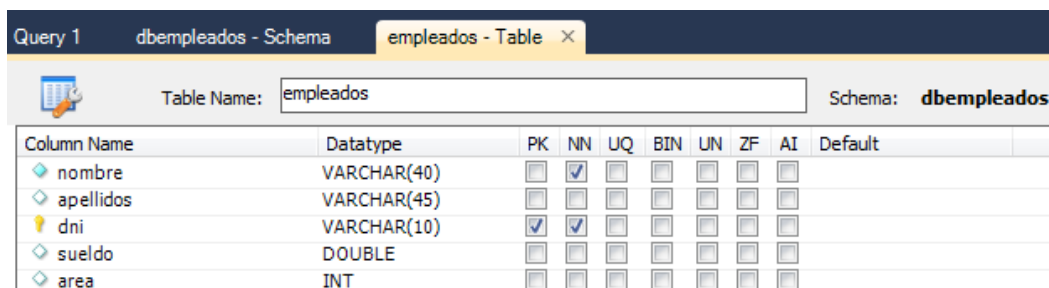


Damos clic en el botón Finish para llegar a la siguiente ventana:

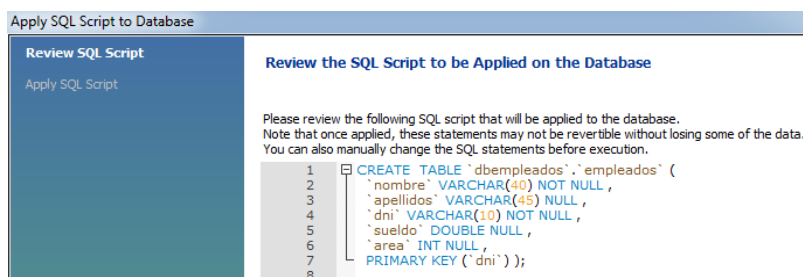


Donde comprobamos que se ha creado un nuevo esquema (dbempleados) donde daremos clic derecho y elegimos set as default schema.

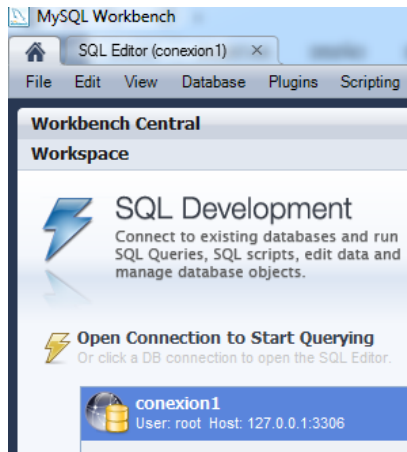
Luego damos clic en el ícono para crear una nueva tabla donde escribimos el nombre y llenamos los campos (column name) conforme a la siguiente ventana:



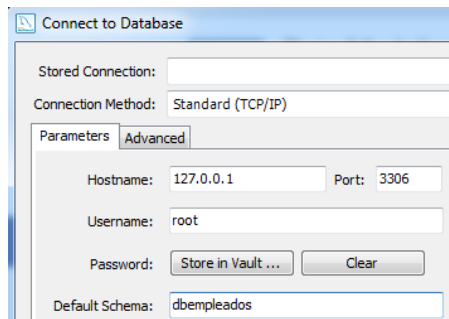
Damos clic en el botón Apply para llegar a la siguiente ventana:



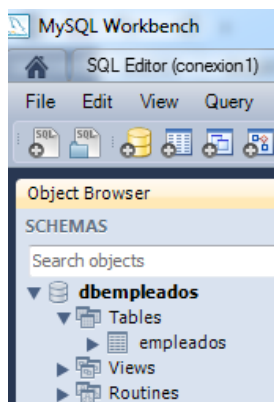
Damos clic en el botón Apply y luego en Finish. Vamos a la ventana de inicio de Workbench donde comprobamos que se ha creado una nueva conexión.



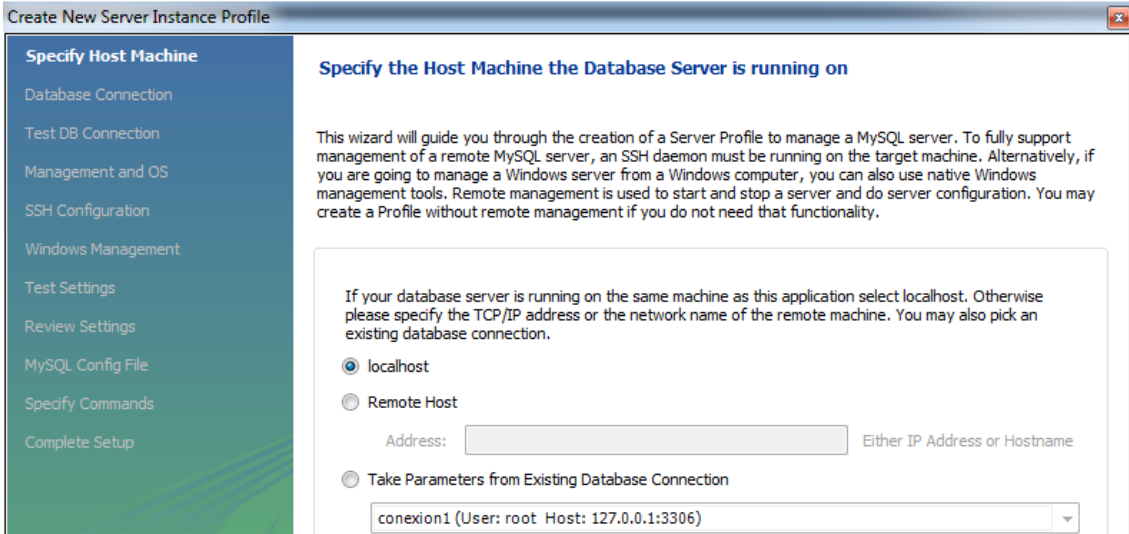
Damos clic en Open Connection to Start Querying para llegar a la siguiente ventana donde escribimos el nombre del esquema (dbempleados):



Damos clic en el botón OK, ingresamos el password para llegar a la siguiente ventana:



Regresamos al inicio de Workbench para crear una nueva instancia de servidor:



Create New Server Instance Profile

Specify Host Machine

Database Connection
Test DB Connection
Management and OS
SSH Configuration
Windows Management
Test Settings
Review Settings
MySQL Config File
Specify Commands
Complete Setup

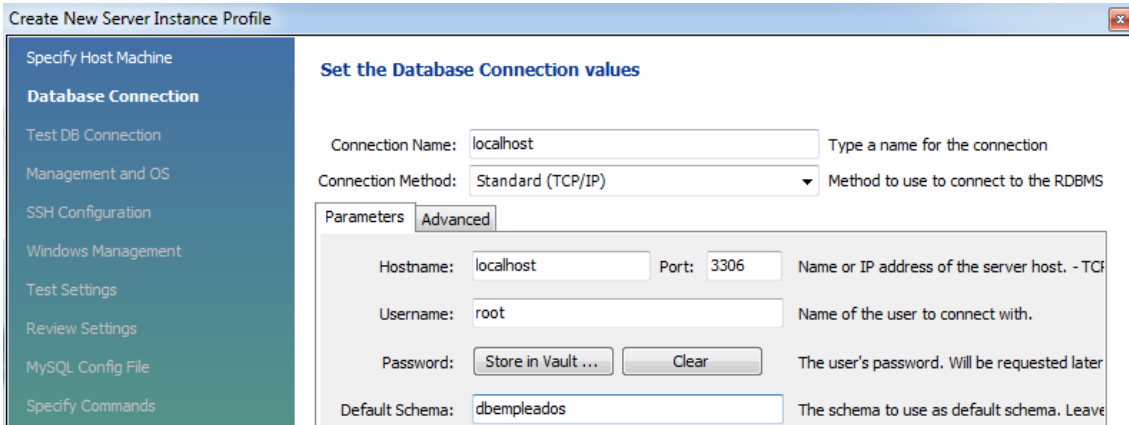
Specify the Host Machine the Database Server is running on

This wizard will guide you through the creation of a Server Profile to manage a MySQL server. To fully support management of a remote MySQL server, an SSH daemon must be running on the target machine. Alternatively, if you are going to manage a Windows server from a Windows computer, you can also use native Windows management tools. Remote management is used to start and stop a server and do server configuration. You may create a Profile without remote management if you do not need that functionality.

If your database server is running on the same machine as this application select localhost. Otherwise please specify the TCP/IP address or the network name of the remote machine. You may also pick an existing database connection.

☒ localhost
☐ Remote Host
 Address: Either IP Address or Hostname
☐ Take Parameters from Existing Database Connection
 conexion1 (User: root Host: 127.0.0.1:3306)

Damos clic en el botón Next y en la siguiente ventana escribimos el nombre del esquema “dbempleados”



Create New Server Instance Profile

Specify Host Machine
Database Connection
Test DB Connection
Management and OS
SSH Configuration
Windows Management
Test Settings
Review Settings
MySQL Config File
Specify Commands

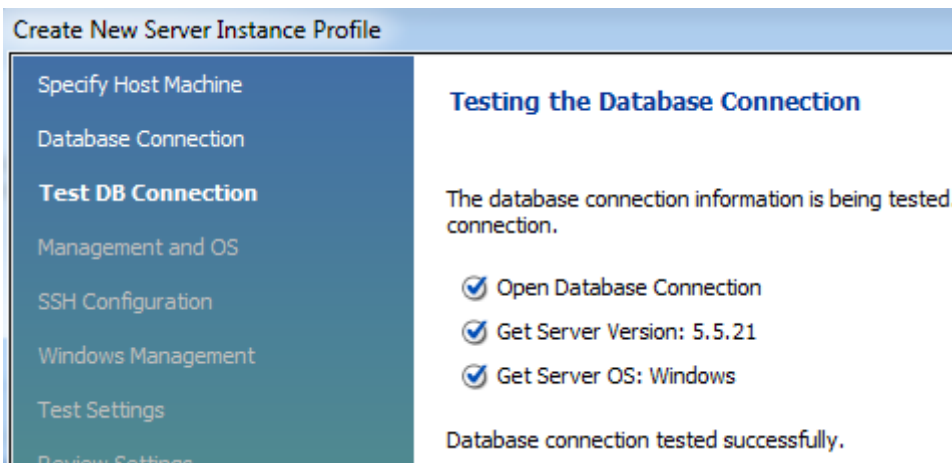
Set the Database Connection values

Connection Name: Type a name for the connection
 Connection Method: Method to use to connect to the RDBMS

Parameters **Advanced**

Hostname: Port: Name or IP address of the server host. - TCP
 Username: Name of the user to connect with.
 Password: The user's password. Will be requested later
 Default Schema: The schema to use as default schema. Leave

Damos clic en el botón Next para llegar a la siguiente ventana:



Create New Server Instance Profile

Specify Host Machine
Database Connection
Test DB Connection
Management and OS
SSH Configuration
Windows Management
Test Settings
Review Settings

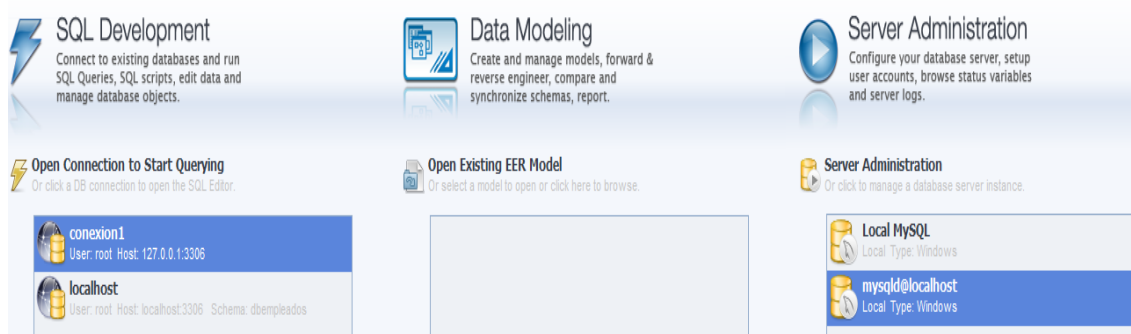
Testing the Database Connection

The database connection information is being tested. connection.

☒ Open Database Connection
☒ Get Server Version: 5.5.21
☒ Get Server OS: Windows

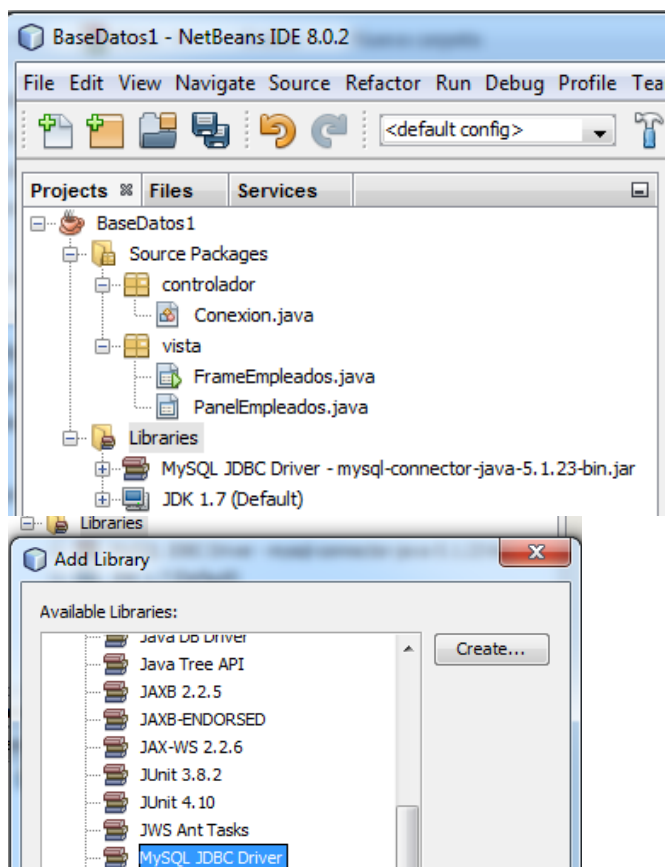
Database connection tested successfully.

Damos clic en el botón Next hasta finalizar. Vamos al inicio del Workbench para verificar que se ha creado la conexión y la instancia del servidor:



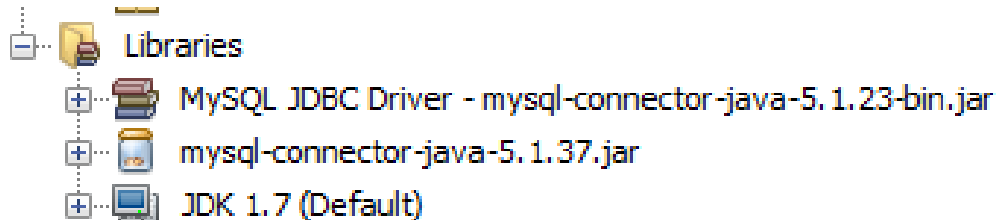
Conexión a una Base de Datos

Entramos a NetBeans, creamos un proyecto para incorporar el driver descargado y la librería en la carpeta librerías de nuestra aplicación: Clic derecho en Libraries de nuestra aplicación y elegimos Library y seleccionamos MySQL JDBC Driver.

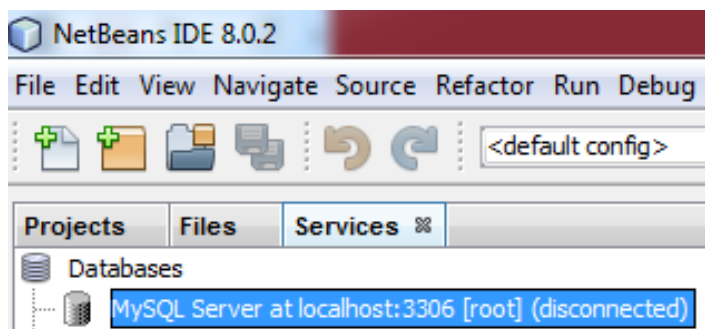


Luego de aceptar, verifique su incorporación en las librerías de nuestra aplicación:

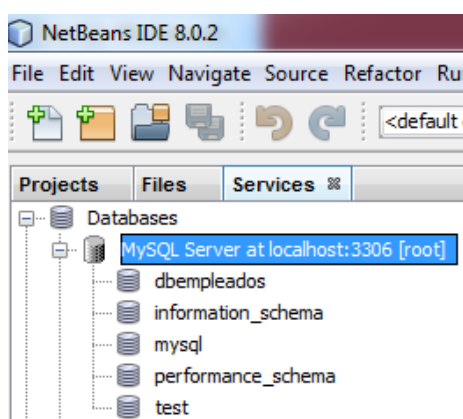
Ahora vamos a incorporar el driver conector que hemos descargado de internet. Nuevamente clic derecho en la carpeta Libraries de nuestra aplicación, elegimos add JAR/folder para que finalmente quede así la carpeta Libraries de nuestra aplicación:



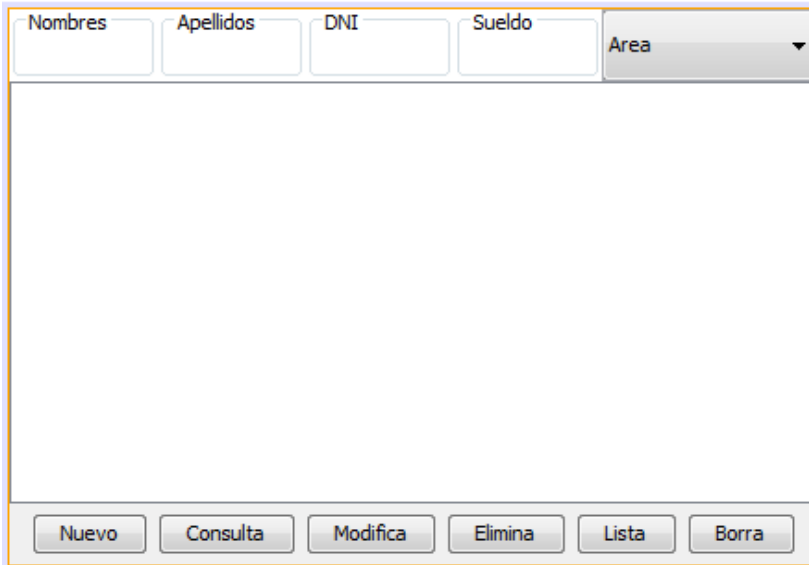
Ahora vamos a la librería Services para conectar el servidor localhost que inicialmente está desconectado:



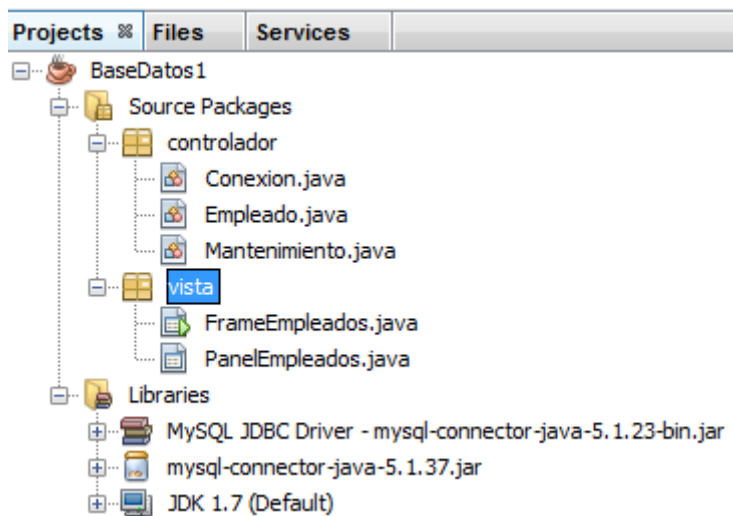
Para conectarlo, le damos clic derecho y elegimos Connect para que quede así: (Observe que nuestra base de datos creada en Workbench (dbempleados) ya es reconocida):



Ahora, vamos a crear una aplicación con un formulario para hacer el mantenimiento de la tabla de empleados considerando el siguiente diseño:



En la ficha Projects de nuestra aplicación creamos un paquete de nombre **controlador** donde diseñamos las clases Empleado, Conexión y Mantenimiento. Luego crearemos el paquete **vista** donde diseñamos un PanelEmpleados para el formulario y un FrameEmpleados para ejecutar el panel, de la siguiente manera:



Las clases Empleado y Mantenimiento lo desarrollaremos en la siguiente sesión. Ahora sólo probaremos la conexión con el diseño de la siguiente clase Conexión:

```
package controlador;
import java.sql.*; // acceso a Connection, Statement y ResultSet
public class Conexion {
    // atributos de uso compartido
    protected static String servidor="jdbc:mysql://localhost/dbempleados";
    protected static String usuario="root";
    protected static String password="root";
    protected static String driver="com.mysql.jdbc.Driver";
    protected static Connection conexion;

    // constructor que realiza la conexión a la base de datos
    public Conexion(){
        try{
            Class.forName(driver);
            conexion=DriverManager.getConnection(servidor,usuario,password);
            System.out.println("conexion exitosa!");
        }catch(ClassNotFoundException | SQLException e){
            System.out.println("Conexion fallida: "+e.getMessage());
        }
    }

    // método que retorna el resultado de la conexión realizada
    public Connection getConexion(){
        return conexion;
    }
}
```

En el constructor del PanelEmpleados escribimos lo siguiente:

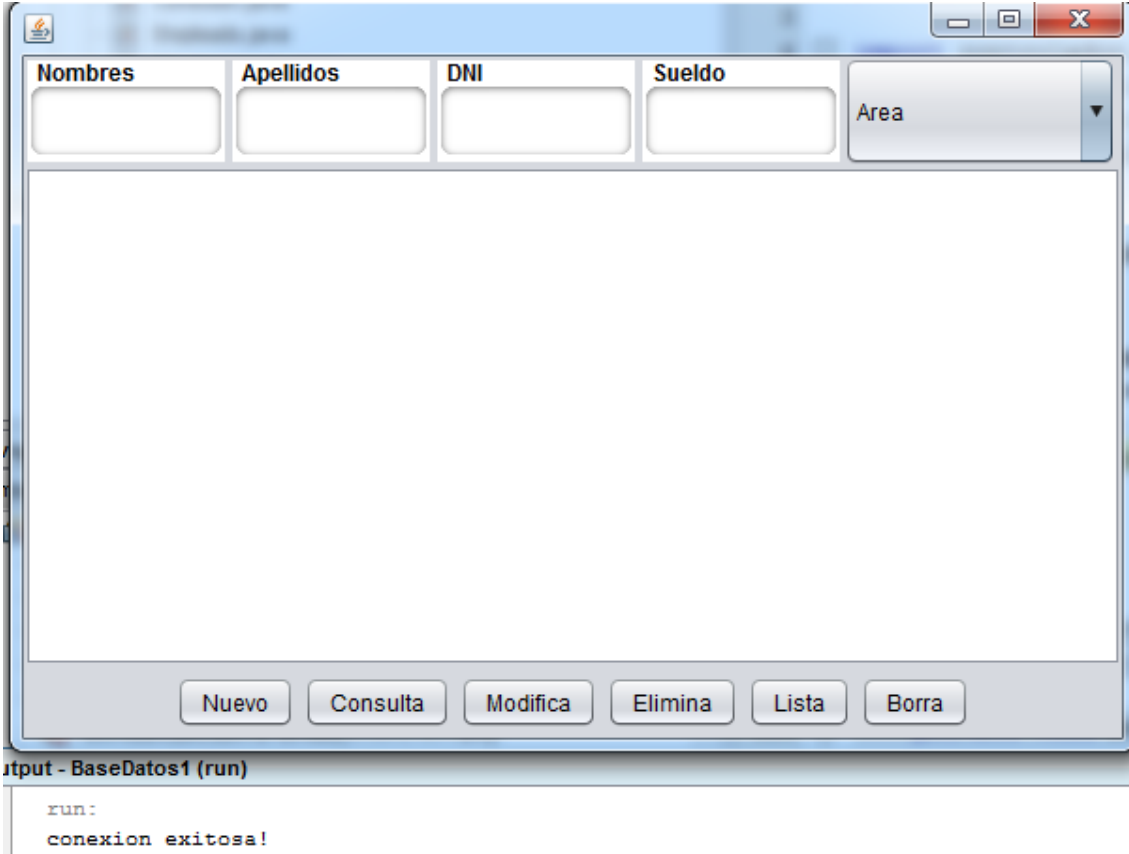
```
package vista;

import controlador.Conexion;

public class PanelEmpleados extends javax.swing.JPanel {

    public PanelEmpleados() {
        initComponents();
        // conecta a la base de datos
        Conexion con = new Conexion();
    }
}
```

Ejecutamos nuestra aplicación:



The image shows a Java Swing window titled "BaseDatos1" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a form with five input fields at the top, labeled "Nombres", "Apellidos", "DNI", "Sueldo", and "Area". Below these fields is a large, empty rectangular area, likely for displaying data. At the bottom of the window, there are six buttons: "Nuevo", "Consulta", "Modifica", "Elimina", "Lista", and "Borra".

Below the Swing window, there is a console output window titled "Output - BaseDatos1 (run)". It displays the following text:

```
run:
conexion exitosa!
```

Observe el mensaje en la salida de consola: "conexión exitosa! "

GUIA DE LABORATORIO 14

Mantenimiento y consultas en una base de datos

Abra el proyecto de la semana anterior y desarrolle las clases que faltan: Mantenimiento y programación de los botones del PanelEmpleados.

Diseñe la clase **Mantenimiento** con el siguiente atributo y constructor:

```
package controlador;

import java.sql.ResultSet;
import java.sql.Connection;
import java.sql.Statement;

public class Mantenimiento {
    protected Conexion conecta;

    public Mantenimiento() {
        conecta = null;
    }
}
```

Luego desarrollamos el método para registrar un nuevo empleado en la base de datos:

```
public void nuevo(Empleado e) {
    String sql;

    sql = "insert into empleados (nombre,apellidos,dni,sueldo,area) "
        + "values ('"+e.getNombre()+"', '"+
        e.getApellidos()+"', '"+
        e.getDni()+"', '"+
        e.getSueldo()+"', '"+
        e.getArea()+"')";

    try{
        conecta = new Conexion();
        Connection con = conecta.getConexion();
        Statement st;
        st=con.createStatement();
        st.executeUpdate(sql);
        con.close();
        st.close();
    }catch(Exception ex){
        System.out.println("Error en el ingreso nuevo: "+ex.getMessage());
    }
}
```

Ahora desarrollamos el método para hacer una modificación en un empleado:

```
public void modifica(Empleado e){
    String sql;

    sql = "update empleados set nombre = '"+e.getNombre()+
        "',apellidos = '"+e.getApellidos()+
        "',dni = '"+e.getDni()+
        "',sueldo = '"+e.getSueldo()+
        "',area = '"+e.getArea()+
        "'where dni = '"+e.getDni()+"'";

    try{
        conecta = new Conexion();
        Connection con = conecta.getConnection();
        Statement st;
        st=con.createStatement();
        st.executeUpdate(sql);
        con.close();
        st.close();
    }catch(Exception ex){
        System.out.println("Error en la modificacion: "+ex.getMessage());
    }
}
```

Ahora desarrollamos el método para hacer una eliminación de un empleado:

```
public void elimina(String dni){
    String sql;

    sql = "delete from empleados where dni = '"+dni+"'";

    try{
        conecta = new Conexion();
        Connection con = conecta.getConnection();
        Statement st;
        st=con.createStatement();
        st.executeUpdate(sql);
        con.close();
        st.close();
    }catch(Exception ex){
        System.out.println("Error en la eliminacion: "+ex.getMessage());
    }
}
```

Ahora desarrollamos el método para hacer una consulta de un empleado:

```
public Empleado consulta(String dni){
    String sql;
    sql = "select * from empleados where dni = '"+dni+"'";
    try{
        conecta = new Conexion();
        Connection con = conecta.getConnection();
        Statement st;
        st=con.createStatement();
        ResultSet rs= st.executeQuery(sql);
        if (rs.next()){ // lo encontré
            return new Empleado(rs.getString(1),
                                rs.getString(2),
                                rs.getString(3),
                                rs.getDouble(4),
                                rs.getInt(5));
        }
        con.close(); st.close();
    }catch(Exception ex){
        System.out.println("Error en la eliminacion: "+ex.getMessage());
    }
    return null;
}
```

Finalmente, desarrollamos el método que retorna la información de toda la tabla de empleados:

```
public String lista(){
    String sql, resultado="";
    sql = "select * from empleados";
    try{
        conecta = new Conexion();
        Connection con = conecta.getConnection();
        Statement st;
        ResultSet rs;
        st=con.createStatement();
        rs = st.executeQuery(sql);
        while (rs.next()){
            resultado += (rs.getString(1) + "\t" +
                        rs.getString(2) + "\t" +
                        rs.getString(3) + "\t" +
                        rs.getDouble(4) + "\t" +
                        rs.getInt(5) + "\n");
        }
        con.close(); st.close();
    }catch(Exception ex){
        System.out.println("Error en la eliminacion: "+ex.getMessage());
    }
    return resultado;
}
```


Ahora vamos a programar los botones del PanelEmpleados:

```
private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Mantenimiento man = new Mantenimiento();
    Empleado e = man.consulta(leeDNI());
    if (e!=null)
        mensaje("DNI repetido!");
    else{
        Empleado nuevo = new Empleado(leeNombre(),
                                       leeApellidos(),
                                       leeDNI(),
                                       leeSueldo(),
                                       leeArea());
        man.nuevo(nuevo);
        btnLista.doClick();
    }
}
```

```
private void btnConsultaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Mantenimiento man = new Mantenimiento();

    Empleado e = man.consulta(leeDNI());
    if (e==null)
        mensaje("DNI no registrado!");
    else{
        txtNombres.setText(e.getNombre());
        txtApellidos.setText(e.getApellidos());
        txtDNI.setText(e.getDni());
        txtSueldo.setText(e.getSueldo()+"");
        cboArea.setSelectedIndex(e.getArea());
    }
}
```

```
private void btnModificaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Mantenimiento man = new Mantenimiento();
    Empleado actual = man.consulta(leeDNI());
    if(actual == null)
        mensaje("DNI no registrado!");
    else{
        actual = new Empleado(leeNombre(),
                              leeApellidos(),
                              leeDNI(),
                              leeSueldo(),
                              leeArea());
        man.modifica(actual);
        btnLista.doClick();
    }
}
```

```
private void btnEliminaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Mantenimiento man = new Mantenimiento();
    Empleado actual = man.consulta(leeDNI());
    if(actual == null)
        mensaje("DNI no registrado!");
    else{
        man.elimina(leeDNI());
        btnLista.doClick();
    }
}
```

```
private void btnListaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Mantenimiento man = new Mantenimiento();
    txtSalida.setText("Nombre\tApellidos\tDNI\tSueldo\tArea\n");
    imprime(man.lista());
}
```

Métodos complementarios:

```
private String leeNombre(){ return txtNombres.getText();}
private String leeApellidos() { return txtApellidos.getText();}
private String leeDNI(){ return txtDNI.getText(); }
private double leeSueldo(){ return Double.parseDouble(txtSueldo.getText());}
private int leeArea(){ return cboArea.getSelectedIndex(); }
private void imprime(String s){
    txtSalida.append(s+"\n");
}
private void mensaje(String s){
    JOptionPane.showMessageDialog(this, s);
}
```

Ejecute su aplicación:

Nombre	Apellidos	DNI	Sueldo	Area
pedro	landauro	32423423	34243.0	6
juan	perez	32424323	1231.0	1
rosalia	marrou	34234334	4322.0	4
jose	bardales	34242343	2921.0	2
karen	swartz	35434534	5435.0	2
josefina	marifios	43234245	3223.0	4
jesus	zapata	53432434	2133.0	6

La información ya está registrada en la base de datos, puede cerrar su aplicación y volver a ejecutarlo para comprobarlo.

Añada un comboBox de reportes para mostrar la siguiente información:

- Empleados que pertenecen a un área seleccionada
- Empleados con un sueldo inferior al sueldo promedio
- Empleados en orden alfabetico por apellidos

ASESORIA PROYECTO

EXPOSICION DE TRABAJOS

EVALUACION INTEGRAL

BIBLIOGRAFIA

1. **CEBALLOS, F. (2008).** Java 2 Interfaces gráficas y aplicaciones para internet. México: Alfa omega. **(Código en Biblioteca 005.133 C43 2008)**
2. **DEITEL, H. (2008).** ¿Cómo programar en Java? México: Pearson. **(Código en Biblioteca: 006.78D46)**
3. **JOYANES L. (2011).** Programación en Java: McGrawHill. **(Código en Biblioteca: 006.78 J79 2011).**
4. **JOYANES L. (2013).** Fundamentos generales de Programación: McGrawHill. **(Código en Biblioteca: 005.133 J79D).**
5. **LOPEZ, L. (2006).** Metodología de la programación orientada a objetos: Alfa omega **(Código en Biblioteca: 005.133.L88)**
6. **VASQUEZ, J. (2010).** Super Java SE for Windows with NetBeans IDE. 2da. Edición. Perú: Fondo Editorial UCH **(Código en Biblioteca: 005.133.V33)**
7. **VILLALOBOS, R. (2008).** Fundamentos de Programación Java. Perú: Macro. **(Código en Biblioteca: 005.133 V66J)**
8. **WU, T. (2008).** Programación en Java introducción a la programación orientada a objetos. México: McGraw Hill. **(Código en Biblioteca: 006.78 W95)**

INDICE

A**abstract**, 55

Acceso a Base de Datos, 68

Arreglo de objetos, 17

B

BIBLIOGRAFIA, 160

BufferedReader, 46**C**

Casting, 55

Clase administradora, 17

Clase Administradora, 27

Clase ArrayList, 17

Clase LinkedList, 27

Clases Abstractas, 55

Colección de objetos polimórficos, 60

Conexión a una Base de Datos, 73

Constructores, 9

Creación de objetos, 10

Creación de una Base de Datos, 68

D

Diseño de clases, 8

E

ENCAPSULAMIENTO, 7, 17

Extends, 39**F****FileReader**, 46**FileWriter**, 47**G**

GUIA DE LABORATORIO 1, 88

GUIA DE LABORATORIO 10, 132

GUIA DE LABORATORIO 12, 137

GUIA DE LABORATORIO 13, 141

GUIA DE LABORATORIO 14, 151

GUIA DE LABORATORIO 2, 95

GUIA DE LABORATORIO 3, 104

GUIA DE LABORATORIO 5, 114

GUIA DE LABORATORIO 6, 121

GUIA DE LABORATORIO 9, 127

H**Herencia**, 38

HERENCIA, 38

Herencia con archivos de texto, 46

I**InstanceOf**, 60

INTRODUCCIÓN, 4

INTRODUCCIÓN A BASE DE DATOS, 68

J**Jerarquía de Clases**, 38**L**

Lista de objetos, 27

M

Mantenimiento y consultas a una base de datos, 78

Métodos, 9

N

Niveles de acceso, 7

P**Palabra reservada this**, 10

Persistencia con archivos de texto, 64

Polimorfismo, 54

POLIMORFISMO, 54

PrintWriter, 47**private**, 7

Programación orientada a objetos, 7

protected, 7**public**, 7

S

Sobrecarga de métodos, 50

StringTokenizer, 46

Super, 39

U

UNIDADES DE APRENDIZAJE, 6