

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

Carrera: Técnico En Ing. De sistemas Informáticos

Modalidad: Presencial y virtual

Materia: Desarrollo de aplicaciones usando tecnologías emergentes



Tema: “Java Swing y POO”

Resultados de aprendizaje:

- Dominar el entorno de desarrollo de NetBeans para el desarrollo de aplicaciones con Java Swing implementando programación orientada a objetos.

Material y equipo:

- Guía práctica.
- Computadora.
- IDE para el desarrollo

¿Qué es Programación Orientada a Objetos?

La programación orientada a objetos es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.

Muchos de los objetos pre-diseñados de los lenguajes de programación actuales permiten la agrupación en bibliotecas o librerías, sin embargo, muchos de estos lenguajes permiten al usuario la creación de sus propias bibliotecas.

Está basada en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento.

Definición Web

Pilares de la POO

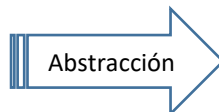


La POO tiene varios pilares para asegurar la simplicidad de código y su reutilización, y aunque diversos autores señalan diversos pilares, en este módulo se considerarán los cuatro que son comunes en la mayoría de textos, estos son: Abstracción, Encapsulamiento, Herencia y Polimorfismo, las dos primeras están más relacionadas con la búsqueda de códigos simples y las dos siguientes con la reutilización.

Abstracción

Consiste en captar las características y acciones de un objeto del mundo real y expresarlas en una clase. Estas características y acciones son las que distinguen al objeto de los demás. Se considerarán las características y acciones que interesan en el procesamiento de la información del objeto real en el sistema informático a realizar.

Ejemplo: Se va realizar un sistema en el cual se procesará información de vehículos para un sistema de venta de vehículos. De cada vehículo hay cierta información que es necesario procesar, se pide aplicar el proceso de abstracción para el diseño de una clase, la cual servirá para del sistema a realizar.



Vehiculo
marca: String modelo: String kilometraje: double precio: Double color: String
Vehiculo () comprar () vender () Etc()



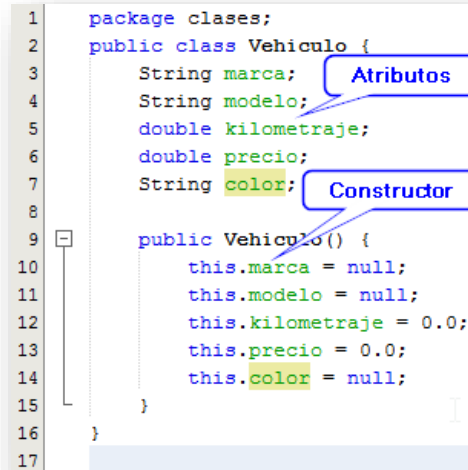
Toda clase consta de: Atributos o campos, métodos y constructores.

Atributos: Para el ejemplo anterior los atributos son las características del objeto (marca, modelo, kilometraje, precio y color)

Métodos: Son las acciones que hace el objeto dentro de nuestro sistema, para el ejemplo anterior son comprar (), vender (), etc... dentro de los paréntesis se escriben los parámetros de entrada si los fuere necesario.

Constructor: Es un método, el cual tiene como función principal inicializar los atributos de la clase.

Ejemplo de una clase en Java:



```
1 package clases;
2 public class Vehiculo {
3     String marca;
4     String modelo;
5     double kilometraje;
6     double precio;
7     String color;
8
9     public Vehiculo() {
10         this.marca = null;
11         this.modelo = null;
12         this.kilometraje = 0.0;
13         this.precio = 0.0;
14         this.color = null;
15     }
16 }
17
```

The image shows a code editor with Java code for a `Vehiculo` class. Two blue callout boxes with arrows point to specific parts of the code: one labeled 'Atributos' points to the attribute declarations (lines 3-7), and another labeled 'Constructor' points to the `public Vehiculo()` method (line 9).

Encapsulamiento

Es una técnica de programación que consiste en organizar atributos y métodos en una estructura específica, conciliando el modo en que el objeto se implementa, es decir, evitando el acceso a datos por cualquier otro medio distinto a los especificados.

El ocultamiento depende de los modificadores de acceso tanto de los métodos como los campos de una clase, estos modificadores de acceso son:

- Public: Permiten que el atributo o método sea accedido desde afuera de la clase.
- Private: Permite que cualquier método de la misma clase acceda al atributo o método, pero no es posible acceder desde afuera de la clase donde se ha declarado.
- Protected: Es parecido al Private con la diferencia que permite el acceso al atributo o clase desde afuera de la clase siempre y cuando se haya aplicado herencia.

Ejemplo: Se ha diseñado una clase que permite procesar información de los Vehículos de una empresa y tiene el siguiente código.

```

1 package clases;
2 public class Vehiculo {
3
4     public String marca;
5     private double precio;
6     protected String modelo;
7
8     public Vehiculo(){
9         this.precio=3000;
10    }
11    public double getPrecio(){
12        if (precio < 2500){
13            precio = precio + (precio * 0.2);
14        }else{
15            precio = precio + (precio * 0.05);
16        }
17        return precio;
18    }
19 }
20

```

Cuando se usa la clase, se crea un objeto, éste es una copia entera de la clase, si se usa la clase el código sería:

```

9 public static void main(String[] args) {
10     Vehiculo miCarro = new Vehiculo();//objeto llamado mi carro
11     //de forma automática se llamó al método
12     //constructor de la clase cuando se crea el objeto
13     System.out.println("El precio del vehiculo es: " +
14         miCarro.getPrecio());
15     //el resultado es "El precio del Vehiculo es: $3000.00"
16 }

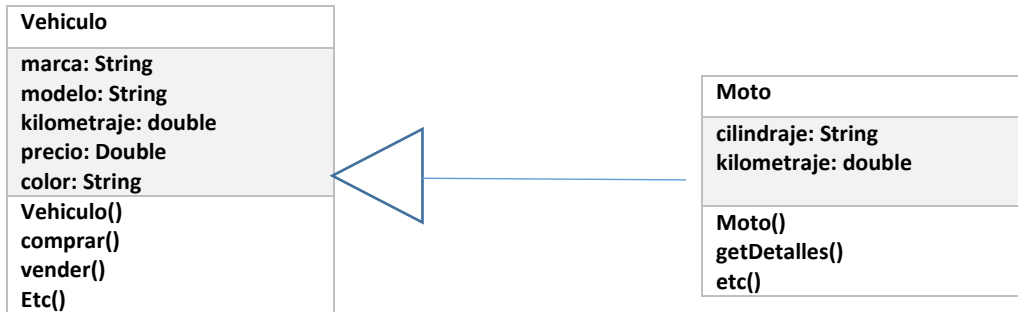
```

Del código anterior se visualiza que al llamar al método **getPrecio ()** de la clase **Vehiculo** se imprime la cantidad que tiene guardado el campo Precio ¿Cómo se calculó ese precio? La respuesta es que esta encapsulado el proceso.

Herencia

Consiste en transmitir atributos y métodos de una clase padre a una clase hija.

Por ejemplo: Se tiene la clase padre **Vehículo** y se aplica herencia hacia la clase **Moto**, eso hace que la clase moto contenga todos los atributos y métodos de la clase **Vehículo**, aunque no estén escritos dentro de la clase **Moto**.



Si se crease un objeto de tipo moto, por medio de este objeto puede llamarse a los métodos comprar (), vender (), etc. El objetivo de la herencia es la reutilización de código. En Java la herencia se aplica por medio del operador: **Ejemplo Class Moto extends Vehiculo { ... }**

Polimorfismo

Polimorfismo significa “la facultad de asumir muchas formas”, refiriendo se a la facultad de llamar a las distintas versiones es que adopta un método definido en una clase usando el mismo medio de acceso.

El polimorfismo se representa por dos caras:

- Sobre carga de métodos
- Sobre escritura de métodos

Sobre carga de métodos

Consiste en escribir métodos con el mismo nombre, pero con diversa cantidad de parámetros de entrada. Esto se hace dependiendo de la necesidad de la programación. Por ejemplo:

```

8 public Vehiculo() {
9     this.precio=3000;
10 }
11
12 public Vehiculo(double nuevoPrecio) {
13     this.precio = nuevoPrecio;
14 }
15
16 public Vehiculo(String marc, String model) {
17     this.marca = marc;
18     this.modelo = model;
19 }
  
```

Sobre escritura de métodos

Esto sucede cuando un método es heredado o implementado y se desea sobre escribir en la clase que lo implementa.... Por ejemplo, en las clases antes mencionadas Vehiculo y Moto, se observa que la clase Moto hereda de Vehiculo, entonces se dice que la clase Moto

posee el método `Comprar()`, pero si por motivos diversos se desea modificar el contenido de este método se puede hacer por sobre escritura.

Clase padre **Vehiculo**

```
23 public double getPrecio() {
24     if (precio < 2500) {
25         precio = precio + (precio * 0.2);
26     } else {
27         precio = precio +
28     }
29     return precio;
}
```

Metodo getPrecio en la clase Vehiculo

Clase hija **Moto**

```
5 @Override
6 public double getPrecio() {
7     //aqui puede escribirse otro codigo
8     /*
9     LO QUE SEA, SEGÚN LA NECESIDAD DE PROGRAMACION
10    */
11    return 0;
12 }
```

uso de @Override

Mismo metodo de la clase Vehiculo sobre escrito en la clase hija

De lo anterior analizamos que la clase **Moto** hereda por medio de la palabra reservada **extends** de la clase **Vehiculo**, por tanto, la clase **Moto** tiene todos los métodos y atributos de la clase **Vehiculo**, pero por diversos motivos se puede reescribir un método en la clase hija... a eso se le llama sobre escritura y se escribe la palabra reservada **@override** para que el intérprete del lenguaje omita el método heredado.

Entonces decimos que polimorfismo significa múltiples formas, y se representa por sobrecarga (mismo método diferente comportamiento, diferente cantidad de parámetros) y sobre escritura de métodos (mismo método en diferente clase con diferente comportamiento)



Si se aplica sobre escritura el método sobre escrito debe tener la misma cantidad de parámetros del método original, sino no es sobre escritura más bien sobre carga.

JAVA SWING

Swing es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, listas desplegables y tablas.

Arquitectura:

Es un framework MVC para desarrollar interfaces gráficas para Java con independencia de la plataforma. Sigue un simple modelo de programación por hilos, y posee las siguientes características principales:

Independencia de plataforma.

- Extensibilidad: es una arquitectura altamente particionada: los usuarios pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto. Se puede extender clases existentes proveyendo alternativas de implementación para elementos esenciales.
- Personalizable: dado el modelo de representación programático del framework de Swing, el control permite representar diferentes estilos de apariencia "look and feel" (desde apariencia MacOS hasta apariencia Windows XP, pasando por apariencia GTK+, IBM UNIX o HP UX, entre otros). Además, los usuarios pueden proveer su propia implementación de apariencia, que permitirá cambios uniformes en la apariencia existente en las aplicaciones Swing sin efectuar ningún cambio al código de aplicación.

Uso de controles comunes

Cajas de Texto (JTextField)

Asignación un valor: `JTextField1.setText("Valor de la caja de texto");`

Extracción del valor: `String valor= jTextField1.getText();`

Listas (JList)

Asignación de ítems:

```
DefaultListModel modeloLista = new DefaultListModel();
modeloLista.addElement("Elemento1");
modeloLista.addElement("Elemento2");
jList1.setModel(modeloLista);
Extracción de valor:
```

`String elemento= jList1.getSelectedValue();`

Tablas (JTable)

Asignación de ítems:

```
String [] columnas = {"COL1", "COL2", "COL3"};
DefaultTableModel modeloTabla = new DefaultTableModel(columnas, 0);
String fila[] = {"dato01", "dato02", "dato03"};
modeloTabla.addRow(fila);
jTable1.setModel(modeloTabla);
```

Extracción de valor:

```
//sacando cada celda por medio de un bucle
String DatoCol1= jTable1.getValueAt(jTable1.getSelectedRow(),0).toString();
//en donde el valor de 0, representa el índice de la columna y va variar
//dependiendo del valor de la celda seleccionada que se quiera recuperar
```

Lista Desplegable (jComboBox)

Asignación de Items:

```
DefaultComboBoxModel modeloCombo = new DefaultComboBoxModel();
modeloCombo.addElement("Elemento1");
modeloCombo.addElement("Elemento2");
modeloCombo.addElement("ElementoN");
jComboBox1.setModel(modeloCombo);
```

Extracción de valores:

```
String valorSeleccionado = jComboBox1.getSelectedItem().toString();
```

Spinner (jSpinner)

Asignación de un valor inicial:

```
jSpinner1.setValue(50);
```

Extracción del valor seleccionado:

```
String val = jSpinner1.getValue().toString();
```

Cuadros de mensaje (JOptionPane)

```
String valor_s = JOptionPane.showInputDialog("ingrese un valor");
JOptionPane.showMessageDialog(this, valor_s);
JOptionPane.showMessageDialog(this,
valor_s,"Titulo",JOptionPane.DEFAULT_OPTION);
JOptionPane.showMessageDialog(this, valor_s,"Titulo",JOptionPane.OK_OPTION);
JOptionPane.showMessageDialog(this, valor_s,"Titulo",JOptionPane.CANCEL_OPTION);
JOptionPane.showMessageDialog(this, valor_s,"Titulo",JOptionPane.CLOSED_OPTION);
JOptionPane.showMessageDialog(this,
valor_s,"Titulo",JOptionPane.ERROR_MESSAGE);
JOptionPane.showMessageDialog(this,
valor_s,"Titulo",JOptionPane.QUESTION_MESSAGE);
JOptionPane.showMessageDialog(this,
valor_s,"Titulo",JOptionPane.WARNING_MESSAGE);
JOptionPane.showMessageDialog(this,
valor_s,"Titulo",JOptionPane.OK_CANCEL_OPTION);
JOptionPane.showMessageDialog(this, valor_s,"Titulo",JOptionPane.NO_OPTION);
JOptionPane.showMessageDialog(this,
valor_s,"Titulo",JOptionPane.YES_NO_CANCEL_OPTION);
JOptionPane.showMessageDialog(this, valor_s,"Titulo",JOptionPane.YES_OPTION);
JOptionPane.showMessageDialog(this, valor_s,"Titulo",JOptionPane.YES_NO_OPTION);
```


USO DE ARRAYLIST PARA ALMACENAMIENTO DE DATOS

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

Los principales métodos para trabajar con los ArrayList son los siguientes:

```
// Declaración de un ArrayList de "String". Puede ser de cualquier otro Elemento u Objeto (float, Boolean, Object, ...)
ArrayList<String> nombreArrayList = new ArrayList<String>();
// Añade el elemento al ArrayList
nombreArrayList.add("Elemento");
// Añade el elemento al ArrayList en la posición 'n'
nombreArrayList.add(n, "Elemento 2");
// Devuelve el numero de elementos del ArrayList
nombreArrayList.size();
// Devuelve el elemento que esta en la posición '2' del ArrayList
nombreArrayList.get(2);
// Comprueba se existe del elemento ('Elemento') que se le pasa como parametro
nombreArrayList.contains("Elemento");
// Devuelve la posición de la primera ocurrencia ('Elemento') en el ArrayList
nombreArrayList.indexOf("Elemento");
// Devuelve la posición de la última ocurrencia ('Elemento') en el ArrayList
nombreArrayList.lastIndexOf("Elemento");
// Borra el elemento de la posición '5' del ArrayList
nombreArrayList.remove(5);
// Borra la primera ocurrencia del 'Elemento' que se le pasa como parametro.
nombreArrayList.remove("Elemento");
// Borra todos los elementos de ArrayList
nombreArrayList.clear();
// Devuelve True si el ArrayList esta vacio. Sino Devuelve False
nombreArrayList.isEmpty();
// Copiar un ArrayList
ArrayList arrayListCopia = (ArrayList) nombreArrayList.clone();
// Pasa el ArrayList a un Array
Object[] array = nombreArrayList.toArray();
```

Ejemplo Práctico: Agregar al proyecto abierto un MainClass, y escribir el siguiente código en el método principal

```
// Declaración el ArrayList
ArrayList<String> nombreArrayList = new ArrayList<String>();
```

```
// Añadimos 10 Elementos en el ArrayList
for (int i=1; i<=10; i++){
    nombreArrayList.add("Elemento "+i);
}

// Añadimos un nuevo elemento al ArrayList en la posición 2
nombreArrayList.add(2, "Elemento 3");

// Declaramos el Iterador e imprimimos los Elementos del ArrayList
Iterator<String> nombrelterator = nombreArrayList.iterator();
while(nombrelterator.hasNext()){
    String elemento = nombrelterator.next();
    System.out.print(elemento+" / ");
}
}
```

EJERCICIOS PRÁCTICOS

- 1) Crear un proyecto que contenga las siguientes clases

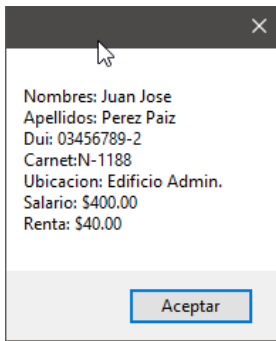
Empleado
nombre: string apellidos: string salario: double dui: string
Double Calcular_salario();

Ordenanza
carnet : int ubicacion: string renta:double Double Calculo_renta()

Aplice herencia de **Empleado** a **Ordenanza** y sobre escriba el método Calcular_salario() en **Ordenanza** de tal forma que siempre reciba un bono extra de 5%.

Cree un formulario como se muestra a continuación:

El objetivo es que en la programación del evento clic del botón, debe crear un objeto que sea instancia a la clase **Ordenanza** y llenará el objeto con los datos del formulario,



posteriormente mostrará en un cuadro de dialogo los datos registrados mostrando también el valor de descuento de renta, tal como se muestra a continuación:

No olvide que debe crear un método dentro de la clase ordenanza que calculará el monto de la renta.

- 2) Del ejercicio anterior, mostrar los datos agregados en un control JTable.
- 3) Investigar sobre el uso de **interfaces y clases abstractas** en Java y presentar un ejemplo básico de su uso en parejas antes de las 4:00 pm
- 4) Almacenar los registros de un empleado en un ArrayList y crear un mantenimiento completo (Agregar, modificar, eliminar, mostrar y salir). Crearlo mediante consola e interfaz gráfica.
- 5) Crear un ArrayList que almacene los resultados de un partido de futbol, tomar en cuenta goles de ambos partidos, número de tarjetas rojas, amarillas, tiros a gol, etc. Crearlo mediante consola e interfaz gráfica.