



**PROGRAMACION ORIENTADA A OBJETOS**

**ACTIVIDAD COLABORATIVA**

**FASE 4 – HERENCIA EN LENGUAJE DE PROGRAMACIÓN JAVA.**

**PRESENTADO POR:**

**PAOLA AMADO ANGULO**

**CC 1103712413**

**TUTOR**

**FRANKLIN LIZCANO CELIS**

**UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA UNAD**

**ESCUELA DE CIENCIAS BASICAS, TECNOLOGIA E INGENIERIA ECBTI**

**PROGRAMACION ORIENTADA A OBJETOS**

**MAYO DE 2019**





## 1. Cada estudiante consulta y entrega definición de los siguientes conceptos:

### -Herencia y Polimorfismo

Herencia en la programación orientada a objetos es la habilidad de extender una funcionalidad existente definiendo una nueva clase que hereda funcionalidad de una clase existente. Lo cual nos ahorra mucho tiempo a los programadores.

Si contamos con una clase que se acerca a lo que necesitamos; no es necesario crear una clase desde cero. Podemos aprovecharla y extenderla para crear nuestra nueva clase. Esta nueva clase se llamará subclase y la clase que ya teníamos se llamará superclase.

La subclase heredará todos los atributos y los métodos que fueron definidos en la clase padre. Si necesitamos cambiar algún método, se puede sobrescribir el comportamiento en nuestra subclase; utilizando el mismo nombre y los mismos argumentos del método que se encuentra en la subclase. O bien si se requiere crear un nuevo método lo podemos incluir en nuestra subclase.

Una clase puede heredar atributos por dos superclases (clases padres). La herencia múltiple puede ser usada para agrupar atributos y métodos de distintas clases en una sola

### -Herencia

La herencia simple es la más típica, la que se puede encontrar en cualquier lenguaje moderno como Java o C#.

La herencia simple es una relación entre una clase padre (clase base) y una clase hija (clase derivada) llamada "es un tipo de", que muchas veces se abrevia como *isA*.

La herencia es simple cuando la clase derivada que estamos considerando sólo tiene una clase base. Simple


### -Herencia Múltiple

hace referencia a la característica de los lenguajes de programación orientada a objetos en la que una clase puede heredar comportamientos y características de más de una superclase. Esto contrasta con la herencia simple, donde una clase sólo puede heredar de una superclase.

### -Herencia de Interfaz

Las interfaces también pueden heredar de otras interfaces, consiguiendo así una nueva interfaz, se emplea el *extends* habitual.

Teniendo así que la clase que implementa a esa nueva interfaz recibirá los métodos tanto





de la interfaz base como la derivada.

### **-Herencia de Implementación**

Como ya se ha visto, las interfaces carecen de funcionalidad por no estar implementados sus métodos, por lo que se necesita algún mecanismo para dar cuerpo a sus métodos.

La palabra reservada `implements` utilizada en la declaración de una clase indica que la clase implementa la interfaz, es decir, que asume las constantes de la interfaz, y codifica sus métodos

### **-Polimorfismo y reutilización**

En programación orientada a objetos se denomina polimorfismo a la capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación. Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa.

### **-Sobrecarga**

La sobrecarga de métodos es la creación de varios métodos con el mismo nombre, pero con diferente lista de tipos de parámetros, diferencia los métodos sobrecargados con base en el número y tipo de parámetros o argumentos que tiene el método y no por el tipo que devuelve

### **-Variables Polimórficas**


En Java las palabras que contienen objetos son variables polimórficas. En Java, una variable polimórfica es una variable que contiene un objeto y se refiere al hecho de que una misma variable puede contener objetos de diferentes tipos del tipo declarado o de cualquier subtipo del tipo declarado por eso se llama polimórfica que literalmente significa: muchas formas.

2. Cada estudiante realiza en un documento el modelo de herencia a aplicar en su proyecto. En este modelo deben especificarse cada una de las clases según el modelo de clases de la fase 2 y posteriormente implementar la herencia donde determine cuáles son las clases padre que quedan y cuáles son las clases hijas que quedan junto con los atributos a heredar.

### **CLASE PADRE EMPLEADOS**

`package clases;`

```
public class usuarios {
```





```
private int cedula;
```

```
private String nombre;
```

```
private Factura[] arrFactura;
```

```
public Cliente(int cedula, String nombre) {
```

```
    this.cedula = cedula;
```

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

```
    this.arrFactura = arrFactura;
```

```
}
```

```
public String mostrarCliente(Cliente cliente) {
```

```
    String mostrarCliente = "Informacion del Cliente" + "\n"
```

```
        + "Nombre: " + cliente.getNombre() + "    Cédula: " + cliente.getCedula();
```

```
    return mostrarCliente;
```

```
}
```

```
public int getCedula() {
```

```
    return cedula;
```


```
}
```

```
public void setCedula(int cedula) {
```

```
    this.cedula = cedula;
```

```
}
```

```
public String getNombre() {
```





```
return nombre;  
}
```


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


```
public Factura[] getArrFactura() {  
    return arrFactura;  
}
```

```
public void setArrFactura(Factura[] arrFactura) {  
    this.arrFactura = arrFactura;  
}  
  
}
```

## CLASES HIJO FACTURA

```
package clases;  
  
public class factura {  
    private Cliente persona;  
    public Producto[] arrProductos;  
    protected int maxiProducto;  
  
    public Factura(Cliente persona, int maxiProducto, Producto[] arrProducto) {
```





```
this.persona = persona;  
this.maxiProducto = maxiProducto;  
arrProductos = arrProducto;
```

```
}
```


```
public void agregarProducto(Producto producto, int posicion) {  
    arrProductos[posicion] = producto;  
}
```


```
public void mostrarFactura() {  
    System.out.print("\n" + persona.mostrarCliente(persona));  
    System.out.print("\n" + "Productos " + "\n");  
    System.out.print("\n" + "Cantidad " + "Codigo " + "Producto " + "Precio " + "  
Descuento " + "Total " + "\n");  
    Producto miProducto = null;  
    for (int i = 0; i < arrProductos.length; i++) {  
        if (arrProductos[i] instanceof SinOferta) {  
            miProducto = new SinOferta();  
            miProducto = arrProductos[i];  
            System.out.println("'" + ((SinOferta) miProducto).mostrarProducto((SinOferta)  
miProducto));  
        }  
        if (arrProductos[i] instanceof EnOferta) {  
            miProducto = new EnOferta();  
            miProducto = arrProductos[i];  
            System.out.println("'" + ((EnOferta) miProducto).mostrarProducto((EnOferta)  
miProducto));  
        }  
    }  
}
```





```
public String calcular(Producto producto) {  
    float totalFactura = 0;  
    int cantidadProducto = 0;  
    String mostrar = "";  
    Producto miProducto = null;  
    for (int i = 0; i < arrProductos.length; i++) {  
        if (arrProductos[i] instanceof SinOferta) {  
            miProducto = new SinOferta();  
            miProducto = arrProductos[i];  
            totalFactura += ((SinOferta) miProducto).calcularPrecioProducto();  
            cantidadProducto += ((SinOferta) miProducto).getCantidad();  
        }  
        if (arrProductos[i] instanceof EnOferta) {  
            miProducto = new EnOferta();  
            miProducto = arrProductos[i];  
            totalFactura += ((EnOferta) miProducto).calcularPrecioProducto();  
            cantidadProducto += ((EnOferta) miProducto).getCantidad();  
        }  
    }  
    mostrar = "La cantidad de Articulos: " + cantidadProducto + " El Total a Cancelar: " +  
totalFactura + "\n";  
    return mostrar;  
}
```





```
public Cliente getPersona() {  
    return persona;  
}
```

```
public void setPersona(Cliente persona) {  
    this.persona = persona;  
}
```

```
public Producto[] getArrProductos() {  
    return arrProductos;  
}
```

```
public void setArrProductos(Producto[] arrProductos) {  
    this.arrProductos = arrProductos;  
}
```

```
public int getMaxiProducto() {  
    return maxiProducto;  
}
```

```
public void setMaxiProducto(int maxiProducto) {  
    this.maxiProducto = maxiProducto;  
}
```

```
}
```







## CLASE HIJO USUARIOS

package clases;

```
public class usuarios {
```

```
    private int cedula;
```

```
    private String nombre;
```

```
    private Factura[] arrFactura;
```

```
    public Cliente(int cedula, String nombre) {
```

```
        this.cedula = cedula;
```

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

```
        this.arrFactura = arrFactura;
```

```
    }
```

```
    public String mostrarCliente(Cliente cliente) {
```

```
        String mostrarCliente = "Informacion del Cliente" + "\n"
```

```
        + "Nombre: " + cliente.getNombre() + "    Cédula: " + cliente.getCedula();
```

```
        return mostrarCliente;
```

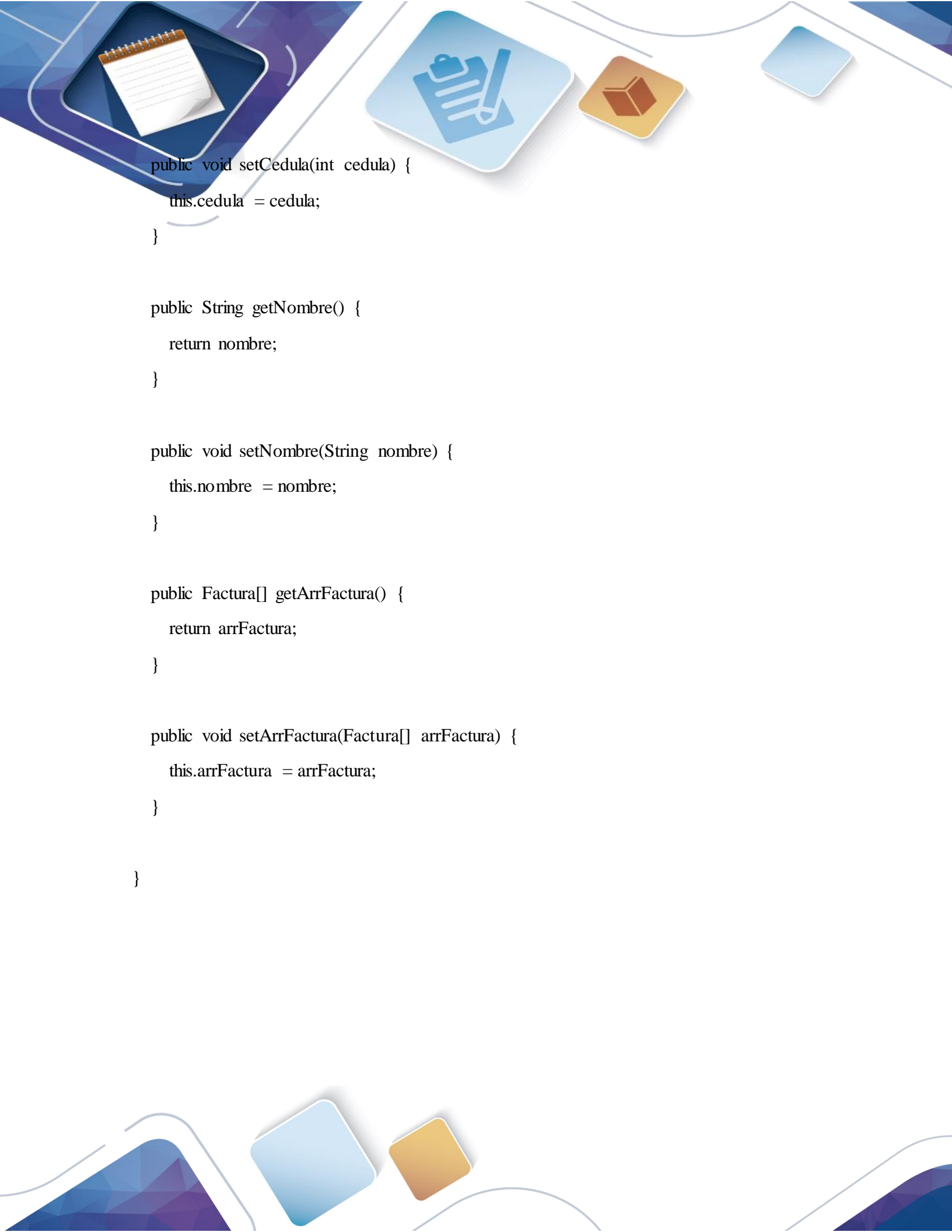
```
    }
```

```
    public int getCedula() {
```

```
        return cedula;
```

```
    }
```





```
public void setCedula(int cedula) {  
    this.cedula = cedula;  
}
```

```
public String getNombre() {  
    return nombre;  
}
```


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

```
public Factura[] getArrFactura() {  
    return arrFactura;  
}
```

```
public void setArrFactura(Factura[] arrFactura) {  
    this.arrFactura = arrFactura;  
}
```

```
}
```





Tutor realizo entrega de la actividad en los siguientes link  
Quedo muy pesado.

Link 1 Drive

Realizo entrega del link Drive donde dejo el archivo.

[https://drive.google.com/open?id=1bbUXnBNMDNcdDwR9bpo5rPK3KMSHg\\_e8k](https://drive.google.com/open?id=1bbUXnBNMDNcdDwR9bpo5rPK3KMSHg_e8k)

Link 2 Github

Realizo entrega del link del Repositorio de Github donde dejo el archivo.

<https://github.com/paolaamadoa/FASE-4-HERENCIA-POO.git>

