

## **FASE 4 HERENCIA EN LENGUAJE DE PROGRAMACIÓN JAVA**



**JULIÁN ALEXANDER ÁNGEL**

**CÓD. 1073506357**

**Grupo:  
301403\_2**

**FRANKLIN LIZCANO  
TUTOR**

**ESCUELA DE CIENCIAS BÁSICAS, TECNOLOGÍA E INGENIERÍA  
PROGRAMACIÓN ORIENTADA A OBJETOS  
ABRIL 2019**



## Actividades a desarrollar

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

- Herencia y Polimorfismo
- Herencia Simple
- Herencia Múltiple
- Herencia de Interfaz
- Herencia de Implementación
- Polimorfismo y reutilización
- Sobrecarga
- Polimorfismo en jerarquías de herencia
- Variables Polimórficas


Desarrollo de la Actividad

### -Herencia y Polimorfismo

La 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 ahorrara 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. Actividades que desarrollara subclase y la clase que ya teníamos se llamara superclase.

La subclase heredara 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.

## **Polimorfismo**

Significa literalmente muchas formas. En programación orientada a objetos es una técnica para optimizar la funcionalidad basada en tipos particulares.

La diferencia entre herencia y polimorfismo es que herencia está relacionada con clases y polimorfismo con métodos.

Existen 3 tipos de polimorfismo:

### **- Sobrecarga:**


Es cuando existen funciones con el mismo nombre, con funcionalidad similar; en clases que son completamente independientes una de la otra.

### **- Paramétrico:**

Existen funciones con el mismo nombre, pero se usan diferentes parámetros (nombre o tipo). Se selecciona el método dependiendo del tipo de datos que se mande.

### **-Inclusión:**

Es cuando se puede llamar a un método sin tener que conocer su tipo, así no se toma en cuenta los detalles de las clases especializadas, utilizando una interfaz común.



## **-Herencia Simple**

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.

## **-Herencia Múltiple:**

Consiste en la utilización de las propiedades de una clase a varias clases mas, lo que significa que en esta propiedad una sola clase padre puede heredarle atributos, u objetos de esta a varias clases hijo sin ninguna limitación entre ellas.

## **-Herencia de Interfaz**

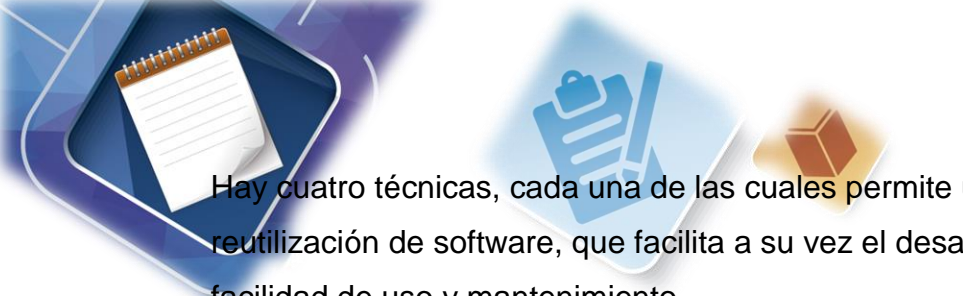
Las interfaces también pueden heredar de otras interfaces, consiguiendo así una nueva interfaz, se emplea el *extends* habitual. Teniendo así que la clase que implementa a esa nueva interfaz recibirá los métodos tanto de la interfaz base como la derivada.

## **-Herencia de Implementación**

La implementación de los métodos es heredada. Puede sobre escribirse en las clases derivadas.

## **-Polimorfismo y reutilización**

Capacidad de una entidad de referenciar distintos elementos en distintos instantes de tiempo. √ El polimorfismo nos permite programar de manera general en lugar de programar de manera específica.



Hay cuatro técnicas, cada una de las cuales permite una forma distinta de reutilización de software, que facilita a su vez el desarrollo rápido, la confianza y la facilidad de uso y mantenimiento.

Sobrecarga

Sobreescritura

Variables polimórficas

Genericidad

-Sobrecarga

Un mismo nombre de mensaje asociado a varias implementaciones v La sobrecarga se realiza en tiempo de compilación (enlace estático) en función de la signatura completa del mensaje.

Dos tipos de sobrecarga:

Basada en ámbito: Métodos con diferentes ámbitos de definición, independientemente de sus signaturas de tipo. Permitido en todos los lenguajes OO.

Un mismo método puede ser utilizado en dos o más clases. v P. ej. Sobrecarga de operadores como funciones miembros.

Basada en signatura: Métodos con diferentes signaturas de tipo en el mismo ámbito de definición. No permitido en todos los lenguajes OO.

Dos o más métodos en la misma clase pueden tener el mismo nombre siempre que tengan distinta signatura de tipos.

**-Polimorfismo en jerarquías de herencia**



## -Variables Polimórficas

Una variable polimórfica es aquella que puede referenciar más de un tipo de objeto y puede mantener valores de distintos tipos en distintos momentos de ejecución del programa.

En un lenguaje débilmente tipado todas las variables son potencialmente polimórficas

En un lenguaje fuertemente tipado la variable polimórfica es la materialización del principio de sustitución.

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 int getMaxiProducto() {
```

```
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;
```

```
}
```

```
}
```





Link de los archivos Solicitados.

[https://drive.google.com/open?id=1pqswu0L\\_TO57Rn2LBUsps6gDlfs8Wr](https://drive.google.com/open?id=1pqswu0L_TO57Rn2LBUsps6gDlfs8Wr)

<https://github.com/julian2019/Herencia-Lenguaje-Java>



## REFERENCIAS BIBLIOGRAFICAS

Weitzenfeld, A. (2005). Programación y Lenguajes Orientados a Objetos. In Ingeniería de Software Orientada a Objetos con UML, Java e Internet (pp. 25-28). Mexico City, Mexico: Cengage Learning. Recuperado

de: <http://bibliotecavirtual.unad.edu.co:2081/ps/i.do?p=GVRL&u=unad&id=GALE|CX3004300018&v=2.1&it=r&sid=GVRL&asid=0d192802>

Ceballos, S. F. J. (2010). Java 2: curso de programación (4a. ed.). Recuperado

de <https://bibliotecavirtual.unad.edu.co:2538/lib/unadsp/reader.action?docID=3228856&query=herencia+en+java>

Flórez, F. H. A. (2012). Programación orientada a objetos usando java (pp. 119). Recuperado

de <https://bibliotecavirtual.unad.edu.co:2538/lib/unadsp/reader.action?docID=3203026&query=programaci%C3%B3n+con+java>

Pérez Menor, J. M., Carretero Pérez, J., García Carballeira, F., & Pérez Lobato, J. M. (2003). Herencia y polimorfismo. In Problemas resueltos de programación en lenguaje Java (pp. [283]-303). Madrid, Spain: Paraninfo. Recuperado

de: <http://bibliotecavirtual.unad.edu.co:2081/ps/i.do?p=GVRL&u=unad&id=GALE|CX2136500015&v=2.1&it=r&sid=GVRL&asid=f6553daa>

Lizcano, F. (25,01,2019). *Concepto de Herencia*. [Archivo de video]. Recuperado de: <http://hdl.handle.net/10596/23547>

A continuación, encontrará los recursos educativos adicionales como apoyo para el desarrollo de las actividades de la unidad 3.

García, L. L. F. (2010). Todo lo básico que debería saber: sobre programación orientada a objetos en Java. Bogotá, CO: Ediciones de la U. Recuperado



de: <https://bibliotecavirtual.unad.edu.co:2538/lib/unadsp/reader.action?docID=3198642>

López, G. J. L. (2014). Programación orientada a objetos C++ y Java: un acercamiento interdisciplinario. México, D.F., MX: Larousse - Grupo Editorial Patria. Recuperado de: <https://bibliotecavirtual.unad.edu.co:2538/lib/unadsp/reader.action?docID=3227905>

Ordax, C. J. M., & Aranzazu, O. D. U. P. (2012). Programación web en java. Madrid, ES: Ministerio de Educación de España. Recuperado de: <https://bibliotecavirtual.unad.edu.co:2538/lib/unadsp/reader.action?docID=3214540>

Prieto, N., Marqués, F., & Llorens, M. (2013). Empezar a programar usando JAVA (2a. ed.). Valencia, ES: Editorial de la Universidad Politécnica de Valencia. Recuperado de: <https://bibliotecavirtual.unad.edu.co:2538/lib/unadsp/reader.action?docID=3217647>

Ruiz, R. R. (2011). Fundamentos de la programación orientada a objetos: una aplicación a las estructuras de datos en Java. Córdoba, AR: El Cid Editor. Recuperado de: <https://bibliotecavirtual.unad.edu.co:2538/lib/unadsp/reader.action?docID=3220512>

Schildt, H. (2009). Java: soluciones de programación. México, D.F., MX: McGraw-Hill Interamericana. Recuperado de: <https://bibliotecavirtual.unad.edu.co:2538/lib/unadsp/reader.action?docID=3191826>

Vélez, S. J., Peña, A. A., & Gortazar, B. P. (2011). Diseñar y programar, todo es empezar: una introducción a la Programación Orientada a Objetos usando UML y Java. Madrid, ES: Dykinson. Recuperado de: <https://bibliotecavirtual.unad.edu.co:2538/lib/unadsp/reader.action?docID=3198514>



<http://www.youtube.com/watch?v=nxwD7hGmIDQ&feature=related>

<http://en.kioskea.net/contents/poo/polymorp.php3#overloading>

<http://www.youtube.com/watch?v=nxwD7hGmIDQ&feature=related>

