

POLIMORFISMO

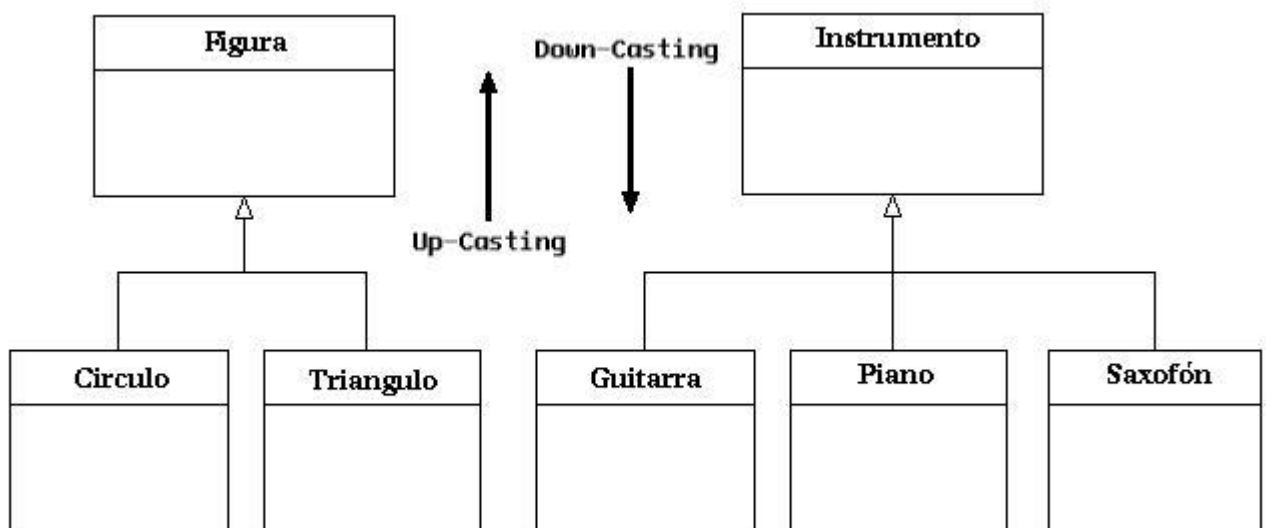
Polimorfismo, casting, up-casting, down-casting, clases Abstractas,

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.

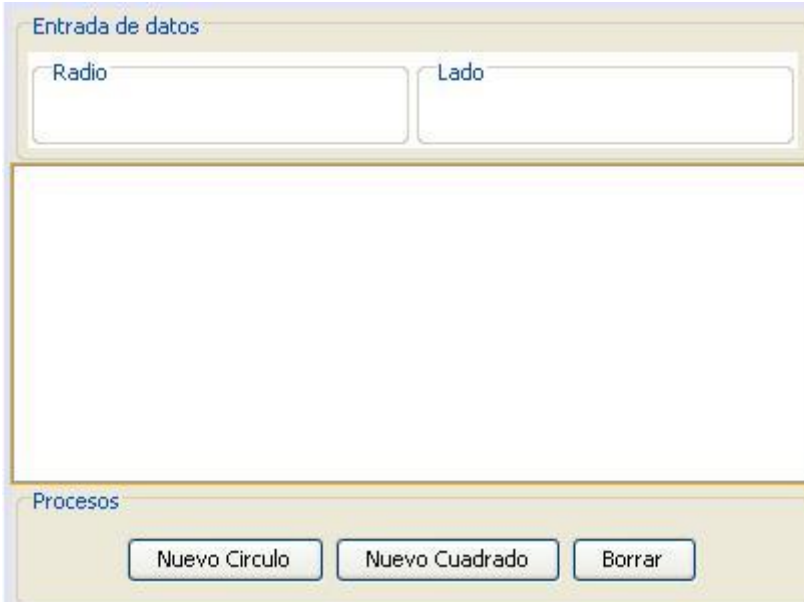
Finalmente, diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente tipo de vehiculo.

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

Cree un proyecto nuevo de nombre **P09E01**. Cree un paquete nuevo de nombre **modelo** y 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 en el paquete **modelo** 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, Cree un paquete nuevo de nombre **vista** y 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;}  
}
```

Cree un paquete nuevo de nombre **vista** e 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() );  
}
```

En el paquete **vista** 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 **modelo** y 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 en el paquete **modelo** 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, cree un paquete nuevo de nombre **vista** y diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente tipo de empleado.

En el paquete **vista** 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 **modelo** y 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, cree un paquete nuevo de nombre **vista** y diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente tipo de celular.

En el paquete **vista** 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 **modelo** y 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, cree un paquete nuevo de nombre **vista** y diseñe una clase de nombre **PanelPrincipal** con la interface necesaria para crear objetos de diferente tipo de vehiculo.

En el paquete vista diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.