

# JAVA

Pulse para añadir texto

Herencia Polimorfismo e Interfaces



TOGETHER. FREE YOUR ENERGIES

# Objetivo

- Lograr que aprendamos a programar en Java, hasta el punto de construir aplicaciones reales en la tecnología.



# Agenda de hoy

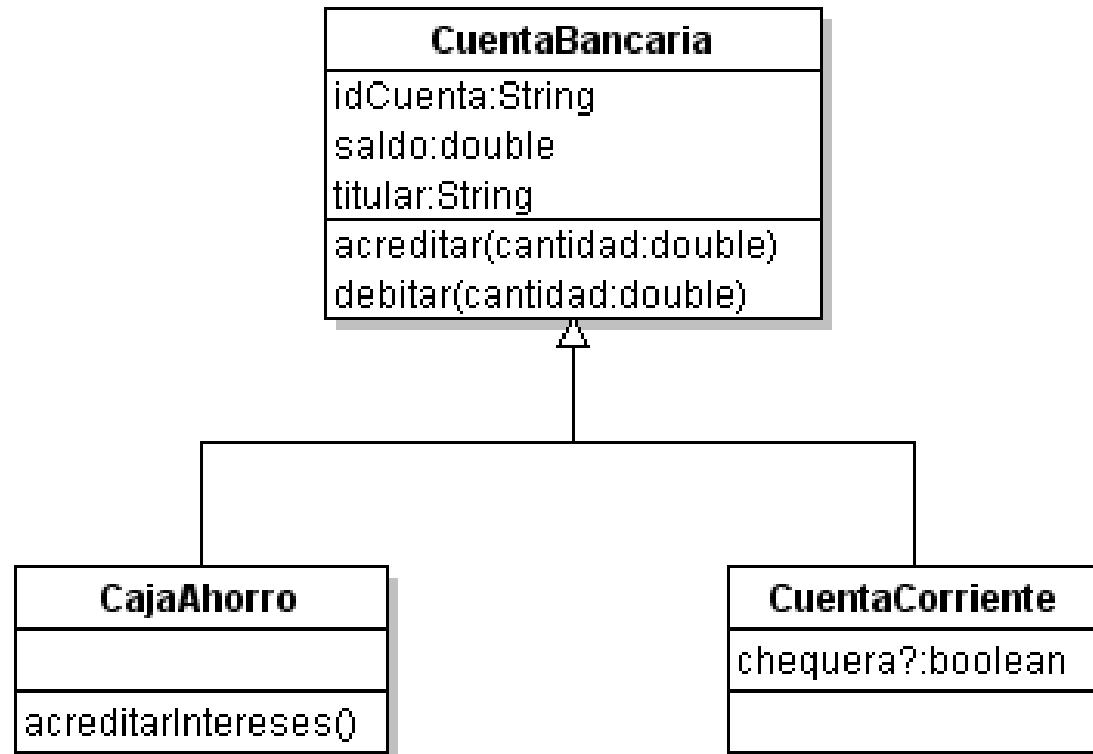
- ✓ Parte 1: Herramientas de base y J2EE

## Módulo 1:

- ✓ El lenguaje Java
  - ✓ POO en Java: Herencia, Polimorfismo, Interfaces

# Herencia I

## Herencia en UML



# Herencia II

## Indicar herencia en Java

Se utiliza la palabra reservada **extends** a continuación del nombre de la clase, para indicar de qué clase se hereda.

```
package modulo1.ejemplos;

public class CuentaBancaria {

    private String idCuenta;
    private double saldo;
    private String titular;

    public void acreditar(double cantidad){
        ;
    }

    public void debitar(double cantidad){
        ;
    }
}
```

```
package modulo1.ejemplos;

public class CajaAhorro extends CuentaBancaria
{

    public void acreditarIntereses() {
        ;
    }
}
```

# Herencia III

## La clase Object

- En Java, todas las clases heredan de otra clase, lo indiquemos o no.
- Si lo especificamos en el código con la palabra `extends`, heredarán de la clase indicada.
- Si no lo especificamos heredan del padre de todas las clases que es la clase `Object`.
- Esto significa que todas nuestras clases heredarán las propiedades y los métodos de la clase `Object`.

# Herencia IV

## Métodos de la clase Object

- Algunos métodos de Object:
- `public boolean equals (Object o)`
- `public String toString()`
- `public Class getClass()`
- `public Object clone()`

# Herencia V

## Ejemplo class Object

```
public class Punto
{
    public int x = 0;
    public int y = 0;

    public Punto(int param1, int param2)
    {
        x = param1;
        y = param2;
    }
}

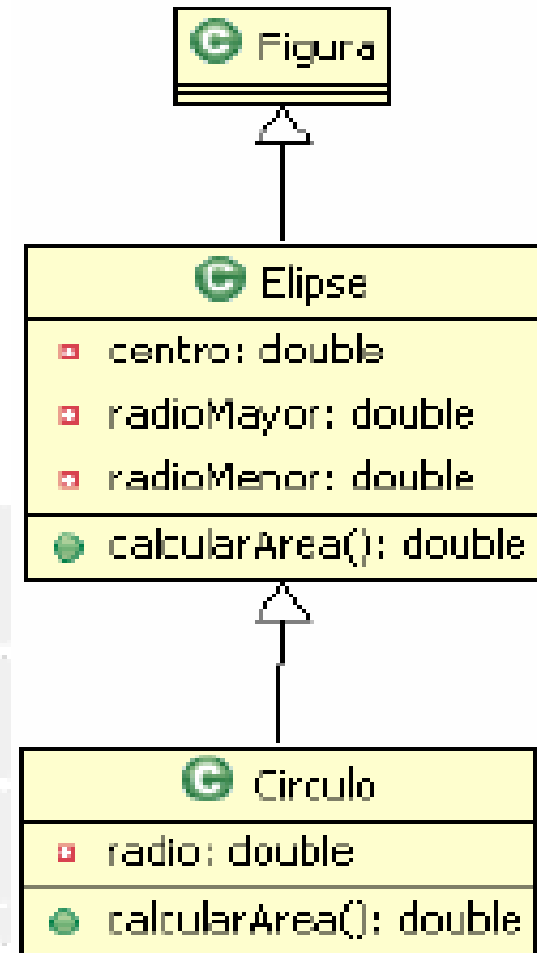
public class Test
{
    public static void main(String[] args)
    {
        Punto pun1 = new Punto(1,2);
        Punto pun2 = new Punto(1,2);
        System.out.println("Punto 1: " + pun1);
        System.out.println("Punto 2: " + pun2);
        if(pun1.equals(pun2))
            System.out.println("Son iguales.");
    }
}
```



# Herencia VI

## Overriding

- Una clase hace **overriding** de un método cuando vuelve a definir las instrucciones de un método heredado.
- Para hacer overriding un método :
  - Mismo nombre
  - Mismos parámetros y tipo de retorno
  - Modificador de acceso no puede ser más restrictivo.



# Herencia VII

## Usos de Herencia

- Debemos usar la herencia cuando haya una clase que sea más específica que otra.
- No debemos utilizar la herencia solamente por el hecho de reutilizar código.
- Si no se cumple la regla Es-un, entonces no debemos aplicar la herencia.

# Herencia VIII

## Acceso a métodos y propiedades de la clase padre

```
public class ClasePadre
{
    public boolean atributo = true;
}

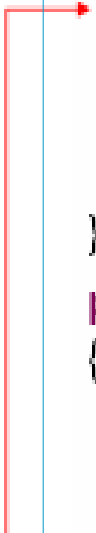
public class ClaseHija extends ClasePadre
{
    public boolean atributo = false;
    public void imprimir()
    {
        System.out.println(atributo);
        System.out.println(super.atributo);
    }
}
```

# Herencia IX

## Acceso al constructor de una superclase

```
public class ClasePadre
{
    public ClasePadre(int param)
    {
        System.out.println(param);
    }
}


public class ClaseHija extends ClasePadre
{
    public ClaseHija(int param)
    {
        super(param + 2);
        System.out.println(param);
    }
}
```



## Acceso a un método de la superclase

```
public class ClasePadre
{
    public void imprimir()
    {
        System.out.println("Método del padre");
    }
}

public class ClaseHija extends ClasePadre
{
    public void imprimir()
    {
        super.imprimir();
        System.out.println("Método del hijo");
    }
}
```



# Herencia X

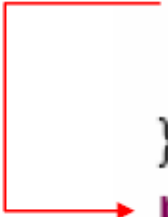
## Ejemplo de uso de this

```
public class MiClase
{
    private int x = 5;
    public void setX(int x)
    {
        System.out.println("x local vale: " + x);
        System.out.println("x atributo vale: " + this.x);
        this.x = x;
        System.out.println("x atributo vale: "
                               + this.x);
    }
}
```

# Herencia XI

## Ejemplo de uso de this

```
public class MiClase
{
    public MiClase()
    {
        this(2);
        System.out.println("Constructor sin");
    }
    public MiClase(int param)
    {
        System.out.println("Constructor con");
    }
}
```



# Ejercicio



## Ejercicio V de la guía



# Clases Abstractas I

## Clases abstractas

- Una clase abstracta no puede ser instanciada.
- Sí pueden definirse subclases concretas de las mismas.
- Heredan sus métodos y propiedades
- Por ejemplo, la clase Figura podría ser abstracta, ya que nunca vamos a hacer una instancia suya, sino de sus hijos (círculos, triángulos, ...)



# Clases Abstractas II

## Métodos abstractos

- Son métodos que no están implementados, es decir, no tienen instrucciones que ejecutar, solamente tienen una signatura.
- Una clase que tenga algún método abstracto tendrá que ser declarada abstracta.
- Cuando una clase hereda de otra que tiene métodos abstractos, la clase hija deberá implementar obligatoriamente todos estos métodos.
- Se utilizan precisamente para obligar a que las clases hijas tengan que implementarlos.

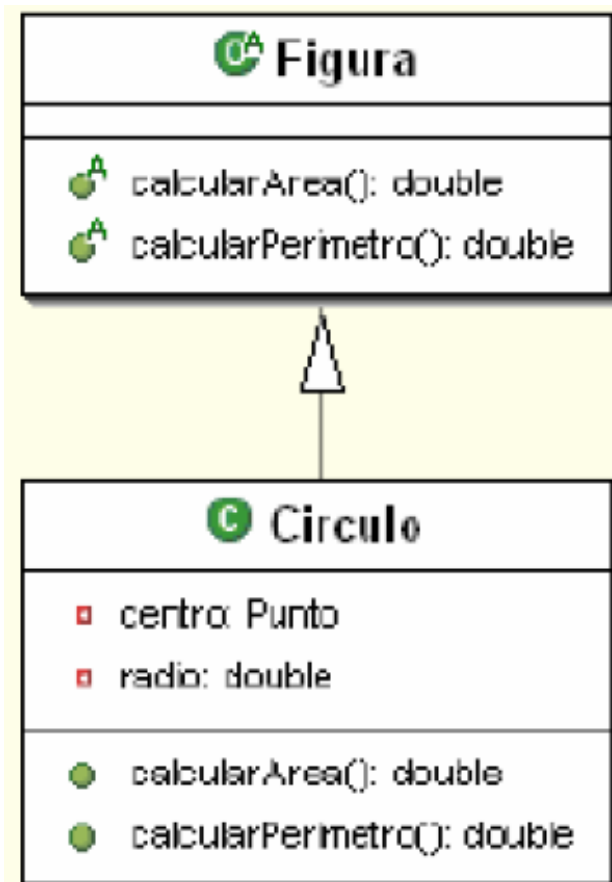
# Clases Abstractas III

```
public abstract class Figura
{
    public abstract double calcularArea();
    public abstract double calcularPerimetro();
}

public class Circulo extends Figura
{
    private Punto centro = null;
    private double radio = 0.0;

    public double calcularArea()
    {
        return Math.PI*radio*radio;
    }

    public double calcularPerimetro()
    {
        return 2*Math.PI*radio;
    }
}
```



# Polimorfismo I

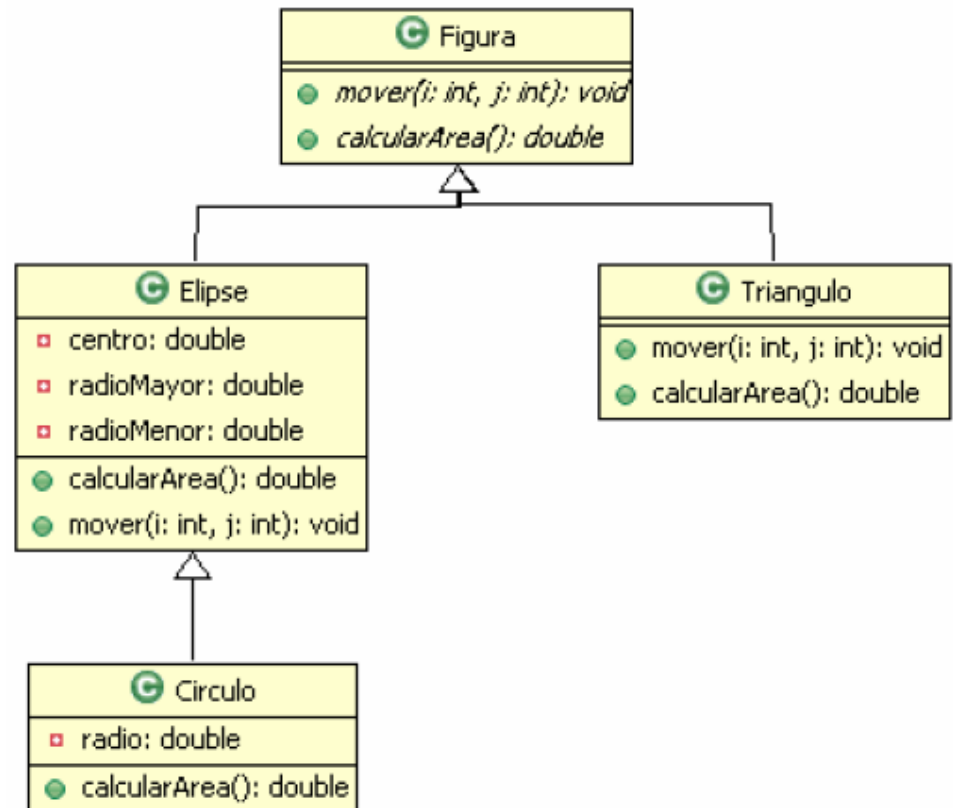
## Polimorfismo

- Es un concepto de teoría de tipos en el que un nombre (una variable) puede denotar objetos de clases diferentes que tienen una superclase común.

# Polimorfismo II

## ■ Ejemplo

```
public static void main(String[] args)
{
    Figura[] figuras = new Figura[4];
    figuras[0] = new Circulo();
    figuras[1] = new Elipse();
    figuras[2] = new Triangulo();
    for (int i=0; i<figuras.length; i++)
        figuras[i].mover(0,0);
}
```



# Interfaces I

- Una interface es una clase especial que:
  - Si tiene alguna propiedad, debe ser una constante.
  - Todos los métodos que tiene han de ser abstractos, es decir, no implementa ninguno.
- Las clases que implementen una interface, deberán escribir todos los métodos que contenga la interface, excepto si es abstracta.

```
public interface MiInterface  
{  
}
```

```
public class MiClase implements MiInterface  
{  
}
```

# Interfaces II

- Polémica

Clase Abstracta

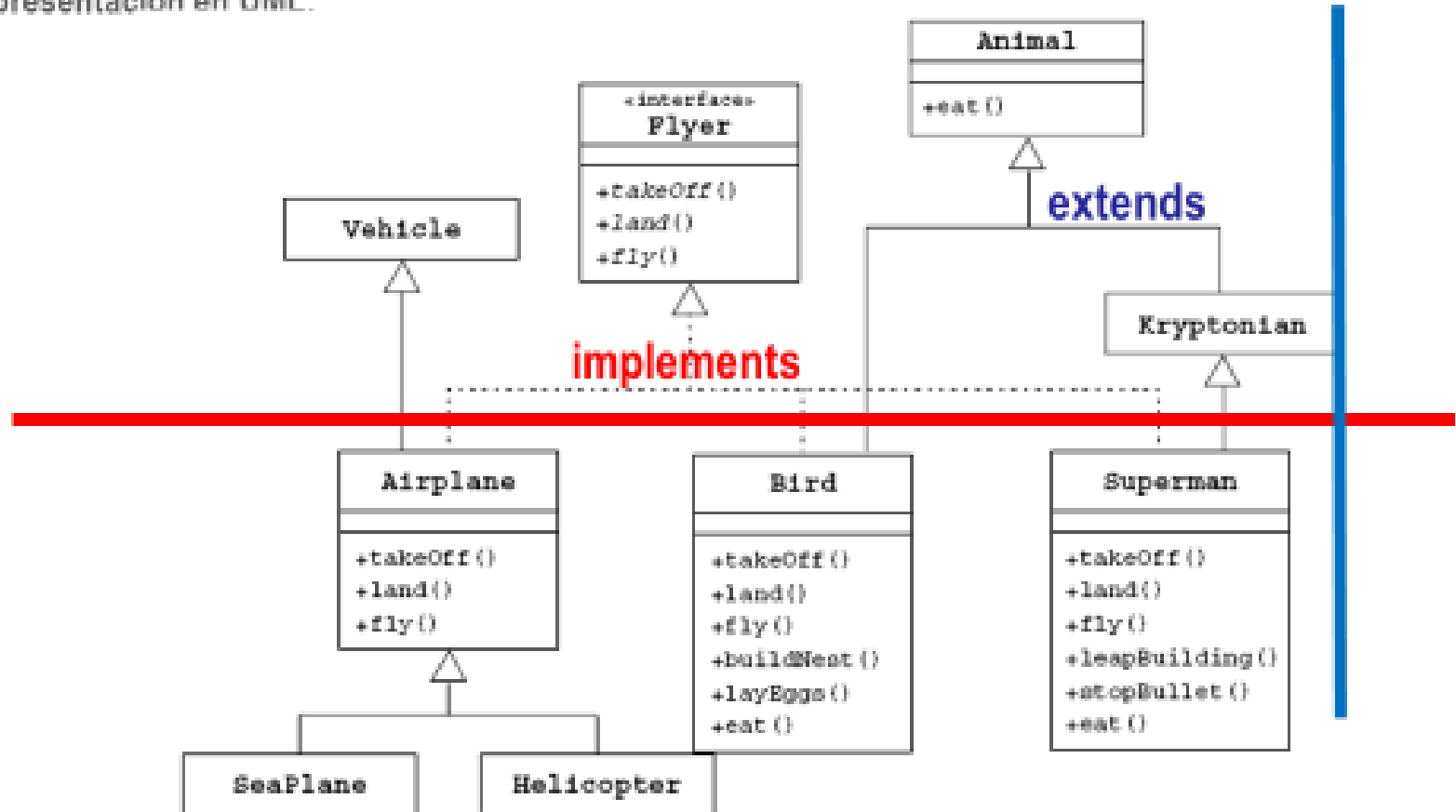
Interface



# Conceptos Básicos de Java

## Interfaces

Representación en UML:



# Ejercicio



## Ejercicio VI de la guía





**Muchas gracias!**