Java

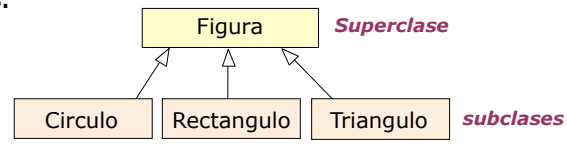
Tema 5

Herencia

INDICE

Fundamentos de la Herencia

- La **herencia** permite la creación de clasificaciones jerárquicas de clases.
- Consiste en crear una clase general o **superclase**, con los atributos y métodos comunes de un conjunto de **subclases** más especializadas.
- Las subclases "heredan" los atributos y métodos de la superclase y añaden a su definición sus propios atributos y métodos.



- No hay herencia multiple, solo se puede heredar de una clase

3

Fundamentos de la Herencia

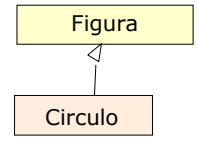
- Sintaxis de declaración de una subclase:

```
class nombre_subclase extends nombre_superclase {
    // cuerpo de la clase
}
```

extiende a, hereda de

- Ejemplo:

```
class Circulo extends Figura {
    // cuerpo de la clase
}
```



4

Ejemplo: proyecto Figuras

```
public class Figura {

   protected double x, y;

   public Figura(){
      this.x=0;
      this.y=0;
   }

}
```

- ✓ El modificador **protected** hace que los atributos de una superclase solo puedan ser accedidos directamente desde ella misma, por sus clases hijas y las clases de su mismo package.

Ejercicio: En el mismo proyecto donde esta la clase Figura, crea otro package, y dentro una java main class declarando un objeto Figura. Comprueba que no tienes acceso a sus atributos protected x e y.

5

Ejemplo: proyecto Figuras

```
public class Circulo extends Figura{

   public double radio;

   Circulo(double radio) {
      this.radio = radio;
   }

   public String coordenadas(){
      return "(" + x + "," + y + ")";
   }

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

}
```

- ✓ La clase Circulo hereda los atributos y métodos de la clase Figura. Tiene acceso directo a ellos si son public o protected

6

Ejemplo: proyecto Figuras

```
public class Rectangulo extends Figura{

   public double base, altura;

   public Rectangulo(double base, double altura) {
     this.base = base;
     this.altura= altura;
   }

   public String coordenadas(){
      return "(" + x + "," + y + ")";
   }

   public double area(){
      return base * altura;
   }
}
```

- ✓ La clase Rectangulo hereda los atributos y métodos de la clase Figura.

Ejemplo: proyecto Figuras

```
public class FiguraApp1 {

  public static void main(String[] args) {

   Circulo     c1 = new Circulo(5.5);
   Rectangulo  r1 = new Rectangulo(3.0,7.0);

   System.out.println("Coord.= ( " + c1.x + ", " + c1.y + " )");
   System.out.println("Radio=" + c1.radio);
   System.out.println("Área circulo: "+ c1.area());

  System.out.println( r1.coordenadas() );
  System.out.println("Base=" + r1.base + " Altura=" + r1.altura);
  System.out.println("Área rectangulo:" + r1.area());

  }
}
```

Acceso a miembros y herencia

- Aunque una subclase hereda todos los atributos de la superclase, no puede acceder directamente a los atributos de su superclase declarados **private**.

```
/* Si declaramos una variable privada en Figura */
class Figura {
    private String color;
    . . .
}

class Circulo extends Figura {
. . .
  public String getColor(){
    return color;
  }
}
```

¡¡¡ERROR !!! color es un atributo privado en Figura

Uso de *super*() para llamar a constructores de una superclase

- Una clase hija puede llamar al constructor de su clase padre:

super (*parámetros _constructor_superclase*);

```
class Circulo extends Figura{

   private double radio;

   Circulo(double radio) {

      this.radio = radio;

   }

}
```

✓Como el constructor de Figura no tiene parámetros, Java crea el objeto Figura internamente haciendo una llamada implícita a su constructor, es decir como si pusiera:

```
super();
```

Uso de *super* para llamar a constructores de una superclase

- Si el constructor de la clase padre sí tiene parámetros, el constructor de la clase hija debe llamarlo explícitamente:

```
class Figura {
  protected double x, y;
  public Figura(double x, double y){
    this.x = x;
    this.y = y;
  }
}
```

```
class Circulo extends Figura{

   private double radio;

   Circulo(double x, double y, double radio) {
     super(x,y);       // llamada al constructor de Figura
     this.radio = radio;
   }
}
```


Referencias a objetos superclase y objetos subclases

- A una variable objeto de la clase padre (Figura), podemos asignarle variables de su clase (otra Figura) ó de cualquier subclase derivada de ella (Circulo, Rectangulo)

```
   Figura figura2;

// Correcto, figura es de la misma clase
   Figura figura = new  Figura(2.0, 3.5);
   figura2 = figura;
   System.out.println ( figura2.x + ", " + figura2.y);

// Correcto, c1 es subclase de Figura
    Circulo c1 = new  Circulo(1.6, 2.5, 4.0);
   figura2 = c1;
   System.out.println ( figura2.x + ", " + figura2.y);

// Correcto, r1 es subclase de Figura
    Rectangulo r1  = new  Rectangulo (1,1,5,7);
   figura2 = r1;
   System.out.println ( figura2.x + ", " + figura2.y);
```

Operador: instanceof

- Mediante el operador lógico (devuelve boolean) **instanceof** podemos averiguar si la variable objeto es una instancia de una clase concreta

variable **instanceof** nombreclase

```
Figura f1  = new  Circulo(1.6, 2.5, 4.0);

if ( f1  instanceof  Circulo)
    System.out.println ( "es un circulo");

if ( f1  instanceof  Rectangulo )
    System.out.println ( "es un rectangulo");
```

- **Ejercicio: ¿Qué saldria por pantalla?**

13

Herencia clase Object

- Todas las clases en Java heredan de la clase **Object**, por tanto heredan sus métodos, entre estos destacamos:
  - String **toString**()
    Devuelve en modo texto el nombre de la clase
  - Object **clone**()
    Crea una copia del objeto
  - Boolean **equals**(Object o)
    Devuelve true si el objeto es igual al objeto pasado como parámetro

14

Redefinición usando @Override

- Un método heredado puede ser definido de nuevo, es decir modificar su implementación. Se indica anteponiendo al método **@Overrride**

```
public class Figura{
  @Override
  public String toString(){
     return  x + "-" + y;
  }
```

- Se puede invocar la implementacion anterior a la redefinición, es decir la de la clase padre, usando **super.nombre_metodo()**

```
public class Circulo{
  @Override
  public String toString(){
     return  super.toString() + "**" + x + "-" + y;
}
```

15

Conceptos de herencia

•Recuperar el nombre de la clase de un objeto

```
Figura f;
System.out.println( f.getClass().getSimpleName());
```

•Conversión explícita a Circulo para poder acceder a sus propiedades especificas.

```
Figura figura = new Circulo(1.6, 2.5, 4.0);

Circulo c = (Circulo) figura;

System.out.println( c.radio);
```

16

Ver documento T5-HerenciaEjemplo.pdf