PROGRAMACIÓN ORIENTADA A OBJETOS - 22951

Universidad Industrial de Santander

#LaUISqueQueremos

Universidad Industrial de Santander

Escuela de Ingeniería de Sistemas e Informática

Programa de Ingeniería de Sistemas

74 años 1948-2022

Clases abstractas y Síntesis Herencia y Polimorfismo e Interfaces

Herencia

- **Objetivo**: reutilizar piezas de software (clases) existentes.
- **Mecanismo**: Crear una nueva clase más especifica (**especializar**) a partir de una más genérica (próxima).
- **Ventaja**: Economía, no se debe definir todo, sólo aquello que es especifico.
- Basada en la **relación «es un»**, esta relación no es simétrica:
  - Todo león **es un** felino.
  - Todo felino **es un** mamífero.
  - Todo mamífero **es un** animal.

Herencia

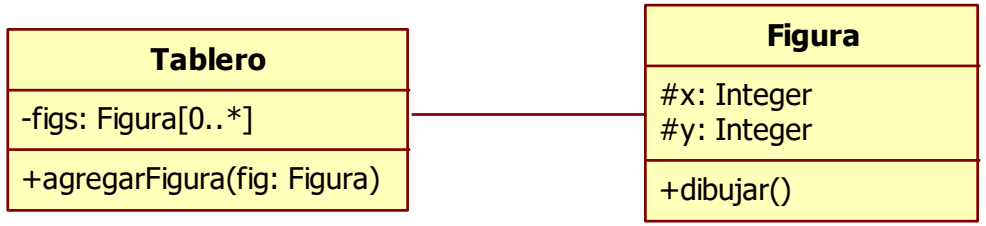
- ¿Qué se hereda?: atributos y métodos.
- ¿Qué se puede acceder directamente?: miembros (atributos y métodos) públicos o protegidos.
- Redefinición de métodos: dar en una subclase una definición mas adaptada, utilizando la misma firma del método de la superclase.
- Acceso a la superclase: la palabra clave ***super*** permite acceder a métodos de la superclase que han sido redefinidos.
- Encadenar constructores: toda instancia de una subclase contiene una instancia de la superclase. Para crear una instancia de la subclase debemos crear una instancia de la superclase usando uno de sus constructores a través de la palabra clave ***super***.

Polimorfismo

- Objetivo: permitir una evolución controlada de un programa OO en el futuro.
- Mecanismo: facultad de una instancia de una clase derivada (subclase) a ser considerada como una instancia de una superclase.
- Ventaja: se puede definir el marco evolutivo de una aplicación.
- Conversión de tipos:
  - Ascendente: de una subclase a una superclase. Implícita y autorizada por el compilador.
  - Descendente: de una superclase a una subclase. Explicita, controlada en ejecución.

Polimorfismo

- **Enlace dinámico:** permite a un objeto referenciado por una variable de una superclase invocar el método definido en la subclase.



Clases Abstractas

Guernica – Pablo Picasso

Clases abstractas

- Suponga que usted debe implementar una jerarquía de clases que representen figuras geométricas (*Rectángulo, Cuadrado, Circulo*, etc.).
- Los métodos que deben tener las clases creadas son
  - Un método para calcular el área.
  - Un método para calcular el perímetro.
  - El método *toString()* que retorne lo siguiente «Área: 20 – Perímetro: 60».
- ¿Cuál sería la clase de base y qué métodos tendría esta clase?
- ¿Que código usaría para implementar los métodos descritos?

Clases abstractas

- Una clase abstracta permite definir parcialmente una clase.
- Algunos de los métodos para los que no se conoce la implementación son declarados abstractos.
- Si hay por lo menos un método abstracto la clase debe abstracta.
- Una clase abstracta puede no contener métodos abstractos.
- No se pueden crear instancias de clases abstractas.
- Una subclase de una clase abstracta debe implementar todos los métodos abstractos o ser definida ella misma como abstracta

Clases abstractas

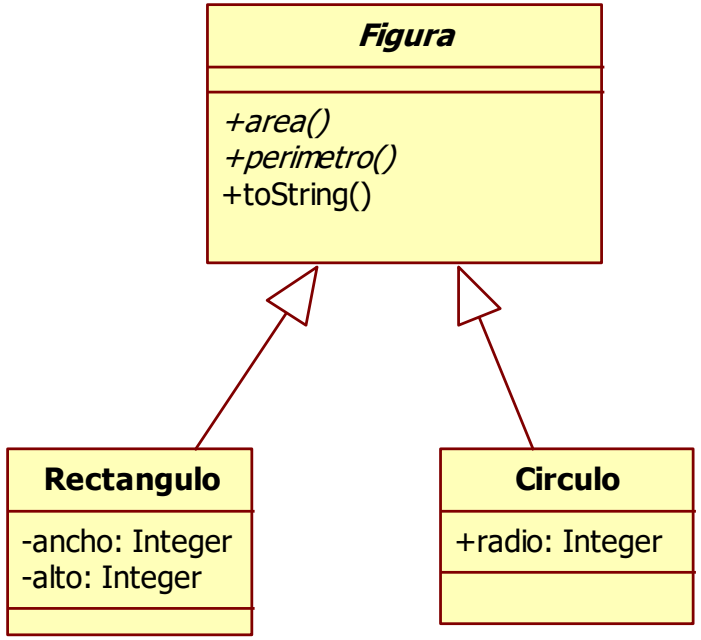
```
public abstract class Figura {

   public abstract int area();

   public  abstract int perimetro();

   public  String toString() {
      return "Area: " + area() + " Perimetro: " + perimetro();
   }

}
```

```
public class Rectangulo extends Figura {

   private int ancho, alto;

   public Rectangulo(int ancho, int alto) {
      this.ancho = ancho;
      this.alto = alto;
   }

   public int area() {
      return ancho * alto;
   }

   public int perimetro() {
      return (2*ancho) + (2*alto);
   }
}
```

```java
public class Circulo extends Figura {

    private int radio;

    public Circulo(int radio) {
        this.radio = radio;
    }

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

    public int perimetro() {
        return (int) Math.PI * 2 * radio;
    }

}
```

```
public void test1() {
    Figura[] figs = new Figura[3];
    figs[0] = new Rectangulo(10, 2);
    figs[1] = new Circulo(10);
    figs[2] = new Rectangulo(5, 3);

    for (int i = 0; i < figs.length; i++) {
       Figura fig = figs[i];
       System.out.println(fig);
    }
}

public void test2() {
    Figura[] figs = new Figura[3];
    figs[0] = new Rectangulo(10, 2);
    figs[1] = new Circulo(10);
    figs[2] = new Figura();

    for (int i = 0; i < figs.length; i++) {
       Figura fig = figs[i];
       System.out.println(fig);
    }
}
```

Clases abstractas en UML



Clases Finales

- Permiten al desarrollador indicar que una clase no podrá ser extendida.

```
public final class Circulo extends Figura {

   private int radio;

   public Circulo(int radio) {
      this.radio = radio;
   }

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

   public int perimetro() { return (int) Math.PI * 2 * radio; }

}
```

Interfaces


INPUT 7
HDMI
>PC<
PD372WJ
ETHERNET
(10/100)
INPUT 6
INPUT 8
AUDIO
DIGITAL
AUDIO
OUTPUT
OUTPUT
R-AUDIO-L
ANT/CABLE
ANALOG
RGB (PC IN)
RS-232C
S-VIDEO
COMPO-
NENT
VIDEO
COMPO-
NENT
Y
AUDIO
L
R
PB
PR
INPUT 4
INPUT 2
INPUT 1

Interfaces

- Suponga que debe posicionar las figuras geométricas en un plano: el centro para el Circulo y el punto superior izquierdo para el Rectángulo.
- Suponga que usted creó una clase ObjetoEnPlano que posee un método setPosicion(int x, int y) para definir la posición de la instancia.
- ObjetoEnPlano debe brindar también un método dibujar(Tablero tab).
- Circulo y Rectángulo deben seguir siendo subclases de Figura.
- Ya que Java no permite herencia múltiple se debe utilizar otro mecanismo: las interfaces

Interfaces

- El objetivo de una Interface es definir la interface o API de una familia de objetos
- La definición de una interface es parecido a la definición de una clase sin la palabra claves class
- Una interface no contiene ninguna implementación, solo la definición de los métodos.
- Todos los métodos de una interface son implícitamente abstractos
- Todos los métodos de una interface son públicos, incluso si la palabra public no es utilizada.
- Las interfaces no tienen atributos
- Las interfaces no tienen constructores

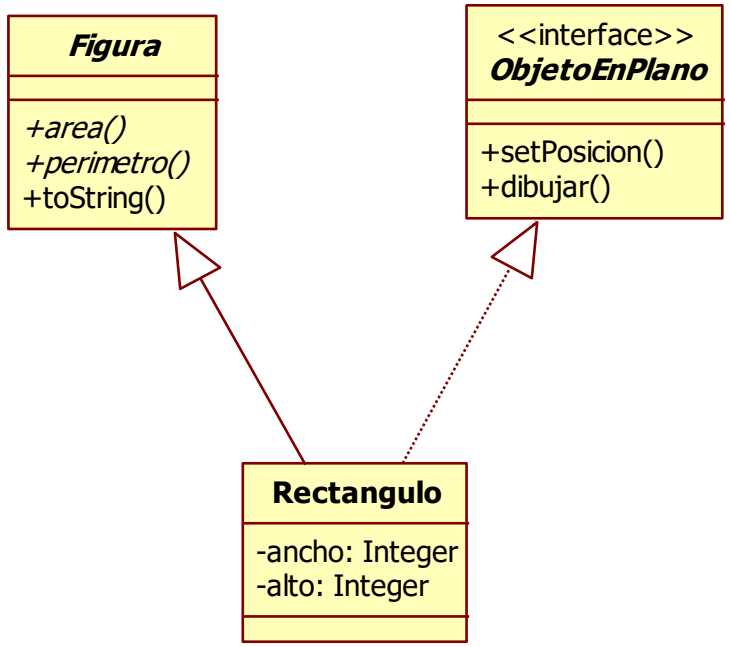
Interfaces

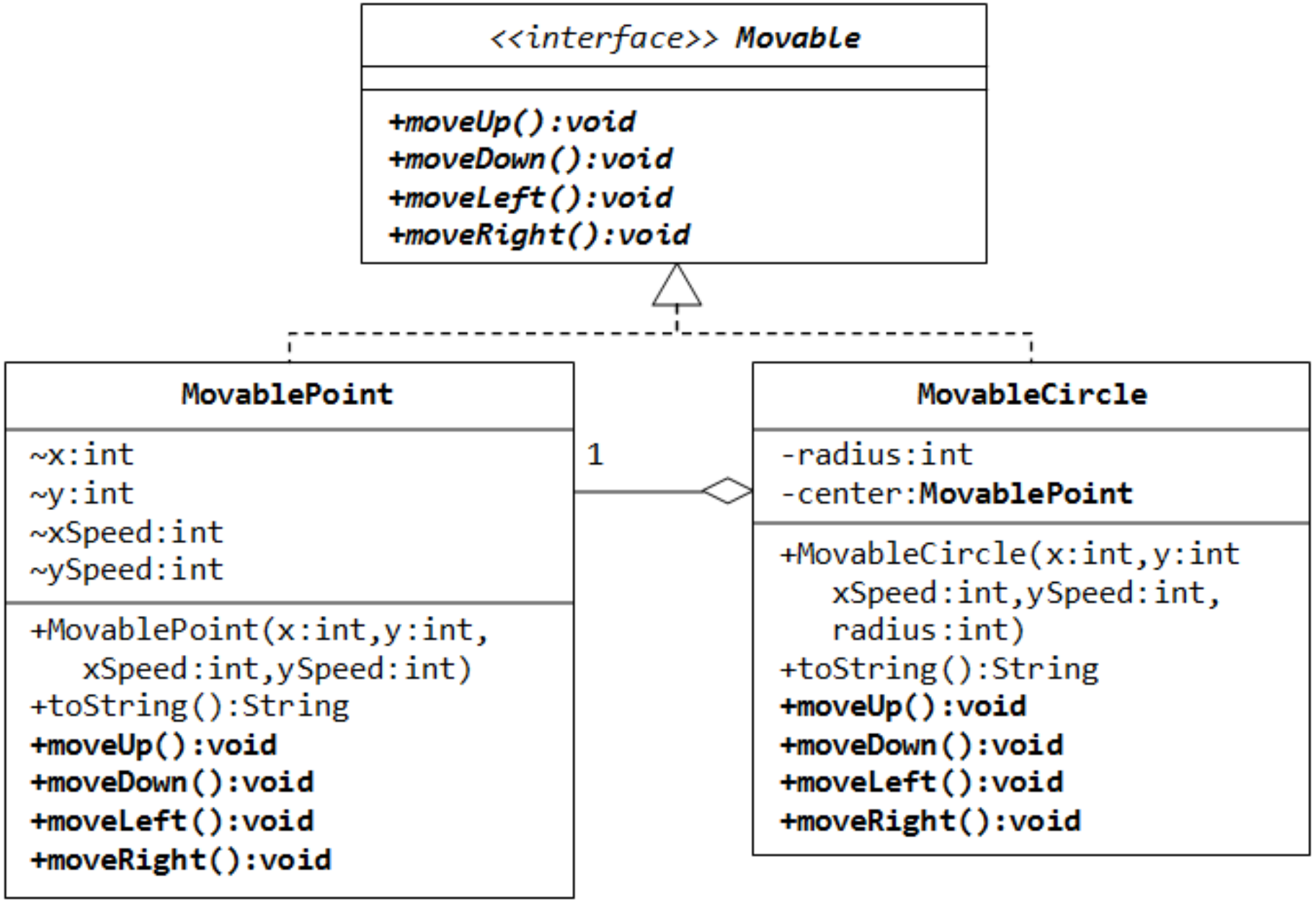
```
public interface ObjetoEnPlano {

   public void setPosicion(int x, int y);

   public void dibujar(Tablero tab);

}
```

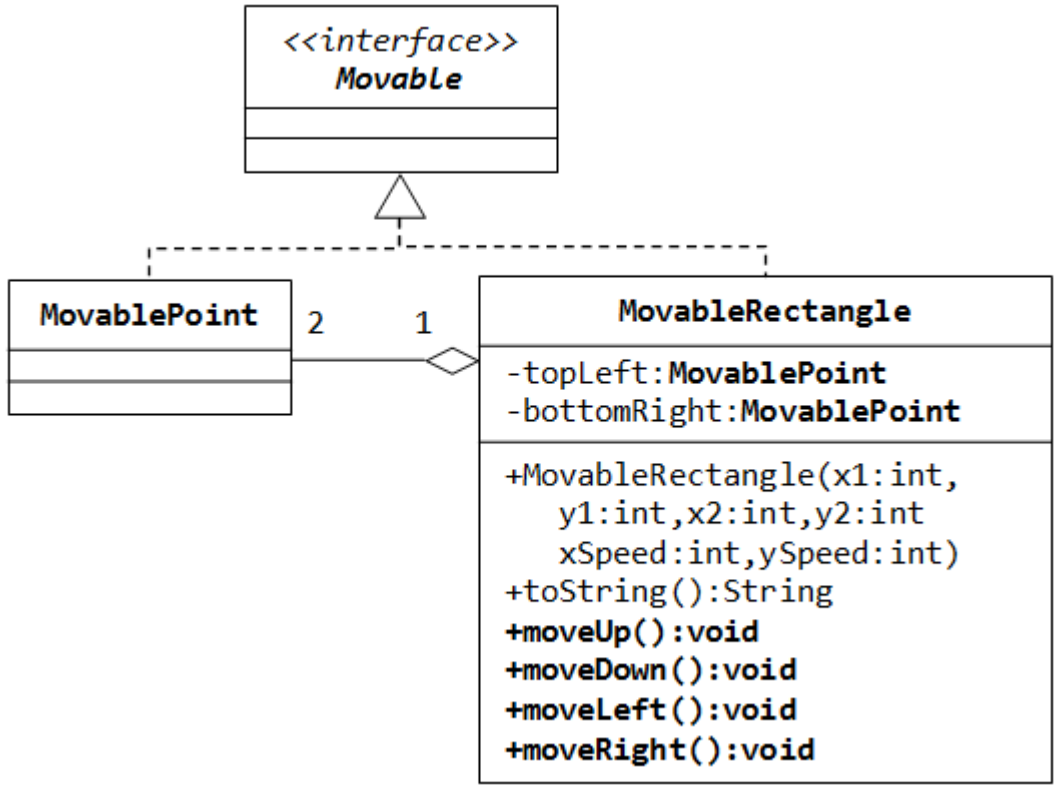
```
public class Circulo extends Figura implements ObjetoEnPlano {

   private int radio, x, y;

   public Circulo(int radio) {
      this.radio = radio;
   }

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

   public int perimetro() { return (int) Math.PI * 2 * radio; }

   public void setPosicion(int x, int y) {
      this.x = x;
      this.y = y;
   }

   public void dibujar(Tablero tab) {
      // Codigo para dibujar
   }

}
```

```
public class Rectangulo extends Figura implements
ObjetoEnPlano {

   private int ancho, alto, x, y;

   public Rectangulo(int ancho, int alto) {
      this.ancho = ancho;
      this.alto = alto;
   }

   public int area() { return ancho * alto; }

   public int perimetro() { return (2*ancho) + (2*alto); }

   public void setPosicion(int x, int y) {
      this.x = x;
      this.y = y;
   }

   public void dibujar(Tablero tab) {
      // Codigo para dibujar
   }
}
```

Interfaces en UML




<<interface>> Movable
+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void
MovablePoint
~x:int
~y:int
~xSpeed:int
~ySpeed:int
+MovablePoint(x:int,y:int,
xSpeed:int,ySpeed:int)
+toString():String
+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void
1
MovableCircle
-radius:int
-center:MovablePoint
+MovableCircle(x:int,y:int
xSpeed:int,ySpeed:int,
radius:int)
+toString():String
+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void


<<interface>>
Movable
MovablePoint
2
1
MovableRectangle
-topLeft:MovablePoint
-bottomRight:MovablePoint
+MovableRectangle(x1:int,
y1:int,x2:int,y2:int
xSpeed:int,ySpeed:int)
+toString():String
+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void

Universidad Industrial de Santander

#LaUISqueQueremos

¡Gracias!

#OrgulloEISI