Lenguajes, Tecnologías y Paradigmas de la programación (LTP)

Práctica 1: Herencia y sobrecarga

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Sergio Pérez
serperu@dsic.upv.es

Introducción

```
public class Circle {
 private double x, y;
 private double radius;
 public Circle(double a,
        double b, double c){
   x = a; y = b; radius = c;
 }

 public boolean equals(Object o){
  if (!(o instanceof Circle)) {
     return false; }
  Circle c = (Circle) o;
  return x == c.x && y == c.y &&
  radius == c.radius;
 }

 public String toString(){
  return ``Circle:\n\t'' +
    ``Position: ('' +x+ ``,'' +
    y+``)\n\tRadius: '' +radius;
 }
}
```

Introducción

```
public class Circle {
 private double x, y;
 private double radius;
 public Circle(double a,
        double b, double c){
   x = a; y = b; radius = c;
 }

 public boolean equals(Object o){
  if (!(o instanceof Circle)) {
     return false; }
  Circle c = (Circle) o;
  return x == c.x && y == c.y &&
  radius == c.radius;
 }

 public String toString(){
  return ``Circle:\n\t'' +
    ``Position: ('' +x+ ``,'' +
    y+``)\n\tRadius: '' +radius;
 }
}
```

```
public class Triangle {
  private double x, y;
  private double base, height;
  public Triangle(double a,
     double b, double c, double d){
   x = a; y = b;
   base = c; height = d;
  }
  public boolean equals(Object o){
    if (!(o instanceof Triangle)){
       return false; }
    Triangle t = (Triangle) o;
    return x == t.x && y == t.y &&
           base == t.base &&
           height == t.height;
  }
  public String toString(){
    return ``Triangle:\n\t'' +
      ``Position: ('' +x+ ``,'' +
      y+``)\n\tBase: '' +base+
      ``\n\tHeight: '' +height;
  }
}
```

Introducción

```
public class Circle {
 private double x, y;
 private double radius;
 public Circle(double a,
        double b, double c){
   x = a; y = b; radius = c;
 }

 public boolean equals(Object o){
  if (!(o instanceof Circle)) {
     return false; }
  Circle c = (Circle) o;
  return x == c.x && y == c.y &&
  radius == c.radius;
 }

 public String toString(){
  return ``Circle:\n\t'' +
    ``Position: ('' +x+ ``,'' +
    y+``)\n\tRadius: '' +radius;
 }
}
```

```
public class Triangle {
  private double x, y;
  private double base, height;
  public Triangle(double a,
     double b, double c, double d){
   x = a; y = b;
   base = c; height = d;
  }
  public boolean equals(Object o){
    if (!(o instanceof Triangle)){
       return false; }
    Triangle t = (Triangle) o;
    return x == t.x && y == t.y &&
           base == t.base &&
           height == t.height;
  }
  public String toString(){
    return ``Triangle:\n\t'' +
      ``Position: ('' +x+ ``,'' +
      y+``)\n\tBase: '' +base+
      ``\n\tHeight: '' +height;
  }
}
```

Introducción

```
public class Circle {
 private double x, y;
 private double radius;
 public Circle(double a,
       double b, double c){
  x = a; y = b; radius = c;
 }

 public boolean equals(Object o){
  if (!(o instanceof Circle)) {
     return false; }
  Circle c = (Circle) o;
  return x == c.x && y == c.y &&
  radius == c.radius;
 }

 public String toString(){
  return ``Circle:\n\t'' +
    ``Position: ('' +x+ ``,'' +
    y+``)\n\tRadius: '' +radius;
 }
}
```

```
public class Triangle {
  private double x, y;
  private double base, height;
  public Triangle(double a,
     double b, double c, double d){
   x = a; y = b;
    base = c; height = d;
  }
  public boolean equals(Object o){
    if (!(o instanceof Triangle)){
       return false; }
    Triangle t = (Triangle) o;
    return x == t.x && y == t.y &&
           base == t.base &&
           height == t.height;
  }
  public String toString(){
    return ``Triangle:\n\t'' +
      ``Position: ('' +x+ ``,'' +
      y+``)\n\tBase: '' +base+
      ``\n\tHeight: '' +height;
  }
}
```

Introducción

```
public class Circle {
 private double x, y;
 private double radius;
 public Circle(double a,
       double b, double c){
  x = a; y = b; radius = c;
 }

 public boolean equals(Object o){
  if (!(o instanceof Circle)) {
     return false; }
  Circle c = (Circle) o;
  return x == c.x && y == c.y &&
  radius == c.radius;
 }

 public String toString(){
  return ``Circle:\n\t'' +
   ``Position: ('' +x+ ``,'' +
   y+``)\n\tRadius: '' +radius;
 }
}
```

```
public class Triangle {
 private double x, y;
 private double base, height;
 public Triangle(double a,
   double b, double c, double d){
  x = a; y = b;
  base = c; height = d;
 }
 public boolean equals(Object o){
  if (!(o instanceof Triangle)) {
     return false; }
  Triangle t = (Triangle) o;
  return x == t.x && y == t.y &&
          base == t.base &&
          height == t.height;
 }
 public String toString(){
  return ``Triangle:\n\t'' +
   ``Position: ('' +x+ ``,'' +
   y+``)\n\tBase: '' +base+
    ``\n\tHeight: '' +height;
 }
}
```

Herencia: Figure

```
public class Figure {
  private double x, y;
  public Figure(double x, double y) {
    this.x = x; this.y = y;
  }
  public boolean equals(Object o) {
    if (!(o instanceof Figure)) { return false; }
    Figure f = (Figure) o;
    return x == f.x && y == f.y;
  }
  public String toString() {
    return ``Position: (`` + x + ``, `` + y + ``)``;
  }
}
```

Herencia: Figure

```
public class Figure {
  private double x, y;
  public Figure(double x, double y) {
    this.x = x; this.y = y;
  }
  public boolean equals(Object o) {
    if (!(o instanceof Figure)) { return false; }
    Figure f = (Figure) o;
    return x == f.x && y == f.y;
  }
  public String toString() {
    return ``Position: (`` + x + ``, `` + y + ``)``;
  }
}
```

Herencia: Figure

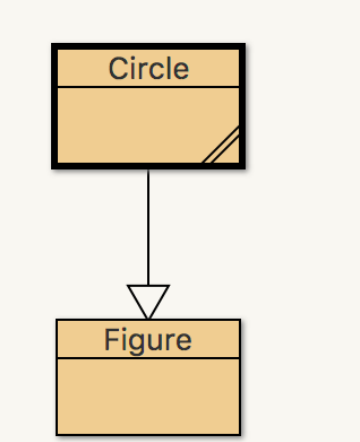
```
public class Figure {
  private double x, y;
  public Figure(double x, double y) {
    this.x = x; this.y = y;
  }
  public boolean equals(Object o) {
    if (!(o instanceof Figure)) { return false; }
    Figure f = (Figure) o;
    return x == f.x && y == f.y;
  }
  public String toString() {
    return ``Position: (`` + x + ``, `` + y + ``)``;
  }
}
```

Herencia: Figure

```
public class Figure {
  private double x, y;
  public Figure(double x, double y) {
    this.x = x; this.y = y;
  }
  public boolean equals(Object o) {
    if (!(o instanceof Figure)) { return false; }
    Figure f = (Figure) o;
    return x == f.x && y == f.y;
  }
  public String toString() {
    return ``Position: (`` + x + ``, `` + y + ``)``;
  }
}
```

Herencia: Circle / Triangle

- Palabra reservada **extends**
- Palabra reservada **super** para referirse a elementos de la clase padre
- En BlueJ:



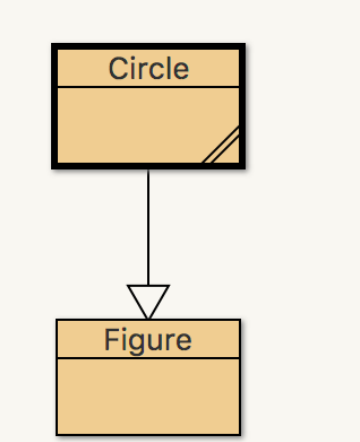

```
public class Circle
   extends Figure {
 private double radius;

 public Circle(double x,
  double y, double r) {
    super(x, y);
    radius = r;
 }

 public String toString() {
  return ‘‘Circle:\n\t’’ +
         super.toString() +
         ‘‘\n\tRadius: ’’ +
         radius;
 }
}
```

Herencia: Circle / Triangle

- Palabra reservada **extends**
- Palabra reservada **super** para referirse a elementos de la clase padre
- En BlueJ:



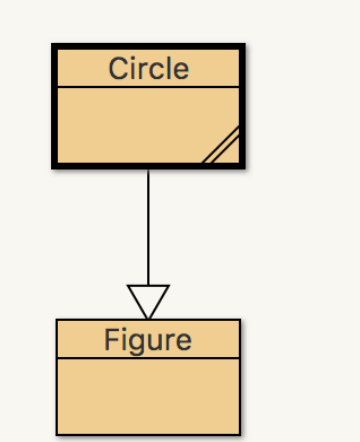

```
public class Circle
    extends Figure {
 private double radius;

 public Circle(double x,
  double y, double r) {
    super(x, y);
    radius = r;
 }

 public String toString() {
  return ‘‘Circle:\n\t’’ +
         super.toString() +
         ‘‘\n\tRadius: ’’ +
         radius;
 }
}
```

Herencia: Circle / Triangle

- Palabra reservada **extends**
- Palabra reservada **super** para referirse a elementos de la clase padre
- En BlueJ:



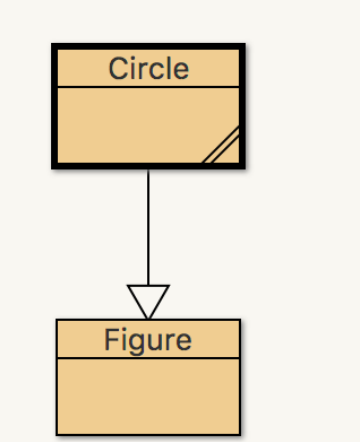

```
public class Circle
    extends Figure {
 private double radius;

 public Circle(double x,
  double y, double r) {
    super(x, y);
    radius = r;
 }

 public String toString() {
  return ‘‘Circle:\n\t’’ +
         super.toString() +
         ‘‘\n\tRadius: ’’ +
         radius;
 }
}
```

Herencia: Circle / Triangle

- Palabra reservada **extends**
- Palabra reservada **super** para referirse a elementos de la clase padre
- En BlueJ:




```
public class Circle
    extends Figure {
 private double radius;

 public Circle(double x,
  double y, double r) {
    super(x, y);
    radius = r;
 }

 public String toString() {
  return ‘‘Circle:\n\t’’ +
          super.toString() +
          ‘‘\n\tRadius: ’’ +
          radius;
 }
}
```

Sobreescribir un método

```
public boolean equals(Object o) {
   if (!(o instanceof Figure)) { return false; }
   Figure f = (Figure) o;
   return x == f.x && y == f.y;
}
```

```
public class Triangle extends Figure {
   private double base, height;
   public boolean equals(Object o){

      // Ejercicio 2

   }
```

Implementando el método equals

```
public boolean equals(Object o) {
  if (!(o instanceof Figure)) { return false; }
  Figure f = (Figure) o;
  return x == f.x && y == f.y;
}
```

Implementando el método equals

1) ¿**`Object o`** es del mismo tipo que yo?

```
public boolean equals(Object o) {
  if (!(o instanceof Figure)) { return false; } 1
  Figure f = (Figure) o;
  return x == f.x && y == f.y;
}
```

Implementando el método equals

1) ¿**`Object o`** es del mismo tipo que yo?
2) Lo convierto a mi tipo

```
public boolean equals(Object o) {
  if (!(o instanceof Figure)) { return false; } 1
2 Figure f = (Figure) o;
  return x == f.x && y == f.y;
}
```

Implementando el método equals

1) ¿**`Object o`** es del mismo tipo que yo?
2) Lo convierto a mi tipo
3) ¿Mis valores y los suyos son iguales?

```
public boolean equals(Object o) {
  if (!(o instanceof Figure)) { return false; } 1
2 Figure f = (Figure) o;
  return x == f.x && y == f.y; 3
}
```

Herencia: FiguresGroup & FiguresGroupUse

```
public class FiguresGroup {
 private static final int NUM_FIGURES = 10;  // constante
 private Figure[] figuresList = new Figure[NUM_FIGURES];
 private int numF = 0;
 public void add(Figure f) { figuresList[numF++] = f; }
 public String toString() {
  String s = ``'';
  for(int i = 0; i < numF; i++) s += ``\n'' + figuresList[i];
  return s;
 }
}
```

Herencia: FiguresGroup & FiguresGroupUse

```
public class FiguresGroup {
 private static final int NUM_FIGURES = 10;  // constante
 private Figure[] figuresList = new Figure[NUM_FIGURES];
 private int numF = 0;
 public void add(Figure f) { figuresList[numF++] = f; }
 public String toString() {
  String s = ``'';
  for(int i = 0; i < numF; i++) s += ``\n'' + figuresList[i];
  return s;
 }
}
```

```
public class FiguresGroupUse {
 public static void main(String[] args) {
  FiguresGroup g = new FiguresGroup();
  g.add(new Circle(10, 5, 3.5));
  g.add(new Triangle(10, 5, 6.5, 32));
  System.out.println(g);
 }
}
```

Ejercicios Herencia

- **Ejercicio 2**

  Clases implicadas: Figure, Circle, Triangle
- **Ejercicios 5 y 6**

  Clases implicadas: Figure, Rectangle, Square
- **Ejercicio 3**

  Clases implicadas: FiguresGroup

NOTA: Para probar todas las implementaciones hay que modificar la clase **FiguresGroupUse** creando objetos y mostrando los resultados

Ejercicios Herencia

EJERCICIO 3

- ¿Cuándo dos objetos **`FiguresGroup fg1`** y **`fg2`** son iguales?

```
public class FiguresGroup {
 ...
 private boolean found(Figure f) {
  for(int i = 0; i <numF; i++) {
     if (figuresList[i].equals(f)) return true;
  }
  return false;
 }
 private boolean included(FiguresGroup g) {
  for(int i = 0; i <g.numF; i++) {
     if (!found(g.figuresList[i])) return false;
  }
  return true;
 }
}
```

Ejercicios Herencia

EJERCICIO 3

- ¿Cuándo dos objetos **`FiguresGroup fg1`** y **`fg2`** son iguales?

- Cuando **TODOS** los elementos de **`fg1`** están en **`fg2`** y **viceversa**

```
public class FiguresGroup {
 ...
 private boolean found(Figure f) {
  for(int i = 0; i <numF; i++) {
     if (figuresList[i].equals(f)) return true;
  }
  return false;
 }
 private boolean included(FiguresGroup g) {
  for(int i = 0; i <g.numF; i++) {
     if (!found(g.figuresList[i])) return false;
  }
  return true;
 }
}
```

Clases Abstractas

```
public class FiguresGroupUse {
 public static void main(String[] args) {
  FiguresGroup g = new FiguresGroup();
  g.add(new Circle(10, 5, 3.5));
  g.add(new Triangle(10, 5, 6.5, 32));
  g.add(new Figure(10, 5));
  System.out.println(g);
 }
}
```

```
public abstract class Figure {
  private double x, y;
  public Figure(double x, double y) {
    this.x = x; this.y = y;
  }
  public boolean equals(Object o) {
    if (!(o instanceof Figure)) { return false; }
    Figure f = (Figure) o;
    return x == f.x && y == f.y;
  }
  public String toString() {
    return ``Position: (`` + x + ``, `` + y + ``)``;
  }
}
```

Clases Abstractas

```
public abstract class Figure {
  private double x, y;
  public Figure(double x, double y) {
    this.x = x; this.y = y;
  }
  public boolean equals(Object o) {
    if (!(o instanceof Figure)) { return false; }
    Figure f = (Figure) o;
    return x == f.x && y == f.y;
  }
  public String toString() {
    return ``Position: (`` + x + ``, `` + y + ``)``;
  }
  public abstract double nonImplementedFunction();
}
```

Ejercicios Clases Abstractas

- **Ejercicio 8**

  Clases implicadas: Figure

- **Ejercicio 9**

  Clases implicadas: Circle, Triangle, Rectangle

- **Ejercicio 10**

  Clases implicadas: FiguresGroup

- **Ejercicio 11**

  Clases implicadas: FiguresGroup

NOTA: Para probar todas las implementaciones hay que modificar la clase **FiguresGroupUse** creando objetos y mostrando los resultados