Object-Oriented Programming in Java

Tecnología de Videojuegos

Objectives

- Introduce specific Java POO mechanisms

Bibliography

1. The Java™ Tutorials. Oracle. (Link)

Table of Contents

Classes

Declaring classes (I)

The general form of a class definition is:

```
[public|private] class ExampleClass {
  // Fields declaration
  [public|protected|private] [type] field;

  // Constructors declaration
  [public|protected|private] ExampleClass(...);

  // Methods declaration
  [public|protected|private] [type] method(...);
}
```

Classes

Declaring classes (II)

Bicycle.java

```
public class Bicycle {
  public int gear;
  private int speed;

  public Bicycle(int gear, int speed) {
    this.gear = gear;
    this.speed = speed;
  }

  public void speedUp() { speed = speed + 5; }

  private void speedDown() { speed = speed - 5; }

  public static void main(String [] args) {
    Bycicle bike = new Bicycle(5, 10);
    bike.gear = 2;
    bike.speed = 5; // Error
    bike.speedUp();
    bike.speedDown(); // Error
  }
}
```

Classes

Declaring member variables (I)

Three types of variables

- Fields: Member variables in a class
- Local variables: Variables in a method or block of code
- Parameters: Variables in method declaration

Three elements in a field declaration

- Access modifiers
  - **Public**: Accesible from all classes (default)
  - **Private**: Accesible only within the own class
  - **Protected**: Accesible from own class and its subclasses
- Variable types (int, float, long, *other classes*, etc)
- Field name (Java naming convention)
  - Class names begin with capital
  - Methods with a lowercase verb, fields with lowecase noun

Example: `public int edad;`

Classes

Defining methods

Example of method definition

```
public double calculateAnswer(double wingSpan, int
    numberOfEngines, double length, double grossTons) {
      //do the calculation here
}
```

Elements of a method definition

- Modifiers (public, protected and private)
- Return type (including void)
- Method name
- Parameters (if any)

Remember naming rules!

- Examples: `run()`, `runFast()`, `getColor()`, `isEmpty()`

Classes

Defining methods: Overloading methods

- **Overloading**: Several methods with the same name
- Signature: Method name and parameters type
  - Method identification
  - No duplicated signatures

DataArtist.java

```
public class DataArtist {
  // Fields
  public void draw(String s) {
    // Body
  }
  public void draw(int i) {
  // Body
  }
  public void draw(double f) {
  // Body
  }
  public void draw(int i,double f) {
    // Body
  }
}
```

Classes

Constructors (I)

Constructor: Method that builds up an object

- First method invoked when an object is created
- Initializes variables and perform initial tasks
- *Same name* than the class *without* return type

A constructor is invoked with the `new` operator

- There is a default constructor for each class
- A constructor of the superclass will be also invoked (even explicitly with `super()`)

Constructor migh contain arguments

Classes

Constructors (II)

Bicycle.java

```
public class Bicycle {
  // Fields ...
  public Bicycle(int startCadence, int startSpeed,
      int startGear) {
    gear = startGear;
    cadence = startCadence;
    speed = startSpeed;
  }
  public Bicycle() {
    gear = 1;
    cadence = 10;
    speed = 0;
  }
   public static void main(String[] args) {
    Bicycle bike1 = new Bicycle();
    Bicycle bike2 = new Bicycle(0, 10, 10);
   }
}
```

Classes

Passing arguments to a method or a constructor

Parameters are local variables in a method

- Passed by value
- They can shadow any other variable with the same name

Another keyword: this

Circle.java

```
public class Point {
    private int x, y;

    public void setX(int x) { this.x = x; }
    public void setY(int y) { this.y = y; }
}
```

Objects

Creating objects (I)

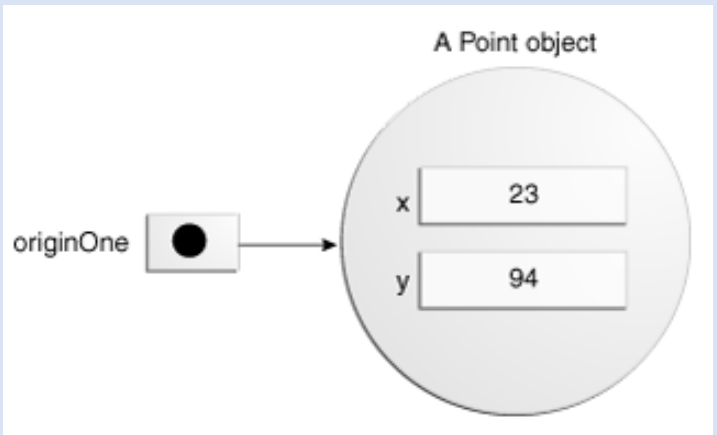
- An object is created in three parts
  1. Declaration
  2. Instantiation
  3. Initialization
- In Java, any object is referenced (≈ pointers)

Declaration

```
Point originOne;
```

Instantiation

```
Point originOne = new Point(23, 94);
```



Objects

Creating objects (II)

- The `new` operator creates a new object
  1. It requires a constructor
  2. It returns a reference
- Valid usages of new
  1. `Point originOne = new Point(23, 94);`
  2. `int height = new Rectangle().height;`

Objects

Using objects

- Fields:
  - Within the object: `width;`
  - Outside the object: `object.width;`
- Methods
  - Within the object: `getArea();`
  - Outside the object: `object.getArea();`
- Be careful with private scope!
- A good thing about Java: Garbage collector
  - When an object is no longer referenced, the garbage collector frees its memory automatically
  - It is invoked by JVM periodically
- Delete an object just assigning it `null`

More on classes

Returning a value from a method

- A method finishes when first:
  - Executes the last statement in the method
  - Executes a return statement
  - Throws an exception
- The return statement indicates the return value
  - ... unless the return type is void

Example

```
public void move(int x, int y) {
  origin.x = x;
  origin.y = y;
}
public int getArea() {
  return width * height;
}
```

More on classes

Using the this keyword

The keyword this represents the current object

Point.java

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

  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public Point(Point p) {
    this(p.x, p.y);
  }
}
```

More on classes

Access control

Modifier	Class	Package	Subclass	World
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
No modifier	Y	Y	N	N
Private	Y	N	N	N

Good practices

- Use the most restrictive access level (`private` by default)
- Avoid `public` fields except for constants

More on classes

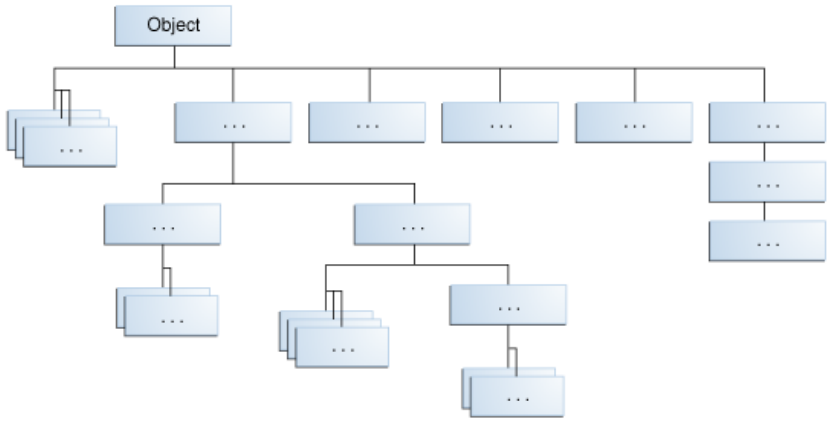
The static keyword

- The meaning of static depends
  - Fields: Field shared by all the objects of that class
    - Example: `public static double PI = 3.14;`
    - Example: `System.out`
  - Methods: They can be invoked without an object
    - Example: `System.out.println()`
- Both require the class name:
  `area = Math.pow(r, 2) * MyClass.PI`
- The keyword final defines a constant field
  - Example: `public final ruedas = 4;`
  - Example: `public static final double PI = 3.14;`

Inheritance

Definitions

- Objective: Derive your new class from an existing class (reusing its code)
  - Subclass, derived class, extended class or child class
  - Superclass, base class or parent class
- Any Java class has a superclass
  - The only exception is `Object`
  - Any class is derived from `Object` (`java.lang.Object`)



Inheritance

Inheritance example (I)

Bicycle.java

```
public class Bicycle {
  private int gear;
  private int speed;
  private int cadence;
  public Bicycle(int startCadence, int startSpeed,
      int startGear) {
    gear = startGear;
    cadence = startCadence;
    speed = startSpeed;
  }
  public void setCadence(int newValue) {
    cadence = newValue;
  }
  public void setGear(int newValue) {
    gear = newValue;
  }
  public void applyBrake(int decrement) {
    speed -= decrement;
  }
  public void speedUp(int increment) {
    speed += increment;
  }
}
```

Inheritance

Inheritance example (II)

MountainBike.java

```
public class MountainBike extends Bicycle {
  private int seatHeight;

  public MountainBike(int startHeight,int startCadence,
      int startSpeed, int startGear) {
    super(startCadence, startSpeed, startGear);
    seatHeight = startHeight;
  }

  public void setHeight(int newValue) {
    seatHeight = newValue;
  }
}
```

Inheritance

Overriding and hiding methods (I)

- Sometimes we need to adapt the behaviour of an inheritanced method: Overriding
  - You might want to use @override to avoid warnings

Animal.java

```
public class Animal {
  public static void testClassMethod() {
    System.out.println("The class"
      + " method in Animal.");
  }
  public void testInstanceMethod() {
    System.out.println("The instance "
      + " method in Animal.");
  }
}
```

Inheritance

Overriding and hiding methods (II)

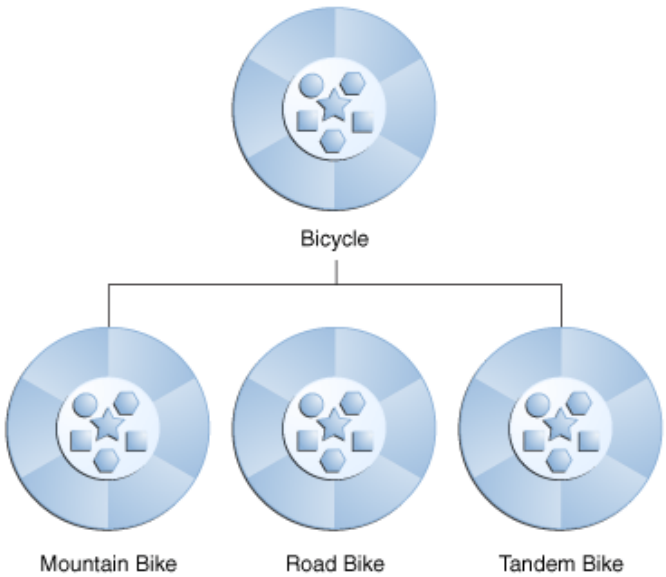
Cat.java

```
public class Cat extends Animal {
  public static void testClassMethod() {
    System.out.println("The class method"
      + " in Cat.");
  }
  public void testInstanceMethod() {
    System.out.println("The instance method"
      + " in Cat.");
  }
  public static void main(String[] args) {
    Cat myCat = new Cat();
    Animal myAnimal = myCat;
    Animal.testClassMethod();
    myAnimal.testInstanceMethod();
  }
}
```

Inheritance

Polymorphism (I)

- **Polymorphism**: Same method signature, diferent implementations
- Define a method in a base class
  - Redefine the same method in several subclasses
  - They can be invoked regardless of the class



Inheritance

Polymorphism (II)

- Example: A superclass (`Bicycle`) with three subclasses
  - `Bicycle` implements a `printDescription()` method
  - Subclasses override `printDescription()`

TestBikes.java

```
public class TestBikes {
  public static void main(String[] args){
    Bicycle bike01, bike02, bike03;
    bike01 = new Bicycle(20, 10, 1);
    bike02 = new MountainBike(20, 10, 5, "Dual");
    bike03 = new RoadBike(40, 20, 8, 23);
    bike01.printDescription();
    bike02.printDescription();
    bike03.printDescription();
  }
}
```

Inheritance

Object as superclass

- The class `Object` is the root of the Java hierarchy
- Any Java class inherits a set of methods from `Object`
  - `protected Object clone() throws CloneNotSupportedException`
  - `public boolean equals(Object obj)`
  - `protected void finalize() throws Throwable`
  - `public final Class getClass()`
  - `public int hashCode()`
  - `public String toString()`

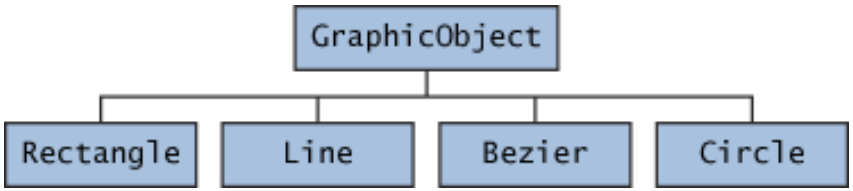
Inheritance

Abstract methods and classes (I)

- Java supports abstract methods and classes
- **Abstract class**: A class that cannot be instanciated
  - Provide a base to develop a class hierarchy
  - It contains common code
- **Abstract method**: A method without implementation
  - It must be overriden by subclasses
  - It defines a shared behaviour

Inheritance

Abstract methods and classes (II)

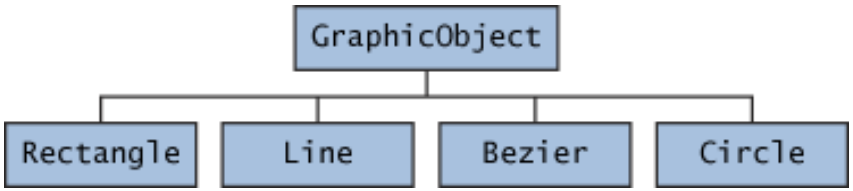


GraphicObject.java

```
abstract class GraphicObject {
  int x, y;
  ...
  void moveTo(int newX, int newY) {
    ...
  }
  abstract void draw();
  abstract void resize();
}
```

Inheritance

Abstract methods and classes (III)



Circle.java

```
class Circle extends GraphicObject {
  void draw() {
    ...
  }
  void resize() {
    ...
  }
}
```