Object-Oriented Programming concepts

Videogames Technology
Asignatura transversal

Departamento de Automática

Universidad de Alcalá
ISG

Objective

- Understand the meaning of paradigm
- Introduce the main OOP concepts
- Basic Java syntax for classes

Bibliography

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

Readings

1. Intro to Object-Oriented Programming for Game Development. (Link)
2. Game Architecture Day 2. (Link)

Table of Contents

1. Programming paradigms
   - Programming, programming language and paradigm
   - Programming paradigms types
   - Declarative programming
   - Imperative programming
   - Object-Oriented Programming
   - OOP objectives
2. Object-Oriented Programming
   - Basic concepts
   - Synthesizing OOP terminology
   - Inheritance
   - Polymorphism
   - Abstraction and encapsulation
3. Java OOP concepts
   - Classes in Java
   - Inheritance in Java
   - Interfaces
   - Packages
   - Javadoc
4. Exercises
   - Exercise 1: Asteroids
   - Exercise 2: Tetris
   - Exercise 3: Pac-Man

Programming paradigms

Programming, programming language and paradigm

Programming

Set of techniques that allow the development of programs using a programming language

Programming language

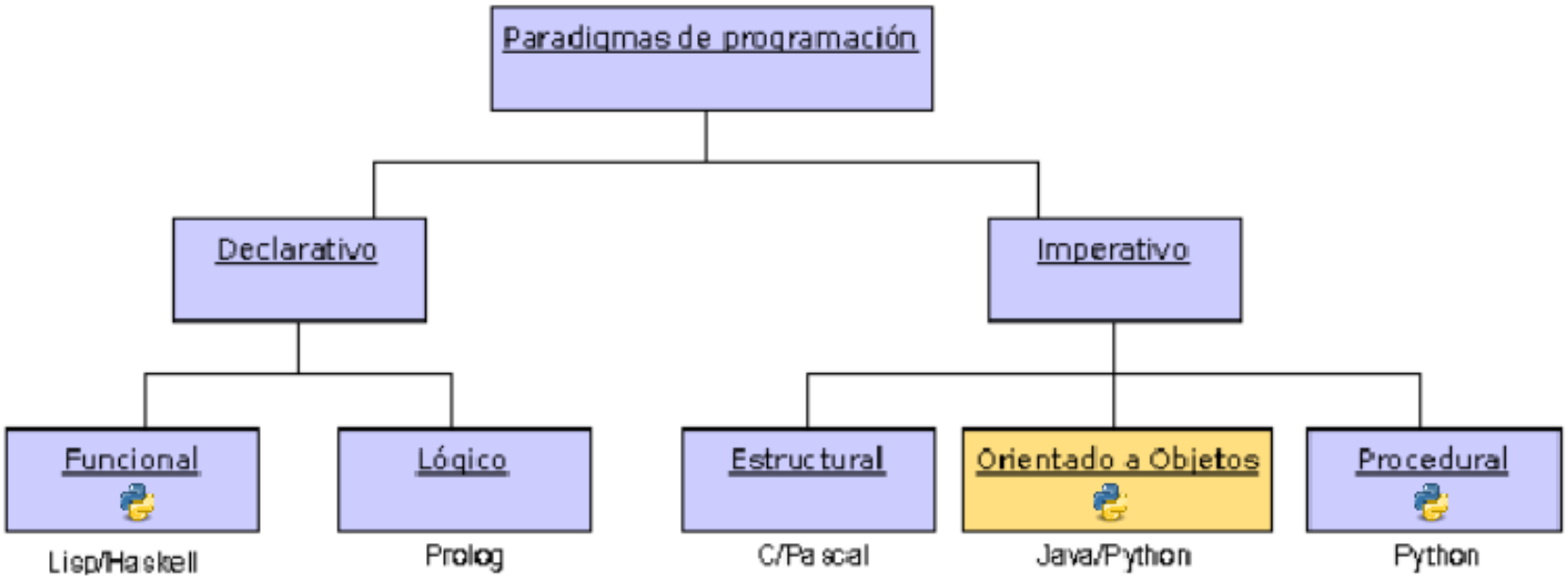
Set of rules and instructions based on a familiar syntax and later translated into machine language which allow the elaboration of a program to solve a problem

Paradigm

Set of rules, patterns and styles of programming that are used by programming languages

Programming paradigms

Programming paradigms types (I)



(Source)

Many other paradigms: Event-Driven programming, Concurrent, Reactive, Generic, etc

Programming paradigms

Declarative programming

Declarative programming

Describe what is used to calculate through conditions, propositions, statements, etc., but does not specify how

- **Logic**: First order logic in order to formalize facts of the real world (Prolog)
  - Example: Anne's father is Raul, Raul's mother is Agnes. Who is Ana's grandmother
- **Functional**: Based on the evaluation of functions (like Maths) recursively (Lisp, Haskell, R)
  - Example: The factorial from 0 and 1 is 1 and n is the factorial from n * factorial (n-1). What is the factorial from 3?

Programming paradigms

Imperative programming

Imperative programming

Describes, by a set of instructions, how the task should be implemented

- **Procedural**: Collections of subroutines related by means of invocations (C, Python)
  - Example: *The cooking process consists of 20 lines of code. When it is used, it only calls the function (1 line)*
- **Structural**: Nesting, loops, conditionals and subroutines. `GOTO` command is forbidden (C, Pascal)
  - Example: *Reviewing products of a shopping list and add the item X to the shopping if it is available*

Programming paradigms

Object-Oriented Programming

Object-Oriented Programming

Evolves from imperative programming. It is based on objects that allow express the attributes and behavior in a closer way to real life (*Java, Python, C++, C#*)

- **Main characteristics**: Abstraction, encapsulation, polymorphism, inheritance, modularity, etc
  - Example: *A car has a set of properties (color, fuel type, model) and a functionality (speed up, shift gears, braking)*

Programming paradigms

OOP objectives

OOP tries to provide

- **Reusability**: Ability of software elements to serve for many applications
- **Modularity**: Capacity to divide a program
- **Extensibility**: Ease of adapting software products to specification changes
- **Usability**: Ease of using the tool

OOP relays on a set of abstract concepts

Object-Oriented Programming

Basic concepts (I)

Class

Generic entity that groups the properties and functions of an entity

Object-Oriented Programming

Basic concepts (II)

Atribute

Individual characteristics that determine the qualities of an object

Object-Oriented Programming

Basic concepts (III)

Method

Function responsible for performing operations

Object-Oriented Programming

Basic concepts (IV)

Object or instance

Specific representation of a class, namely, a class member with their corresponding attributes

Object-Oriented Programming

Basic concepts (V)

Constructor

Method called when an object is created. It allows the initialization of attributes

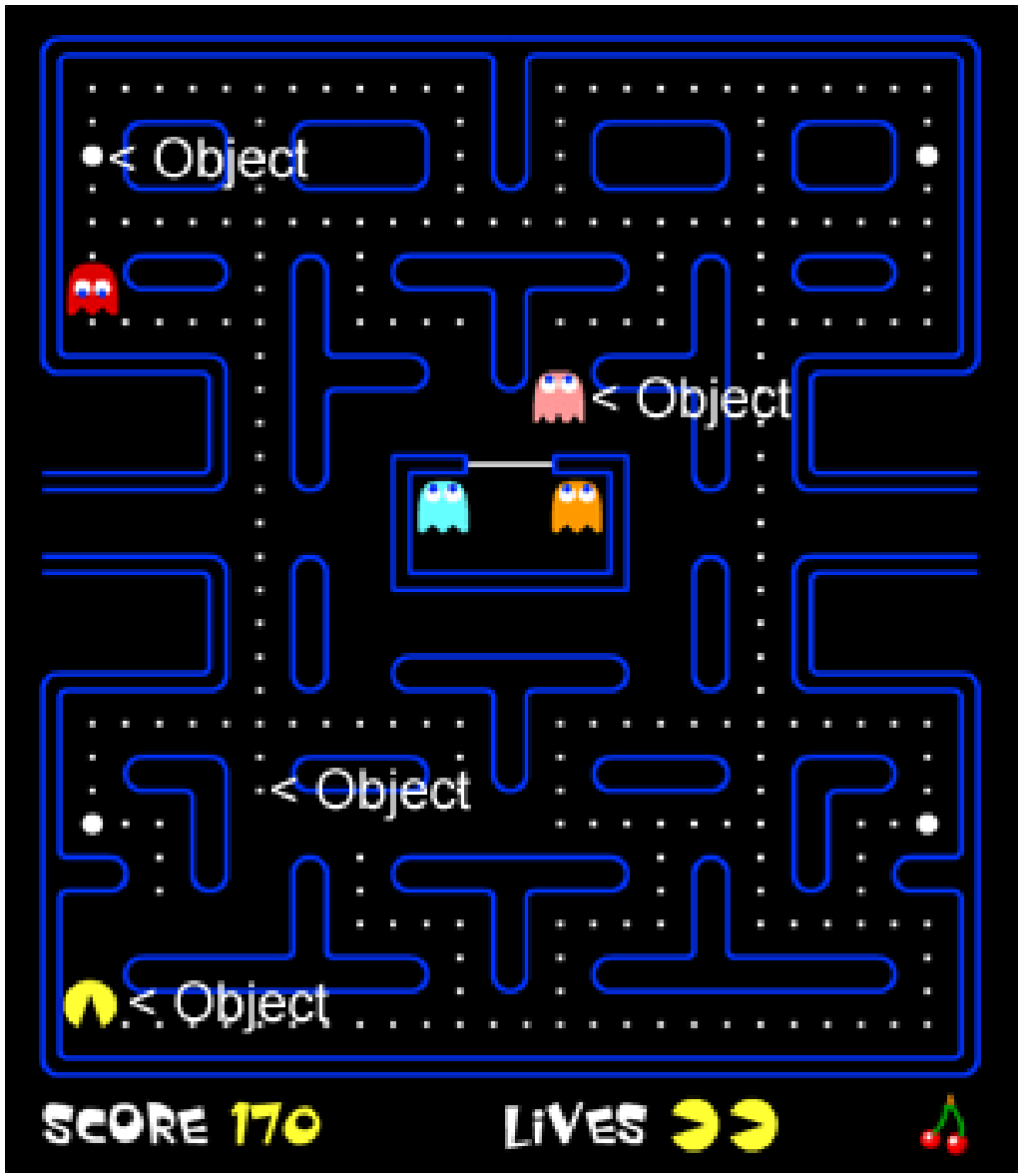
Object-Oriented Programming

Synthesizing OOP terminology (I)

- Software objects mimics physical objects
  - An object contains *attributes* and a *behaviour*
  - Example: A dog has a name (attribute) and may bit (behaviour)
- A class is a set of objects with common characteristics and behaviour
- An object is called an instance of a class
- Members of a class:
  - Properties: Data describing an object
  - Methods: What an object can do

Object-Oriented Programming

Synthesizing OOP terminology (II)



(Source)

Object-Oriented Programming

Synthesizing OOP terminology (III)

Object-Oriented Programming

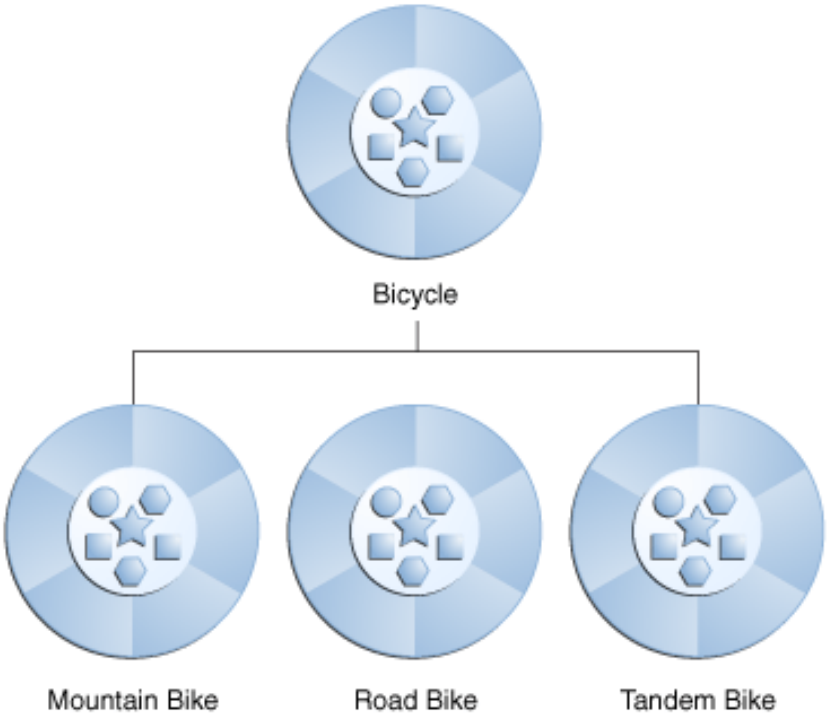
Inheritance (II)

Mechanism of reusing code in OOP. Consists of generating child classes from other existing (super-class) allowing the use and adaptation of the attributes and methods of the parent class to the child class

- *Superclass*: ``Father'' of a class
- *Subclass*: ``Child'' of a class
- A subclass inherits all the attributes and methods from its superclass
- *Class hierarchy*: A set of classes related by inheritance

Object-Oriented Programming

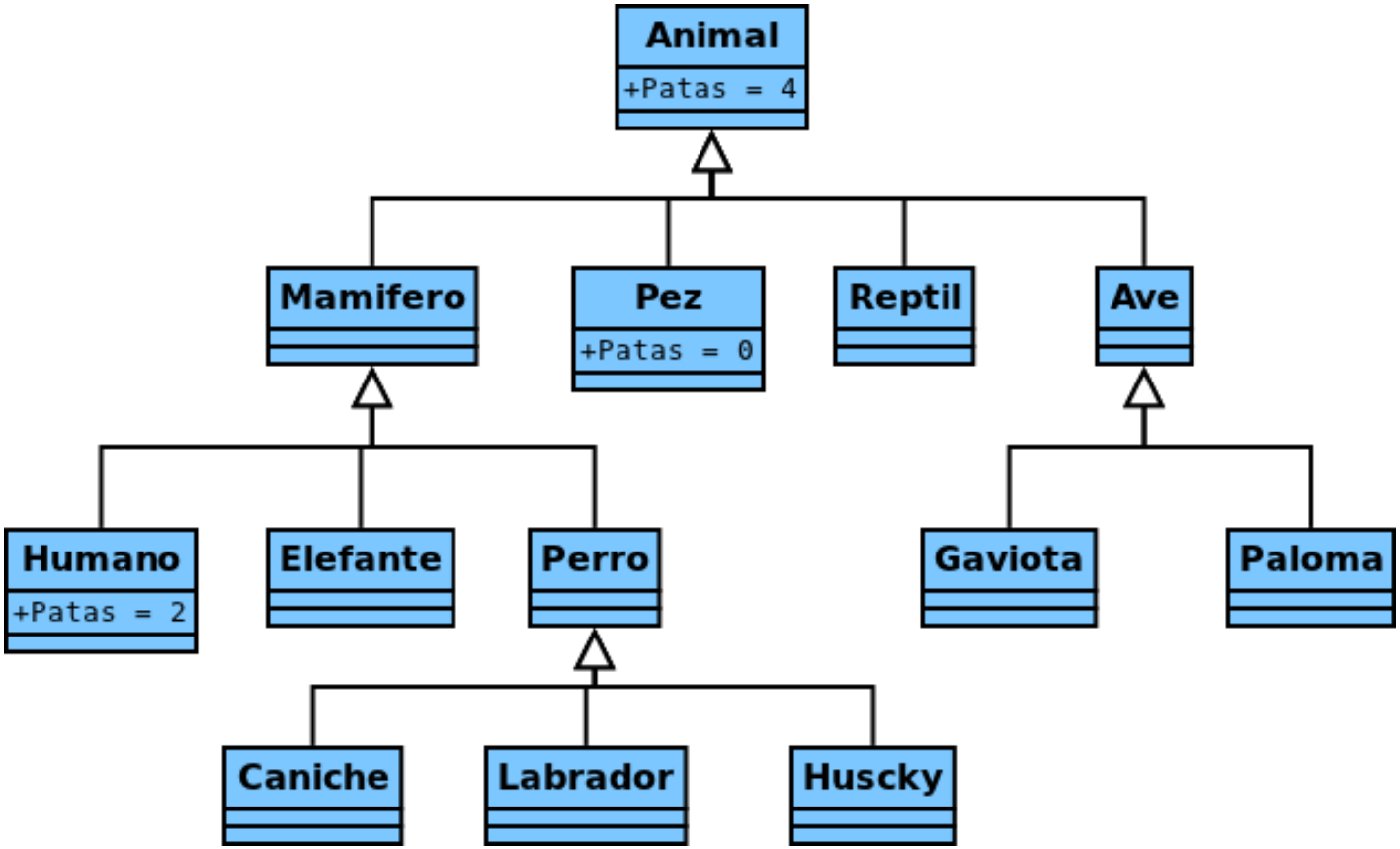
Inheritance (II)



(Source)

Object-Oriented Programming

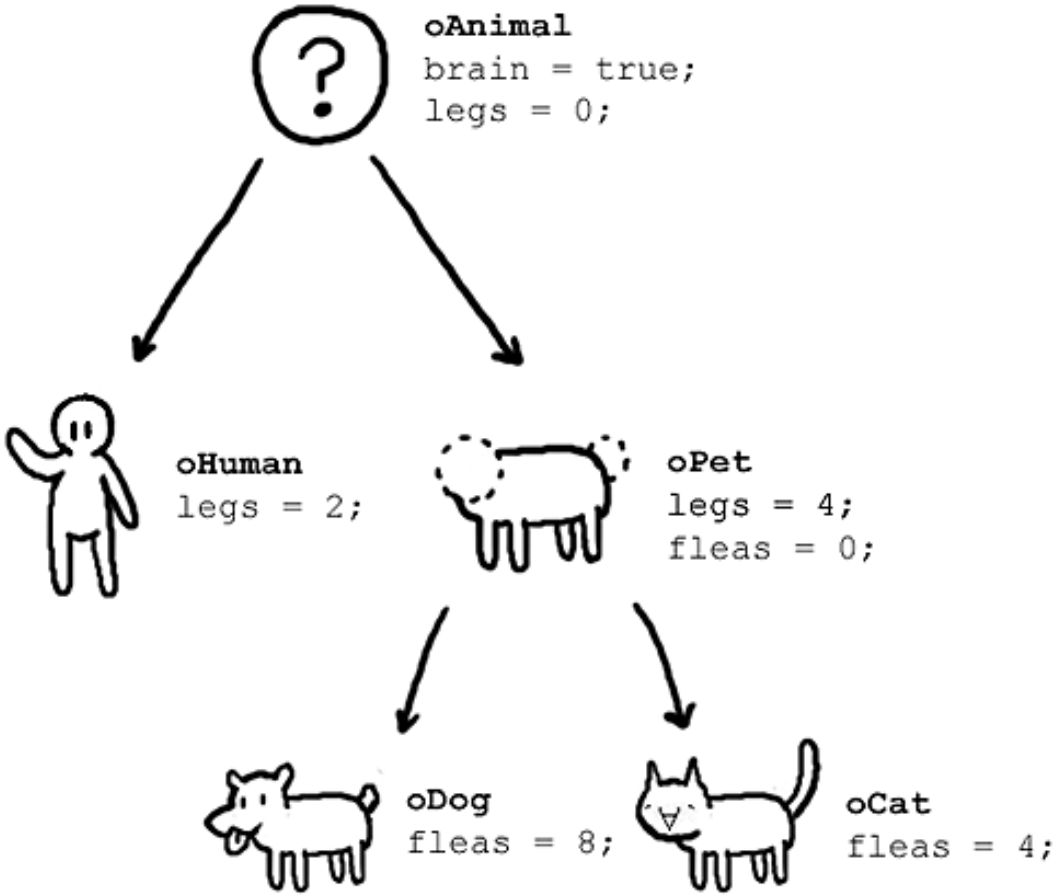
Inheritance (III)



(Source)

Object-Oriented Programming

Inheritance (IV)

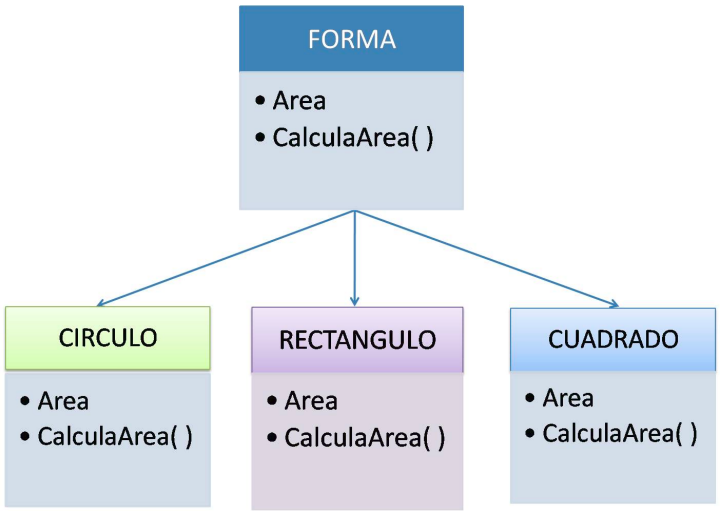


Object-Oriented Programming

Polymorphism (I)

Polymorphism

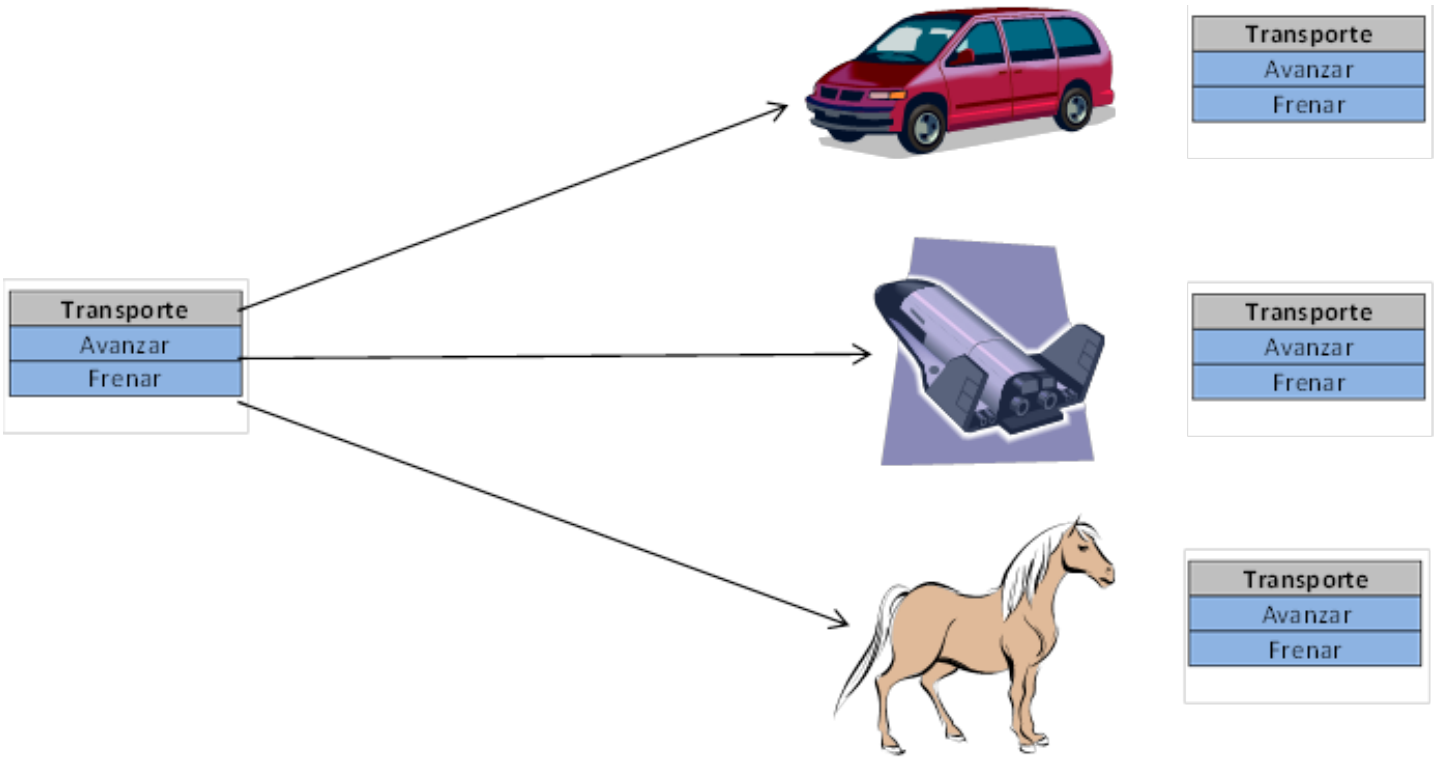
Invoke a method whose implementation will depend on the object that contains it



(Source)

Object-Oriented Programming

Polymorphism (II)



(Source)

Object-Oriented Programming

Abstraction

Abstraction

Mechanism that allows the isolation of the not relevant information to a level of knowledge

- A driver does not need to know how the carburator works
- To talk on the phone does not need to know how the voice is transferred
- To use a computer do not need to know the internal composition of their materials

Object-Oriented Programming

Encapsulation

Encapsulation

Provide an access level to methods and attributes for avoiding unexpected state changes. This mechanism is used to limit the visibility of the attributes and to create methods controlling them (`set()` y `get()`).

The most common access levels are:

- **public**: visible for everyone
- **private**: visible for the creator class
- **protected**: visible for the creator class and its descendents

Java OOP concepts

Classes in Java (I)

Bicycle.java

```
public class Bicycle {
  private int cadence = 0;
  private int speed = 0;
  private int gear = 1;

  public void changeCadence(int newValue) {
    cadence = newValue;
  }
  public void changeGear(int newValue) {
    gear = newValue;
  }
  public void speedUp(int increment) {
    speed = speed + increment;
  }
  public void applyBrakes(int decrement) {
    speed = speed - decrement;
  }
}
```

Java OOP concepts

Classes in Java (II)

BicycleDemo.java

```
class BicycleDemo {
    public static void main(String[] args) {
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);

        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeGear(3);
    }
}
```

OOP concepts

Inheritance in Java (I)

BicycleDemoInheritance.java

```
class MountainBike extends Bicycle {
    // new fields and methods defining
        // a mountain bike would go here
}
```

Java OOP concepts

Interfaces (I)

Interface: Set of methods without implementation

- Methods related to each other
- Expose a behaviour
- All methods in a interface must be implemented
- *Imitates multiple inheritance*
- Interfaces can be instanciated

Java OOP concepts

Interfaces (II)

Bicycle.java

```
interface Bicycle {
  //  wheel revolutions per minute
  void changeCadence(int newValue);
  void changeGear(int newValue);
  void speedUp(int increment);
  void applyBrakes(int decrement);
}
```

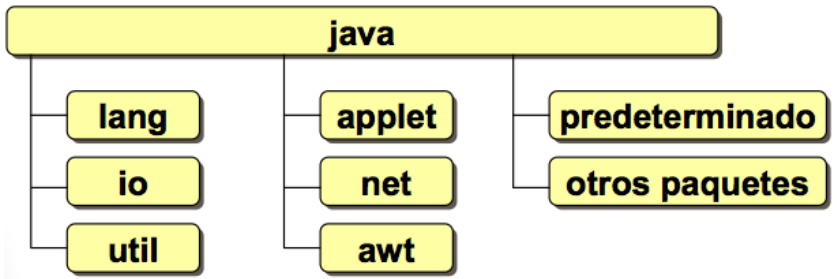
ACMEBicycle.java

```
class ACMEBicycle implements Bicycle {
  // remainder of this class
  // implemented as before
}
```

Java OOP concepts

Packages (I)

- A typical Java project might content thousands of classes
  - Some kind of classes organization is needed
- **Package**: A collection of related classes
  - Packages are ``folders'' of classes
  - ... there is a direct mapping package-folder, indeed
  - Technically speaking, packages separate namespaces
  - Packages are defined using package in the begining of a class
- Packages must be declared to be used within a class
  - Reserved word import



Java OOP concepts

Packages (II)

bicycles/ACMEBicycle.java

```
package bicycles;

class ACMEBicycle implements Bicycle {
        // Some other code goes here
}
```

demo/BicycleDemo.java

```
import bicycles.*;

class BicycleDemo {
  public static void main(String[] args) {
    Bicycle bike = new Bicycle();
    bike1.changeCadence(50);
  }
}
```

Java OOP concepts

Packages (III)

Compile with

- `javac bicycles/Bicycle`
- `javac demo/BicycleDemo`

Execute with

- `java demo.BicycleDemo`

Packages imported by default

- Package `java.lang`
- Same package

Watch out the CLASSPATH!

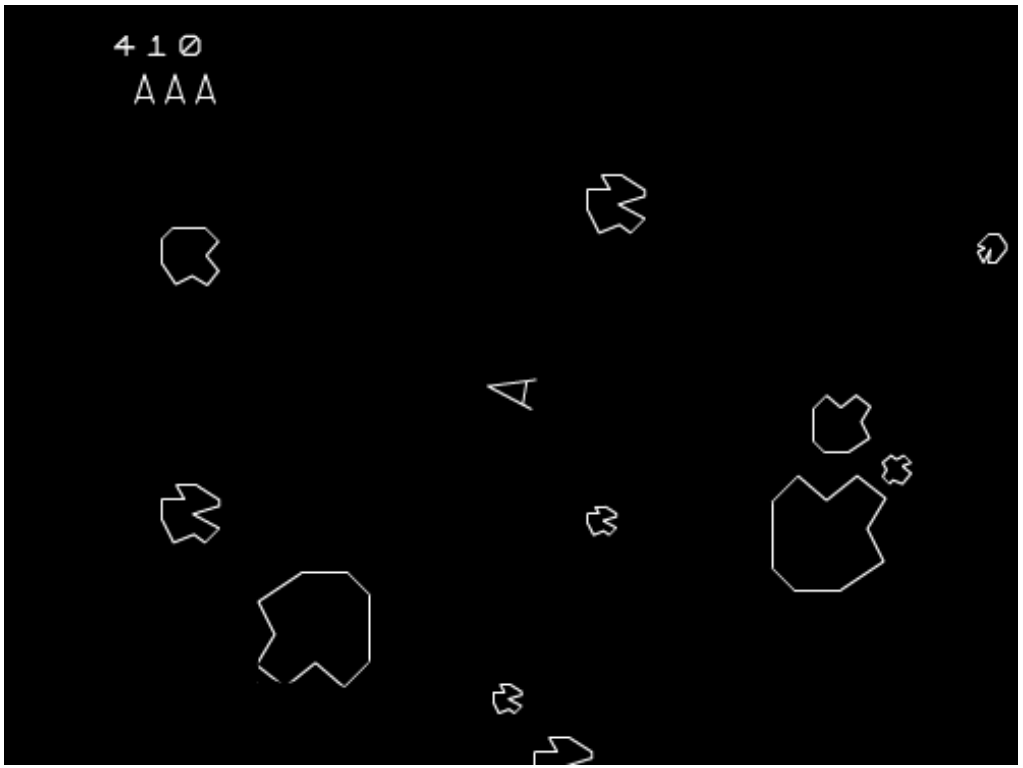
- CLASSPATH: Environment variable, Java version of PATH
- It stores the packages (and classes) locations

Java OOP concepts

Javadoc

- Javadoc generates documentation from the source code
- Comments begining with /** are Javadoc comments
  - They are included in the Javadoc documentation
- Getting used to handle Javadoc is critical!
  - Reference API documentation in any Java library is in Javadoc format
  - Try to use it in your code

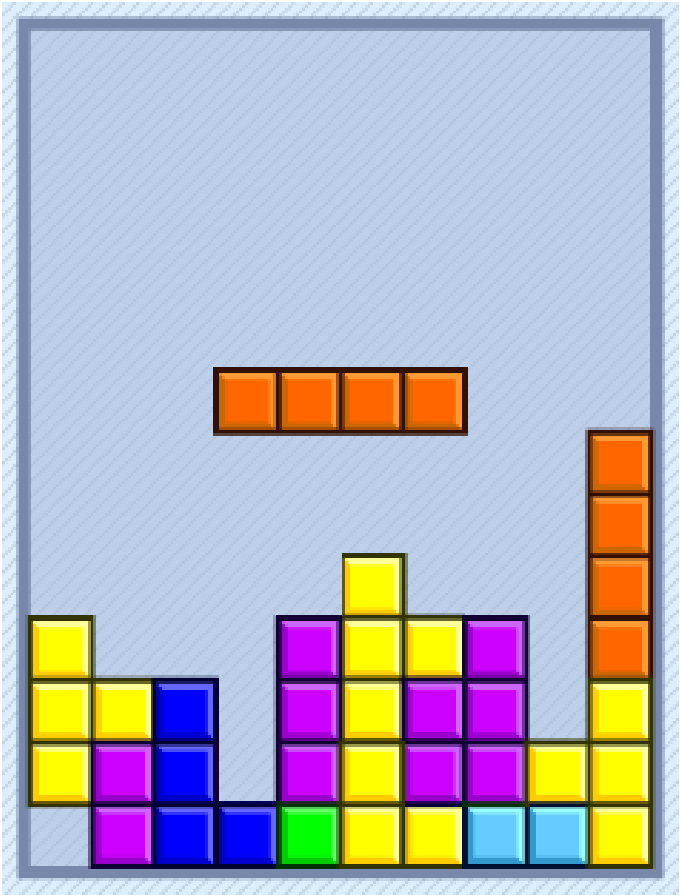
Exercise 1: Asteroids



(Source)

1. Identify the classes in the Asteroids videogame
2. Identify attributes contained in the previous classes
3. Identify methods contained in the previous classes

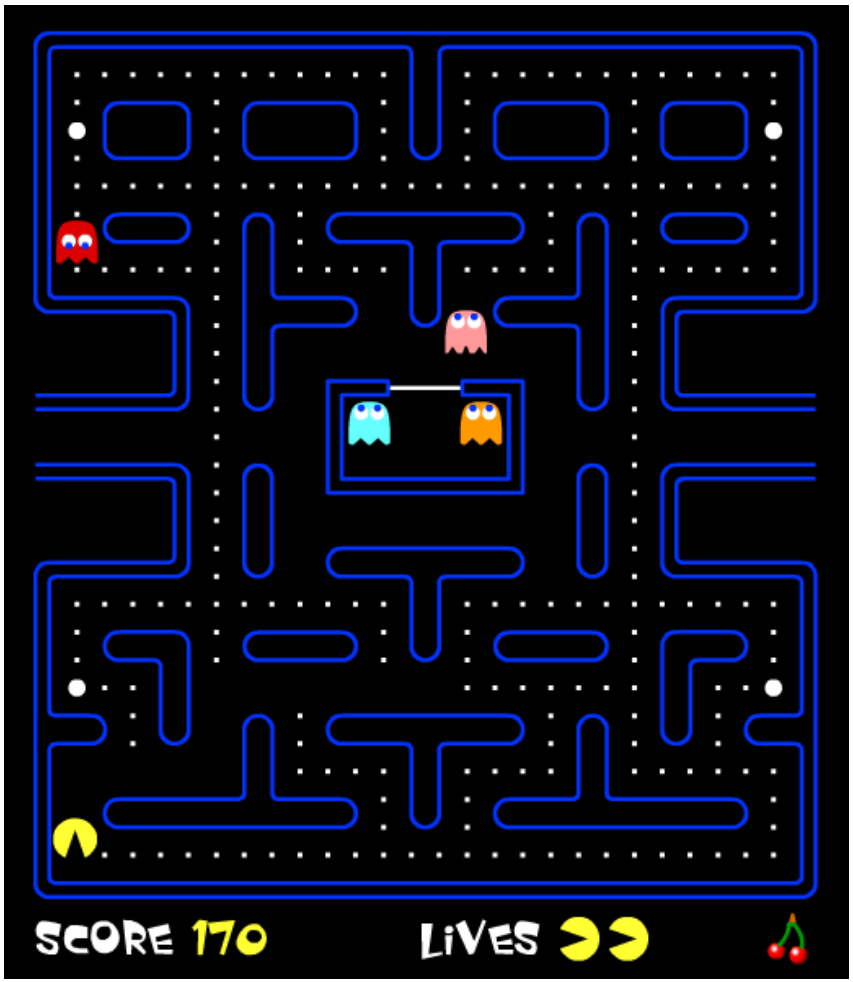
Exercise 2: Tetris



(Source)

1. Identify the classes in the Tetris videogame
2. Identify attributes contained in the previous classes
3. Identify methods contained in the previous classes

Exercise 3: Pac-Man



(Source)

1. Identify the classes in the Pac-Man videogame
2. Identify attributes contained in the previous classes
3. Identify methods contained in the previous classes