Tutorials — Object Oriented - Part 1

version **#1.0.0**

Yet Another Kind of Assistants 2026
<assistants@tickets.assistants.epita.fr>

Copyright

This document is for internal use at EPITA (website) only.

Contents

*https://intra.forge.epita.fr

1 Introduction

Object-oriented programming (OOP) is a programming paradigm based on encapsulation, the process of grouping data into a single entity. The paradigm uses *classes* to define models in order to create *objects*.

Java is intrinsically an object-oriented language, except for primitive types, everything is an object.

2 Classes

A class is a model that defines attributes and methods to build an object. It can be seen as a mold or a set of specifications. Each object created is called an instance of a class.

First things to do when creating a class are to give a name to the class, and then add specifications.

For example, think about a book. The `Book` class could be implemented like this:

```
class Book {

   // Defining attributes
   int numberOfPages;
   String author;
   String title;

   // Defining methods
   void open() {
      System.out.println("Opening " + title + "...");
   }

   void close() {
      System.out.println("Closing " + title + "...");
   }

   void read(int pageNumber) {
      if (pageNumber < 1 || pageNumber > numberOfPages) {
         System.out.println("Given page number is out of range!");
      }
      else {
         System.out.println("Start reading " + title + " on page " + pageNumber + "!");
      }
   }
}
```

Be careful!

A comparison between attributes vs variables, and methods vs

functions, can be easily made but you should pay attention to the words you use while describing your problem during a debug if you want to be clearly understood by assistants.

3 Objects

An object is an instance of a class. An object can be described in the same way as we describe an object in daily life. Think about how you would describe an object around you. An object in OOP is quite similar. It represents an entity with attributes that define its characteristics, and methods, which are actions or behaviors that can be performed with or on the object.

Tips

In the syntax node, methods are compared to functions. That is true, but now you should understand that a method depends on an object.

Consider our previous example about a book. These attributes could be its number of `pages`, its `author`, and its `title`, whereas its methods could be `open`, `close` or `read`. This is exactly the same principle when describing an object in programming.

Tips

Note that two or more instances of the same class (two different objects) are independent. If you have a red book and a blue book, opening the red book should not cause the blue one to open.

Now that you understand what an object is, let us see how you can define how it should be built.

3.1 Constructor

A constructor is a special method used to initialize objects. When you create a new instance of a class, the constructor sets up the object with initial values. Constructors have the same name as the class they belong to, and they do not have a return type.

There are various types of constructors in *Java*. A no-arg constructor is a constructor that takes no parameters. This is the default constructor: if no constructor is explicitly defined, it will always be available. It initializes an object and sets all attributes to their default value. If no explicit initialization is provided, primitives are initialized with *0* and classes with *null*.

A parameterized constructor is a constructor that takes parameters. The all-args constructor initializes all attributes with values passed as arguments. You can also define a constructor that takes any number of parameters according to your needs. It allows you to pass specific values when creating an object, for example it can be used to initialize the values of your attributes in your class. You can create multiple constructors.

If you do not define any constructor in a class, *Java* automatically provides one with no argument.

```java
// filename: Book.java

class Book {

   // Defining attributes
   int numberOfPages;
   String author = "Montesquieu";
   String title;

   // No-arg constructor
   Book() {
   }

   // All args constructor
   Book(int numberOfPages, String author, String title) {
      this.numberOfPages = numberOfPages;
      this.author = author;
      this.title = title;
   }

   // Constructor with only 2 args
   Book(int numberOfPages, String title) {
      this.numberOfPages = numberOfPages;
      this.title = title;
   }
}
```

```java
// filename: Main.java

public class Main {
   public static void main(String[] args) {
        Book book1 = new Book(); // numberOfPages = 0; author = "Montesquieu"; title = null;
        Book book2 = new Book(162, "William Shakespeare", "Hamlet"); // numberOfPages = 162;␣
↪author = "William Shakespeare"; title = "Hamlet";
        Book book3 = new Book(448, "Lettres persanes"); // numberOfPages = 448; author =
↪"Montesquieu"; title = "Lettres persanes";

   }
}
```

You might have noticed the use of the keywords `this` and `new` in the example. The `this` keyword is a reference variable that points to the current object. It can be used within a method or a constructor to access the attributes or methods of the current

object. The `new` keyword is used to create an instance of a class.

Be careful!

You can write classes in any file that you want as long as you follow the naming convention. Thus a `.java` filename must match the declared class name, otherwise the compiler will generate an error (e.g., `LinkedList.java` must implement a class named `LinkedList`).

4 Encapsulation

With encapsulation comes visibility of attributes and methods defined in classes. In fact, you can choose whereas your data should be exposed or not.

In *Java*, different access permissions can be set for class members:

- `public`: accessible by any other object or class
- `protected`: accessible by the class, subclasses and classes in the same package
- *default*: used if you do not specify any visibility; the member is accessible by any class in the same package
- `private`: accessible only by instances of the class

Summary:

Modifier	Class	Package	Subclass	World
`public`	Yes	Yes	Yes	Yes
`protected`	Yes	Yes	Yes	No
default	Yes	Yes	No	No
`private`	Yes	No	No	No

Tips

To indicate a **default** visibility, there is no need for a keyword.

Be careful!

You should check this complementary video resource out to enhance your understanding and practice of the subject: video #01. It features a live-coding that will help you write your first class.

Being a hero means fighting back even when it seems impossible.