Contents

[Java Object (POJO) 1](#_Toc190867455)

[Fields 1](#_Toc190867456)

[Constructor 1](#_Toc190867457)

[Getters and setters 2](#_Toc190867458)

[To String 2](#_Toc190867459)

[Setting Breakpoint 3](#_Toc190867460)

[Starting the Debugger 3](#_Toc190867461)

[Debugging tools in Eclipse 4](#_Toc190867462)

[**Summary Table** 5](#_Toc190867463)

[Debugging Constructor 6](#_Toc190867464)

# Java Object (POJO)

This code base here is an example of a plain old Java object (POJO). POJOs are straightforward Java objects with fields, constructors, getters, setters, and possibly some methods.

# Fields

In the context of a POJO (Plain Old Java Object), fields are the variables that store the data attributes of the object. These fields represent the state of the object and are typically declared as private to maintain encapsulation.

**Fields in a POJO**

* **Private Access Modifier:** Fields in a POJO are usually marked as private to encapsulate the object's state. This means that these fields cannot be directly accessed from outside the class. Instead, access is controlled through getter and setter methods.
* **Data Types:** The fields can be of any data type, such as primitive types (e.g., int, double, boolean) or reference types (e.g., String, other objects).

**private** **int** id;

**private** String name;

**private** String description;

**private** String price;

# Constructor

In Java, a constructor is a special method used to initialize objects. When you create an instance of a class, the constructor sets the initial state of the object. Let's look at how constructors work in the context of a POJO (Plain Old Java Object).

**Characteristics of a Constructor**

1. **Name:** The name of the constructor is the same as the name of the class.
2. **No Return Type:** Constructors do not have a return type, not even void.
3. **Initialization:** They are used to set initial values for the object's fields.
4. **Overloading:** You can have multiple constructors with different parameters (constructor overloading)

**public** Product() {

**super**();

}

**public** Product(**int** id, String name, String description, String price) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.description = description;

**this**.price = price;

}

# Getters and setters

Getters and setters are methods used in POJOs (Plain Old Java Objects) to access and modify the private fields of the object. They encapsulate the fields and provide controlled access to them. Let's dive into how they work, using our Product example:

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** i) {

**this**.id = i;

}

# To String

In Java, the toString method is a special method defined in the Object class. It is used to provide a string representation of an object.

@Override

**public** String toString() {

**return** "Product [id=" + id + ",

name=" + name + ",

description=" + description + ",

price=" + price + "]";

}

# Setting Breakpoint

We'll start by placing a breakpoint here. *Right-click in Gutter.*

A computer screen shot of a computer screen

AI-generated content may be incorrect.

## Starting the Debugger

Let's run this with the debugger. *Right-Click on file.* Go to "Debug As" -> Java Application."

A screenshot of a computer

AI-generated content may be incorrect.

Let's step into this code base.

# Debugging tools in Eclipse

These functions are essential tools in Eclipse for navigating and understanding your code during debugging sessions. Happy debugging! 😊

**Step into**

A screenshot of a computer

AI-generated content may be incorrect.

* **Purpose:** To dive deeper into the code execution within a method.
* **Usage:** When you reach a method call and want to see its internal workings, use "Step into" (or press F5). Eclipse will take you inside the method, allowing you to execute it line by line.
* **Scenario:** Useful for debugging complex logic and understanding how data is being *manipulated within the method*.

**Step Over**

A screenshot of a computer

AI-generated content may be incorrect.

* **Purpose:** To execute the current line of code and move to the next one, without stepping into any called methods.
* **Usage:** When you're on a method call but don't need to see its internal details, use "Step Over" (or press F6). Eclipse will execute the method and move to the next line.
* **Scenario:** Handy when you want to *continue execution without entering every method call*, saving time.

**Step Return**

A screenshot of a computer

AI-generated content may be incorrect.

* **Purpose:** To complete the current method and return to the caller.
* **Usage:** When you're inside a method and want to finish it quickly to go back to the caller, use "Step Return" (or press F7). Eclipse will execute the rest of the method and take you back to where it was called.
* **Scenario:** Ideal when you're done inspecting a method's execution and *want to return to the higher-level context*.

**Summary Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Debug Action | Purpose | Shortcut | Scenario |
| Step into | Enter the method and execute line by line | F5 | Debugging method internals |
| Step Over | Execute the current line, skip method details | F6 | Continuing without entering methods |
| Step Return | Finish the current method and return to caller | F7 | Returning to the higher-level context |

# Debugging Constructor

You can see what's going on with the constructor and the values with the *Step into* Debugging Tool. You can see the main attributes: name, description, and price.

We are running this line of code. Step over. Step over. You'll see the values are now assigned. This is the step to return, which brings us back to the code. Let's go to the next constructor, "Product 2." Check the values of "Product 1."

Step into the constructor. The green highlight indicates the line of code being executed. Step into it. Watch the values get assigned. Initially, the values are empty. Step over, and you'll see the values update.

Name and description are now assigned, but price is not. Step over to execute that line of code. Now, the price is assigned.

Return to the main function. We're running the next line of code. Step over. This getter method runs the function "get" and returns the name value.

Step return to go back to the main function. You can see in the console that the output is already there. Step over this code to print the output to the console.

We see the name is "smartphone." Print this to the console. Change "Product 1" name to "desktop." Step over. The value changes to yellow, indicating the update. Step over to print the updated value.

Here's the "toString" method. It concatenates the values into a string. Run the "toString" method. Step over, and this string is printed to the console.

This is "Product 1" printed. This method includes the constructor, getters, setters, "toString," and properties.