Java Part 4: Methods and Reusable Code

A method is a named block of code that we write once and use many times. Without methods, the same calculation is copied into several places, and a later change has to be repeated everywhere. Four steps make methods work: define, pass in, return and keep small. In this part, Rania, an administrator in a hospital, uses them to calculate overtime pay in one place, and each step answers one practical question.

1. Define

A method definition has a return type, a name, a list of parameters in parentheses and a body in braces. The return type says what comes back to the caller, and the word void means that nothing comes back. The name should start with a verb, written in camel case, such as calculateOvertimePay, so that the call reads like an action. We use a method by calling it by name, and Java then runs its body and returns to the line after the call. Methods live inside a class, and the main method we met in the first part is also a method. The question for this step is: what is the single job of this method?

Example: A shop owner finds the same three lines for printing a receipt header on the till screen, in the emailed receipt and in the end-of-day report. She moves them into one method called printReceiptHeader. Each place now calls it by name, and the header looks exactly as before. When the shop changes its address, she edits the method once, and every receipt shows the new address.

At work: Rania finds the overtime calculation copied into three reports: nursing, radiology and the laboratory. She writes one method, `static BigDecimal overtimePay(BigDecimal hourlyRate, BigDecimal hours)`, and moves the calculation into its body. The three reports now call the method by name instead of carrying their own copy. She had written the job of the method in one sentence first: calculate overtime pay from a rate and hours.

2. Pass In

A parameter is a variable in the method definition, and an argument is the actual value we give when we call it. The arguments must match the parameters in number, order and type. Java passes arguments by value, which means the method works with a copy of each value. For an object, the copy is a copy of the reference, so the method can change the content of the object, but it cannot make the caller's variable point to a different object. Overloading lets several methods share one name when their parameter lists differ, for example one version for hours and one for hours with a bonus rate. A different return type alone is not enough to tell overloaded methods apart. The question for this step is: what does the method need to know, and nothing more?

Example: A payroll clerk calls a method named formatName with a first name and a last name, and it returns the full name, such as Sara Nasser. A second method with the same name accepts a title as well, so a call with three values returns the name with the title. The compiler picks the right version from the number and type of the arguments in the call. This is overloading, and the clerk only has to remember one name.

At work: The overtime method needs two inputs: the hourly rate and the number of overtime hours. Rania first passes the department too, then removes it, because the calculation never uses it. She also adds a second version with the same name that takes a bonus rate for night shifts. A call with two arguments uses the first version, and a call with three arguments uses the second.

3. Return

The return statement ends the method and sends a value back to the caller. The value must match the declared return type, and a method whose type is not void must return a value on every possible path.

This document is a property of Asaseeyat Academy - © Copy Right Protected.

The caller can store the result in a variable, use it in a calculation or pass it straight into another method. A void method is called for what it does, such as printing or saving, rather than for a value. One method returns only one value, so when several values belong together we return an object, which we build in a later part. The question for this step is: what does the caller get back?

Example: A shipping method receives the weight of a parcel and returns a fee. The caller stores the fee in a variable and adds it to the invoice. The method returns a small fee for a light parcel and a higher fee for a heavy one, and each branch ends with its own return. If the method forgot to return a value on the heavy branch, the compiler would stop with a missing return message.

At work: The overtime method returns the pay as a BigDecimal, so each report can store it or add it up. Rania adds a check at the top: when the hours are zero or less, the method returns zero at once. On her first try the compiler refuses the code with a missing return message, because one branch has no return. She adds the final return, and the message disappears.

4. Keep Small

Variables declared inside a method exist only while that method runs, and other methods cannot see them. This is called scope, and it keeps methods independent. A good method does one job, fits on one screen and has a name that says what it does. Static methods belong to the class itself and can be called without creating an object, which is why main is static and why small helper methods in the same class are often static too. A short comment above a method, in the Javadoc style, describes what it does, what each parameter means and what it returns. Small methods can be tested one by one, which is the topic of the last part. The question for this step is: could a colleague understand this method from its name and its comment alone?

Example: A developer finds one method that reads a file, calculates the tax and prints a report. Any change to the tax rule means reading through all three jobs. She splits it into three methods called readFile, calculateTax and printReport. Each one can now be changed and checked on its own, so a change to the tax touches only calculateTax.

At work: Rania's report code has grown into one long main method. She extracts the overtime pay, the department total and the printing into three separate methods. She adds a short Javadoc comment to the overtime method that describes the rate, the hours and the returned pay. When the hospital changes the overtime rule, she edits one method and all three reports are correct.

Conclusion

To recap, a method is a named block of code that we write once and call many times. We define it with one clear job, pass in only what it needs, return a result of the declared type and keep it small. Arguments are passed by value, overloading lets one name serve several parameter lists, and variables stay inside their method. Rania's overtime rule now lives in one place, and all three reports follow it. Next time, we will store many values together in arrays, lists and maps.

Sources

- Oracle: The Java Tutorials, Defining Methods – https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html
- Oracle: The Java Tutorials, Passing Information to a Method or a Constructor – https://docs.oracle.com/javase/tutorial/java/javaOO/arguments.html
- Oracle: The Java Tutorials, Understanding Class Members (static) – https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html
- Oracle: The Java Tutorials, Returning a Value from a Method – https://docs.oracle.com/javase/tutorial/java/javaOO/returnvalue.html
- Oracle: The Java Tutorials, A Closer Look at the Hello World Application (main) – https://docs.oracle.com/javase/tutorial/getStarted/application/index.html
- dev.java: Learn Java – https://dev.java/learn/
- Java is a trademark of Oracle and/or its affiliates. This video is independent training material and is not affiliated with Oracle.

This document is a property of Asaseeyat Academy - © Copy Right Protected.