# Java Part 3: Conditions and Loops

Programs become useful when they decide and repeat. A decision runs some code only when a condition is true, and a loop runs code again and again until a condition changes. Four steps cover this: decide, combine, choose and repeat. In this part, Tariq, operations manager of an online shop, uses them to give loyalty discounts to the right orders, and each step answers one practical question.

## 1. Decide

A decision uses the word if, a condition in parentheses and a block of code in braces. The condition must be a boolean value, so Java never accepts a number there as a replacement for true or false. We build conditions with comparison operators: equal to, not equal to, greater than, less than, and the versions of the last two that include equality. Adding else gives the code a second path, and else if lets us test several conditions in order, stopping at the first one that is true. For text we compare the content with equals, never with the double equals sign. The question for this step is: what exactly must be true before this code runs?

**Example:** A hotel booking system checks one yes or no question for every request: is the room the guest asked for free on those dates? If the answer is yes, the system confirms the stay, and otherwise it offers the next available date, just like the two paths of an if with an else. Only one of the two paths runs for each request, so no guest is ever confirmed and refused at the same time.

**At work:** Tariq writes his first rule: if the order total is above the discount threshold, the order gets ten percent off. A first test shows that an order exactly at the threshold gets no discount, because the rule says above, not at or above. He changes the comparison to greater than or equal to, and the test passes. He had answered the step question in one sentence before he wrote any code.

## 2. Combine

Real rules rarely have one condition, so we combine them. We use the logical operators, which we read as and, or and not: and needs both conditions to be true, or needs at least one, and not reverses a condition. Java evaluates from left to right and stops as soon as the result is known, so in an and the right side is not evaluated when the left side is already false. This is useful for safe checks, such as testing that a value exists before using it. The and operator is stronger than the or operator, so parentheses make a rule clear to every reader. Complex rules are easier to read when each piece gets a boolean variable with a meaningful name. The question for this step is: which conditions must hold together, and which are alternatives?

**Example:** A hospital pharmacy releases a medicine only if the prescription is valid and the patient is registered, which is an and rule that joins two conditions. If either check fails, nothing is released, because an and needs both conditions to be true. A boolean variable named prescriptionIsValid keeps the rule readable, so the pharmacist sees the first condition at a glance and the second one beside it.

**At work:** Tariq's discount applies when the customer is loyal and the total reaches the threshold, but never when the order is a return. He first writes it in one long line and cannot read it a week later. So he creates three named boolean variables: isLoyal, reachesThreshold and isReturn. The rule now reads almost like a sentence, and a colleague checks it in a minute.

## 3. Choose

When one value decides between many outcomes, a chain of else if becomes long. The switch statement is the cleaner tool. It looks at one value, such as a customer tier or a shipping method, and jumps to the



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

matching case. In the modern arrow form each case stands alone, and a switch can even return a value that we store in a variable. A default case covers every value we did not list, so nothing is silently ignored. In the older colon form each case must end with break, or execution falls through into the next case, which is a well-known source of bugs. The question for this step is: is one value choosing between many outcomes?

**Example:** A courier company sets the delivery fee by service level: standard, express or same day, so one value, the level, decides between three outcomes. One switch maps each level to its fee, and each case stands alone with its own fee. A new level is added with one more case, and the default case handles any unknown level, so no request is ever left without a fee.

**At work:** Tariq's shop has four customer tiers: bronze, silver, gold and platinum, each with its own discount rate. He replaces a long chain of else if with one switch that returns the rate. He adds a default case that returns no discount, so an unknown tier simply gets nothing. The whole logic now fits on one screen.

## 4. Repeat

A loop repeats code. The for loop is used when we know the number of rounds, and it has a start, a condition and a step. The while loop checks its condition before every round and runs while it is true, and the do while loop checks after the round, so it always runs at least once. The enhanced for loop, also called for each, visits every item of a collection or an array without a counter, which removes counting mistakes. Inside a loop, break leaves it immediately and continue skips to the next round. A loop whose condition never becomes false runs forever, so every loop needs something that changes the condition. The question for this step is: what repeats, and what makes it stop?

**Example:** A payroll system must process every employee in a list, so the same steps repeat once for each person. A for each loop visits each employee in turn, takes the person's data and calculates the pay, without any counter to manage. When the list ends, the loop stops by itself, which answers the question of this step: it repeats for each employee and stops at the end of the list.

**At work:** Tariq receives the day's orders as a list, which we meet properly in a later part, and writes a for each loop over them: `for (Order order : orders)`. Inside it he places the decision, the combined rule and the switch from the first three steps. He uses continue to skip returns and keeps a running total of the discounts given. The report at the end of the loop shows how many orders received a discount and how much it cost.

## Conclusion

To recap, a program decides with if and else, combines conditions with and, or and not, chooses between many outcomes with switch, and repeats work with loops. Every condition must be a boolean, text is compared with equals, and every loop needs a way to stop. Tariq's rules now run over the whole day's orders and give each loyal customer the right discount. Next time, we will package this kind of logic into methods that can be reused and tested. A good habit is to test the edges of a rule: the value just below the limit, exactly at the limit and just above it.

### Sources


- Oracle: The Java Tutorials, Control Flow Statements – https://docs.oracle.com/javase/tutorial/java/nutsandbolts/flow.html
- Oracle: The Java Tutorials, Equality, Relational, and Conditional Operators – https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op2.html
- Oracle: The Java Tutorials, The switch Statement – https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html
- dev.java: Learn Java – https://dev.java/learn/
- OpenJDK: Switch Expressions (arrow form) – https://openjdk.org/jeps/361

- 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.