Java Part 2: Variables, Types and Calculations

Every program remembers values: amounts, names, counts and yes or no answers. Java stores each value in a variable, and each variable has a type that says what it may hold. Four steps make this reliable: declare, calculate, protect precision and work with text. In this part, Lina, an accountant at a distribution company, uses them to total a month of invoices without losing a cent. Each step answers one practical question.

1. Declare

A variable is a named place in memory that holds one value. In Java we declare it with a type, a name and usually a starting value, and the type cannot change later. The main types for everyday work are int for whole numbers, long for very large whole numbers, double for decimal numbers, boolean for true or false, and char for a single character. Text uses String, which is a class rather than a basic type, and its name starts with a capital letter. By convention names use camel case, such as invoiceTotal, and the keyword final turns a variable into a constant that cannot be changed. In modern Java the word var lets the compiler work out the type of a local variable from its value, but the variable still keeps one fixed type. The question for this step is: what kind of value is this, and what should it be called?

Example: A warehouse supervisor needs to store a pallet count, a weight in kilograms and a flag that says whether the pallet is fragile. She chooses int for the count, double for the weight and boolean for the flag. Each name says what the value is, so anyone can read the code.

At work: Lina lists what her program must remember: the invoice count, the customer name, the quantity and whether the invoice is paid. She picks long for the invoice count, String for the customer name, int for the quantity and boolean for the paid flag. She marks the tax rate as final, because it should never change while the program runs. The step question is answered before she writes a single calculation.

2. Calculate

Java calculates with plus, minus, star for multiplication, slash for division and the percent sign for the remainder. Multiplication and division are done before addition and subtraction, and parentheses change the order. When both sides of a division are whole numbers, the result is a whole number and the fraction is dropped. So seven divided by two gives three, not three and a half, unless one side is a decimal number. Java converts a smaller type to a larger one automatically, but going the other way needs an explicit cast, which cuts off the decimals. The remainder operator is useful for questions such as how many items are left after filling the full boxes. The question for this step is: which type will the result have, and have we lost anything on the way?

Example: A packing planner has one hundred twenty items and boxes that hold fifty each. Dividing two whole numbers gives two full boxes, and the remainder operator gives twenty items left over. If she needs the exact share as a decimal, she converts one side to a decimal number first. The two answers together tell her how many boxes to fill and how many items go into the last one.

At work: Lina divides the total quantity of items by the count of invoices to find the average quantity per invoice, and the first result looks too round. She notices that both values are whole numbers, so Java dropped the fraction. She converts the total to a decimal number before dividing, and the average now shows its decimals. The step question made her check the result type.

3. Protect Precision

The double type stores decimals in binary form, and many everyday decimal fractions, such as a tenth, cannot be stored exactly. Adding such values can leave a tiny error far behind the decimal point, which is acceptable for measurements but not for money. For money, Java provides the BigDecimal class, which keeps exact

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

decimal digits. We create it from text, not from a double, because the text holds the exact digits we mean. Because BigDecimal is an object, we calculate with its methods, such as add, subtract and multiply, and when we divide we should state how many decimals to keep and how to round, because an exact division is not always possible. We also compare two amounts with compareTo, because equals treats a different number of decimals as a different value. The question for this step is: does this value need to be exact?

Example: A payroll clerk adds a long list of small allowances stored as double, such as ten cents and twenty cents, and finds a tiny fractional difference against the bank file. She switches every amount to BigDecimal created from text, so the digits are exact. The difference disappears, and the report matches the bank file to the last cent.

At work: Lina rebuilds the invoice line so that the price and the quantity are BigDecimal values created from text. The line total becomes `BigDecimal total = unitPrice.multiply(quantity);`, a multiplication done with a method, not with the star. For the tax she multiplies by the rate, then sets the scale to two decimals with half up rounding. Her month total now matches the accounting system exactly.

4. Work with Text

A String holds text, and once it is created it cannot be changed, so methods such as toUpperCase and trim return a new String. The plus sign joins strings together, and it also joins a string with a number. To compare the content of two strings we use equals, not the double equals sign, because the double equals sign checks whether both names point to the very same object. Useful methods are length, trim, substring, contains and replace. For clean output, printf and String format let us write a template with placeholders for text, whole numbers and decimals, so the columns line up. The question for this step is: are we comparing the content, and how should the result look?

Example: A call-centre supervisor reads customer names from a file, and some have extra spaces at the end. A comparison with the double equals sign may say that two names that look identical are different. After trim removes the spaces and equals compares the content, the names match and the duplicates are found.

At work: Lina prints a customer line from three pieces: the name, the invoice count and the total. She joins them with the plus sign, but the columns do not line up. She switches to printf with a template that gives the name a fixed width and the total two decimals. Now the whole month prints as a tidy table that she can paste into her report.

Conclusion

To recap, Java remembers values in variables, and every variable has one fixed type. We declare with the right type, calculate while watching the result type, protect money with BigDecimal, and treat text with care. Whole-number division drops the fraction, double carries tiny errors, and equals compares the content of text. Lina's month total now matches the accounting system, and every line prints cleanly. Next time, her program will start to make decisions and repeat work with conditions and loops. A good habit is to name every value by its meaning, because clear names save more time than clever code.

Sources

- Oracle: The Java Tutorials, Primitive Data Types – https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
- Oracle: The Java Tutorials, Operators – https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html
- Oracle: Java Platform API Specification, BigDecimal – https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html
- Oracle: The Java Tutorials, Strings – https://docs.oracle.com/javase/tutorial/java/data/strings.html

- 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.