Basics

Java is one of the most popular programming languages in the world. With Java you can build various types of applications such as desktop, web, mobile apps and distributed systems.

Java Development Kit

We use Java Development Kit (JDK) to build Java applications. JDK contains a compiler, the Java Runtime Environment (JRE) and a library of classes that we use to build applications.

Java Editions

We have four editions of Java, each used for building a different type of application:

- **Java Standard Edition (SE)**: the core Java platform. It contains all of the libraries that every Java developer must learn.
- **Java Enterprise Edition (EE)**: used for building very large scale, distributed systems. It's built on top of Java SE and provides additional libraries for building fault-tolerant, distributed, multi-tier software.
- **Java Micro Edition (ME)**: a subset of Java SE, designed for mobile devices. It also has libraries specific to mobile devices.
- **Java Card**: used in smart cards.

How Java Code Gets Executed

The Java compiler takes Java code and compiles it down to Java Bytecode which is a cross-platform format. When we run Java applications, Java Virtual Machine (JVM) gets loaded in the memory. It takes our bytecode as the input and translates it to the native code for the underlying operating system. There are various implementations of Java Virtual Machine for almost all operating systems.

Architecture of Java Applications

The smallest building blocks in Java programs are **methods** (also called functions in other programming languages). We combine related methods in **classes**, and related classes in **packages**. This modularity in Java allows us to break down large programs into smaller building blocks that are easier to understand and re-use.

5 Interesting Facts about Java

1. Java was developed by James Gosling in 1995 at Sun Microsystems (later acquired by Oracle).
2. It was initially called Oak. Later it was renamed to Green and was finally renamed to Java inspired by Java coffee.
3. Java has close to 9 million developers worldwide.
4. About 3 billion mobile phones run Java, as well as 125 million TV sets and every Blu-Ray player.
5. According to indeed.com, the average salary of a Java developer is just over $100,000 per year in the US.

Types

Variables

We use variables to temporarily store data in computer's memory. In Java, the type of a variable should be specified at the time of declaration.

In Java, we have two categories of types:

- **Primitives:** for storing simple values like numbers, strings and booleans.
- **Reference Types:** for storing complex objects like email messages.

Primitive Types

Type	Bytes	Range
byte	1	[-128, 127]
short	2	[-32K, 32K]
int	4	[-2B, 2B]
long	8	
float	4	
double	8	
char	2	A, B, C, …
boolean	1	true / false

Declaring Variables

```
byte age = 30;
long viewsCount = 3_123_456L;
float price = 10.99F;
char letter = 'A';
boolean isEligible = true;
```

- In Java, we terminate statements with a semicolon.

- We enclose characters with single quotes and strings (series of characters) with double quotes.
- The default integer type in Java is int. To represent a long value, we should add L to it as a postfix.
- The default floating-point type in Java is double. To represent a float, we should append F to it as a postfix.

Comments

We use comments to add notes to our code.

```
// This is a comment and it won't get executed.
```

Reference Types

In Java we have 8 primitive types. All the other types are reference types. These types don't store the actual objects in memory. They store the reference (or the address of) an object in memory.

To use reference types, we need to allocate memory using the new operator. The memory gets automatically released when no longer used.

```
Date now = new Date();
```

Strings

Strings are reference types but we don't need to use the new operator to allocate memory to them. We can declare string variables like the primitives since we use them a lot.

```
String name = "Mosh";
```

Useful String Methods

The String class in Java provides a number of useful methods:

- startsWith("a")
- endsWith("a")
- length()

- indexOf(“a”)
- replace(“a”, “b”)
- toUpperCase()
- toLowerCase()

Strings are immutable, which means once we initialize them, their value cannot be changed. All methods that modify a string (like toUpperCase) return a new string object. The original string remains unaffected.

Escape Sequences

If you need to use a backslash or a double quotation mark in a string, you need to prefix it with a backslash. This is called escaping.

Common escape sequences:

- \\
- \”
- \n (new line)
- \t (tab)

Arrays

We use arrays to store a list of objects. We can store any type of object in an array (primitive or reference type). All items (also called elements) in an array have the same type.

```
// Creating and and initializing an array of 5 elements
int[] numbers = new int[3];
numbers[0] = 10;
numbers[1] = 20;
numbers[2] = 30;

// Shortcut
int[] numbers = { 10, 20, 30 };
```

Java arrays have a fixed length (size). You cannot add or remove new items once you instantiate an array. If you need to add new items or remove existing items, you need to use one of the collection classes.

The Array Class

The Array class provides a few useful methods for working with arrays.

```
int[] numbers = { 4, 2, 7 };
Arrays.sort(numbers);
String result = Arrays.toString(numbers);
System.out.println(result);
```

Multi-dimensional Arrays

```
// Creating a 2x3 array (two rows, three columns)
int[2][3] matrix = new int[2][3];
matrix[0][0] = 10;

// Shortcut
int[2][3] matrix = {
                        { 1, 2, 3 },
                        { 4, 5, 6 }
                    };
```

Constants

Constants (also called final variables) have a fixed value. Once we set them, we cannot change them.

```
final float INTEREST_RATE = 0.04;
```

By convention, we use CAPITAL LETTERS to name constants. Multiple words can be separated using an underscore.

Arithmetic Expressions

```
int x = 10 + 3;
```

```
int x = 10 - 3;
int x = 10 * 3;
int x = 10 / 3;                    // returns an int
float x = (float)10 / (float)3;  // returns a float
int x = 10 % 3;   // modulus (remainder of division)
```

Increment and Decrement Operators

```
int x = 1;
x++;    // Equivalent to x = x + 1
x--;    // Equivalent to x = x - 1
```

Augmented Assignment Operator

```
int x = 1;
x += 5;  // Equivalent to x = x + 5
```

Order of Operations

Multiplication and division operators have a higher order than addition and subtraction. They get applied first. We can always change the order using parentheses.

```
int x = 10 + 3 * 2;        // 16
int x = (10 + 3) * 2;      // 26
```

Casting

In Java, we have two types of casting:

- **Implicit**: happens automatically when we store a value in a larger or more precise data type.
- **Explicit**: we do it manually.

```
// Implicit casting happens because we try to store a short
// value (2 bytes) in an int (4 bytes).
short x = 1;
int y = x;
```

```
// Explicit casting
int x = 1;
short y = (short) x;
```

To convert a string to a number, we use one of the following methods:

- Byte.parseByte(“1”)
- Short.parseShort(“1”)
- Integer.parseInt(“1”)
- Long.parseLong(“1”)
- Float.parseFloat(“1.1”)
- Double.parseDouble(“1.1”)

Formatting Numbers

```
NumberFormat currency = NumberFormat.getCurrencyInstance();
String result = currency.format(“123456”); // $123,456
```

```
NumberFormat percent = NumberFormat.getPercentInstance();
String result = percent(“0.04”); // 4%
```

Reading Input

```
Scanner scanner = new Scanner(system.in);
double number = scanner.nextDouble();
byte number = scanner.nextByte();
String name = scanner.next();
String line = scanner.nextLine();
```

Control Flow

Comparison Operators

We use comparison operators to compare values.

```
x == y    // equality operator
x != y.   // in-equality operator
x > y
x >= y
x < y
x <= y
```

Logical Operators

We use logical operators to combine multiple boolean values/expressions.

- x && y (AND): if both x and y are true, the result will be true.
- x || y (OR): if either x or y or both are true, the result will be true.
- !x (NOT): reverses a boolean value. True becomes false.

```
bool hasHighIncome = true;
bool hasGoodCredit = false;
bool hasCriminalRecord = false;
bool isEligible = (hasHighIncome || hasGoodCredit) && !isEligible;
```

If Statements

Here is the basic structure of an if statement. If you want to execute multiple statements, you need to wrap them in curly braces.

```
if (condition1)
   statement1
else if (condition2)
   statement2
else if (condition3)
   statement3
```

```
else
    statement4
```

The Ternary Operator

```
String className = (income > 100_000) ? “First” : “Economy”;
```

This is a shorthand to write the following code:

```
String className;

if (income > 100_000)
     className = “First”;
else
     className = “Economy”;
```

Switch Statements

We use switch statements to execute different parts of the code depending on the value of a variable.

```
switch (x) {
   case 1:
      …
      break;

   case 2:
      …
      break;

   default:
      …
}
```

After each **case** clause, we use the break statements to jump out of the switch block.

For Loops

For loops are useful when we know ahead of time how many times we want to repeat something. We declare a loop variable (or loop counter) and in each iteration we increment it until we reach the number of times we want to execute some code.

```
for (int i = 0; i < 5; i++)
    statement
```

While Loops

While loops are useful when we don't know ahead of time how many times we want to repeat something. This may be dependent on the values at run-time (eg what the user enters).

```
while (someCondition) {
    …
    if (someCondition)
       break;
}
```

We use the **break** statement to jump out of a loop.

Do..While Loops

Do..While loops are very similar to **while** loops but they executed at least once. In contrast, a while loop may never get executed if the condition is initially false.

```
do {
    …
} while (someCondition);
```

For-each Loops

For-each loops are useful for iterating over an array or a collection.

```
int[] numbers = {1, 2, 3, 4};
for (int number : numbers)
```