Introduction to Java

H.1. Introduction to Java

This material is a direct copy of java basics course on hyperskill.org from JetBrains
H.x. notation signifies this

Ever wondered why Java's logo is a steaming cup of coffee? The creators of Java, while brainstorming a name for their new language, chose 'Java', a slang term for 'coffee'. Just as coffee fuels our day, Java powers the tech world with its robust and versatile features.

In this topic, we will explore why Java has been a popular choice among developers for over two decades and how it has brewed success in various domains. We will also introduce you to your very first Java program. So, grab your cup of coffee and join us on this exciting journey into the world of Java!

What is Java

Java is a high-level, class-based, object-oriented[1] programming language. James Gosling at Sun Microsystems (now part of Oracle Corporation) designed it, and it was released in 1995. The language was developed with the **"Write Once, Run Anywhere"** (WORA) philosophy. This principle underscores Java's key feature - **platform independence**, allowing the same Java program to run on multiple platforms without modifications.

Java is designed to be both **simple** and **powerful**. It borrows its syntax[2] from C and C++, but eliminates certain low-level programming complexities, such as explicit memory management and multiple inheritance found in C++. Unlike these two languages, Java does not require you to

manually clean the application memory, as it has a garbage collector that performs this task automatically. Known for its robustness, security, and simplicity, Java has become a popular choice among developers worldwide. It supports different programming techniques, including generic programming, multithreaded and concurrent programming, and functional programming.

Where is Java applied

Let's go through a typical day and see how Java impacts our lives without us even realizing it.

Imagine waking up to your Android alarm. As you reach out to snooze it, you're interacting with an application built using Java. You decide to work on a project using a development tool like IntelliJ IDEA or Eclipse. As you write and compile[3] your code, Java is there, forming the backbone of these development tools. During lunch, you enjoy a Netflix show or Spotify music, both services powered by Java. Later, you finish the project and receive payment. Behind the scenes, Java is working diligently, processing your request. In the evening, you unwind with a game of Minecraft, yet another Java-based application.

Java is like a silent friend, aiding us and making our lives easier in numerous ways, from the moment we wake up till we call it a day.

A sample of Java

Let's create the classic "Hello, World!" program, a friendly greeting from your computer.

Here's the program:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Don't worry if it looks a bit cryptic now. You'll get the hang of it soon.

This program simply prints the phrase "Hello, World!" to the console. But there's a lot going on here:

- **public class HelloWorld** -In Java, all the code you write will be inside classes. We're telling Java we're creating a new public class (a kind of blueprint) and we're naming it HelloWorld. Every Java application has to have at least one class.
- **public static void main(String[] args)** - This is the heart of our program, where the execution begins.
- **System.out.println("Hello, World!");** - These are our program's first words! This command [4] instructs Java to print "Hello, World!" to the console via the `System.out.println()` method, providing instant feedback.

Note that in Java code, we use different types of brackets, and if you open any bracket, you are required to close it.

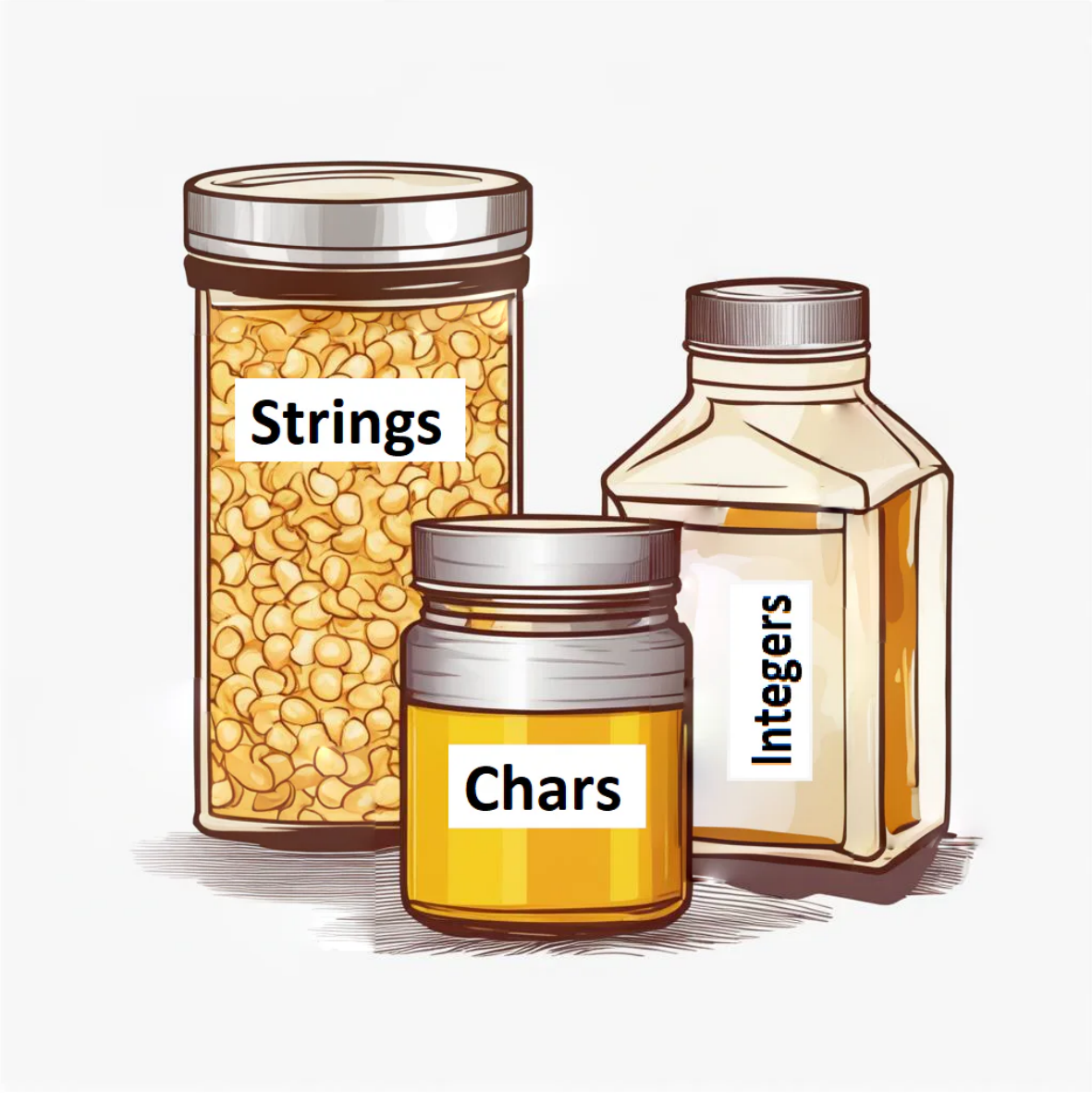
And there you have it, your first Java program! It's a modest step, but it marks the beginning of an exciting journey into Java programming.

---

1. In Java, object-oriented programming (OOP) is a programming paradigm that revolves around the concept of objects, which can represent real-world entities or abstract concepts. Each object has its own state, which is stored in fields, and behavior, which is defined by methods. Objects are instances of classes, which serve as blueprints that define the properties and methods of their corresponding objects. OOP emphasizes encapsulation, inheritance, and polymorphism, making software design more reusable and maintainable. It is a good practice to use interface-oriented design, which means relying on interfaces instead of concrete implementations. Interfaces define a contract that classes must adhere to, allowing for greater flexibility and modularity in programming. Java is primarily an object-oriented programming language, but it also supports other programming paradigms, such as functional programming. OOP provides a way to create well-structured and readable programs by allowing developers to define the behavior of objects and classes using methods. ↩
2. In Java, syntax refers to a set of rules that define how a program needs to be written in order to be valid. It includes the order of components, such as statements written from top to bottom in a sequential manner, and the use of specific symbols, such as semicolons to terminate a statement. Java has its own specific syntax, which includes the use of keywords, identifiers, literals, operators, and separators. These tokens must adhere to the specific rules of the Java syntax and are separated by whitespace. ↩
3. In Java, a compile is the process of translating source code into bytecode that can be executed by the Java Virtual Machine (JVM). | During compilation, the Java compiler checks the code for errors, such as typos, incorrect method invocations, and mismatched variable declarations. If any errors are found, the code will not compile and an error message will be generated. To avoid compile-time errors, programmers can use tools like Integrated Development Environments (IDEs) with static code analyzers, which can identify and highlight potential errors before the code is compiled. ↩
4. In Java, a command is a behavioral design pattern that encapsulates a request or action as an object, allowing for greater flexibility and decoupling between the invoker and the implementer of the command. | The command pattern is commonly used in GUI button and menu items, networking, and transactional behavior. It can also be used to implement undo functionality and create macro commands. The command pattern typically involves an interface with a method that performs the command, and one or more classes implementing this interface that encapsulate the data needed to execute the command. ↩

H.2. Basic literals

Regardless of its complexity, a program always performs operations on numbers, strings, and other values. These values are called **literals**. There are many different sorts of literals in Java, but in this topic we will focus only on a few of them: the ones that surround us all the time in everyday life.



Consider literals as groceries. To use them, usually you need to store them somewhere. Typically, they are stored in variables [1], which you can think of as containers designed to hold a specific type of data.

Variables can only store matching data. You wouldn't want to accidentally put honey in a cardboard cereal box or pour cereal into a salt shaker. To prevent such mistakes, learn to distinguish between the basic literals: integer numbers, strings, and characters.

Integer numbers

You use these numbers to count things in the real world as natural numbers. Integer numbers also include zero and negative ones. Here are several examples of valid integer number literals separated by commas: `0, 1, 2, 10, 11, -100`.

Here is how integers can be used in code:

```
int numApples = 1000;
```

Reading code is crucial for anyone in IT, so let's parse it together. Here you put the integer 1000 into a variable of an integer type, called numApples. This is similar to filling a container with its designated contents!

You can increase code readability by dividing the digit into blocks with underscores: `1_000_000` is more readable than `1000000`. So let's pack our apples to make selling them easier:

```
int numPackedApples = 1_000_000;
```

Fear not if these code snippets aren't 100% clear to you yet! They aim to help you develop the skill of code reading. Just grasp the overall meaning and follow your study plan, and you'll be writing your own code in no time!

Characters

A character is a single symbol, denoted with single quotes. You can use character literals[2] to represent single letters like `'A', 'x'`, digits from `'0'` to `'9'`, whitespaces `(' ')`, and other characters or symbols like `'$'`.

Be mindful of quotes and avoid confusing characters representing digits with the digits themselves:

```
char charOne = '1'
int numOne = 1
```

Fun fact: characters sit between integers and strings: they resemble strings, yet you can do math with them.

Strings

A string is a sequence of characters, encapsulated by double quotes. It represents text-based information, such as an advertising line, a webpage address, or a website login name. Here are some valid examples: `"text", "I want to know Java", "123456", "e-mail@gmail.com"`. As you can see, a string can include letters, digits, whitespaces, and other characters altogether.

A string consisting of a single character like `"A"` is also a valid string, but do not confuse it with the `'A'` character. Note the difference in quotes!

```
char singleQuoted = 'A'
String doubleQuoted = "A"
```

Printing literals using variables

In Java, assigning literals to variables and printing them using `System.out.println` is a fundamental operation. To assign a literal to a variable, you first declare[3] the variable with its type and then initialize it with a literal value. For instance, you might declare an integer variable like `int number = 42;`, where `42` is the literal value assigned to `number`. Similarly, for a string variable, you could write `String greeting = "Hello";`, where `"Hello"` is the string literal assigned to `greeting`. Once the variables are initialized, you can use `System.out.println` to print their values to the console:

```
public class Main {
    public static void main(String[] args) {
        int number = 42;
        String greeting = "Hello";

        System.out.println(number);
        System.out.println(greeting);
    }
}
```

For example, `System.out.println(number);` will output the value `42`, and `System.out.println(greeting);` will display `Hello`. This approach allows you to store and manipulate literal values in variables and then output them as needed for various purposes in your Java programs.

---

1. In Java, a variable is a named storage location that is used to store a value of a specific type. It is declared with a specific data type, which determines the kind of value that can be stored in it. Every variable has a unique name, also known as an identifier, which is used to access its value. Variables can be declared and initialized in a single statement, and their value can be accessed and modified using the name. It's important to note that variables can be changed, meaning you can assign a new value to a variable without having to declare it again. ↩
2. In Java, a character literal is a single symbol, denoted with a single quote, that represents a single character such as a letter, digit, whitespace, or other symbol. It's important to distinguish character literals that represent digits from the digits themselves. Character literals are similar to strings, but you can perform mathematical operations with them. A string, on the other hand, is a sequence of characters, enclosed by double quotes, that represents text-based information. Character literals are a type of literal, along with number and string literals, that are used to represent values in Java ↩
3. In Java, a declaration is a statement that introduces a variable, method, or class into the program. | It specifies the type of the entity being declared, its name, and other relevant information such as access modifier, return type (for method), and initial value (for variable). For example, the declaration of a method may include its name, return type, access modifier, and parameter list. Declarations are essential for Java to understand the structure and behavior of the program. Incorrect declarations can lead to compilation errors. In the context of variables, a declaration specifies the type, name, and initial value of a variable. The type of a variable determines what possible operations can be performed on the variable and which value can be stored in it. Every variable has a name (also known as an identifier) to distinguish it from others. Before using a variable, it must be declared. The general form of a variable declaration is: `type identifier = value;` It's important to note that a declaration is not the same as an assignment. ↩

H.3 Writing first program

In this topic, we will build our very first Java program. Our program will simply print **"Hello, World!"** on the screen (a tradition by most programmers when learning new languages). Our code may not seem too exciting at first, however, we will learn about the basic template that all Java programs need to follow.

The Hello World program

Here is the Java code of this program:

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

You can type this code in the **Your Code** section here and then press the **execute** button. In the **result** section, you will see:

```
Hello, World!
```

If you have already installed Java, you can run the program on your computer. If not, there is no need to install it right now. We will do that later.

The basic terminology

Now that you have seen the result, let's learn some basic terminology and then try to understand this program.

- **Program** – a sequence of instructions (called statements), which are executed one after another in a predictable manner. Sequential flow is the most common and straightforward sequence of statements, in which statements are executed in the order that they are written – from top to bottom in a sequential manner;
- **Statement** – a single action (like print some text) terminated by a semicolon (`;`);
- **Block** – a group of zero, one or more statements enclosed in a pair of braces `{...}`; There are two such blocks in the program above;
- **Method** – a sequence of statements that represents a high-level operation (also known as a subprogram or procedure);
- **Syntax** – a set of rules that define how a code needs to be written in order to be valid; Java has its own specific syntax that we will learn.
- **Keyword** – a word that has a special meaning in the programming language (`public`, `class`, and many others). These words cannot be used as variable names in your own program;
- **Identifier or name** – a word that refers to something in a program (such as a variable or a function name);

- **Comment** – a textual explanation of what the code does. Java comments start with `//` or `/*`;
- **Whitespace** – all characters that are not visible (space, tab, newline, etc.).

The Hello World program under a microscope

The **Hello World** program illustrates the basic elements of Java programs. For now, we will discuss only the most important elements.

1. **The public class**.** It is the basic unit of a program. Every Java program must have at least one class. The definition of a class consists of the `class` keyword followed by the class name. A class can have any name, such as `App`, `Main`, or `Program`, but it must not start with a digit. A set of braces `{...}` encloses the body of a class.

```
public class Main {
    // ...
}
```

The text after `//` is just a comment, not a part of the program. We will learn about comments in detail in later topics.

2. The main method.** To make the program runnable, we put a method named `main` inside a class, otherwise, it will not run. It is the entry point for a Java program. Again, the braces `{...}` enclose the body of the method, which contains programming statements.

```
public static void main(String[] args) {
    // statements go here
}
```

The keywords `public`, `static`, and `void` will be discussed later, so just remember them for now. The name of this method (`main`) is predefined and should always be the same. Capitalization matters: if you name your first method **Main**, **MAIN** or something else, the program cannot start.

The element `String[] args` represents a sequence of arguments passed to the program from the outside world. Don't worry about them right now.

3. Printing "Hello, World!". The body of the method consists of programming statements that determine what the program should do after starting. Our program prints the string **"Hello, World!"** using the following statement:

```
System.out.println("Hello, World!"); //  each statement has to end with ;
```

This is one of the most important things to understand from the **Hello World** program. We invoke a special method `println` to display a string followed by a new line on the screen. We will often use this approach to print something of interest to the screen. The text is printed without double quotes.

It is important that **"Hello, World!"** is not a keyword or an identifier; it is just some text to be printed.

Keywords

As you can see, even a simple Java program consists of many elements, including **keywords**that are parts of the language. In total, Java provides more than 50 keywords which you will gradually learn on this platform. The full list is here, though you don't need to remember all of them at this moment.

Note, `main` is outside the given list because it is not a keyword.

Congratulations on learning how to create your first Java program! This simple "Hello, World!" example is just the beginning. As you progress, you'll build more complex projects that you can add to your portfolio on GitHub and showcase to potential employers. Feel free to complete projects you enjoy and share them with others. Remember, learning is a journey, and we're here to help you every step of the way!

Conclusion

We have discussed the simplest program you can write in Java. It has a single class with a single `main` method. Every Java program must have a `main` method as it is the first to be executed when the program runs. Don't worry about memorizing every single term used in the topic (syntax, statement, block). These terms will reappear in further materials. Do not forget to use the provided **Hello World** program as a template in your own programs.

H.4 Printing data

When you write programs, you often need to print calculation results, text, or any other type of data. Also, throughout this educational platform, you will write a lot of programs that print data on the screen. Let's learn how to do that using a standard approach in Java.

Displaying text using println() and print()

Standard output is a receiver to which a program can send information as text. It is supported by all common operating systems. Java provides a special `System.out` object to work with the standard output. We will often use it to print data.

The `println` method displays the passed string followed by a new line on the screen (**print-line**). For example, the following code snippet prints four lines.

```
System.out.println("I ");
System.out.println("know ");
System.out.println("Java ");
System.out.println("well.");
```

Here is the output we get:

```
I
know
Java
well.
```

All the strings were printed as they are, without double quotes.

This method allows you to print an empty line when no string is given:

```
System.out.println("Java is a popular programming language.");
System.out.println(); // prints empty line
System.out.println("It is used all over the world!");
```

And here is the output:

```
Java is a popular programming language.

It is used all over the world!
```

The `print` method displays the value that was passed in and places the cursor (the position where we display a value) after it. For example, the code below outputs all strings in a single line.

```
System.out.print("I ");
System.out.print("know ");
```

```
System.out.print("Java ");
System.out.print("well.");
```

We receive the following output:

```
I know Java well.
```

Pay attention to the spaces between words. We pass them to the method for printing.

4 letters **sout** and pressing the **Tab** key in the IntelliJ IDEA development environment will automatically write `System.out.println()` for you

New line using escape characters

In many programming languages, including Java, you can use an escape character `\n` which moves the cursor to the beginning of the next line of output. When you include `\n` in a string passed to `System.out.print`, it breaks the text at that point and continues printing on the following line. For example, the statement `System.out.print("Hello\nWorld");` will output the following:

```
Hello
World
```

This option is useful when you need to break a long text into new lines in multiple places or when you need to have multiple new lines:

```
Hello



World
```

The following can be achieved by using three `\n` characters:

```
System.out.print("Hello\n\n\nWorld");
```

Printing numbers and characters

Both `println` and `print` methods allow a program to print not only strings and characters, but also numbers.

Let's print two secret codes.

```
System.out.print(108);   // printing a number
System.out.print('c');   // printing a character that represents a letter
System.out.print("Q");   // printing a string
System.out.println('3'); // printing a character that represents a digit

System.out.print(22);
```

```
System.out.print('E');
System.out.print(8);
System.out.println('1');
```

Here is our output:

```
108cQ3
22E81
```

As is the case with strings, none of the printed characters contain quotes.

Concatenating data

In practice, you may need to concatenate strings or other literals inside `System.out.println` into a single output. This can be achieved by simply placing them adjacent to each other and use the + operator to join them. For example, if you want to print a greeting with a name, you might write the following:

```
public class Main {
    public static void main(String[] args) {
        String name = "John";

        System.out.println("Hello, " + name + "!"); // Hello, John!
    }
}
```

The `+` operator merges these elements into one cohesive string, which is then passed to `System.out.println` to display the result on the console. Similarly, you can concatenate strings with integers or other types:

```
public class Main {
    public static void main(String[] args) {
        int age = 33;

        System.out.println("Hello, I am " + age + " years old."); // Hello, I am
33 years old.
    }
}
```

In such cases, the value of that type will be converted to a string and merged with the string to the left of the `+` operator.

Take in account that when you concatenate a string with multiple integers in Java, the integers are converted to their string representations and concatenated in sequence. Here's how it works step-by-step:

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, I am " + 3 + 3 + " years old."); // Hello, I
```

```
am 33 years old.
    }
}
```

You might expect the result to be "**Hello, I am 6 years old**" but when you concatenate the first integer with the string, you get "**Hello, I am 3**" as a result. The second `+` operator then merges this string with another `3`, converting the integer to a string again.

Conclusion

In Java, you can print data via the standard output using the `System.out` object. You can use the `println` method to display the passed string in a print-line and the `print` method to output all passed strings in a single line. Both of these methods also allow for printing numbers as well as characters.