What is Java?

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

Java is a powerful, general-purpose programming environment.

The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms.

Java is –

- **Object Oriented** – In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** – Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code.
- **Simple** – Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** – With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** – Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable** – Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable.
- **Robust** – Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded** – With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously.
- **Interpreted** – Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

- **High Performance** – With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed** – Java is designed for the distributed environment of the internet.
- **Dynamic** – Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

The Way of the Programmer

One of goals of this class is to get students to think like a computer scientist. This way of thinking combines some of the best features of mathematics, engineering, and natural science. Like mathematicians, computer scientists use formal languages to denote ideas.

The single most important skill for a computer scientist is problem solving. Problem solving means the ability to formulate problems, think creatively about solutions, and express a solution clearly and accurately. As it turns out, the process of learning to program is an excellent opportunity to practice problem-solving skills

What is a program?

A program is a sequence of instructions that specifies how to perform a computation. The computation might be something mathematical, such as solving a system of equations or finding the roots of a polynomial, but it can also be a symbolic computation, such as searching and replacing text in a document or (strangely enough) compiling a program. The details look different in different languages, but a few basic instructions appear in just about every language:

- Input: Get data from the keyboard, a file, or some other device.
- Output: Display data on the screen or send data to a file or other device.
- Math: Perform basic mathematical operations like addition and multiplication.
- Conditional execution: Check for certain conditions and execute the appropriate code.
- Repetition: Perform some action repeatedly, usually with some variation.
-

You can think of programming as the process of breaking a large, complex task into smaller and smaller subtasks until the subtasks are simple enough to be performed with one of these basic instructions.

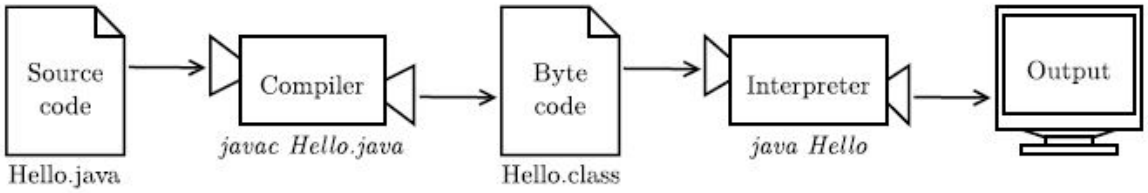
Programming Languages

Before they can run, programs in high-level languages have to be translated into a low-level language, also called \machine language". This translation takes some time, which is a small disadvantage of high-level languages. But high-level languages have two advantages:

- . It is much easier to program in a high-level language. Programs take less time to write, they are shorter and easier to read, and they are more likely to be correct.
- . High-level languages are portable, meaning they can run on different kinds of computers with few or no modifications. Low-level programs can only run on one kind of computer, and have to be rewritten to run on another

Two kinds of programs process high-level languages into low-level languages:
Interpreters processes the program a little at a time, alternately reading lines and performing computations.

Compilers translates source code into object code, which is run by a hardware executor the object code or the executable.



Java is both compiled and interpreted. Instead of translating programs directly into machine language, the Java compiler generates byte code. Similar to machine language, byte code is easy and fast to interpret. But it is also portable, so it is possible to compile a Java program on one machine, transfer the byte code to another machine, and run the byte code on the other machine. The interpreter that runs byte code is called a **Java Virtual Machine" (JVM).**

Beginner Friendliness

Generally, Java was designed to be relatively beginner-friendly in that it assumes the programmer is not that smart or careful, so programmers will be less likely to shoot themselves in the foot when coding Java apps.

Relatively Simple to Use

Java is a high level language, which means Java abstracts away (i.e. handles for you) most of the complex details of the machine (computer) such as memory management, etc. Thus, you can focus on programming instead of worrying about the little details many consider both tedious and difficult.

Less Rewarding from the Start

Java is very stubborn and verbose, which means you need to write a lot of code trying to convince the language to build some feature. Thus, you may need to spend a lot of time coding before you can get a working app. This may be demotivating for coding beginners.

Scalability

Easier to Maintain

Java is a statically-typed language, which means your code will have to be checked for errors before it can be built into an app. This means errors will be easier to track down. Furthermore, since statically-typed languages are also more strict with the definitions of things, you will have less strange and unexpected errors, which means your codebase will be easier to maintain as it grows in size and complexity.

Fast

As a statically typed language, Java is faster than dynamically typed languages because things are more clearly defined. Thus, when the app is running, your machine's resources will not be wasted on checking the definition of something in your code.

Optimized Performance

Modern Java programs are now even more performant thanks to mature JIT (just in time) compilers and improved JVMs, as performance can be optimized in real time to help a Java program run faster. This is very helpful as your app grows larger or needs to handle more processes.

Community

Community size is important because the larger a programming language community is, the more support you'd be likely to get. As you step into the programming world, you'll soon understand how vital support is, as the developer community is all about giving and receiving help. Moreover, the larger a community, the more people will be building useful tools to make development in that particular language easier.

The hello world program

Traditionally, the first program you write when learning a new programming language is called the hello world program.

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

When this program runs it displays:

```
Hello, World!
```

A statement is a line of code that performs a basic operation. In the hello world program, this line is a print statement that displays a message on the screen:

```
System.out.println("Hello, World!");
```

Java is \case-sensitive", which means that uppercase and lowercase are not the same. In this example, System has to begin with an uppercase letter; system and SYSTEM won't work.

A method is a named sequence of statements. This program defines one method named main:

```
public static void main(String[] args)
```

The name and format of main is special: when the program runs, it starts at the first statement in main and ends when it finishes the last statement. Later, we will see programs that define more than one method.

A class is a collection of methods. This program defines a class named Hello. You can give a class any name you like, but it is conventional to start with a capital letter. The name of the class has to match the name of the _le it is in, so this class has to be in a _le named Hello.java.

Java uses squiggly braces (f and g) to group things together. In Hello.java, the outermost braces contain the class definition, and the inner braces contain the method definition.

The line that begins with two slashes (//) is a comment, which is a bit of English text that explains the code. When the compiler sees //, it ignores everything from there until the end of the line. Comments have no effect on the execution of the program, but they make it easier for other programmers (and your future self) to understand what you meant to do.

Displaying strings

You can put as many statements as you like in main. For example, to display more than one line of output:

```
public class Hello {
	public static void main(String[] args) {
		// generate some simple output
		System.out.println("Hello, World!"); // first line
		System.out.println("How are you?"); // another line
	}
}
```

As this example shows, you can put comments at the end of a line as well as on lines all by themselves. Phrases that appear in quotation marks are called strings, because they contain a sequence of \characters" strung together. Characters can be letters, numbers, punctuation marks, symbols, spaces, tabs, etc. System.out.println appends a special character, called a newline that moves to the beginning of the next line. If you don't want a newline at the end, you can use print instead of println:

```
public class Goodbye {
	public static void main(String[] args) {
		System.out.print("Goodbye, ");
		System.out.println("cruel world");
	}
}
```

In this example, the first statement does not add a newline, so the output appears on a single line as Goodbye, cruel world. Notice that there is a space at the end of the first string, which appears in the output.

Escape sequences

It is possible to display multiple lines of output in just one line of code. You just have to tell Java where to put the line breaks.

```
public class Hello {
	public static void main(String[] args) {
		System.out.print("Hello!\nHow are you doing?\n");
	}
}
```

The output is two lines, each ending with a newline character:
Hello!
How are you doing?

The \n is an escape sequence, which is a sequence of characters that represents a special character. The backslash allows you to \escape" the strings literal interpretation. Notice there is no space between \n and How. If you add a space there, there will be a space at the beginning of the second line.

Another common use of escape sequences is to have quotation marks inside of strings. Since double quotes indicate the beginning and end of strings, you need to escape them with a backslash. System.out.println("She said \"Hello!\" to me.");

\n	newline
\t	tab
\”	Double quote
\\	backslash

Debugging code

Whenever you are experimenting with a new feature, you should also try to make mistakes. For example, in the hello world program, what happens if you leave out one of the quotation marks? What if you leave out both? What if you spell println wrong? These kinds of experiments help you remember what you read. They also help with debugging, because you learn what the error messages mean. It is better to make mistakes now and on purpose than later on and accidentally.

Programming and debugging should go hand in hand. Don't just write a bunch of code and then perform trial and error debugging until it all works. Instead, start with a program that does something and make small modifications, debugging them as you go, until the program does what you want. That way you will always have a working program, and it will be easier to isolate errors.

- **Problem solving**: The process of formulating a problem, finding a solution, and expressing the solution.
- **Program**: A sequence of instructions that specifies how to perform tasks on a computer.
- **Programming**: The application of problem solving to creating executable computer programs.
- **Computer** science: The scientific and practical approach to computation and its applications.
- **Algorithm**: A procedure or formula for solving a problem, with or without a computer.
- **Bug**: An error in a program.
- **Debugging**: The process of finding and removing errors.
- **High-level language**: A programming language that is designed to be easy for humans to read and write.
- **Low-level language**: A programming language that is designed to be easy for a computer to run. Also called \machine language" or \assembly language".
- **Portable**: The ability of a program to run on more than one kind of computer.
- **Interpret**: To run a program in a high-level language by translating it one Line at a time and immediately executing the corresponding instructions.
- **Compile**: To translate a program in a high-level language into a low-level
- **Language**, all at once, in preparation for later execution.
- **Source code**: A program in a high-level language, before being compiled.
- **Object code**: The output of the compiler, after translating the program.
- **Executable**: Another name for object code that is ready to run on specific hardware.
- **Byte code**: A special kind of object code used for Java programs. Byte code is similar to a low-level language, but it is portable like a high-level language.
- **Statement**: Part of a program that specific one step of an algorithm.

- **Print statement**: A statement that causes output to be displayed on the screen.
- **Method**: A named sequence of statements.
- **Class**: For now, a collection of related methods. (We will see later that there is more to it.)
- **Comment**: A part of a program that contains information about the program but has no effect when the program runs.
- **String**: A sequence of characters; the primary data type for text.