|  |
| --- |
| Java Fundamentals |
|  |

In today's world, social media websites have become an integral part of one's daily life. Sharing details of every moment of one's day, appreciating other's viewpoints or posts, sharing one's opinions, etc are some of the activities that we perform on a daily basis. And to be a part of this, one has to first log in to the particular website. Any leading social website has more than a billion users operating on it every day. The data shown to one user will be exclusive to that user and may not be shown to any other. Due to this, the amount of data that would have to be stored for this purpose would be huge.

Due to the huge amount of data involved, developing an application similar to a social media website would have to follow a few checkpoints,

* It should be large-scale
* It should have the potential to accept new functionalities after development
* It should be secure
* It should also be split into multiple isolated layers.

All these can be achieved by developing an Enterprise application.

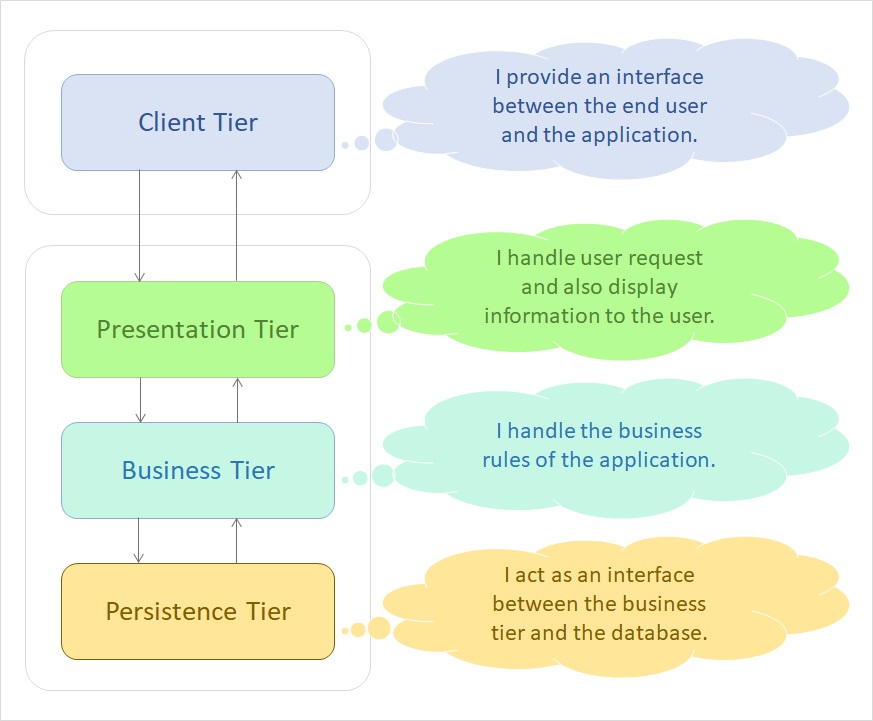
An Enterprise Application is a software application that helps enterprises in managing their business-related activities. They are large and complex and solve business-related problems to fulfill the needs of the enterprises.

Few of the enterprise application you would have encountered till now are,

* Sparsh
* Yammer
* Microsoft Outlook
* Microsoft Skype
* Yours truly (Lex)

Let us see the different layers in an enterprise application.

Since the enterprise application has been used for data manipulation of a large amount of data, it has become huge and complex. Maintaining such a complex application is challenging. And to resolve this, the enterprise application will be divided into different tiers.



There are 4 tiers that the application is split into,

**Client-Tier:**  
Acts as an interface to the end-user. Clients can be a web browser or standalone applications which is present in Computer, Mobile etc.

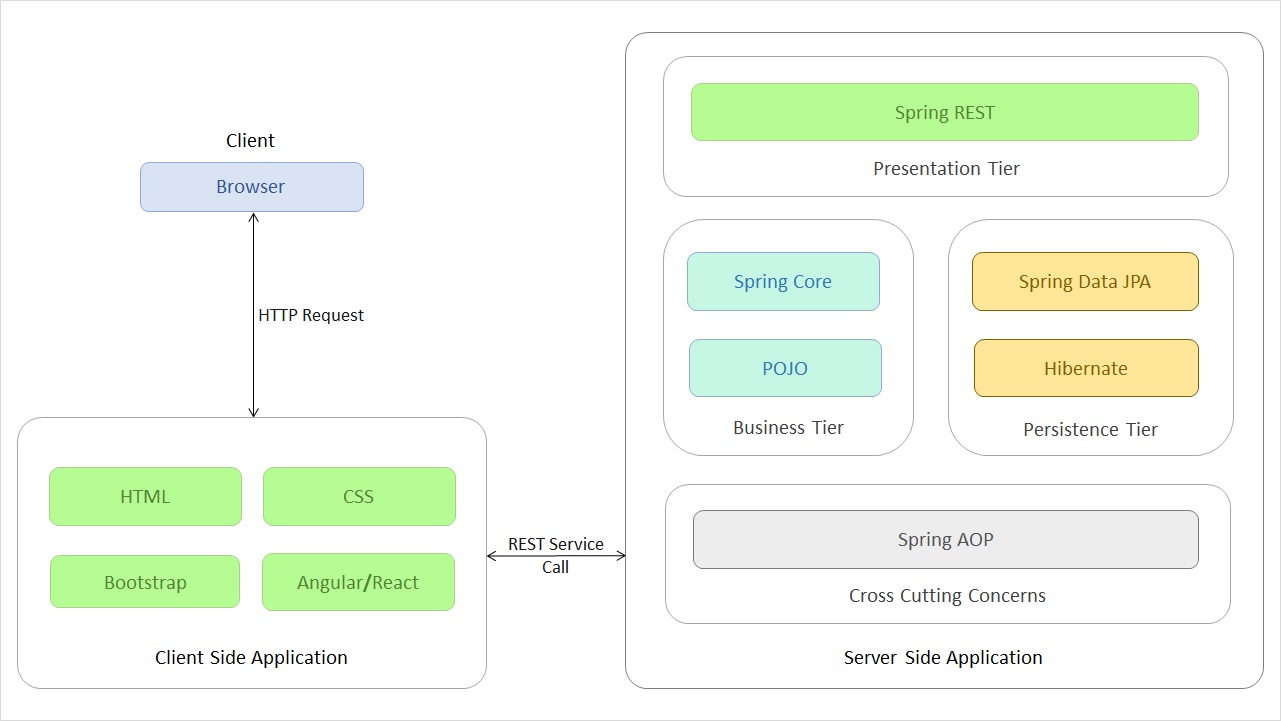
**Presentation-Tier:**  
This tier links the business logic of the application and the end user. It also takes care of displaying dynamic content to the end-user and accepting various inputs from the user.

**Business-Tier:**  
This is the most vital part of the application. This tier represents the complete business logic and requirements of the application. All the main functionalities, like data validations and data manipulations lie in this tier of the application.

**Persistence-Tier:**  
This acts as a go-between for the business tier and the database. It gets the requirements from the business tier, and based on it accesses data from the database.

Let us see which technology we can use to set up an enterprise application.

You have seen the different tiers that an enterprise application is split into. Will Java be used in all the tiers? Or are there any other technologies used? Which technologies can one use in the multiple tiers created?  
There are many technologies that can be used in different tiers, but we will be working with a specific set in this training.



As you can see above, for different tiers, we are using different technologies. In this course, we will focus on the POJO(Plain Old Java Object) part of the Business Tier. POJO is a Java object which has not been bounded to any framework.

The other technologies and tiers will also be discussed in this training.

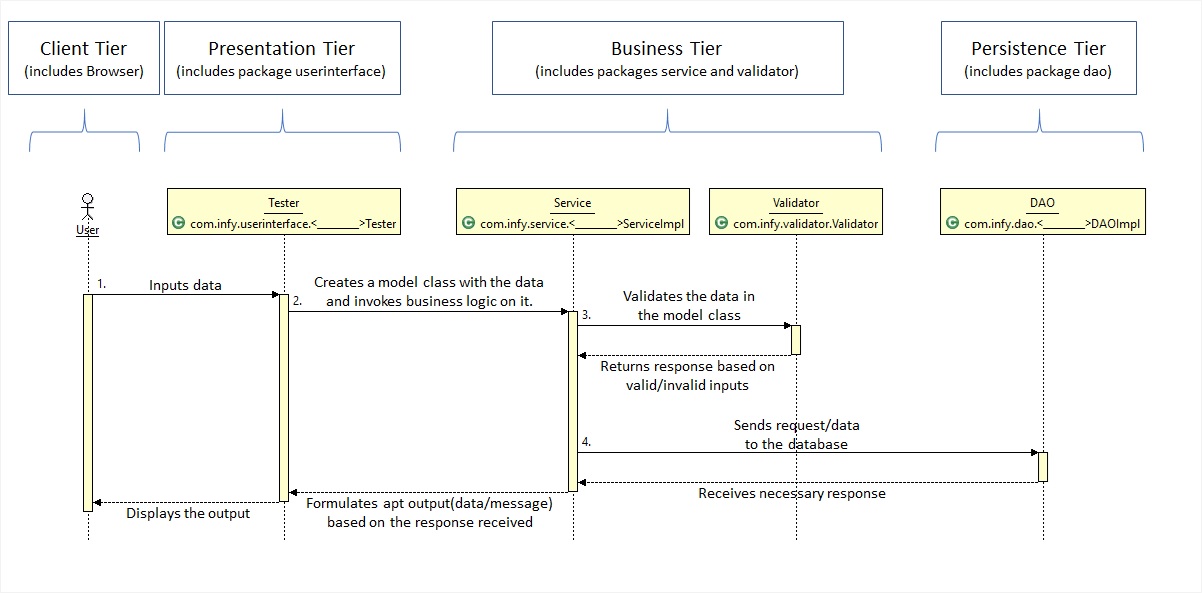
This course talks about how to create a Java application. Let us see how we split the Java application into the four tiers and how the data flows in the project.

As you saw in the previous page, an enterprise application is divided in four tiers (Client, Presentation, Business and Persistence) with each having a specific functionality. But how do we split up our project to represent these tiers? Which aspect of our project goes to which tier?

For depicting this distinction, we have four different groups,

* **User Interface** – This is the group that belongs to the Presentation Tier. This group contains the userinterface package with a class called Tester. It takes care of processing the inputs from the user and relaying them to the concerned functionalities. For now, we are using a normal Java class. But in future we will use an HTML file to do so.
* **Service**– This is the group that belongs to the Business Tier. This group contains the service package with many Service interfaces and classes. It takes care of authenticating the user inputs, as well as performing all the business logics needed for the application.
* **Validator** – This along with the Service group belongs to the Business Tier. This group contains the validator package with different Validator classes. This class receives user input to be validated for proper format and data.
* **Data Access Object(DAO)** – This is the group that belongs to the Persistence Tier. This group contains the dao package with many DAO interfaces and classes. It helps the application in interacting with the Database. In this course, we will be mocking the database. In the future courses, we will use the actual database.
* **Model**– This group contains classes which are used to transfer data from one class to the other.

That being said, how does the control flow through each of these tiers? To demonstrate that, please have a look at the below image.



The different steps taking place in the image shown are,

1. User inputs the data. This data is sent to the Tester class (or the Presentation Tier).
2. The Tester class converts the raw data to a proper Model class object (Objects used to transfer data from one class to another) and send the object to the Service Class (Business Tier).
3. The Service class sends the object to the Validator class to get the inputs validated. If the inputs are in valid format, the Validator returns a successful response, else a failure response.
4. The Service class performs any business operation to be performed on the data before sending it to the DAO class (Persistence Tier).
5. The Persistence Tier interacts with the database and returns the response of the database.
6. Based on the responses from the Validator and the DAO classes, the Service class formulates  either a successful output or a failure output and return the same to the Tester class.
7. The Tester class then displays this output to the User.

These steps are the fundamental procedures in any application. Different functionalities can have one or more of the above shown steps in a repeated pattern, but these steps will be there.

About Java:

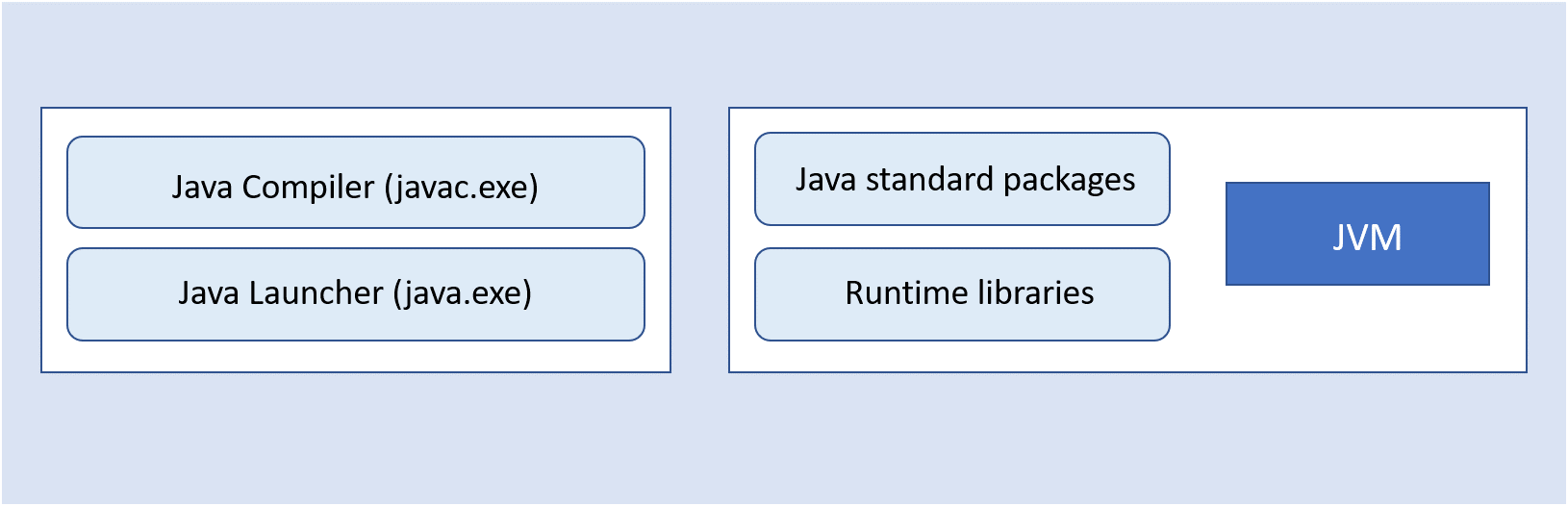
In this course, we will be learning Java, one of the most popular programming language which is used to build an application.

* Java has been evolving since 1991 with different editions. In this course, the features of Java till Java 11 has been discussed.
* According to the [TIOBE Programming Community Index](https://www.tiobe.com/tiobe-index/java/), Java has been one of the top 5 programming languages for several years.
* Billions of devices and enterprise desktops are powered by Java technology.

Let us look into the Java Architecture and Features next.

Java Architecture:

A Java program requires a Java development kit (JDK) for library support and development tools.



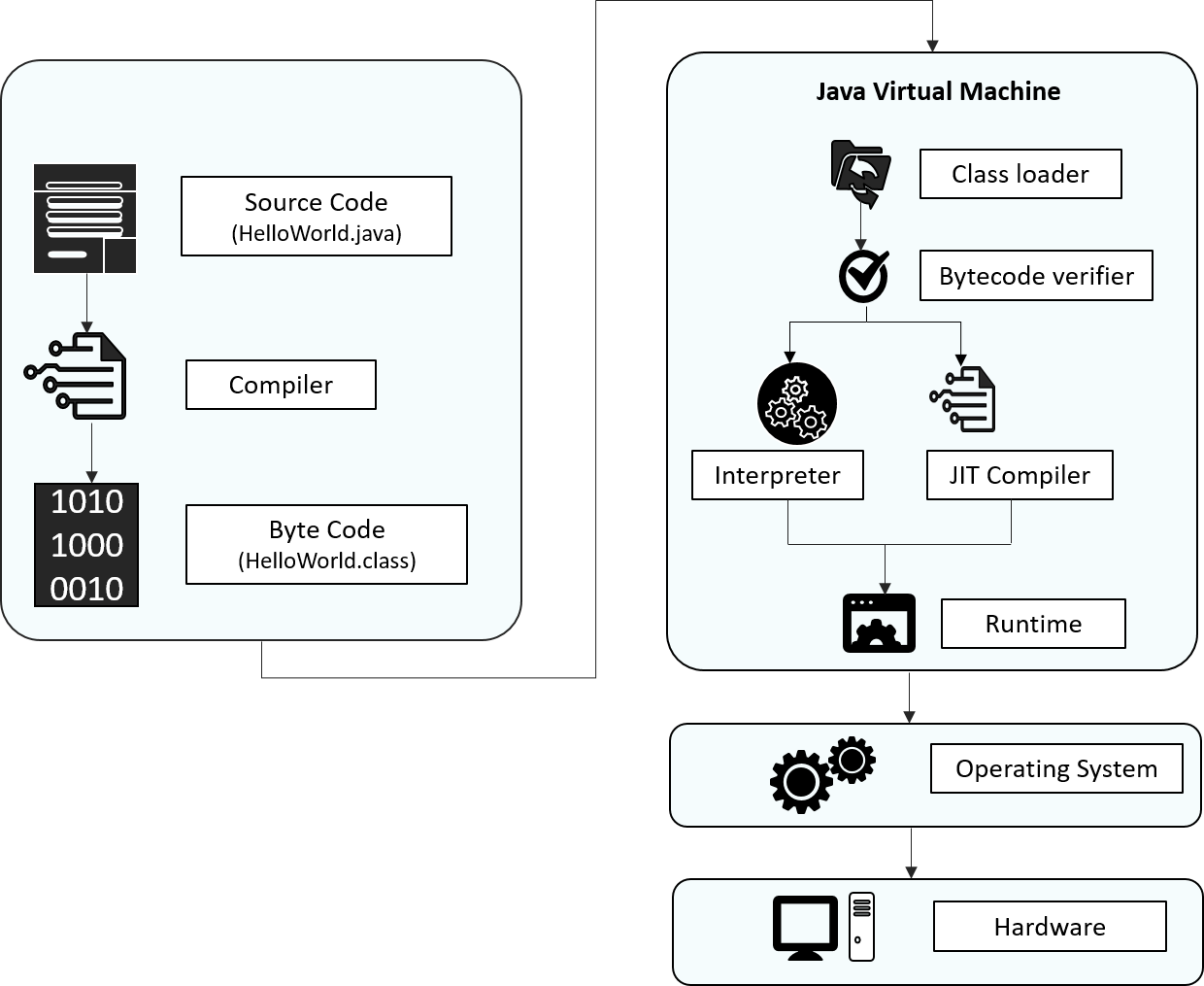
**Java compiler (javac.exe):**A Java compiler is a program that converts source files into the respective byte code. The byte code is platform-independent.

**Java launcher (java.exe):**Java launcher launches the Java application.

Java standard packages and runtime libraries contain the necessary code for executing Java applications.

**Note:** Previously, JDK came with a separate implementation for JRE (Java Runtime Environment). Now, starting from Java 11, Adopt OpenJDK does not come with separate JRE implementation; JDK will include JRE implementation.

Let us look into the internal work of the Java program, how it is compiled and executed as shown below:



**Source code:**Program written in Java language

**Byte code:** A .class file is generated after the Java code is compiled

**ClassLoader:**Loads all the class files needed for execution

**Byte code verifier:** Checks code for fragments for  illegal code

**Interpreter:**Converts byte code instruction to machine code

**Compiler:**Compiles reusable byte code instructions to machine code

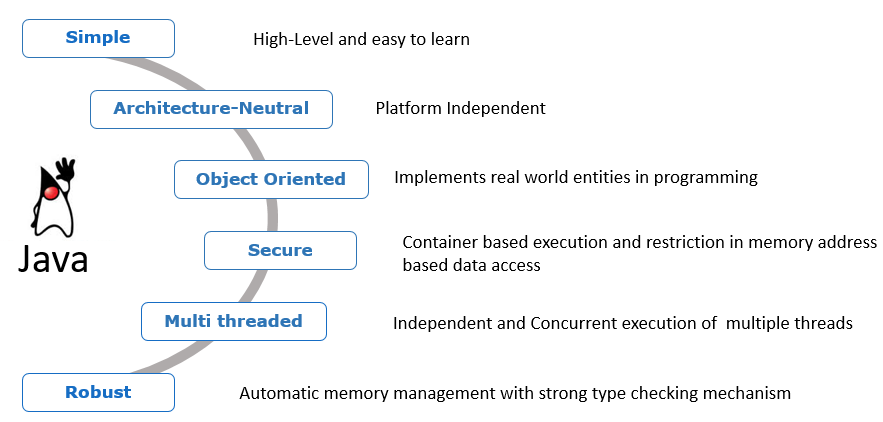
**Runtime:**The overall execution of the program is assisted by Runtime.

**Note:**Graal is a new JIT compiler written entirely in Java. Graal is a high-performance JIT compiler. Graal is independent of HotSpot VM and produces faster native code, which is used to get optimized performance.

Next, let us discuss the different features of Java.

Features Of Java:

There are many features of java which contribute to its unique, distinguished and super exciting nature:



**Simple:**

Previously there were many complicated and rarely used features, which have been now removed, thus making Java easy to learn and understand. Apart from that, because of automatic garbage collection in Java, we don't have to deallocate unreferenced objects.

**Architecture Neutral:**

The compiler first compiles the code written in java and then converts into a "bytecode" (machine-understandable language) file. This feature makes Java platform-independent, as the bytecode file can be run on any platform (Windows, Linux, Mac/OS, Sun Solaris etc.)

**Object-Oriented:**

Object-Oriented concepts like Object, Class, Inheritance, Polymorphism, Abstraction, and Encapsulation is supported by Java.

**Secure:**

Java programs run inside a virtual machine sandbox. This enables us to develop more secure systems using Java.

**Multi-threaded:**

A thread is a separate program that executes concurrently. We can break down a monolithic program handling multiple tasks, into smaller multiple thread-like Java sub-programs that can deal with all the tasks at a given point in time. The main advantage of multi-threading is that threads share a common memory area. Threads are most importantly used for multi-media, web applications, etc.

**Robust:**

Java is considered robust because

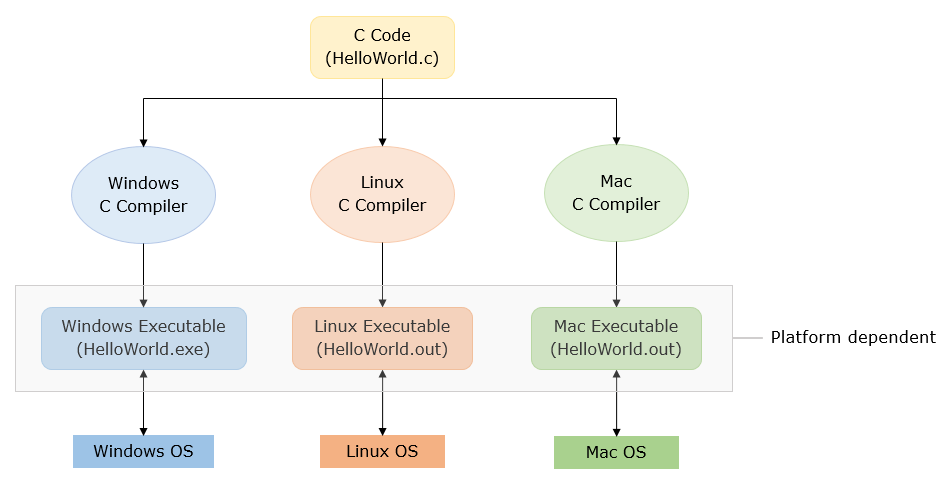
* there is strong memory management in Java
* Java doesn't use pointers which thus avoid security problems
* Java automatically garbage collects unused objects
* it supports exception handling and type checking mechanism

Let us understand the concept of platform independence next.

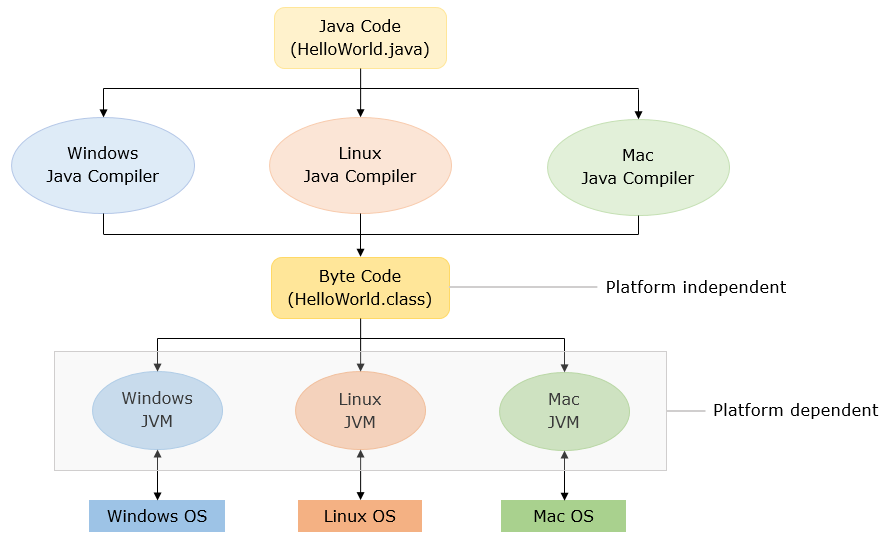
If a program written on a particular platform can run on other platforms without any recompilation, it is known as platform independence.

Since Java is platform-independent, any program written using Java on Windows will execute without any recompilation on any other platform.

To see what platform independence actually is and how it matters, let us compare Java to a platform-dependent language like C.



Java is platform-independent whereas JVM is platform dependent. Let us see how Java code executes in different platforms.



Now that we have discussed the features of Java, let us start coding in Java!

First Java Program:

Now that we have discussed a few features of Java, let us begin by coding a Java program that prints "Hello World!".

The Java code for printing "Hello World!" is,

1. public class DemoClass {
3. public static void main(String[] args) {
4. System.out.println("Hello World!");
5. }
7. }

**Note:** "main()"  is the starting point of execution of all the programs.

The first thing that might come to your mind seeing this code might be, what are all those terms? I just started Java, and have no idea about class, static, public, etc. Those terms that you saw are the basic constructs in the Java language. Without them, Java does not work.

But you wanted to just print a normal "Hello World" and you just started learning Java. Would you have to learn all that to just print "Hello World"? Yes, you have to. To print a normal "Hello World", you do have to write a huge code as shown. But is that the only way to learn Java programming?

No, it isn’t. Due to a new addition in the Java language (from Java 9), learning of basics is simplified. That addition is the JShell.

Let us now see what's JShell.

**What is JShell?**

JShell is a command-line tool which can execute Java code snippets, and it does not require to write a complete Java program. It is Java REPL (Read-Evaluate-Print-Loop) like other languages such as Scala, Groovy, etc. With JShell, not only will the beginners find it easy to start learning the Java programming language, but the developers will also find it easy to test code snippets and incrementally build prototypes. JShell helps us try out code on different options, to test individual statements, and to explore different APIs. You can write the program elements one by one and see the results immediately.

**How JShell works?**

If you are wondering how snippets in JShell can be executed without having a class or a method, JShell uses the following steps to execute the code:

1. Code is parsed and its type is determined (variable declaration, expressions, method, class, etc.)  
2. The code is wrapped inside synthetic methods and classes. Import statements are used as is  
3. The wrapped source code is analyzed and compiled by the regular Java compiler  
4. The generated bytecode is executed in a JVM, and the result is displayed by the JShell tool

Next, we will be using JShell to learn the fundamentals of Java.

How to work in JShell?

Now let us code the Hello World program using JShell.To execute the code in JShell, JShell tool needs to be launched in command prompt as shown below in steps

**Step-1:** Starting.JShell  
To start JShell, enter the JShell command, JShell, in the terminal of IDE or command prompt. The following example shows the command and the response from JShell:

1. D:\Java11>jshell
2. |  Welcome to JShell -- Version 11.0.1
3. |  For an introduction type: /help intro
4. jshell>

JShell can also be launched with --start DEFAULT and --start PRINTING options which provide several library files made available by default.

PRINTING option to make the print methods available as top-level methods, allowing us to use methods like print(), println(), printf() directly on JShell. But unlike DEFAULT this will not add any frequently used libraries (for example Data Structure Libraries like List).

Using **--start DEFAULT**option

1. D:\Java11>jshell --start DEFAULT
2. |  Welcome to JShell -- Version 11.0.1
3. |  For an introduction type: /help intro
4. jshell>

Using **--start PRINTING** option

1. D:\Java11>jshell  --start PRINTING
2. |  Welcome to JShell -- Version 11.0.1
3. |  For an introduction type: /help intro
4. jshell> println("Hello World!");
5. Hello World!

**Step 2:** After launching JShell write the following code on JShell command prompt and press enter to get the output:

1. jshell> System.out.println("Hello World!");
2. Hello World!
3. jshell>

Congratulations! Your first Java Program is executed using JShell.

**Note:**Every code written in JShell will have a specific id allotted to it. Using /list command, you can get a list of code snippets executed in JShell. As you can see below, we have executed only the print statement, and it has its id.

1. jshell> /list
2. 1 : System.out.println("Hello World!");
3. jshell>

**Step 3:** Closing JShell

We can exit from the JShell tool using the /exit command which will return you to the command prompt.

1. jshell> /exit
2. | Goodbye
3. D:\Java11>

**Exceptions in JShell**

In JShell, there is no need to handle checked exceptions. If a snippet throws a checked exception, the stack trace will be shown and JShell will continue. Though if a snippet throwing a checked exception is used inside a method, it will have to be handled or declared to be thrown.

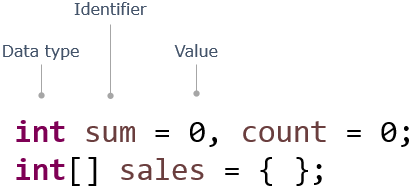
**Note:**Checked Exceptions are the exception encountered during compile-time. We will discuss them further in the course.

For further reading on JShell, you can refer to the Oracle Docs by clicking [Java Shell User’s Guide](https://docs.oracle.com/javase/10/jshell/commands.htm#JSHEL-GUID-34165A38-E6D0-459D-9947-4DEA9845A71A).

Let us see the Java Installations next.

Keywords And Datatypes:

In programs, data values are stored in memory locations identified by names (identifiers). Such named memory locations are called *variables*.  
Notice how variables are declared using types, identifiers, and assigned values:



As you have seen, a program is composed of several components, blocks, and words.

Some of these words are reserved and have a special meaning in Java, e.g. class, public, void, for, int, static. These words are called *keywords*. There are 50 keywords in Java 8.

Apart from keywords, there are other words that are used as names to identify components in a program, e.g. sum, index. These words are called *Identifiers*.

Identifiers are names used for identifying components of a program like classes, methods, interfaces, enums, and variables. Once declared, these names can be used to identify those components later in the program.

Now use JShell to execute the below code. Copy and paste the below code in JShell command prompt

1. int pizzaPrice = 250, totalCost = 0;
2. int quantity = 3;
3. System.out.println("TotalCost:" + pizzaPrice\*quantity);

**Rules for naming identifiers**:

1. case sensitive
2. should not start with a number
3. should start with a letter, $ or \_
4. should not have spaces
5. should not be a Java keyword or a literal
6. no restriction on the length

Note: The identifier should not have the **only \_** in its name.

**Example :**

**Valid Identifiers:** CustomerName, grade\_in\_first\_attempt, marksScoredIn3rdAttempt

**Invalid Identifiers:** model number, 1ofAKind, double

Next, let us see how we can declare a variable.

Now let us see how to declare variables in Java. While declaring variables, we have to specify the kind of data they will hold.

A data type defines the type of data a variable can hold, its memory needs, and the operations that can be performed on it.

A variable can hold values only of the type specified at the time of its declaration.

**Syntax:  <data type> <identifier> = <value>;**

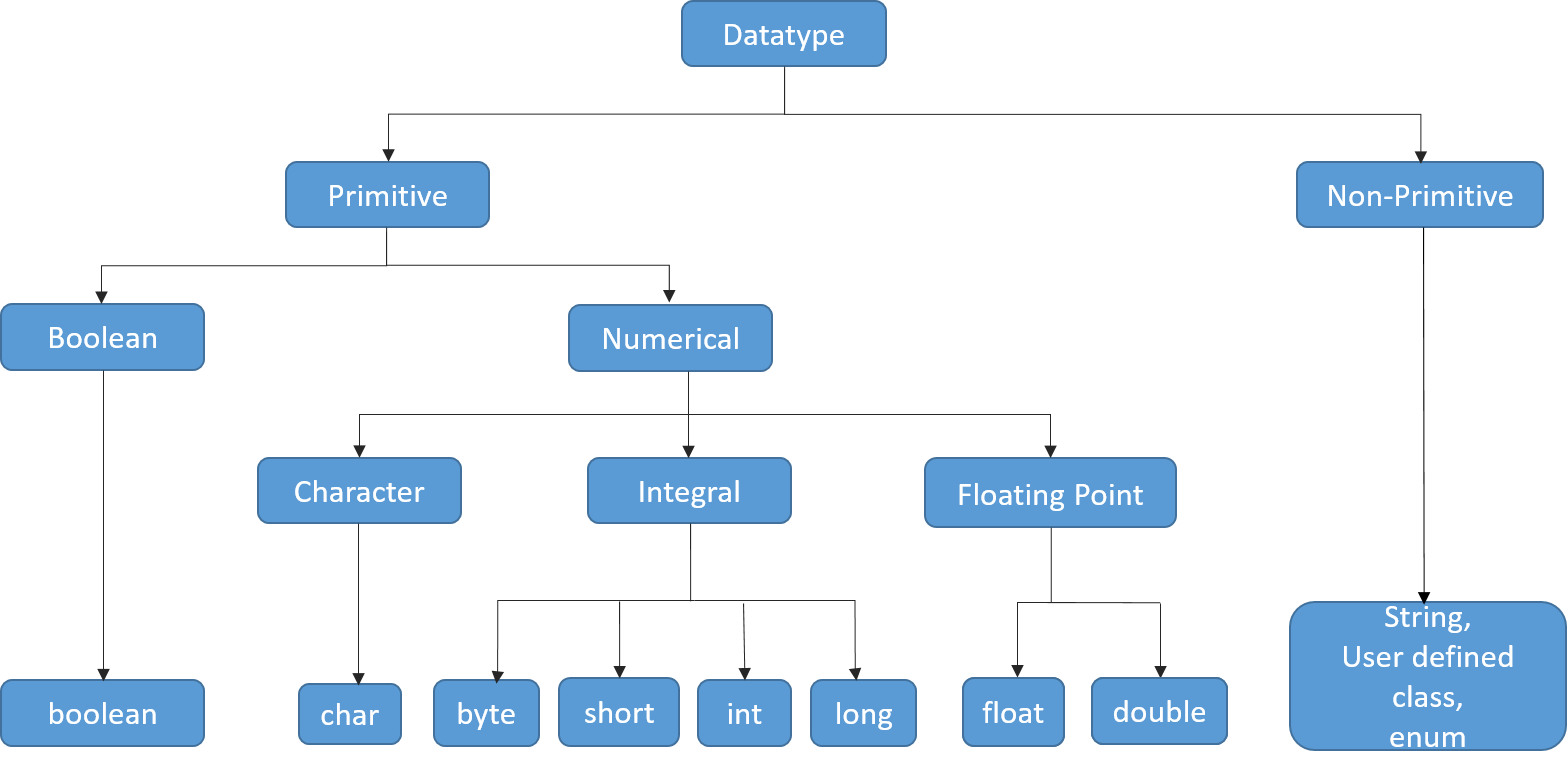
For Example:

1. long totalCost = 0;

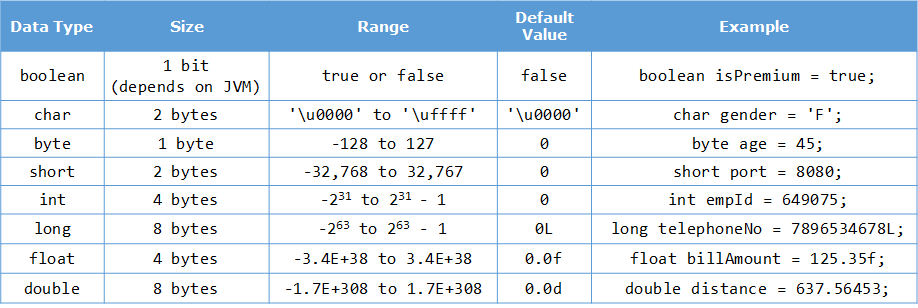
Data types are primarily of two kinds:

* **Primitive data types**: Variables hold the value of data item. char in Java uses Unicode characters
* **Non-Primitive/Reference data types**: Variables hold the reference of the memory location where the data item is stored.    Note: Non-primitive types such as String, user-defined types, enum will be discussed later in the course.

Let us have look at different data types in the below diagram in detail.



The default value and size occupied by each datatype with an example below.



Next, let us see how we can specify the datatype of a variable.

By now, you have seen that to declare a variable, you need to specify the data type of the variable but from Java 10 onwards, a new feature of local variable type inference has been introduced.

This feature allows you to specify **var**instead of the data type while declaring a variable as shown below.

1. var number=11;

In the above code snippet, you can notice that instead of a data type, var has been specified while declaring the variable.

In this case, the compiler infers the data type of the variable based on the value initialized to it.

Here, the compiler will infer the data type as int as an integer value has been initialized to the variable number.

Please note that var can only be used when a variable is initialized during declaration. var cannot be used when a variable is just declared without initialization.

The code snippet given below will result in a compilation error.

1. var number;

Once a variable declared with var has been initialized, you cannot even change the type of the value assigned to the variable.

1. var number=11; *//Line 1*
2. number="Java"; *//Line 2*

The above code snippet will result in a compilation error as in Line 2, the value 'Java' of type String is assigned to the variable number but initially the type inferred for the variable number was int. This is not allowed.

Similarly a var variable cannot be initialized to null, which will result in compilation error if done. This is because it cannot determine the type of the variable from the value null.

1. var number=null;

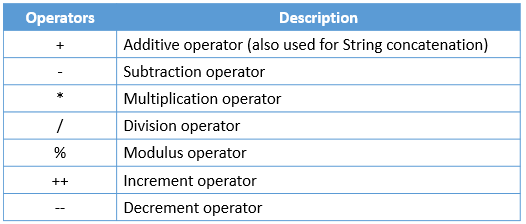
You can now try these out in tryouts given later.

Let us discuss about operators next.

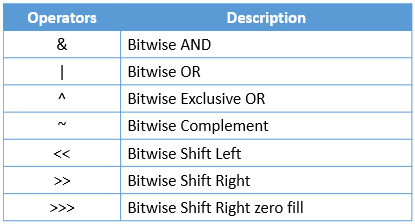
To manipulate variables and to do some operations on the data, we require operators.

Operators are symbols that perform specific operations on values and return a result. Java is loaded with a huge set of operators:

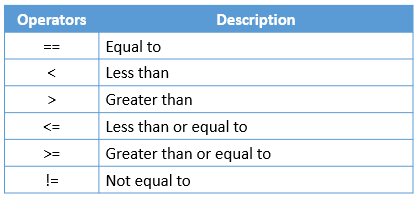
Arithmetic operators:



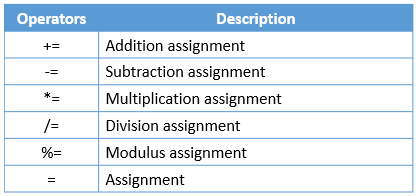
Bitwise operators:



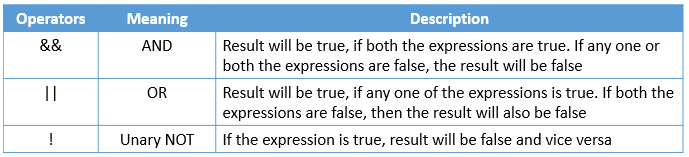
Relational operators:



Assignment operators:



Logical operators:



Ternary Operator: It is a short form of if-then-else statement

            Syntax:  <condition> ? <statement if true> : <statement if false>;

            For Example:

1. int a=30, b=50;
2. String result= a>b?"A is greater":"B is greater";

Use JShell to execute the below code

1. int pizzaPrice=280;
2. int quantity=2;
3. long totalCost = pizzaPrice \* quantity;
4. if(totalCost>2000 && totalCost<=10000){
5. float totalCostWithDiscount = totalCost - (totalCost \* (float)5/100);
6. }
7. else if(totalCost>10000){
8. float totalCostWithDiscount = totalCost - (totalCost \* (float)10/100);
9. }
10. else{
11. System.out.println("No Discount");
12. }

See more about operators in Java [here](https://lex.infosysapps.com/viewer/lex_1957914491484875953478).

Next, let us see how we can assign a value from one data type to another.

In Java, **type conversion** or **type casting**refers to the process of assigning a value from one data type to another.  In most cases, to prevent the loss of information, any of the operand's value can be converted to a higher accommodating type.

Java supports two types of type conversion:

* **Explicit type casting**

Explicit conversions are generally used to prevent data loss in mathematical operations. Also, when a value of larger data type needs to be stored as a value of smaller data type, despite the possibility of data loss, we should explicitly specify the conversion. In such a case, it is called narrowing conversion.

For Example:

1. float totalCostWithDiscount = totalCost - (totalCost \* (float)5/100); *// Explicit Type Casting*

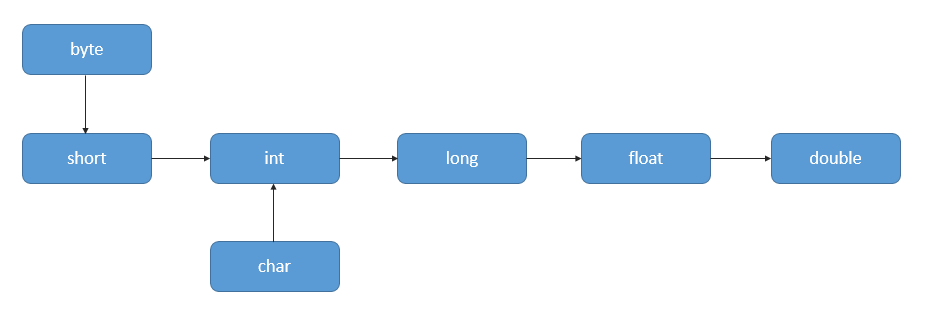
* **Implicit type casting**

Implicit conversion happens when a value of smaller data type needs to be used as a value of larger compatible data type.

For Example:

1. int intValue = 1002;
2. long longVariable = intValue;
3. *//Here type casting is automatically done*

  Also called widening conversion, it is done automatically by Java. It follows the sequence below:



Next, let us see few tryouts on datatypes, var and operators.

# **Data Types - Tryout**

Play around with different data types and get familiar with declaring and initializing variables.

Data types are primarily of two kinds:

* Primitive data types: Variables hold the value of data item. In Java char type uses Unicode characters
* Non-Primitive/Reference data types: Variables hold the reference of the memory location where the data item is stored. Example: String

Java has **var** keyword to declare variables, which allows you to declare a variable without their type.

Below code is a simple Java program to demonstrates different ways of declaring and initializing a variable in  Java

class DataTypesDemo{

public static void main(String[] args) {

int price = 100; //primitive data type

double discount = 25.5;

String size = "Medium"; //non-primitive data type

System.out.println("Prize after discount: "+price\*(1-discount/100));

//Variable declared using var keyword

var customerName = "Rob Jones"; // infers String

System.out.println( customerName +" ordered a "+size+" pizza");

}

}

# **var - Tryout**

Problem Statement

Try executing the code given below to understand local variable type inference. Uncomment the code and observe the result, to understand better.

class Tester{

public static void main(String[] args){

//Variable declared using var instead of data type

var numberOne=11; //The type of the variable is inferred by the compiler

System.out.println("The value of numberOne is "+numberOne);

//Uncomment the code given below and observe the output

//var numberTwo;

//Uncomment the code given below and observe the output

//numberOne="Java";

}

}