Unit –1: Introduction to Java Programming Language

1. Introduction to Java and Brief history, java features, java Applications

1.1 Introduction to Java and Brief history:-

- ❖ Java is a general purpose, class based, and robust, secure, safe, object oriented programming language.
- ❖ It was developed by sun micro system in 1991 by James Gosling its name was **Oak**. In 1995 its name has been changed to **Java** because of some legalissues.
- ❖ Java was designed for development of software for consumer electronic devices like TVs, VCRs, toaster etc.
- ❖ Java removes limitation like portability and reliability of C and C++.
- ❖ Java program would run in fundamentally different execution environments.
- ❖ Java is strongly typed language.
- ❖ Compile time contains translating programs into a machine independent byte code representation.
- ❖ Runtime activities include loading and linking of classes needed to execute a program, optional machine code generation, dynamic optimization of program.
- ❖ Most striking feature of java is platform′ independence. Java is not tied to any particular hardware or OS. Java programs can be executed anywhere on any system.

1.2 Java features

- ❖ There are various features described by sun micro system. Some of them are given here:

1. Compiled and interpreted
   - ❖ Computer language is either compiled or interpreted. Java combines both so it is two stage system First java compiler translates source code into byte code. Java interpreter generates machine code that can be directly executing by machine that is running the program.

2. Platform independent
   - ❖ Programs written by user are called source code and executable program is called object code. Object can be executed by computer and object is specific to a particular CPU. So it cannot be executed on different platform. Java removes this limitation. When java program is compiled java compiler produces on object file which contains bytecode. This byte codes are not machine/CPU specific, it can be interpreted by JVM (Java Virtual Machine).so that code can be run on any JVM of computer system. The Same byte codes can be executed by any JVM on′ any platform. Thus java is platform independent.

3. Portable
   - ❖ Java program can be easily moved from one computer system to another, anywhere and anytime. Changes /upgrade in OS .processors and system resources will not force any changes in java program.
   - ❖ We can also download applet from a remote computer onto our local system via internet and execute it locally.
   - ❖ Java provides Write Once Run Anywhere (WORA), which makes the java language portable provided that the system must have interpreter for the JVM . Java also has the standard data size irrespective of operating system or the processor .These features make the java as a portable language.

4. Object oriented

- ❖ Almost everything in java is an object. All program code and data reside within object and classes. Object model in java is simple and easy to extend. Java comes with extensive set of classes, arranged in packages that we can use in our program by inheritance. Java is true (pure) object oriented language because object is the outer most level of data structure in java. Everything in java is object even the primitive data′ types can also be converted into object by using the wrapper class.

5. Robust and secure

- ❖ It provides safe guards to ensure reliable code. Java has strict compile time and runtime checking for data types. Java also incorporates concept of exception handling, which captures series errors and eliminate any risk crashing system. So, java is robust. Java system verify not only all memory access but also ensures no viruses are communicated with applet. Absence of pointer in java ensures that program cannot gain access to memory location without proper authorization.

6. Distributed

- ❖ Java is designed as distributed language for creating applications on networks. Java applications can open and access remote objects′ on internet .These multiple programmers at multiple remote locations to collaborate and work together on a single project.

7. Simple

- ❖ Java is small and simple. Many features of C and C++ those are redundant or sources of unreliable code are not part of java. It doesn't support pointer, pre-processor , header′ files, goto statement and other It eliminates operator overloads, multiple′ inheritance, thus java is simplified version of C++. It includes automatic storage management,′ typically using a garbage collector, to avoid safety problems of explicit de-allocation of memory references.

8. Multithreaded and interactive

- ❖ Java is multithreaded programming language. Multithreading means a single program having different threads executing independently at the same time. Multiple threads execute instructions according to the program code in process or a program. Java supports multithreaded programs. This means we need not wait for the application to finish one task before beginning other. Java runtime comes with tools that support microprocessor synchronization and construct smoothly running interactive system.

9. High performance

- ❖ High performance ,by intermediate byte code Multithreading enhances the overall execution′ speed of java program

10. Dynamic and extensible

- ❖ Java is dynamic language It is capable of dynamically linking in new class libraries, method and object. Java program support function written in other language such as C and C++.This function known as native methods. Native methods are linked dynamically at′ runtime

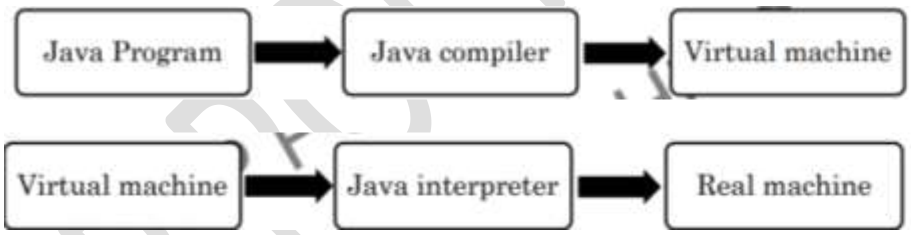
1.3 Java Applications

- Java program may contain many classes of which only one class defines a main method.
- Class contain data members and methods that operate on data members of class.
- Method may contain data type declaration and executable statements.java program define classes and put them together.
- the applications of Java are:
  1. Mobile Applications
  2. Desktop GUI Applications
  3. Web-based Applications
  4. Enterprise Applications
  5. Scientific Applications

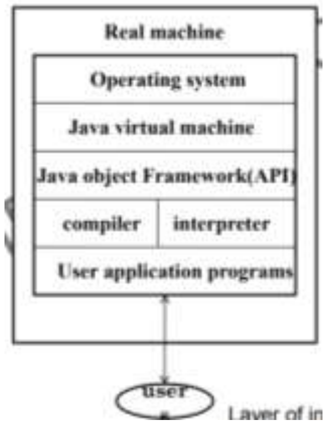
2. Java components: Java Virtual Machine (JVM), Java Runtime Environment (JRE), JDK (Java Development Kit). Importance of byte code and Garbage Collection

2.1 Java Virtual Machine (JVM)

- All language compilers translates source code into machine code for a specific computer. Java is platform independent or machine neutral!!! Java compiler produces an intermediate code (byte′ code or virtual machine code) for a machine that does not exist. that machine is Java Virtual Machine(JVM). JVM exists only inside the computer memory. JVM is a simulated computer within the computer and′ does all major functions of real computer.



- Byte code is not machine specific.
- Machine code is generated by the java interpreter by′ acting as intermediately between the virtual machine and real machine.

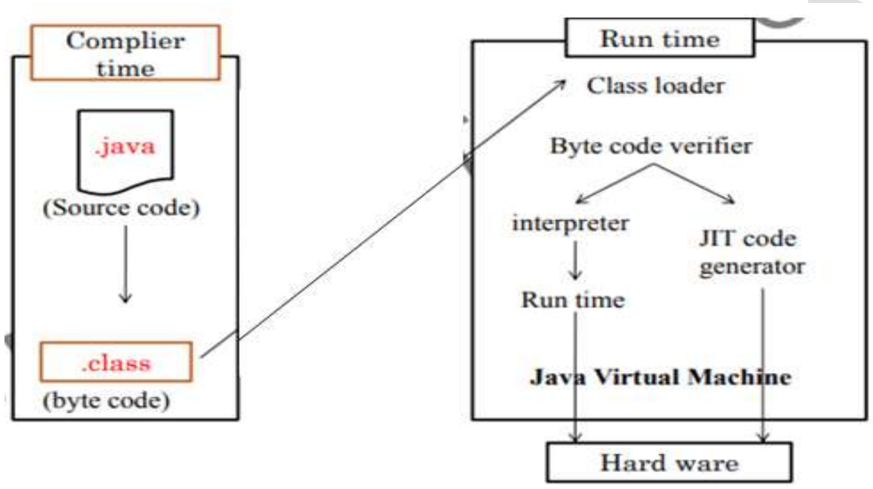


❖ JIT (JUST IN TIME COMPILER) JVM includes optional JIT. JIT dynamically compiles byte code into executable code. It takes byte code ,compiles them into machine code(native code) for machine. It compiles program on a method by method basis just′ before they are called. JIT is faster than JVM.

2.2 JRE (JAVA RUNTIME ENVIRONMENT)

❖ JRE consists of JVM and the program on the other . It runs code compiled for JVM by:

1. Loading the .class files
2. Verifying byte code
3. Executing the code



2.3 JDK (JAVA DEVELOPMENT TOOLKIT)

❖ It is a collection of tools that are used for developing′ and running java program. JDK includes′ applet viewer (for viewing java applets)

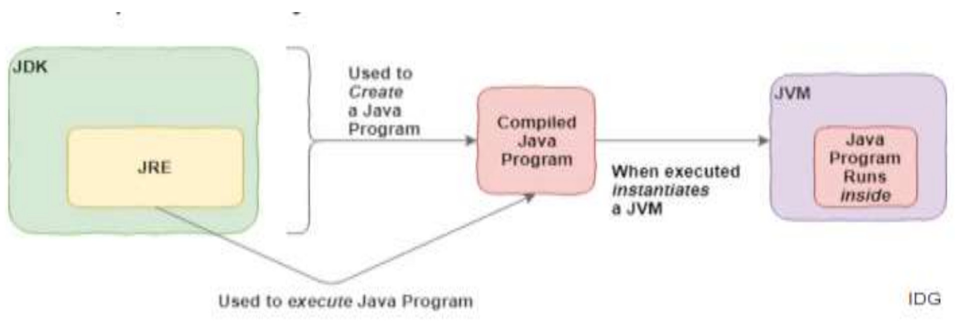
1. javac (java compiler)
2. java (java interpreter)
3. javap (java disassembler)
4. javah (for C header files)
5. javadoc (for creating HTML documents)
6. jdb (java debugger)

❖ The Java Development Kit (JDK) is one of three core technology packages used in Java programming, along with the JVM (Java Virtual Machine) and the JRE (Java Runtime Environment). It's important to differentiate between these three technologies and understand how they're connected:

❖ The JVM is the runtime that hosts running programs.

❖ The JRE is the on-disk part of Java that creates the JVM and loads programs into them.

❖ The JDK provides the tools necessary to write Java programs that can be executed and run by the JVM and JRE.

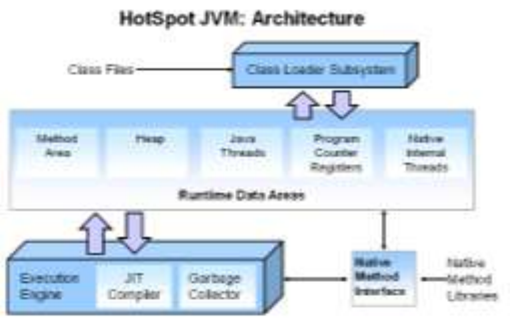

JDK
JRE
Used to Create a Java Program
Compiled Java Program
When executed instantiates a JVM
JVM
Java Program Runs inside
Used to execute Java Program
IDG

Importance of byte code:

- ❖ A java program is compiled, java bytecode is generated. In more apt terms, java bytecode is the machine code in the form of a .class file. With the help of java bytecode we achieve platform independence in java.
- ❖ Importance of byte code:
  - ❖ Bytecode is a highly optimized set of instructions that is executed by the Java Virtual Machine.
  - ❖ Bytecode helps Java achieve both portability and security.
  - ❖ Bytecode is essentially the machine level language which runs on the Java Virtual Machine.

Garbage Collection:

- ❖ Garbage collection in Java is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.



- ❖ How Does Garbage Collection in Java works?
- ❖ Java garbage collection is an automatic process. Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. An in-use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused or unreferenced object is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed. The programmer does not need to mark objects to be deleted explicitly. The garbage collection implementation lives in the JVM.
- ❖ Advantages of Garbage Collection in Java
  1. The advantages of Garbage Collection in Java are:

2. It makes java memory-efficient because the garbage collector removes the unreferenced objects from heap memory.
3. It is automatically done by the garbage collector(a part of JVM), so we don't need extra effort.

3. Java environment setup; Structure of java program; Compilation and execution of java program, , Comment Syntax

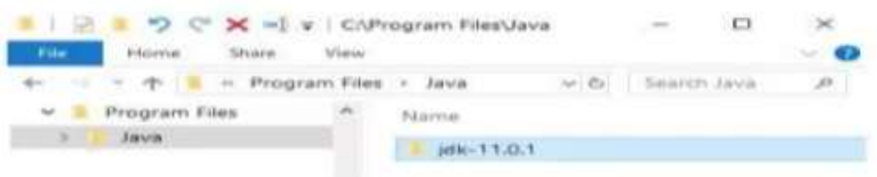
Java environment setup:

Step 1: First uninstall the older version of JDK/JRE

- We recommend that only the latest JDK be installed. Although it is possible to install multiple versions of JDK / JRE simultaneously, it is messy.
- If you have an older version(s) of JDK/JRE in your system, uninstall ALL of them. Go to control panel and uninstall java.

Step 2: Now download latest JDK

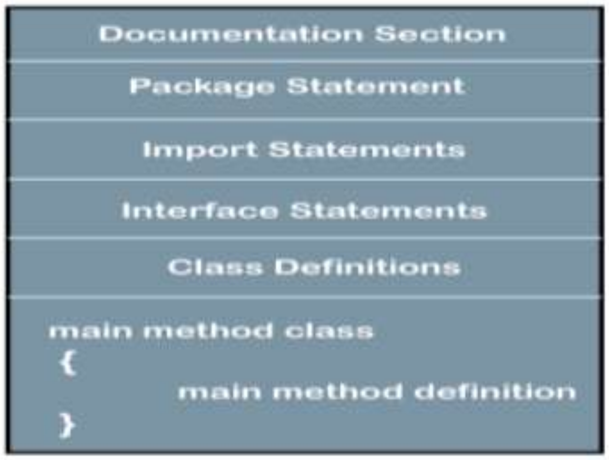
- Visit https://www.oracle.com/java/technologies/javase-downloads.html
- Click on download button for JDK version you want to install. Ex. JDK 11.
- Click on the checkbox to "Accept License Agreement".
- Choose the JDK for your operating system, i.e., "Windows". Download the ".exe" installer.
- Execute the downloaded installer which will install both the JDK and JRE.
- In the "C:\Program Files\Java\jdk-11.0.{v }" directory, where {v} denotes the update number, JDK is installed by default. Accept the defaults and follow the JDK installation instructions on the screen.



- Step 3: Set path for JDK
- Path and Classpath are variables of the environment at the operating system level. Path is used to define where the executables(.exe) files can be found by the system and classpath is used to specify the files.class location. In your java program, the path is set to use java tool such as java, javac, javap. Javac is used for code compilation.
- Open the System Properties.
- In the Properties Window, find the Advanced Tab. Click Environment Variables.
- Scroll down the variables of the system and find the variable PATH. Select the variable PATH and click Edit.
- Add the Java installation path to the PATH variable and save it.
- Step 4: Check if JDK installed correctly
- Go to Control Panel-->Program and Features and check if Java /JDK is listed there.

- ❖ Open command prompt and type java -version. If you get the version info, Java is installed correctly and PATH is also set correctly.
- ❖ Go to start menu-->System-->Advanced-->Environment Variables. Check if both PATH and JAVA_HOME are set correctly.

Structure of java program:



Structure of Java Program

- ❖ A typical structure of a Java program contains the following elements:
  1. Documentation Section
  2. Package Declaration
  3. Import Statements
  4. Interface Section
  5. Class Definition
  6. Main Method Class
- ❖ Documentation Section:
- ❖ The documentation section is an important section but optional for a Java program. It includes basic information about a Java program. The information includes the author's name, date of creation, version, program name, company name, and description of the program. It improves the readability of the program. Whatever we write in the documentation section, the Java compiler ignores the statements during the execution of the program. To write the statements in the documentation section, we use comments. The comments may be single-line, multi-line, and documentation comments.
- ❖ Package Declaration:
- ❖ The package declaration is optional. It is placed just after the documentation section. In this section, we declare the package name in which the class is placed. Note that there can be only one package statement in a Java program. It must be defined before any class and interface declaration. It is necessary because a Java class can be placed in different packages and directories based on the module they are used. For all these classes package belongs to a single parent directory. We use the keyword package to declare the package name.

- Import Statements:
- The package contains the many predefined classes and interfaces. If we want to use any class of a particular package, we need to import that class. The import statement represents the class stored in the other package. We use the import keyword to import the class. It is written before the class declaration and after the package statement. We use the import statement in two ways, either import a specific class or import all classes of a particular package. In a Java program, we can use multiple import statements
- Interface Section:
- It is an optional section. We can create an interface in this section if required. We use the interface keyword to create an interface. An interface is a slightly different from the class. It contains only constants and method declarations. Another difference is that it cannot be instantiated. We can use interface in classes by using the implements keyword. An interface can also be used with other interfaces by using the extends keyword.
- Class Definition:
- In this section, we define the class. It is **vital** part of a Java program. Without the class, we cannot create any Java program. A Java program may conation more than one class definition. We use the **class** keyword to define the class. The class is a blueprint of a Java program. It contains information about user-defined methods, variables, and constants. Every Java program has at least one class that contains the main() method.
- Main Method Class:
- In this section, we define the **main() method.** It is essential for all Java programs. Because the execution of all Java programs starts from the main() method. In other words, it is an entry point of the class. It must be inside the class. Inside the main method, we create objects and call the methods. We use the following statement to define the main() method:
- For example:

```
public class Student //class definition
{
    public static void main(String args[])
    {
            //statements
    }
}
```

Compilation and execution of java program:

- Type 'javac MyFirstJavaProgram.java' and press enter to compile your code. If there are no errors in your code, the command prompt will take you to the next line (Assumption: The path variable is set).
- Now, type ' java MyFirstJavaProgram ' to run your program.

```
C:\> javac MyFirstJavaProgram.java
C:\> java MyFirstJavaProgram
Hello World
```

Comment Syntax:

- we use comments. The comments may be single-line, multi-line, and documentation comments.

1. Single-line Comment: It starts with a pair of forwarding slash (//). For example:

- //First Java Program

2. Multi-line Comment: It starts with a /* and ends with */. We write between these two symbols. For example:

- /*It is an example of

3. multiline comment*/

- Documentation Comment: It starts with the delimiter (/**) and ends with */. For example:
- /**It is an example of documentation comment*/

4. Primitive Data Types : byte, short, int, long, float, double, char, Boolean

- A primitive data type specifies the size and type of variable values, and it has no additional methods.
- There are eight primitive data types in Java:

1. byte type

- The byte data type can have values from **-128** to **127** (8-bit signed two's complement integer).
- If it's certain that the value of a variable will be within -128 to 127, then it is used instead of int to save memory.
- Default value: 0

2. short type

- The short data type in Java can have values from **-32768** to **32767** (16-bit signed two's complement integer).
- If it's certain that the value of a variable will be within -32768 and 32767, then it is used instead of other integer data types (int, long).
- Default value: 0

3. int type

- The int data type can have values from **-231** to **231-1** (32-bit signed two's complement integer).
- If you are using Java 8 or later, you can use an unsigned 32-bit integer. This will have a minimum value of 0 and a maximum value of 232-1. To learn more, visit How to use the unsigned integer in java 8?
- Default value: 0

4. long type

- The long data type can have values from **-263** to **263-1** (64-bit signed two's complement integer).
- If you are using Java 8 or later, you can use an unsigned 64-bit integer with a minimum value of **0** and a maximum value of **264-1**.
- Default value: 0

5. double type

- The double data type is a double-precision 64-bit floating-point.
- It should never be used for precise values such as currency.

- Default value: 0.0 (0.0d)

6. float type

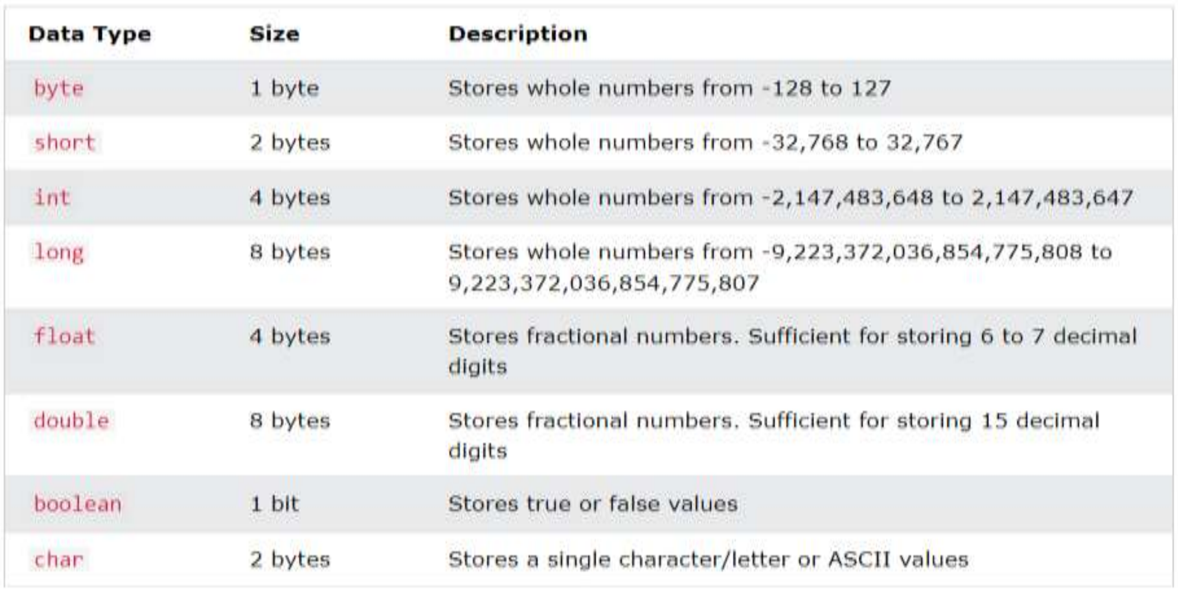
- The float data type is a single-precision 32-bit floating-point. Learn more about single-precision and double-precision floating-point if you are interested.
- It should never be used for precise values such as currency.
- Default value: 0.0 (0.0f)

7. boolean type

- The boolean data type has two possible values, either true or false.
- Default value: false.
- They are usually used for **true/false** conditions.

8. char type

- The char data type use to store character & string value.
- Default value: null



Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

5. Identifiers, Declarations of constants & variables, Type Conversion and Type Casting, Scope of variables

Identifiers:

- Identifiers in Java are symbolic names used for identification. They can be a class name, variable name, method name, package name, constant name, and more. However, In Java, There are some reserved words that can not be used as an identifier.
- Following are some rules and conventions for declaring identifiers:

- A valid identifier must have characters [A-Z] or [a-z] or numbers [0-9], and underscore(_) or a dollar sign ($). for example, @javatpoint is not a valid identifier because it contains a special character which is @.
- There should not be any space in an identifier. For example, java tpoint is an invalid identifier.
- An identifier should not contain a number at the starting. For example, 123javatpoint is an invalid identifier.
- An identifier should be of length 4-15 letters only. However, there is no limit on its length. But, it is good to follow the standard conventions.
- We can't use the Java reserved keywords as an identifier such as int, float, double, char, etc. For example, int double is an invalid identifier in Java.
- An identifier should not be any query language keywords such as SELECT, FROM, COUNT, DELETE, etc.
- Following are some examples of valid identifiers in Java:
  - TestVariable
  - testvariable
  - a
  - i
  - Test_Variable
  - _testvariable
  - $testvariable
  - sum_of_array
  - TESTVARIABLE
  - jtp123

Declarations of constants

- Constants in Java are used when a **‘**static**‘** value or a permanent value for a variable has to be implemented. Java doesn’t directly support constants. To make any variable a constant, we must use ‘static’ and ‘final’ modifiers in the following manner:
- **Syntax to assign a constant value in java:**
  static final datatype identifier_name = constant;

- The static modifier causes the variable to be available without an instance of it’s defining class being loaded
- The final modifier makes the variable unchangeable
- Example : static final int MIN_AGE = 18;

Variables:

- A variable is a location in memory (storage area) to hold data.
- To indicate the storage area, each variable should be given a unique name (identifier). Learn more about Java identifiers.

- Create Variables in Java
  datatype variablename;
- For example,
- int a;

Java Type Casting

- Type casting is when you assign a value of one primitive data type to another type.
- In Java, there are two types of casting:

- **Widening Casting** (automatically) - converting a smaller type to a larger type size
  byte -> short -> char -> int -> long -> float -> double

- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
  double -> float -> long -> int -> char -> short -> byte

1. Widening Casting

- Widening casting is done automatically when passing a smaller size type to a larger size type:
- Example:

```
public class Test
{
 public static void main(String[] args)
{
   int m = 9;
   double d = m; // Automatic casting: int to double

   System.out.println(m);     // Outputs 9
   System.out.println(d);   // Outputs 9.0
  }
}
```

2. Narrowing Casting

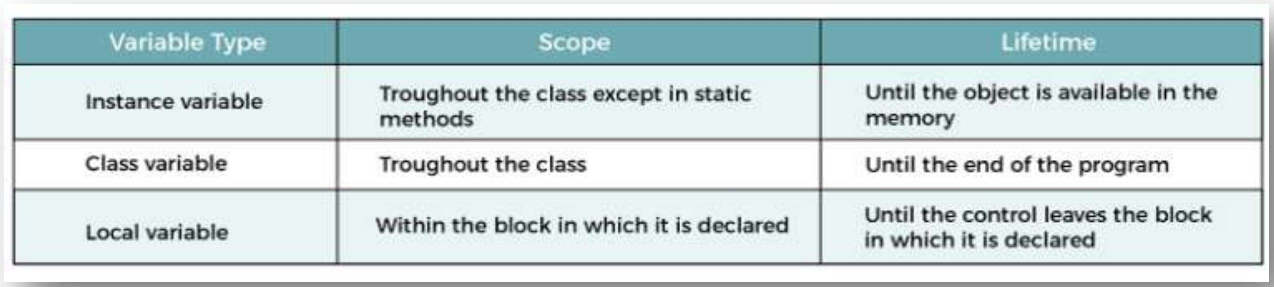
- Narrowing casting must be done manually by placing the type in parentheses in front of the value:
- Example

```
public class Test
 {
  public static void main(String[] args)
{
   double d = 9.78d;
   int m = (int) d; // Manual casting: double to int

   System.out.println(d);   // Outputs 9.78
   System.out.println(m);     // Outputs 9
  }
}
```

Scope of the variable

- ❖ In programming, a variable can be declared and defined inside a class, method, or block. It defines the scope of the variable i.e. the visibility or accessibility of a variable. Variable declared inside a block or method are not visible to outside. If we try to do so, we will get a compilation error. Note that the scope of a variable can be nested.
- ❖ We can declare variables anywhere in the program but it has limited scope.
- ❖ A variable can be a parameter of a method or constructor.
- ❖ A variable can be defined and declared inside the body of a method and constructor.
- ❖ It can also be defined inside blocks and loops.
- ❖ Variable declared inside main() function cannot be accessed outside the main() function



Variable Type	Scope	Lifetime
Instance variable	Troughout the class except in static methods	Until the object is available in the memory
Class variable	Troughout the class	Until the end of the program
Local variable	Within the block in which it is declared	Until the control leaves the block in which it is declared

6. Arrays of Primitive Data Types, Types of Arrays : one-dimensional and twodimensional array

Java Arrays

- ❖ Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- ❖ Type of Array
  1. One Dimensional Array
  2. MultiDimensional Array

1. One Dimensional Array

- ❖ To declare an array, define the variable type with **square brackets**:
- ❖ String[] cars;
- ❖ We have now declared a variable that holds an array of strings. To insert values to it, you can place the values in a comma-separated list, inside curly braces:
- ❖ String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
- ❖ To create an array of integers, you could write:
- ❖ int[] myNum = {10, 20, 30, 40};

- ❖ **Access the Elements of an Array**
- ❖ You can access an array element by referring to the index number.
- ❖ This statement accesses the value of the first element in cars:
- ❖ ExampleGet your own Java Server
  String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

```
System.out.println(cars[0]);
// Outputs Volvo
```

- ❖ **Array Length**
- ❖ To find out how many elements an array has, use the length property:
- ❖ Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length);
```

2. Multidimensional Arrays

- ❖ A multidimensional array is an array of arrays.
- ❖ Multidimensional arrays are useful when you want to store data as a tabular form, like a table with rows and columns.
- ❖ To create a two-dimensional array, add each array within its own set of **curly braces**:
- ❖ Example:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

myNumbers is now an array with two arrays as its elements.

- ❖ **Access Elements**
- ❖ To access the elements of the **myNumbers** array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of myNumbers:
- ❖ Example

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
System.out.println(myNumbers[1][2]);
```

7. Different Operators: Arithmetic, Bitwise, Rational, Logical, Assignment, Conditional, Ternary, Increment and Decrement

- ❖ Operators are symbols that perform operations on variables and values. For example, + is an operator used for addition, while * is also an operator used for multiplication.
- ❖ Operators in Java can be classified into 5 types:

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators
6. Conditional/Ternary Operators
7. Increment and Decrement

1. Java Arithmetic Operators

- ❖ Arithmetic operators are used to perform arithmetic operations on variables and data.
- ❖ For example, a+b;
- ❖ Here, the + operator is used to add two variables a and b. Similarly, there are various other arithmetic operators in Java.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

2. Java Assignment Operators

- ❖ Assignment operators are used in Java to assign values to variables.
- ❖ For example,
  int age;
  age = 5;
- ❖ Here, = is the assignment operator. It assigns the value on its right to the variable on its left. That is, **5** is assigned to the variable age.
- ❖ Let's see some more assignment operators available in Java.

Operator	Example	Equivalent to
=	a = b;	a = b;
+ =	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

3. Java Logical Operators

- ❖ Logical operators are used to check whether an expression is true or false. They are used in decision making.

Operator	Example	Meaning
&& (Logical AND)	expression1 && expression2	true only if both expression1 and expression2 are true
|| (Logical OR)	expression1 || expression2	true if either expression1 or expression2 is true
! (Logical NOT)	!expression	true if expression is false and vice versa

4. Java Relational Operators

- ❖ Relational operators are used to check the relationship between two operands.
- ❖ For example,

```
// check if a is less than b
a < b;
```

- ❖ Here, < operator is the relational operator. It checks if a is less than b or not.
- ❖ It true or false.

Operator	Description	Example
==	Is Equal To	3 == 5 returns false
!=	Not Equal To	3 != 5 returns true
>	Greater Than	3 > 5 returns false
<	Less Than	3 < 5 returns true
>=	Greater Than or Equal To	3 >= 5 returns false
<=	Less Than or Equal To	3 <= 5 returns true

5. Java Bitwise Operators

- Bitwise operators in Java are used to perform operations on individual bits.
- For example,
  Bitwise complement Operation of 35

```
    35  = 00100011 (In Binary)
    36  ~ 00100011
          ________
         11011100  = 220 (In decimal)
```

- Here, ~ is a bitwise operator. It inverts the value of each bit (0 to 1 and 1 to 0).
- The various bitwise operators present in Java are:

Operator	Description
~	Bitwise Complement
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR

6. Java Conditional/Ternary Operator

- Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.
- Java Ternary Operator Example

```
public class Example
{
public static void main(String args[])
{
    int a=2;
    int b=5;
    int min=(a<b)?a:b;
       System.out.println(min);
}
}
```

7. Increment and Decrement

- In programming (Java, C, C++, JavaScript etc.), the increment operator ++ increases the value of a variable by 1. Similarly, the decrement operator -- decreases the value of a variable by 1.
- a = 5
- ++a;   // a becomes 6
- a++;   // a becomes 7
- --a;   // a becomes 6
- a--;   // a becomes 5

8. Decision & Control Statements: Selection Statement (if, if...else, switch), Loops (while, do-while, for), Jump

❖ Java has the following Decision & Conditional statements:

1. Use if to specify a block of code to be executed, if a specified condition is true
2. Use else to specify a block of code to be executed, if the same condition is false
3. Use else if to specify a new condition to test, if the first condition is false
4. Use switch to specify many alternative blocks of code to be executed

1. The if Statement

- Use the if statement to specify a block of Java code to be executed if a condition is true.
- Syntax:

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

- Note that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.
- Example below, we test two values to find out if 20 is greater than 18. If the condition is true, print some text:

```
if (20 > 18) {
  System.out.println("20 is greater than 18");
}
```

2. The if..else Statement

- Use the else statement to specify a block of code to be executed if the condition is false.
- Syntax

```
if (condition)
{
  // block of code to be executed if the condition is true
}
 else
 {
  // block of code to be executed if the condition is false
}
```

- Example

```
int time = 20;
if (time < 18)
{
  System.out.println("Good day.");
}
else
{
  System.out.println("Good evening.");
}
```

3. The else if Statement

- Use the else if statement to specify a new condition if the first condition is false.
- Syntax

```
if (condition1)
{
  // block of code to be executed if condition1 is true
}
else if (condition2)
{
  // block of code to be executed if the condition1 is false and condition2 is true
}
else
{
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

- Example

```
int time = 22;
if (time < 10)
{
  System.out.println("Good morning.");
}
else if (time < 18)
 {
  System.out.println("Good day.");
}
else
{
  System.out.println("Good evening.");
}
```

4. Java Switch Statements

- ❖ Instead of writing **many** if..else statements, you can use the switch statement.
- ❖ The switch statement selects one of many code blocks to be executed:
- ❖ Syntax

```
switch(expression)
{
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
      // code block
}
```

- ❖ This is how it works:
- ❖ The switch expression is evaluated once.
- ❖ The value of the expression is compared with the values of each case.
- ❖ If there is a match, the associated block of code is executed.
- ❖ The break and default keywords are optional, and will be described later in this chapter
- ❖ The example below uses the weekday number to calculate the weekday name:
- ❖ Example

```
int day = 4;
switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
  case 4:
    System.out.println("Thursday");
    break;
  case 5:
    System.out.println("Friday");
    break;
  case 6:
    System.out.println("Saturday");
    break;
  case 7:
    System.out.println("Sunday");
    break;
}
```

Loops

1.While Loop

- Loops can execute a block of code as long as a specified condition is reached.
- Loops are handy because they save time, reduce errors, and they make code more readable.
- The while loop loops through a block of code as long as a specified condition is true:
- Syntax

```
while (condition)
{
  // code block to be executed
}
```

- In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5:
- Example

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
```

2. The Do/While Loop

- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- Syntax

```
do {
  // code block to be executed
}
while (condition);
```

- The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:
- Example

```
int i = 0;
do {
  System.out.println(i);
  i++;
}while (i < 5);
```

3. Java For Loop

- When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:
- Syntax

```
for (statement 1; statement 2; statement 3)
{
  // code block to be executed
}
```

- **Statement 1** is executed (one time) before the execution of the code block.
- **Statement 2** defines the condition for executing the code block.

- **Statement 3** is executed (every time) after the code block has been executed.
- The example below will print the numbers 0 to 4:
- Example

```
for (int i = 0; i < 5; i++)
 {
  System.out.println(i);
}
```

Jump Statement

1. **Java Break**

- You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.
- The break statement can also be used to jump out of a **loop**.
- This example stops the loop when i is equal to 4:
- Example

```
for (int i = 0; i < 10; i++) {
  if (i == 4)
{
    break;
  }
  System.out.println(i);
}
```

2. **Java Continue**

- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- This example skips the value of 4:
- Example

```
for (int i = 0; i < 10; i++)
{
  if (i == 4)
{
    continue;
  }
  System.out.println(i);
}
```