Advance Object Oriented Programming (4340701)

Semester – 4th, Diploma in Computer Engineering

Prepared by: N. R. TRIVEDI   Edited By: S. B Malli

Lecturer, Computer Engineering Department

Institute Vision and Mission

- **Vision:**
  - *To cater skilled engineers having potential to convert global challenges into opportunities through embedded values and quality technical education.*

- **Mission:**
  - *M1: Impart quality technical education and prepare diploma engineering professionals to meet the need of industries and society.*
  - *M2: Adopt latest tools and technologies for promoting systematic problem solving skills to promote innovation and entrepreneurship.*
  - *M3: Emphasize individual development of students by inculcating moral, ethical and life skills.*

Department Vision and Mission

- Vision:
  - *Develop globally competent Computer Engineering Professionals to achieve excellence in an environment conducive for technical knowledge, skills, moral values and ethical values with a focus to serve the society.*

- Mission:
  - *M1: To provide state of the art infrastructure and facilities for imparting quality education and computer engineering skills for societal benefit.*
  - *M2: Adopt industry oriented curriculum with an exposure to technologies for building systems & application in computer engineering.*
  - *M3:To provide quality technical professional as per the industry and societal needs, encourage entrepreneurship, nurture innovation and life skills in consonance with latest interdisciplinary trends.*

Examination Scheme

EXAMINATION SCHEME				
THEORY MARKS		PRACTICAL MARKS		TOTAL MARKS
ESE	CA	ESE	CA	
70	30 (20 Avg of Two Mid Sem + 10 Micro Project)	25	25	150

Course Outcomes

Advance Object Oriented Programming (4340701)	
CO1	Write simple java programs for a given problem statement.
CO2	Use object-oriented programming concepts to solve real world problems.
CO3	Develop an object-oriented program using inheritance and package concepts for a given problem statement.
CO4	Develop object-orientedted program using multithreading and exception handling for a given problem statement.
CO5	Develop an object-oriented program by using the files and collection framework.

Introduction to Java Programming Language

Unit- I

Topics to be Covered...

- Introduction & Brief history of Java
- Java Features and Applications
- Java Components (JVM, JRE, JDK) & Importance of Byte Code
- Garbage Collection
- Java Environment Setup
- Structure of Java Program
- Compilation and Execution of Java Program
- Comments in Java Program
- Data Types, Identifiers, Constants ,Variables & Scope of Variables
- Type Conversion and Type Casting
- Arrays in Java (1D and 2D)
- Operators in Java
- Decision & Control Statements

Introduction and Brief History of Java

- Java is a **Purely Object-Oriented High-Level** Programming language.

- **James Gosling, Mike Sheridan,& Patrick Naughton** (small team of sun engineers) initiated the Java language project in June 1991.

- Firstly, it was called **"Greentalk"** by James Gosling and file extension was **.gt**

- Then it was renamed to Oak.

- The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

- According to James Gosling "Java was one of the top choices". Since java was so unique, most of the team members preferred java.

Why We Use Java?

- Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)

- It is one of the most popular programming language in the world.

- It has a large demand in the current job market.

- It is easy to learn and simple to use.

- It is **open-source and free**.

- It is secure, fast and powerful.

- It has a huge community support (tens of millions of developers).

- Java is an object-oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs.

Applications of Java Programming Language

1. **Mobile App Development:**

    - The Java programming language can be considered as the official language for mobile application development.

    - Most of the android applications are built using Java.

    - The most popular android app development IDE **Android Studio** also uses Java for developing android applications.

    - e.g: Twitter, Tinder,Netflix,Uber, QRReader and Spotify.

2. **Desktop GUI Applications:**

    - We can develop a GUI application using Java.

    - Java provides AWT, JavaFX, and Swing for developing the GUI based desktop application.

    - The tools contain the pre-assembled components like list, menu, button.

    - e.g: *Acrobat Reader, ThinkFree, Media Player, Antiviruses, etc.*

Applications of Java Programming Language

3. **Web-based Applications:**

   - It is also used for developing the web-based application because it provides vast support for web development through Servlet, JSP, and Struts.

   - e.g: LinkedIn, AliExpress, web.archive.org, IRCTC, etc.

4. **Game Development:**

   - Java is widely used by game development companies because it has the support of the open-source most powerful 3D engine.

   - There are some popular Frameworks and Libraries available for Game Development, like - LibGDX and OpenGL.

   - e.g: Minecraft, Mission Impossible III

5. **Big Data Technology:**

   - The tool Hadoop HDFS platform for processing and storing big data applications is written in Java.

   - In big data, Java is widely used in ETL (Extract transform load) applications such as Apache Camel and Apache Kafka.

Java Features

- **Simple**
  - Easy to Learn and understand, No Use of
  - C Language like Syntax
- **Object-Oriented**
  - Supports all features of OOP.
  - Everything is an object in Java.
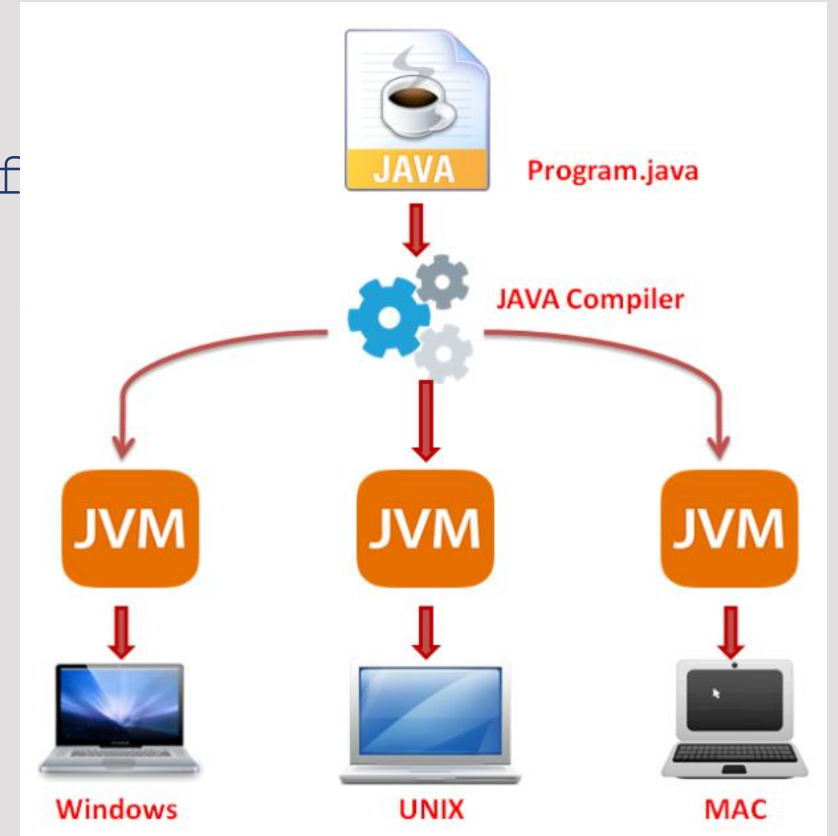- **Platform Independent**
  - Write and Compile once, Run Anywhere.
  - It can run on all available OS.
- **Multithreaded**
  - Java supports multiple threads to be running on a single program to execute more than one tasks concurrently.
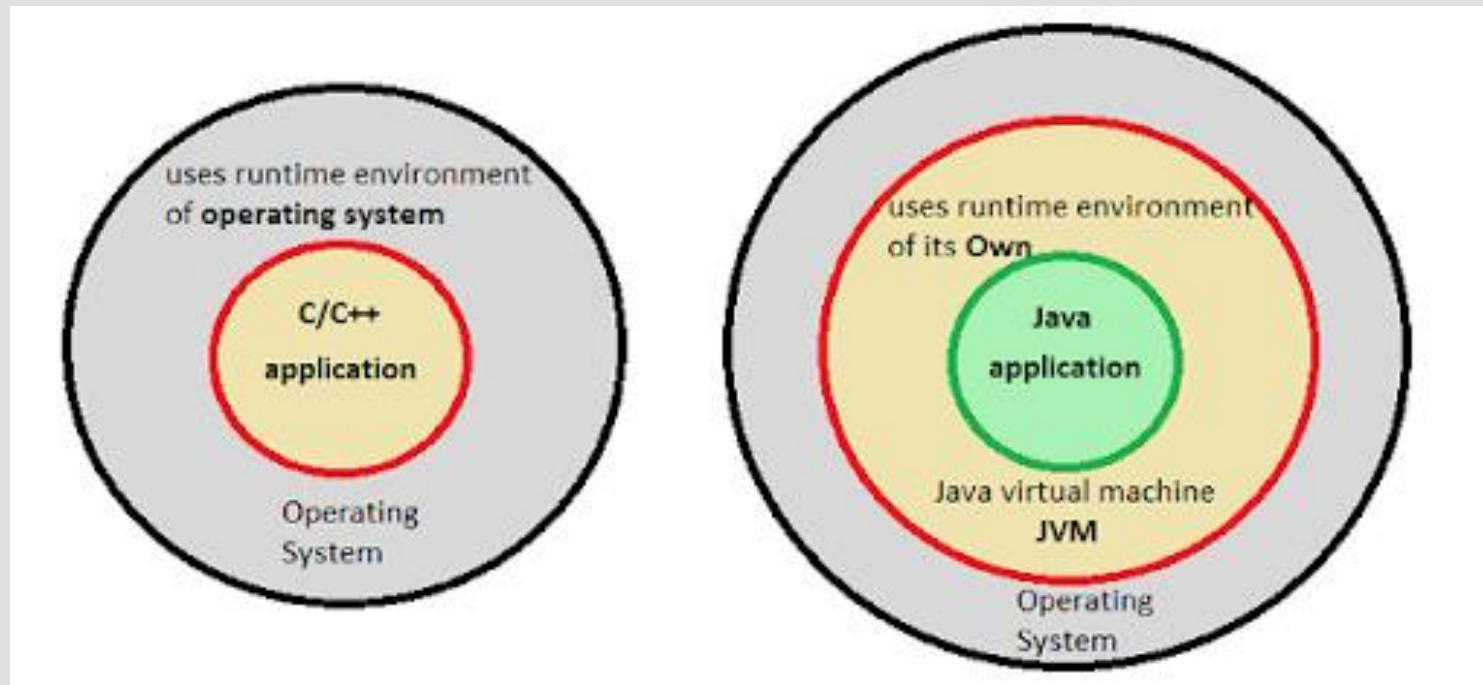- **Robust**
  - Strong memory management.
  - Automatic garbage collector inside JVM.

Secured

- Programs run in JVM (Similar to Sandbox), not directly on OS.
- No Pointers (i.e. No Direct/Unrestricted access to memory)

- **Architecture Neutral**
  - It can be run on any available processors in the real world without considering architecture and providers irrespective to its development and compilation.
  - Example: Data type sizes are fixed in java whereas in C language sizes differ for 32- bit and 64-bit compiler architecture.
- **Portable**
  - Platform Independent + Architecture Neutral
- **High-Performance**
  - Better performance than any other interpreted programming language because the byte code is "close" to the native code.
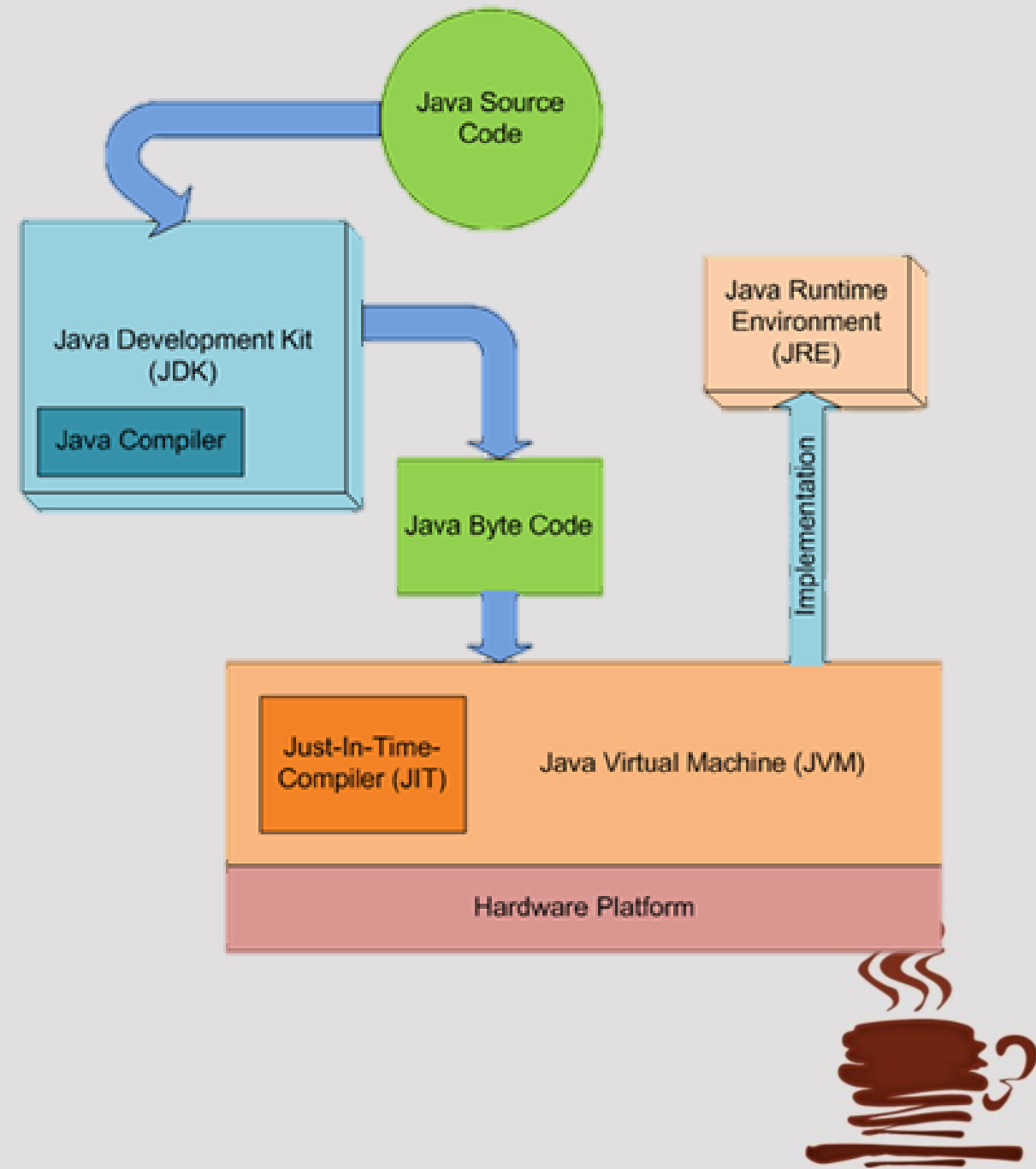- **Distributed**
  - You can write a distributed application in java.

Java Components

- JDK (Java Development Kit)
  - It is a collection of Programming tools that are used for developing and running java programs
  - They Include
    - Javac (Java Compiler)
    - Java Interpreter
    - Jdb (java Debugger)
    - Javadoc (for HTML Docs)
    - Applet Viewer
    - Javah (Header Files)

- **JVM (Java Virtual Machine)**
  - It is a platform or virtual machine that provides runtime environment in which java bytecode can be executed.
  - Functions of JVM
    - Load the Code
    - Verify the Code
    - Execute the Code
    - Provide Runtime Environment for Execution

- **JRE (Java Runtime Environment)**
  - It is a part of JDK and consist set of libraries and tools for developing java applications

- **JIT (Java in time Compiler)**
  - It is a part of JVM that is used to speed up the execution time.

# Comparison between JDK and JRE

- **JDK**

1. JDK(Java Development Kit) is used to develop Java applications. JDK also contains numerous development tools like compilers, debuggers, etc.

2. Its main functionality is development.

3. JDK = JRE + other development tools.

4. JDK is meant for programmers.

5. It contains one JRE.

6. It contains tools like javac, jar, javap, etc.

- **JRE**

1. JRE(Java Runtime Environment) is the implementation of JVM(Java Virtual Machine) and it is specially designed to execute Java programs.

2. It is mainly used for creating an environment for code execution.

3. JRE = JVM + other class libraries.

4. JRE is meant for users.

5. JRE is a part of JDK.

6. It contains tools like java, javaw, etc.

Conversion of Source code to Machine code
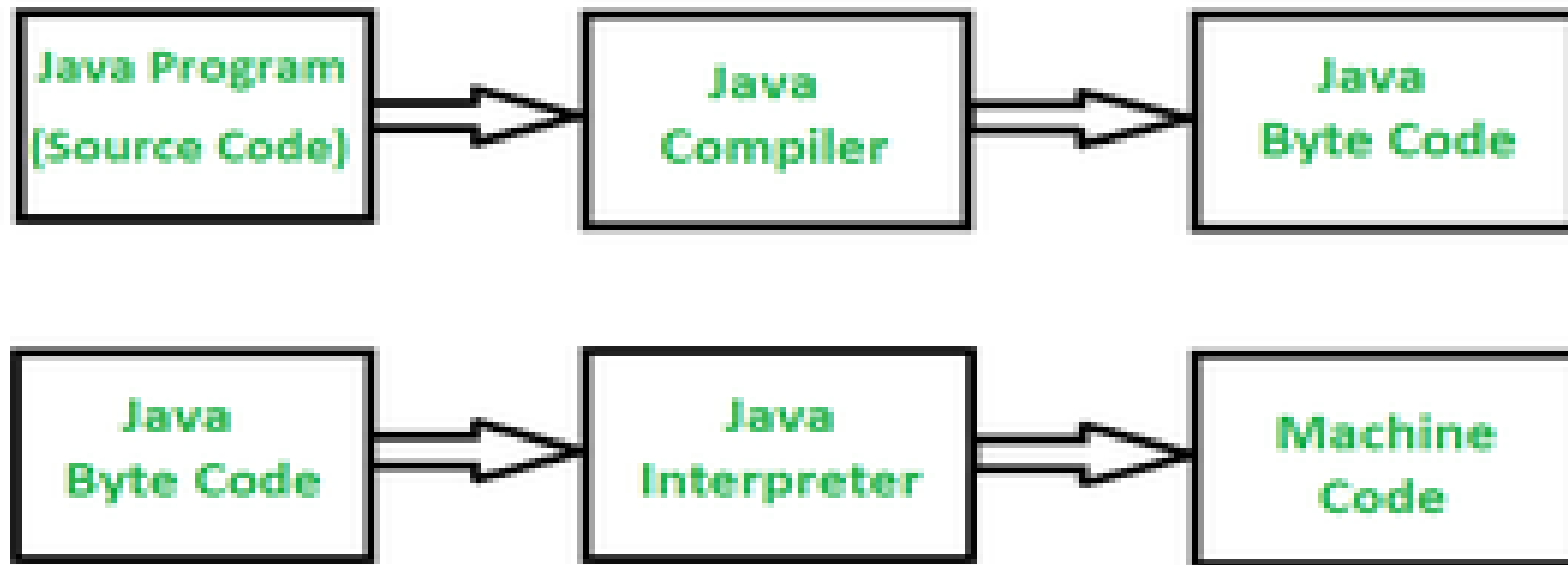
Byte Code /Importance of Byte Code

- When we write any program in a editor, we use java compiler to compile it, a java compiler javac translates Java Source Code into Java Byte Code, The Java runtime system does not compile your source code directly into machine language code i.e.. .exe file

- The JVM is intended to execute java byte code

- **Platform independent**, i.e. same for all OS.

- File extension is **.class**, Byte code file is **generated one per class.**

- Not human readable form, binary
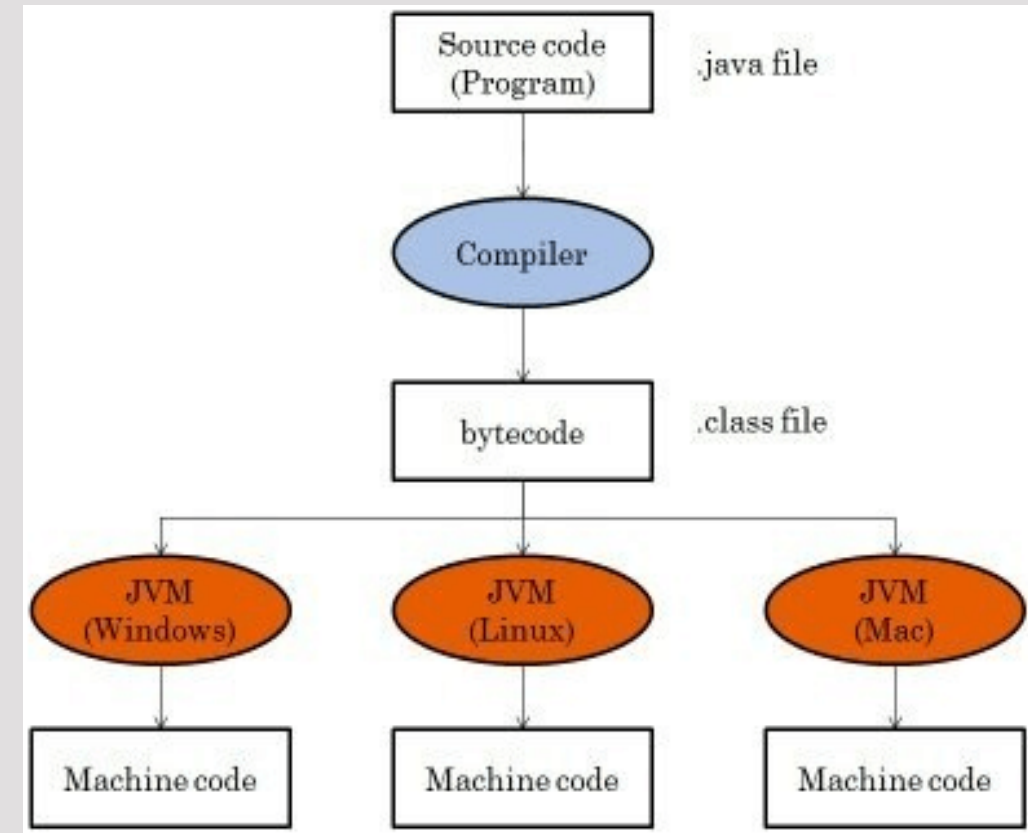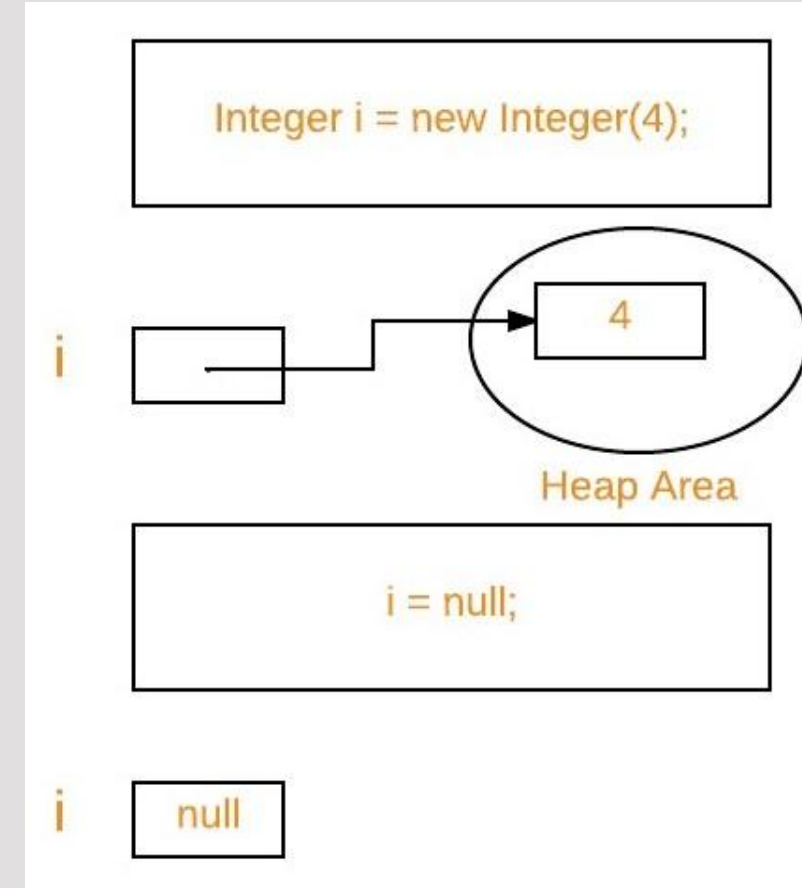
Garbage Collection

- In Java, the programmers do not need to take care of destroying the objects that are out of use. The Garbage Collector takes care of it.

- Garbage Collector is a **"Daemon thread"** that keeps running in the background. The garbage collector frees up heap memory by destroying all unreachable objects.

- **Unreachable objects** are the ones that are no longer referenced by any part of the program.

- We can choose the garbage collector for our java program through **JVM options**.

Garbage Collection: Eligibility

Integer i = new Integer(4);

// the new Integer object is reachable via the reference in 'i'

i = null;

// the Integer object is no longer reachable.

- An object is said to be eligible for GC(garbage collection) if it is unreachable.

- In above image, after i = null; integer object 4 in heap area is eligible for garbage collection.

Garbage Collection: How???

- Automatic Garbage collection is a process of looking at the **Heap memory**, identifying (also known as "marking") the unreachable objects, and destroying them with compaction.

- **The Young generation** heap space is the new where all the new Objects are created. Once it gets filled up, minor garbage collection (also known as, **Minor GC**) takes place. Which means, all **the dead objects** from this generation are **destroyed.** The surviving objects in young generation age and eventually moves to the older generations.

- **The Old Generation** is used to store long surviving objects. Typically, a threshold is set for young generation object and when that age is met, the object gets moved to the old generation. Eventually, the old generation needs to be collected. This event is called a **Major GC (major garbage collection)**. Often it is much **slower** because it involves all live objects.

How can an object be unreferenced?

1. By nulling the reference

Employee e=**new** Employee();

e=**null**;

2. By assigning a reference to another:

Employee e1=**new** Employee();

Employee e2=**new** Employee();

e2=e1;//now the first object referred by e1 is available for garbage collection

3. By anonymous object:

new Employee();

Methods for Garbage collection

1. **finalize() method**

    The finalize() method is invoked each time before the object is garbage collected.

    This method can be used to perform cleanup processing and it is defined in Object class as:

    protected void finalize(){}

* **Note:** The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing.

2. **gc() method**

    The gc() method is used to invoke the garbage collector to perform cleanup processing.

    The gc() is found in System and Runtime classes.

Example of garbage collection

```java
public class TestGarbage1{
        public void finalize()
        {System.out.println("object is garbage collected");}
        public static void main(String args[]){
                TestGarbage1 s1=new TestGarbage1();
                TestGarbage1 s2=new TestGarbage1();
                s1=null;
                s2=null;
                System.gc();
        }
}
```
Output: object is garbage collected
   object is garbage collected

Java Environment Setup

- Download setup file of JDK from
 https://www.java.com/en/download/
  OR
  https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.exe
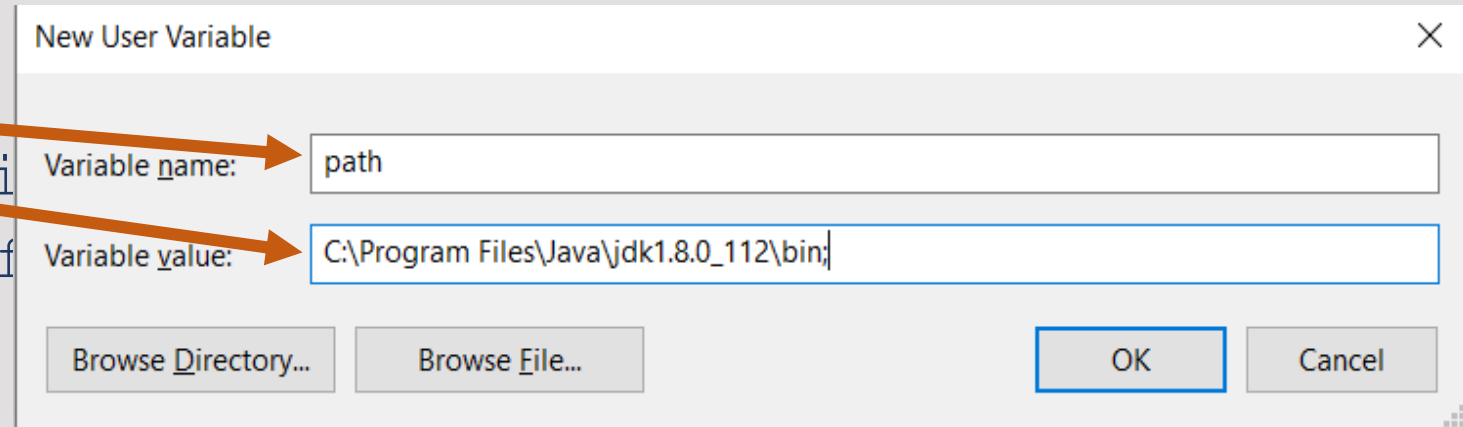 Install the setup using setup wizard.
- Once it is installed, we need to set up environment variable in the system.
  - Right Click on "This PC" icon Properties Option
  - Click on "Advanced System Sett
  - This will open a Dialog.
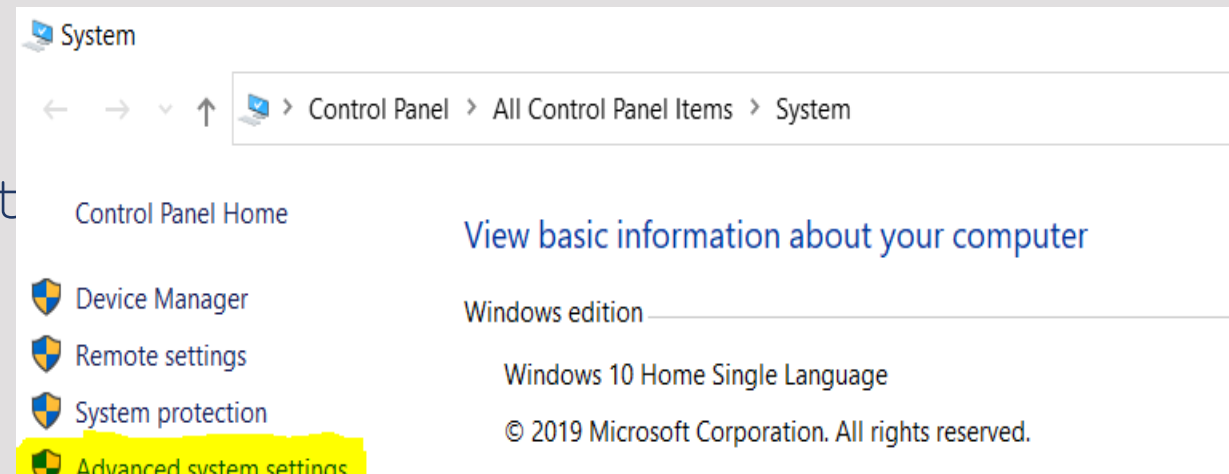
- Go to the "Advanced" tab as shown in figure.
- Click on "Environment Variables…" button.
- On the new Dialog Click on Button "New" to make new environment variable.
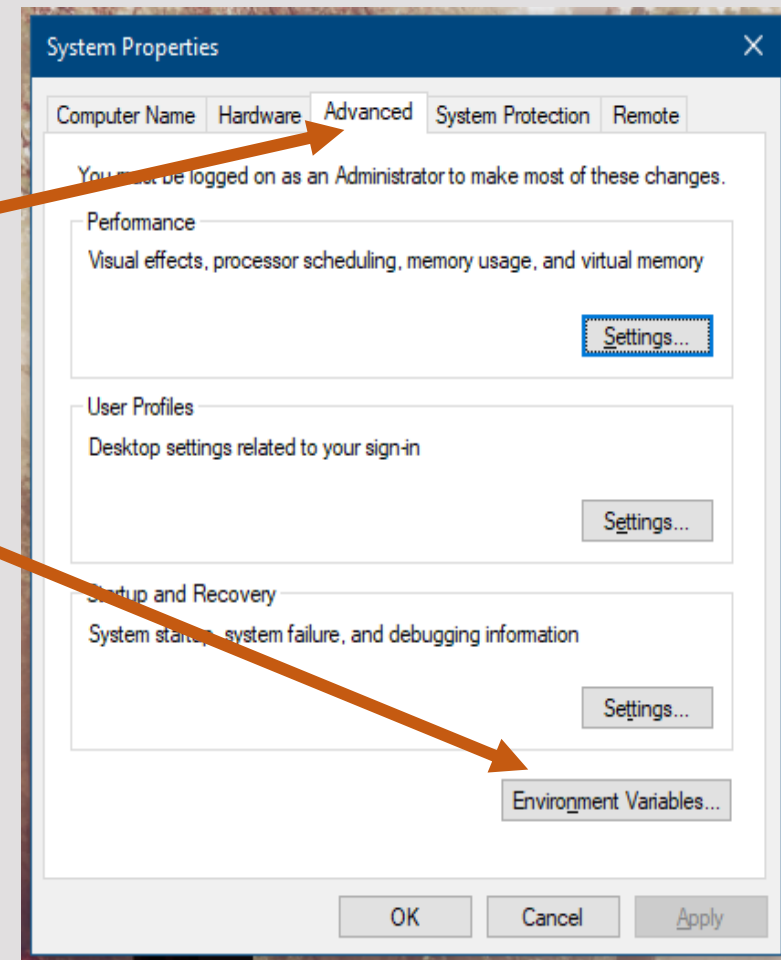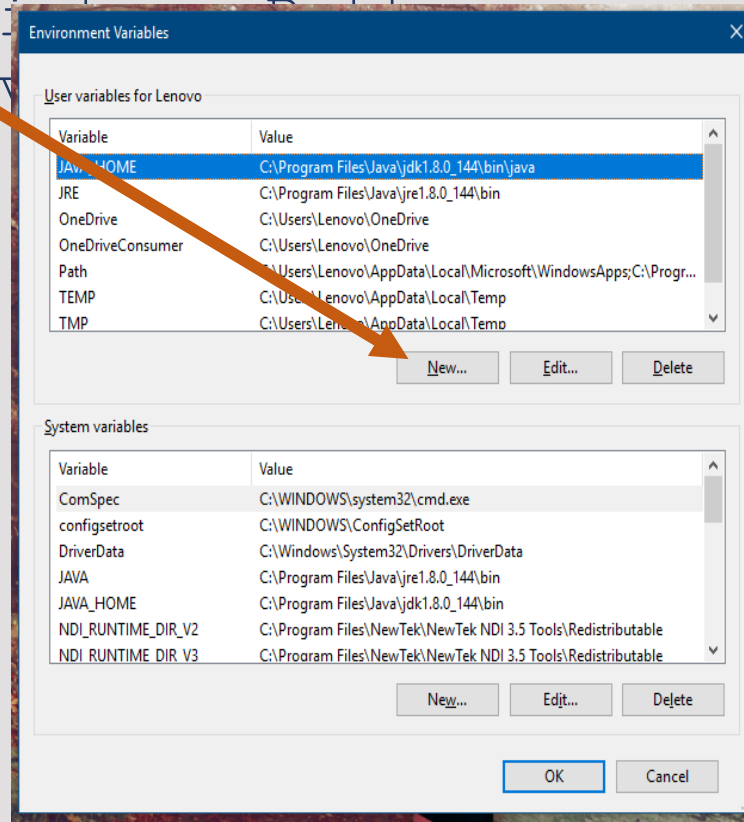
- Give name as "path"
- Value as the installation path up to bin folder followed by a ";" (semi-colon).
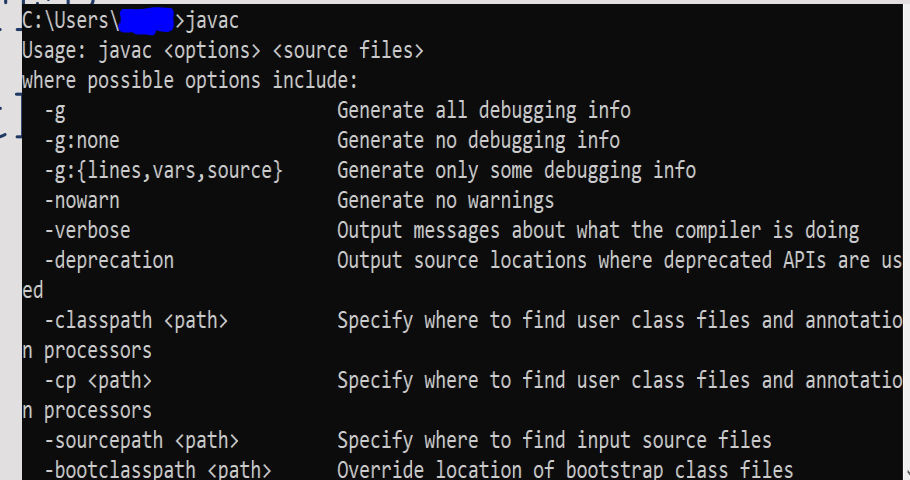- Click on "OK" button.

This finishes setup of Java environment on your computer to check the installation simply open "CMD"

Type command "javac" and press ENTER

If this output is generated then t

Structure of Java Program

Documentation Section	← Suggested
Package Section	← Optional
Import Section	← Optional
Interface Section	← Optional

```
Class Definition Section
class <classname> {
        public static void main(String[] args) {
                //Main method statements…
        }
}
```
← Essential

- **Document Section:**
  - It is a set of comment lines giving the details about the program, details about the author and the other details, which the programmer would like to include.

Package Section:

- This statement declares a package and informs the compiler that the classes defined here belong to this package.

Import Section:

- Using import statement ,we can access to classes that part of other named packages.

  Example : import student.test;

  This statement instructs the interpreter to load the test class   contained in package student.

Interface Section:

- An interface is like a class but includes group of methods declaration.

Class Definition Section:

- Class define in this section.
- Java application program requires main method as starting point.
- There must be one class with main method in java program

main() method syntax

public   static   void   main   (String[]   args)

Access Modifier

Method is called from outside of class also so "public"

Static Makes it Possible for Us to Run these functions without creating an object of the class.

Does not return any value so "void"

Default Function

Compiler needs to find method named main as the entry point of the program.

It is an array of string class that stores java command line arguments, Here the name of the string array is args, shortly from arguments

println() method syntax

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

Name of java utility class

Object which belongs to system class

It is a utility method name that is used to send any string to console.

Naming Conventions in Java

- For Classes Declarations: PascalConvention are used.
  Example: AddTwoNumber

- For   Functions/Methods   Declarations:
  camelCaseConvention are used.
  Example: addTwoNumber

Compile & Run First Java Program

- Create a java file as <filename>.java

- Every line of code that runs in Java must be inside a class, In our example, we named the class Helloworld.

- A class should always start with an uppercase first letter, Note that Java is case-sensitive: "MyClass" and "myclass" has different meaning.

- The name of the java file must match the class name name if the class is declared **public,** otherwise can keep other names also.

- When saving the file, save it using the class name and add **".java"** to the end of the filename.

```java
//Sample Java Program

package chapter1;


import java.util.Scanner;
import java.io.PrintWriter;


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

Documentation Section

Package Section

Import Section

Main class Definition Section

- To execute above program.
  - Open CMD.
  - Go to Directory where the program is stored.
  - Execute **javac <filename>.java** to compile the java program (The class file will be generated here)
  - Execute **java <classname>** (Java Interpreter will execute the class file / program and generated the output.)

```
E:\JAVA\Programs>javac HelloWorld.java

E:\JAVA\Programs>java HelloWorld
Hello World!!!
```

Comments

- Comments in Java, are the lines in code that do not show up in the executable program, i.e. the compiler ignores these lines during the compilation process and that is used to programmers understanding purpose and to add some meta-data regarding the code.

- There are three ways to do comments:
  - Single Line Comment
    - Single-line comments start with two forward slashes //
    - Any text between // and the end of the line is ignored by Java.
    - Example:

      int x;   // a comment
  - Multiline Comment
    - You can use the /* and */ comment delimiters that let you block off a longer comment.
    - Example:

      /*
      The variable x is an integer:

- Document Section Comment
  - Finally, a third kind of comment is used to generate documentation automatically. This comment uses a /** to start and a */ to end.
  - Example:

```
/**
x -- an integer representing the x
coordinate
*/
int x;
```

Keyword

- Java has a set of keywords that are reserved words that cannot be used as variables, methods, classes, or any other identifiers:

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	null	package	private	protected
public	return	short	static	strictfp
super	switch	synchronized	this	throw
throws	transient	try	void	volatile
while				

Data Types

- Datatype Specifies the type of Data that a variable can store.
- Primitive data types are predefined types of data, which are supported by the programming language.
- Non-primitive types are created by the programmer and is not defined by Java.
- Primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
- List are given below;

Data Types in java

Primitive Data Types

- Integers
  - byte
  - short
  - int
  - long
- Floating-Point
  - float
  - double
- Character ——— char
- Boolean ——— boolean

Non-primitive Data Types

- String
- Array
- List
- Set
- Stack
- Vector
- Dictionary
- All user-defined classes
- etc..
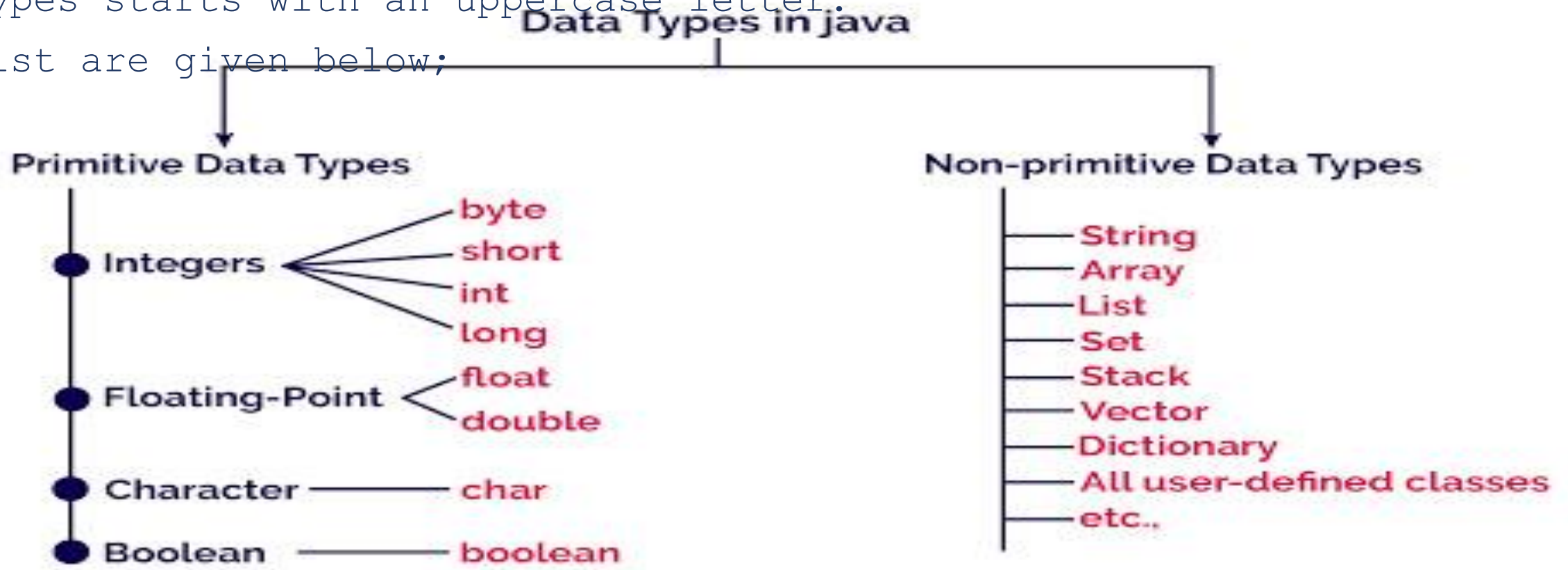
Type	Size	Range	Example
Byte	8 bit (1 byte)	-128 to 128 Default Value: 0	byte a; byte a=100;
Short	16 bit (2 byte)	-32768 to 32767 Default Value: 0	short a; short a=1000;
Int	32 bit (4 byte)	-2147483684 to 2147483647 Default Value: 0	int a; int a=100000;

Type	Size	Range	Example
Double	64 bit (8 byte)	1.7e-308 to 1.7e+308 Default Value: 0.0d	double a; double a=123.4;
Float	32 bit	3.4e-308 to 3.4e+308	float a;

Type	Size	Range	Example
Char	16bit	0 to 65536 Default Value: '\u0000'	char a; Char a='A';

Type	Size	Range	Example
Boolean	1bit	True or false value Default Value: false	boolean a; Boolean a= true;

User Defined Data Types

- Types that are declared by the user as per their need are known as User Defined Data Types.

- Those types include: Derived Data Types like Arrays and User defined types like Classes and Interfaces.

Identifiers

- All Java components require **names**, Names given to the variables, Methods, Constants are known as Identifiers.
- An identifier is a sequence of characters, comprising uppercase and lowercase letters (a-z, A-Z), digits (0-9), underscore "_", and dollar sign "$".
- Rules for identifiers:
  - All identifiers must start with either a letter(a to z or A to Z) or an underscore.
  - After the first character, an identifier can have any combination of characters.
  - A Java keyword cannot be used as an identifier.
  - Identifiers in Java are case sensitive; foo and Foo are two different identifiers.

Literals

- Literal is a specific **constant value** or **data** that is used in java program, such as 123, 3.14, 'a', "Hello", true.
- Integer Literals
  - Refers to a Sequence of digits which Includes only negative or positive.
  - Example:
    ```
    byte b = 100;
    short s = 1000;
    int i = 555;
    ```
- Double/Floating Point Literals
  - Example:
    ```
    float f = 10.2f;
    double d = 23.57d;
    ```

- Character Literals
  - Character is Single Alphabet a single digit or a Single Symbol that is enclosed within Single inverted commas.
  - Example:

    char c = 'A';
- String Literals
  - String is a Sequence of Characters Enclosed between double Quotes, these characters may be digits, Alphabets or any other special symbols.
  - Example:

    string s = "Hello";

Variables

- Variables are containers for storing data values, and this values can be changed during the execution of the program.

- A variable has a name , e.g. radius, area, age, the name is used to uniquely identify each variable.

- In java variables can be declared as,

  <access_modifier> <other_modifiers> <data_type> variable_name;

  OR

  <access_modifier> <other_modifiers> <data_type> variable_name = value;

- Rules for Defining Variable Name;
  - Name Must not begin with a digit.
  - Name is case Sensitive.
  - Name should not be a keyword.
  - Whitespace in Name are not allowed
  - Name can contain alphabets, special character and digits if the other mentioned condition are met.

Declaration of Variables

■ A variable is used for storing a value either a number or a character and a variable also vary its value means it may change his value Variables are used for given names (known as Identifiers) to locations in the Memory of Computer where the different constants are stored. These locations contain Integer, Real or Character Constants.

■

| **Instance Variables**	**Local Variables**	**Class Variables**
•The Variables those are declared in a class are known as instance variables When object of Class is Created then instance Variables are Created	•The Variables those are declared in Method of class are known as Local Variables.	•The Variables Those are declared inside a class are called as Class variables or also called as data members of the class . •They are friendly by default means they are Accessible to main Method or any other Class Which inherits.

- Example:
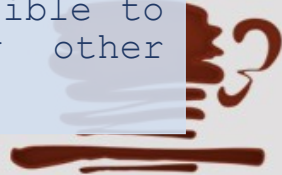
```java
int myNum = 5;                    // Integer Variable
float myFloatNum = 5.99f;         // Floating Point Variable
double myDoubleNum = 6.2;         // Double Variable
char myLetter = 'D';              // Character Variable
boolean myBool = true;            // Boolean Variable
String myText = "Hello";          // String Variable
```

- Example for that shows use of datatype and variable:

```java
public class Main {
    public static void main(String[] args) {
        int a;
        a = 15;
        int b = 14;
        int x=1,y=2,z=3;
        System.out.println(a);
        System.out.println(b);
        System.out.println(x + y + z);

    }
}
```

Output: 15
          14
          6

Scope of Variables/Types of Variable

- A variable's scope is the region of a program within which the variable can be accessed, so scope determines when the system creates and destroys memory for the variable.
  - **Local Variable:**
    - These variables are declared within a method, constructor, or block, and their scope is limited to that specific block.
    - They must be initialized before they are used.
  - **Instance Variable:**
    - These variables are declared within a class but outside any method, constructor, or block.
    - They are initialized when the object is created.
    - Each instance of the class gets its own copy of instance variables.

Scope of Variables/Types of Variable

- **Class Variable:**
  - These variables are declared with the **static** keyword within a class but outside any method, constructor, or block.
  - They are initialized only once at the start of the execution and shared among all instances of the class.
- **Constants:**
  - Variables declared with the **final** keyword are constants and cannot be changed once initialized.
  - Because java does not have a constant, so you can use final variable instead.
  - They must be initialized either at the time of declaration or within the constructor.

Constants

- Constants are Declared as follows:

- The Constants are declared using the word "final".

 \<access-modifier\> final \<data_type\> member_name

- Constants are those who are not changed through the program. They remain same and they are defined.

- "final" can be applied to Variable, Object, Method or Even Class (Details will be discussed in Unit-4).

- If a variable is declared final in any class then it can be initialized only at declaration time or in the constructor of that class. Further initialization of change in value results in an error.

Operators

- In Java, operators are symbols that perform operations on operands.
- Java provides a rich operator environment, that is listed below,
  1. Arithmetic Operator
  2. Bitwise Operator
  3. Relational Operator
  4. Logical Operator
  5. Assignment Operator
  6. Conditional/ Ternary Operator
  7. Increment and Decrement Operator

Arithmetic Operators

▪ Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, division, and modulus.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y

- Example:

```
public class Airthmetic {
        public static void main(String args[]) {
                int i = 12;
                int j = 3;
                        System.out.println(i + j);    // Output: 15
                        System.out.println(i - j);    // Output: 9
                        System.out.println(i * j);    // Output: 36
                        System.out.println(i / j)     // Output: 4
                        System.out.println(i % j);    // Output: 0
        }
    }
```

Bitwise Operators

- Bitwise operators perform operations on individual bits of integers

Operators	Description	Use
&	Bitwise AND	op1 & op2
|	Bitwise OR	op1 | op2
^	Bitwise Exclusive OR	op1 ^ op2
~	Bitwise Complement	~op
<<	Bitwise Shift Left	op1 << op2
>>	Bitwise Shift Right	op1 >> op2
>>>	Bitwise Shift Right zero fill	op1 >>> op2

- Example:

```
public class Bit {
    public static void main(String args[]) {
        int a = 3;           // 0 1 1
        int b = 6;           // 1 1 0
        System.out.println(" a & b = " + (a & b)); // a & b = 2
        System.out.println(" a | b = " + (a | b)); // a | b = 7
        System.out.println(" a ^ b = " + (a ^ b)); // a ^ b = 5
        System.out.println(" ~a = " + (~a));          // ~a = -4
    }
}
```

- Example:

```java
public class Example {
    public static void main(String args[]) {
        int a = 4;          // 0 0 0 0 0 1 0 0
        int b = 1;          // 0 0 0 0 0 0 0 1
        int c = 2;          // 0 0 0 0 0 0 1 0
        System.out.println("a << b : "+ (a << b) );   // a << b : 8
        System.out.println("a >> b : "+ (a >> b) );    // a >> b : 2
        System.out.println("a >>> b : "+ (a >>> b) ); // a >>> b : 2
        System.out.println("c >> b : "+ (c >> b) );    // c >> b : 1
        System.out.println("c >>> b : "+ (c >>> b) ); // c >>> b : 1
    }
}
```

Relational/Comparison Operators

- Relational operators are used to perform used to compare two values (or variables).

- The return value of a comparison is either true or false.

Operator	Description	Usage
==	Equal to	expr1 == expr2
!=	Not Equal to	expr1 != expr2
>	Greater than	expr1 > expr2
>=	Greater than or equal to	expr1 >= expr2
<	Less than	expr1 < expr2
<=	Less than or equal to	expr1 >= expr2

- Example:

```java
public class Rel {
    public static void main(String args[]) {
        int a=2 ,b=5;
        System.out.println("a < b : " + (a<b));   // a < b : true
        System.out.println("a > b : " + (a>b));   // a > b : false
        System.out.println("a <= b : " + (a<=b));// a <= b : true
        System.out.println("a >= b : " + (a>=b));// a >= b : false
        System.out.println("a != b : " + (a!=b));// a != b : true
        System.out.println("a == b : " + (a==b));// a == b : false
    }
}
```

Logical Operators

- Logical operators are used to perform logical operations.
- || and && are called short circuit operators because :
  - In case of || if first condition is true then it will not check the second condition and the whole condition becomes true
  - In case of && if first condition is false the it will not check the second one and the whole condition becomes false

Operator	Description	Usage
!	Logical NOT	! a
||	Logical OR	a || b
&&	Logical AND	a && b
^	Logical XOR	a ^ b

- Example:

```
public class Logic {
    public static void main(String args[]) {
    boolean a = true , b = false;
    System.out.println(" ! a : " + (!a));      // ! a : false
    System.out.println("a || b : " + (a||b));// a || b : true
    System.out.println("a && b : " + (a&&b));// a && b : false
    System.out.println("a ^ b : " + (a^b));   // a ^ b : true
    }
}
```

Assignment Operators

- Assignment operators are used to assign values to variables.

Operator	Example	Equivalent Expression
$=$	$m = 10$	$m = 10$
$+=$	$m += 10$	$m = m + 10$
$-=$	$m -= 10$	$m = m - 10$
$*=$	$m *= 10$	$m = m * 10$
$/=$	$m /=$	$m = m/10$
$\% =$	$m \% = 10$	$m = m\%10$
$<<=$	$a <<= b$	$a = a << b$
$>>=$	$a >>= b$	$a = a >> b$
$>>>=$	$a >>>= b$	$a = a >>> b$
$\& =$	$a \& = b$	$a = a \& b$
$\wedge =$	$a \wedge = b$	$a = a \wedge b$
$\| =$	$a \| = b$	$a = a \| b$

- Example:
```
public class Assign {
    public static void main(String args[]) {
        int x = 5;
        int y = 5;
        y += 3;
        System.out.println(x); // x = 5
        System.out.println(y); // y = 8
    }
}
```

Ternary Operator / Short Hand If…Else Statement

- Syntax:

    variable = (condition) ? expressionTrue : expressionFalse;

- Here, according to condition statement is executed, if condition is true then expressionTrue portion will be executed, else expressionFalse portion will be executed.

- Example:

```
public class Tern {
    public static void main(String[] args) {
        int time = 20;
        String result = (time < 18) ? "Good day" : "Good evening";
        System.out.println(result);
    }
}
Output: Good evening
```

Increment(++) and Decrement(--) Operator

- Increment operator (++) increases its operand by one and decrement operator (--) decreases its operand by one.

- These operators are unique in that they can appear both in postfix form, where they follow the operand (x++ / x--) as just shown, and prefix form, where they precede the operand (++x/--x).

  - In the prefix form, the operand is incremented or decremented before the value is obtained for use in the expression.

  - In postfix form, the value is obtained for use in the expression, and then the operand is modified.

- Example:

  x = x + 1;

  can be rewritten like this by use of the increment operator: x++;

  x = x - 1;

  can be rewritten like this by use of the increment operator: x--;

- Some More Demonstration:

```
int x = 42;
int y = ++x;        // y=43


int x = 42;
int y = x++;        // y=42


int x = 42;
Int y = --x;        // y=41


int x = 42;
Int y = x--;        // y=42
```

- Example:

```java
public class Inc {
    public static void main(String[] args) {
        int a = 5,b = 4,c = 5,d = 4;
        ++a;
        b++;
        --c;
        d--;
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
    }
}
```

Output: 6

              5

              4

              3

Decision & Control Statements: Selection Statement

- **Java's Selection statements**
  - **if**
  - **if-else**
  - **nested-if**
  - **if-else-if**
  - **switch-case**
  - **jump – break, continue**
  - **return**

```java
if(condition)
{
    // Statements to execute if
    // condition is true
}
```

```java
if (condition)
{

    // Executes this block if
    // condition is true
}
else
{

    // Executes this block if
    // condition is false
}
```

Switch Case

```
switch (expression)
{
  case value1:

    statement1;

    break;

  case value2:

    statement2;

    break;

  …

  case valueN:

    statementN;

    break;

  default:

    statementDefault;

}
```

Expression can be of type byte, short, int char or an enumeration. Beginning with JDK7, *expression* can also be of type String.

Dulplicate case values are not allowed.

The default statement is optional.

The break statement is used inside the switch to terminate a statement sequence.

Switch Case

```
class SwitchCaseDemo
{
    public static void main(String args[])
    {
        int i = 9;
        switch (i)
        {
        case 0:
            System.out.println("i is zero.");
            break;
        case 1:
            System.out.println("i is one.");
            break;
        case 2:
            System.out.println("i is two.");
            break;
        default:
            System.out.println("i is greater
than 2.");
        }
    }
}
```

Output
i is greater than 2.

Switch variable in java can only work with byte, short, char, int and Enumerated Data types

Loops

for loop

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

while loop

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

do-while loop

The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

Loops: While

- A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

- Syntax :
  ```
  while (boolean condition)
   {
      //loop statements...
   }
  ```
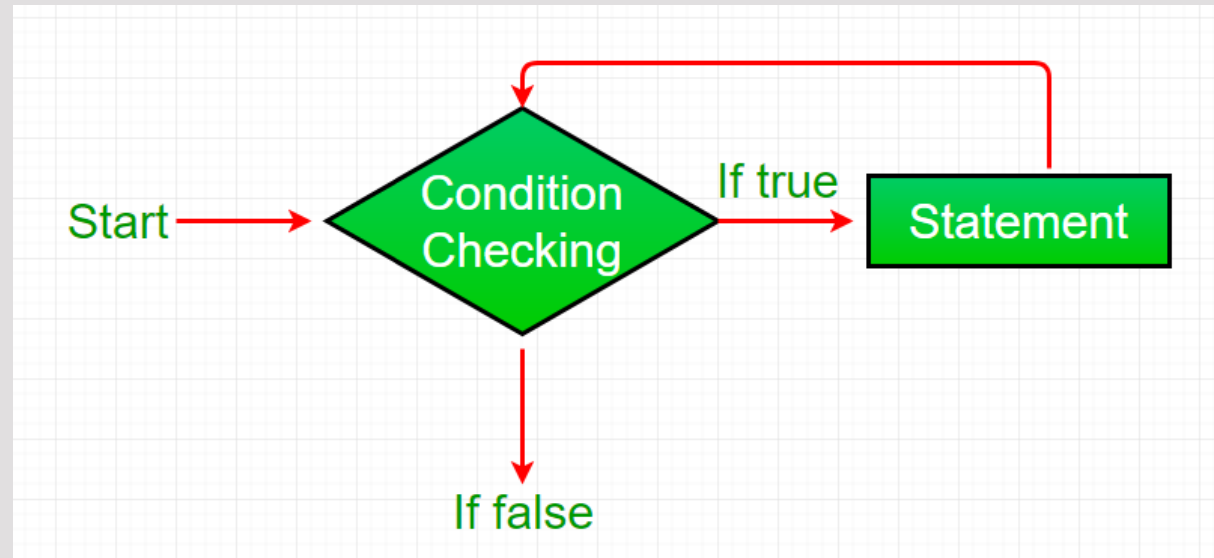
Loops: While

```
class whileLoopDemo
{
    public static void main(String args[])
    {          int x = 1;
        // Exit when x becomes greater than 4
        while (x <= 4)
        {

            System.out.println("Value of x:" + x);

            x++;

        }

    }

}
```

Output:

Value of x:1
Value of x:2
Value of x:3
Value of x:4

Loops: For

- for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

- Syntax:

```
for (initialization condition; testing condition; post increment/decrement)
{
    //loop statement(s)
}
```

Loops: For

```java
public class MyClass {
    public static void main(String[] args) {
        for (int i = 0; i <= 10; i = i + 2) {
            System.out.println(i);
        }
    }
}
```

Result:
```
0
2
4
6
8
10
```

Loops: Do…While

- do…while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.

- Syntax:

```
do
{
    //loop statements..
} while (condition);
```
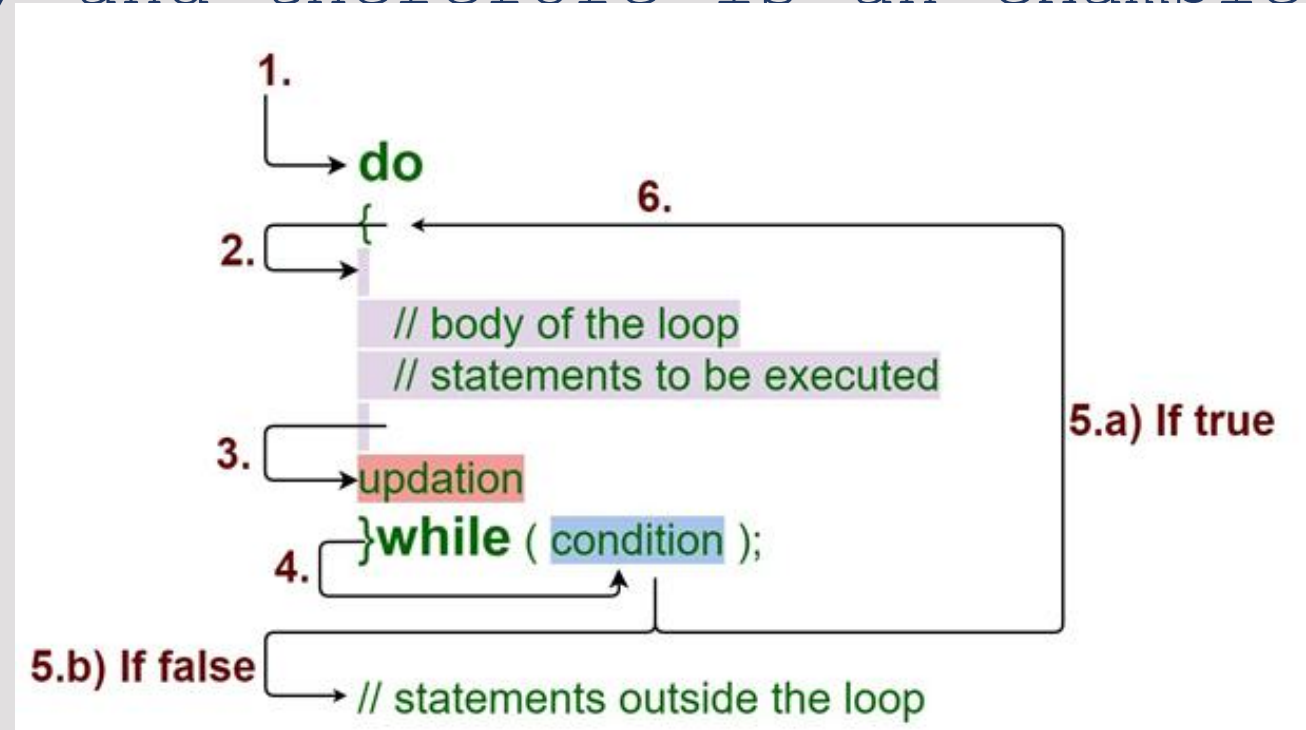
Loops: Do...While

```java
class dowhileloopDemo {

    public static void main(String args[])
    {

        // initialisation expression
        int i = 1;
        do {

            // Print the statement
            System.out.println("Java");

            // update expression
            i++;
        }
        // test expression
        while (i < 6);

    }
}
```

Dry-Running Example 1: The program will execute in the following manner.

1. Program starts.
2. i is initialized with value 1.
3. Execution enters the loop
  3.a) "Java" gets printed 1st time.
  3.b) Updation is done. Now i = 2.
4. Condition is checked. 2 < 6 yields true.
5. Execution enters the loop
  5.a) "Java" gets printed 2nd time.
  5.b) Updation is done. Now i = 3.
6. Condition is checked. 3 < 6 yields true.
7. Execution enters the loop
  7.a) "Java" gets printed 3rd time
  7.b) Updation is done. Now i = 4.
8. Condition is checked. 4 < 6 yields true.
9. Execution enters the loop
  9.a) "Java" gets printed 4th time
  9.b) Updation is done. Now i = 5.
10. Condition is checked. 5 < 6 yields true.
11. Execution enters the loop
  11.a) "Java" gets printed 5th time

Output:
Java
Java
Java
Java
Java

Loops: Nested For Loop

```java
public class PyramidExample {
    public static void main(String[] args) {
        for(int i=1;i<=5;i++){
            for(int j=1;j<=i;j++){
                System.out.print("* ");
            }
            System.out.println();//new line
        }
    }
}
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

For-Each Loop

- **"for-each"** loop, which is used exclusively to loop through elements in an **array**.

```
for (type variableName : arrayName) {
    // code block to be executed
}
```

```
String[] cars = {"Audi", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

Output
Audi
BMW
Ford
Mazda

break

```java
class BreakLoopDemo
{
    public static void main(String args[])
    {
        // Initially loop is set to run from 0-9
        for (int i = 0; i < 10; i++)
        {
            // terminate loop when i is 5.
            if (i == 5)
                break;
            System.out.println("i: " + i);
        }
        System.out.println("Loop complete.");
    }
}
```

break keyword is used
to break the loop
whenever that
statement is

Output
i: 0
i: 1
i: 2
i: 3
i: 4
Loop complete.

Labelled break

```
// Java program to illustrate using break with goto

class BreakLabelDemo
{

    public static void main(String args[])

    {

            boolean t = true;


            // label first

            first:

            {

                    // Illegal statement here
as label second is not

                    // introduced yet break
second;

                    second:

                    {
```

```
third:
{
        // Before break
        System.out.println("Before the break
statement");

        // break will take the control out of
        // second label
        if (t)
                break second;
                System.out.println("This won't
execute.");
        }
        System.out.println("This won't
execute.");
}

// First block
        System.out.println("This is after second
block.");
} } }
```

continue statement

```java
// Java program to illustrate using
// continue in an if statement
class ContinueDemo
{
    public static void main(String args[])
    {
        for (int i = 0; i < 10; i++)
        {
            // If the number is even
            // skip and continue
            if (i%2 == 0)

                continue;


            // If number is odd, print it
            System.out.print(i + " ");

} } }
```

continue statement is used to break only the current iteration of the loop, however the loop can continue executing its next iteration

Output
1 3 5 7 9

return statement

```java
// Java program to illustrate using return
class Return
{
    public static void main(String args[])
    {
        boolean t = true;
        System.out.println("Before the return.");

        if (t)
            return;

        // Compiler will bypass every statement
        // after return
        System.out.println("This won't execute.");
    }
}
```

Return statement is used to return the call of the function from that line. Anything written after that line will be considered as unreachable code because it will never be executed by the compiler.

Output

Before the return.

Type Conversion & Casting

- The process of converting a value from one data type to another is known as **type conversion.**

- If the two data types are **COMPATIBLE** with each other, Java will perform such conversion automatically or implicitly for you.

- Type Conversion can be done in 2 types
  - Implicit conversion is also known as type promotion in java.
  - For example, If you assign an int value to a long variable, It is compatible with each other but an int value cannot be assigned to a byte variable.

  ```
  int a = 20;
  long b = a;    //Automatic conversion
  byte c = a;    //Type mismatch
  ```

- Why???

Type Conversion & Casting

- Type Promotion/Conversion can be done implicitly only if
    - Source and Destination types are Similar/Compatible
    - Source data type is smaller or of equal size as destination.
- Otherwise an error will be generated. (Because of loss of precision, May be!!!)
- But if we know the data can fit in to the new variable then Explicit Type Conversion can be used known as **Type Casting.**
- So Implicit Conversion can also be known as **Widening Conversion** and Explicit Conversion can also be known as **Narrowing Conversion.**

Type Conversion & Casting

```java
class Test {
    public static void main(String[] args) {
        int i = 15;

        //implicit type conversion
        long l = i;

        //implicit type conversion
        float f = l;

        System.out.println("Int value "+i);
        System.out.println("Long value "+l);
        System.out.println("Float value "+f);
    }
}
```

Output

Int value 15

Long value 15

Float value 15.0

Type Conversion & Casting

```java
//Java program to illustrate incompatible data
// type for explicit type conversion
public class IncompatibleCasting {
  public static void main(String[] args) {
    char ch = 'c';
    int num = 88;
    ch = num;
  }
}
```

Output

```
7: error: incompatible types: possible lossy conversion from int to char
    ch = num; <-1 error
```
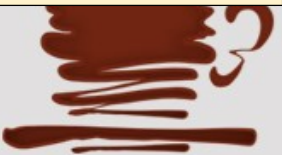
Type Conversion & Casting

```java
//Java program to illustrate explicit type conversion
class ExplicitDemo {
    public static void main(String[] args) {
        double d = 115.44;

        //explicit type casting
        long l = (long)d;

        //explicit type casting
        int i = (int)l;
        System.out.println("Double value "+d);

        //fractional part is lost
        System.out.println("Long value "+l);

        //fractional part is lost
        System.out.println("Int value "+i);
    }
}
```

Output

Double value 115.44

Long value 115

Int value 115

Type Promotion in Expressions

- While evaluating expressions, the intermediate value may exceed the range of operands and hence the expression value will be promoted. Some conditions for type promotion are:

- Java automatically promotes each byte, short, or char operand to int when evaluating an expression.

- If one operand is a long, float or double the whole expression is promoted to long, float or double respectively.

Type Promotion

```java
//Java program to illustrate Type promotion in Expressions
class TypePromotion {

    public static void main(String args[]) {
        byte b = 42;
        char c = 'a';
        short s = 1024;
        int i = 50000;
        float f = 5.67f;
        double d = .1234;

        // The Expression
        double result = (f * b) + (i / c) - (d * s);

        //Result after all the promotions are done
        System.out.println("result = " + result);
    }
}
```

Output

result = 626.7784146484375

Arrays

- An array is a collection of similar type of elements which have a contiguous memory location.

- Java array is an object which contains elements of a similar data type.

- Additionally, The elements of an array are stored in a contiguous memory location.

- It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Arrays

- Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

- In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces.

- We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimensional   multidimensional arrays in Java.

- Moreover, J   anonymous arrays which
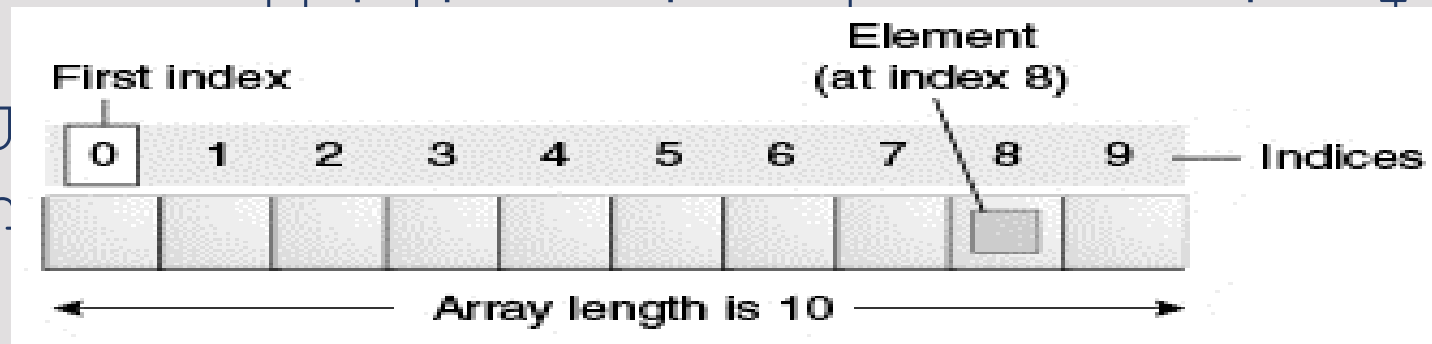
Arrays

- **Advantages**
  - Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.
  - Random access: We can get any data located at an index position
- **Disadvantages**
  - Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Arrays

- 2 Types of Arrays:
  - Single Dimensional Array
  - Multidimensional Array

Single Dimensional Array

- **Syntax to Declare an Array i**
  - *dataType[] arr; (or)*
  - *dataType []arr; (or)*
  - *dataType arr[];*

- **Instantiation of an Array i**
  - *arrayRefVar=new datatype[size];*

```java
class Testarray{
public static void main(String args[])
{
int a[]=new int[5];
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)
System.out.print(" " +a[i]);
}}
```

Output:

10 20 70 40 50

Multi Dimensional Array

- In such case, data is stored in row and column based index (also known as matrix form).

- Syntax to Declare Multidimensional Array in Java
  - dataType[][] arrayRefVar; (or)
  - dataType [][]arrayRefVar; (or)
  - dataType arrayRefVar[][]; (or)
  - dataType []arrayRefVar[];

- Example to instantiate Multidimensional Array in Java
  - int[][] arr=new int[3][3];   //3 row and 3 column

Multi Dimenstional Array

```java
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
    System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}
}
```

Output:
```
1 2 3
2 4 5
4 4 5
```

Jagged Array

- If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.
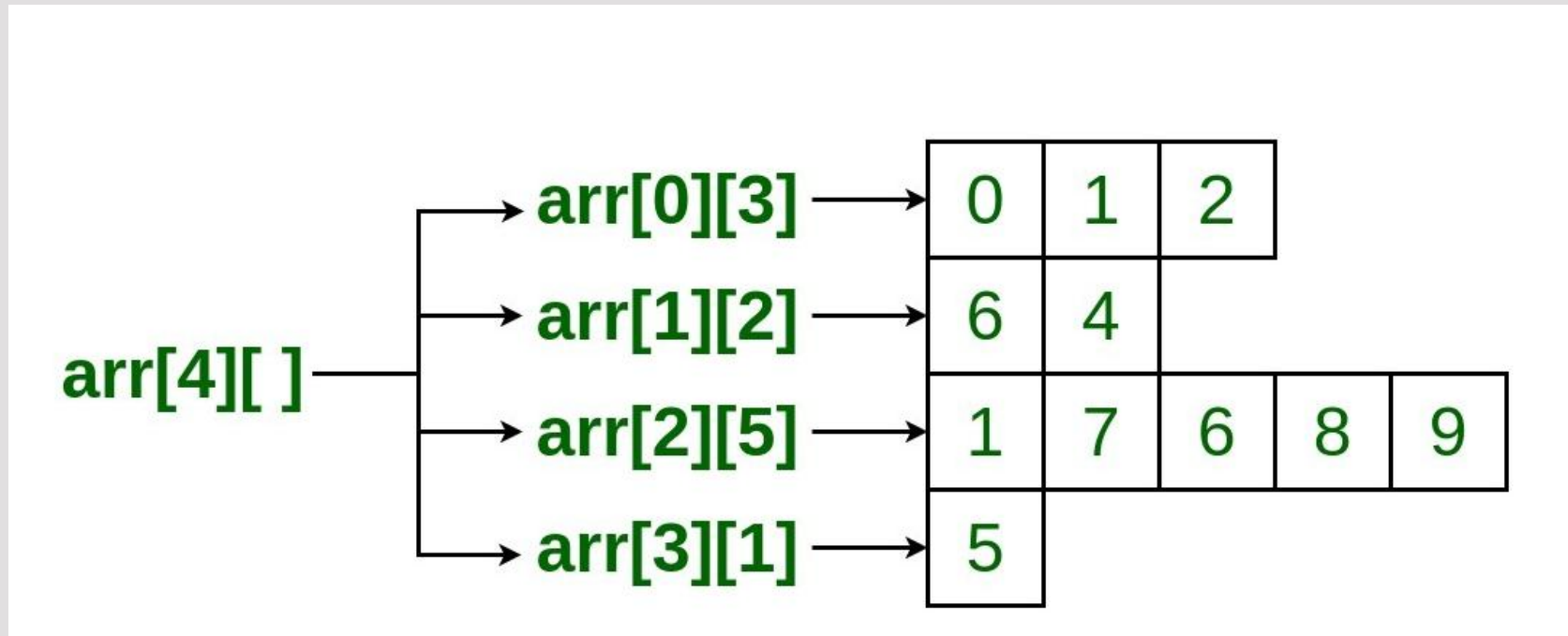
Jagged Array

```java
//Java Program to illustrate the jagged array
class TestJaggedArray{
    public static void main(String[] args){

        //declaring a 2D array with odd columns

        int arr[][] = new int[3][];
          arr[0] = new int[3];

          arr[1] = new int[4];

          arr[2] = new int[2];

        //initializing a jagged array

        int count = 0;

        for (int i=0; i<arr.length; i++)
                for(int j=0; j<arr[i].length; j++)
                    arr[i][j] = count++;

            //printing the data of a jagged array
            for (int i=0; i<arr.length; i++){
                for (int j=0; j<arr[i].length; j++){
                    System.out.print(arr[i][j]+" ");
                }
                System.out.println();//new line
            }
        }
}
```

Output
0 1 2
3 4 5 6
7 8

Matrix Multiplication in Java using Array

Matrix Multiplication in Java using Array

```java
public class MatrixMultiplicationExample{

public static void main(String args[]){
//creating two matrices
int a[][]={{1,1,1},{2,2,2},{3,3,3}};
int b[][]={{1,1,1},{2,2,2},{3,3,3}};

//creating another matrix to store the
multiplication of two matrices
int c[][]=new int[3][3];  //3 rows and 3
columns
```

```java
//multiplying and printing multiplication of 2
matrices
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
c[i][j]=0;
for(int k=0;k<3;k++)
{
c[i][j]+=a[i][k]*b[k][j];
}//end of k loop
System.out.print(c[i][j]+" ");  //printing
 matrix element
}//end of j loop
System.out.println();//new line
}
}}
```
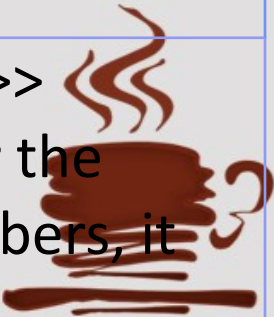
```
Output
6 6 6
12 12 12
18 18 18
```

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java.

Comparison Index	C++	Java
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
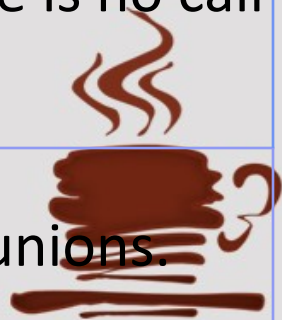
Comparison Index	C++	Java
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.

Comparison Index	C++	Java
Inheritance Tree	C++ always creates a new inheritance tree.	Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.