Programming in Java [TCSCCS0302T]

~ by Asst. Prof. Shivkumar Chandey
Department of Computer Science
Thakur College of Science & Commerce (Autonomous)

Objective

- The objective of this course is to teach the learner how to use building blocks of Core Java Language to implement a real-world and functional CUI and GUI based Application by using Object Oriented paradigm.

Syllabus of Unit I

Unit I	Introduction to Java: Java Architecture and It's components, Java Platform, Java Availability, JDK, Collection. Working with Java Development Kit [JDK]: Downloading JDK 8, Setting the path environment, variable[A], Exploring JDK Directories, Tools available in JDK[A], Text Editor or an IDE. Understanding Java Programs: Java Application[A] Basic Fundamentals: Statement, White space, Case Sensitivity, Identifiers, Keywords, Comments, Braces and Code blocks, Variables[A], Data types[A], Object Reference Types[A], String Handling[A], Arrays[A]. Operators[A], Control Flow Statements[A], Iterations[A].

Expected Learning Outcomes

1. Learner will get a strong knowledge of the structure and model of the Java programming language
2. Implement Various Java Applications and Software by using concepts of Java programming language.
3. Develop a strong foundation of OOPs Concept, AWT, Threads, Java APIs.
4. Evaluate user requirements and Design responsive GUI based applications

What is Java?

- Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.
- Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995.
- *James Gosling* is known as the father of Java.
- Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the Oak name to Java.

Platform

- Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Some of the Applications of Java

- According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:
  - Desktop Applications such as acrobat reader, media player, antivirus, etc.
  - Web Applications such as irctc.co.in, javatpoint.com, etc.
  - Enterprise Applications such as banking applications.
  - Mobile
  - Embedded System
  - Smart Card
  - Robotics
  - Games, etc.

Types of Java Applications

1) Standalone Application: Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

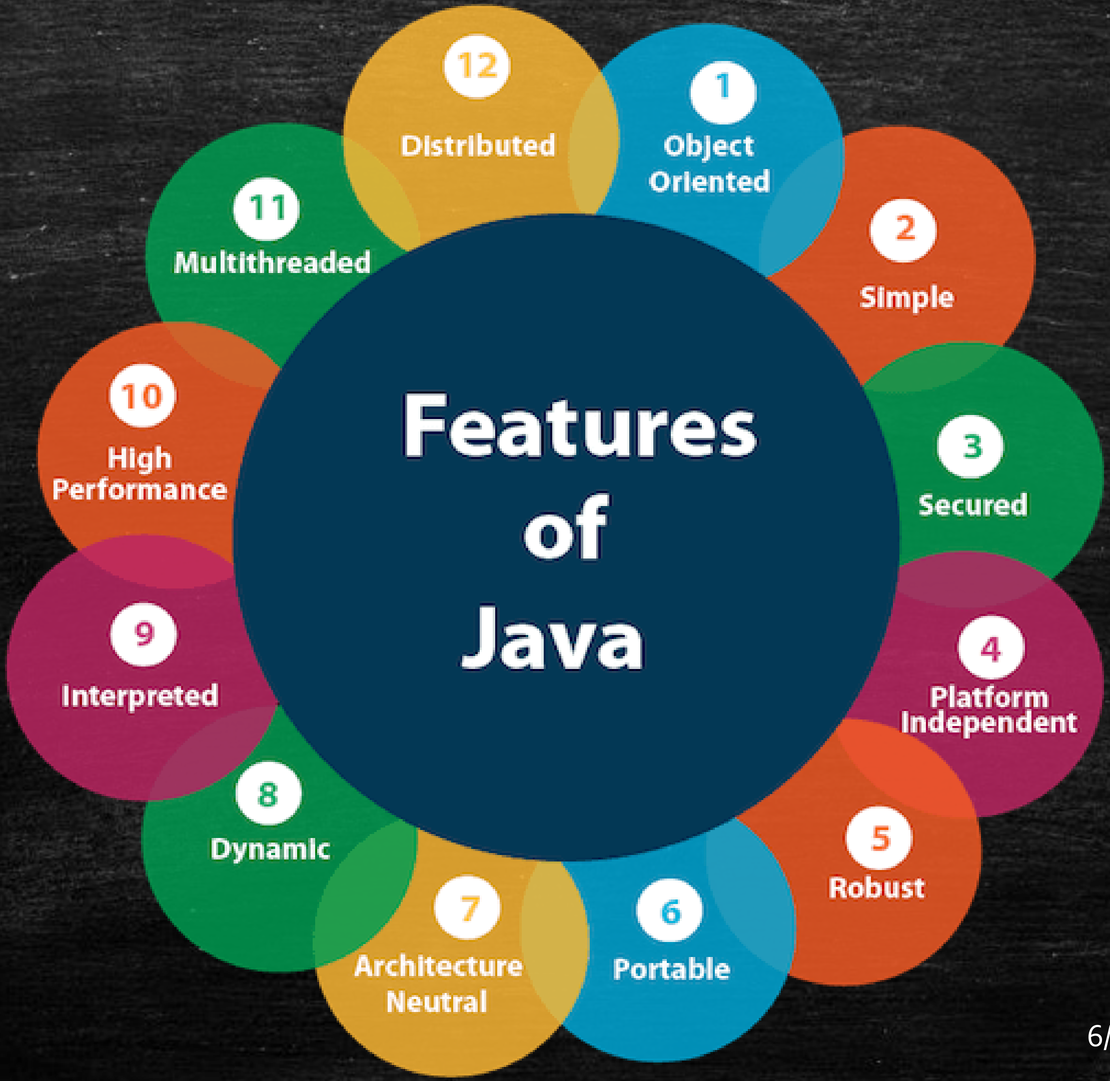
2) Web Application: An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

Types of Java Application (Cont.)

3) Enterprise Application: An application that is distributed in nature, such as banking applications, etc. is called enterprise application. It has advantages of the high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application: An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Characteristics of Java



Characteristics of Java (Cont.)

1) Object- oriented:

- Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.
- Basic concepts of OOPs are:
  - Object, Class, Inheritance, Polymorphism, Abstraction, Encapsulation.

Characteristics of Java (Cont.)

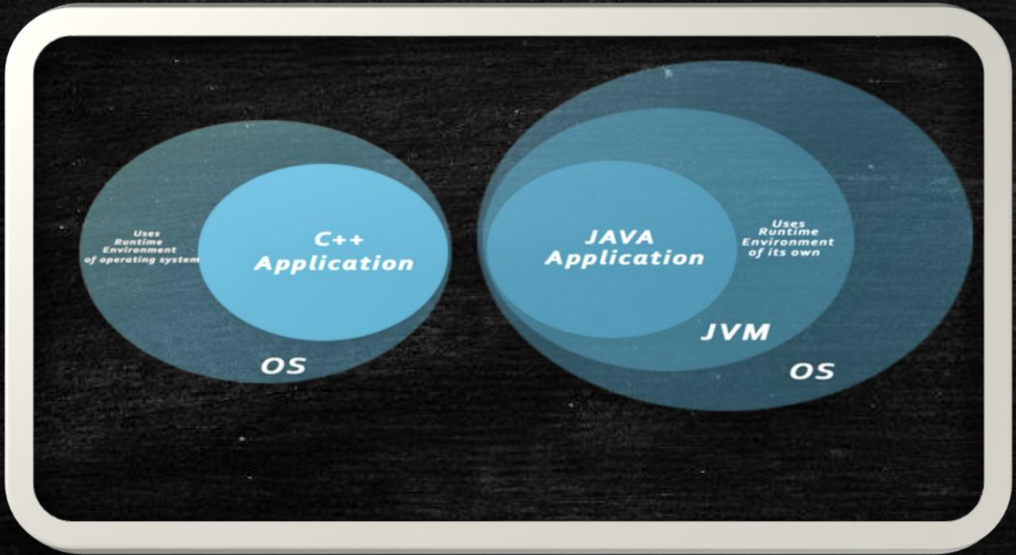
2) Simple:

- Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:
  - Java syntax is based on C++ (so easier for programmers to learn it after C++).
  - Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
  - There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Characteristics of Java (Cont.)

3) Secured:

- Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:
  - **No explicit pointer**
  - **Java Programs run inside a virtual machine sandbox**


Uses Runtime Environment of operating system
C++ Application
OS
JAVA Application
Uses Runtime Environment of its own
JVM
OS

Characteristics of Java (Cont.)

3) Secured (continued..):

- **Class loader: It** is the subsystem of JVM that is used to load class files.
- **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.
- Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

Characteristics of Java (Cont.)

4)Platform Independent:

- Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language.
- There are two types of platforms software-based and hardware-based. Java provides a software-based platform.
- The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:
  - Runtime Environment
  - API (Application Programming Interface)

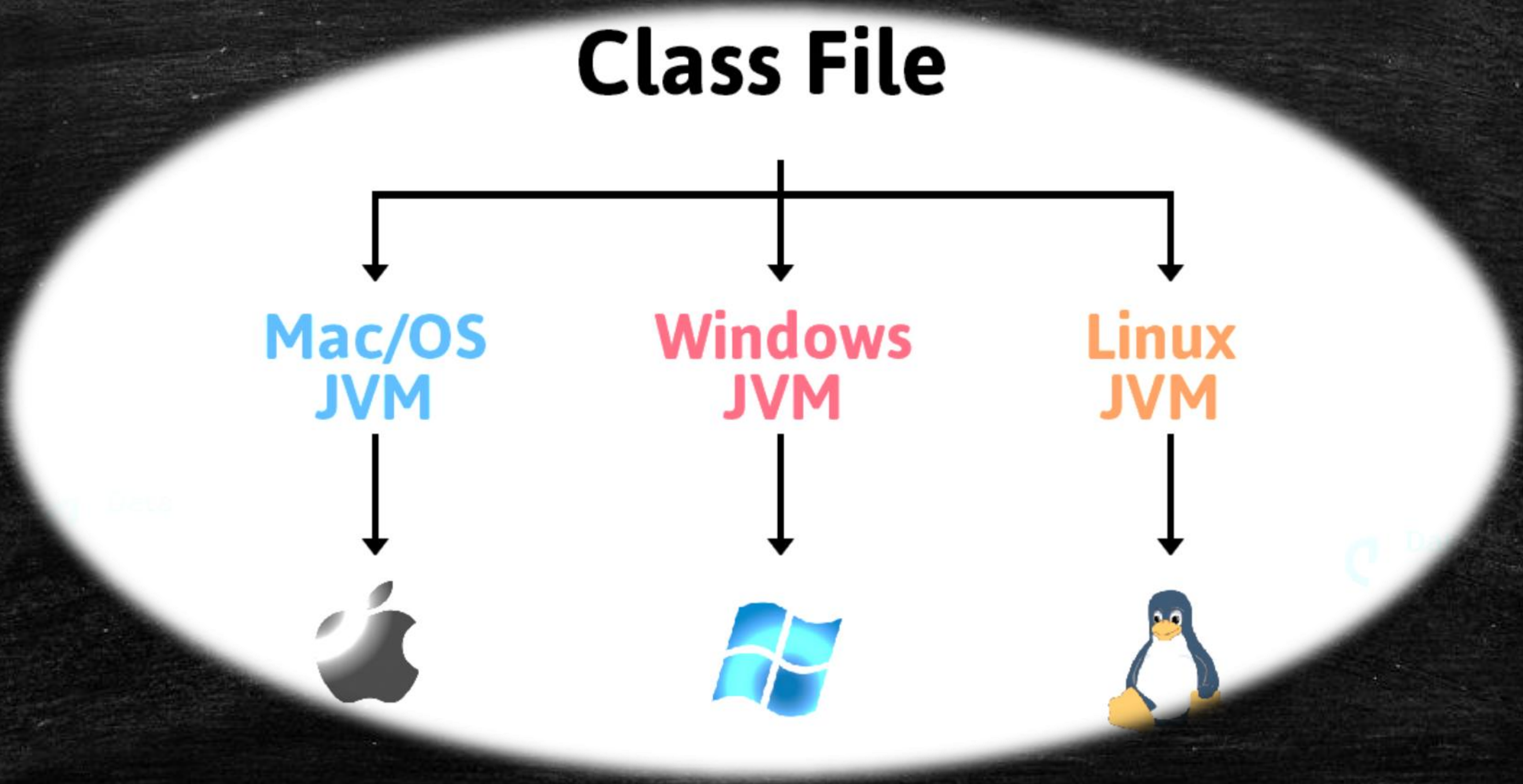
Characteristics of Java (Cont.)

4)Platform Independent (Continued...):

- Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode.
- This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

Characteristics of Java (Cont.)

4)Platform Independent (Continued…):



Characteristics of Java (Cont.)

5) Robust:

- Robust simply means strong. Java is robust because:
  - It uses strong memory management.
  - There is a lack of pointers that avoids security problems.
  - There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
  - There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Characteristics of Java (Cont.)

6) Portable :

- Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

Characteristics of Java (Cont.)

7) Architecture Neutral :

- Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
- In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Characteristics of Java (Cont.)

8) Dynamic :

- Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.
- Java supports dynamic compilation and automatic memory management (garbage collection).

Characteristics of Java (Cont.)

9) Interpreted:

- Java uses compiler and interpreter both. 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.
- Interpreter reads bytecode stream then execute the instructions.

Characteristics of Java (Cont.)

10) High Performance :

- Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Characteristics of Java (Cont.)

11) Multithreaded:

- A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Characteristics of Java (Cont.)

12) Distributed :

- Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

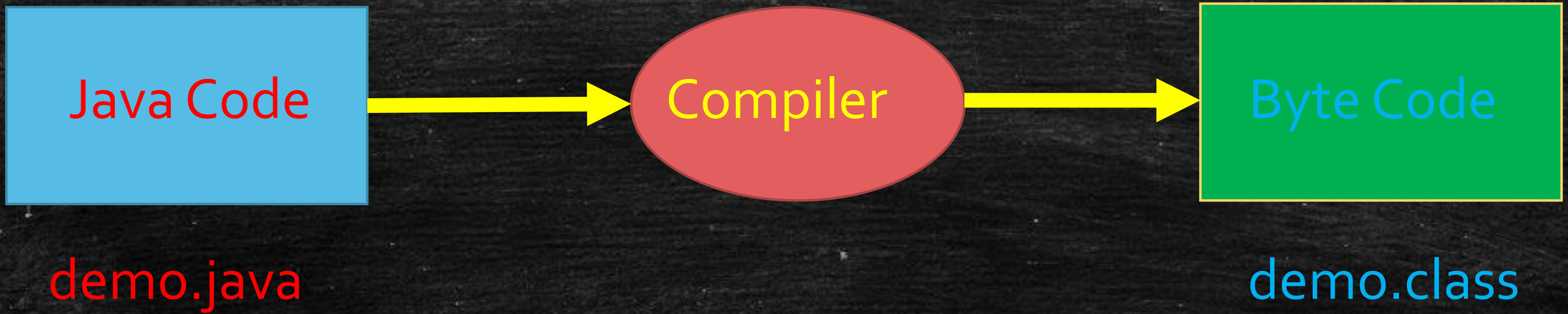
Execution and Compilation

- Install the JDK if you don't have installed it, https://www.oracle.com/in/java/technologies/javase/javase-jdk8-downloads.html and install it.
- Set path of the jdk/bin directory.
- Create the java program.
- Compile and run the java program.

Steps to Compile & Execute Java Program

- To Compile:
  - javac filename.java
  - e.g. javac demo.java
- To Execute:
  - java filename
  - e.g. java demo

Compilation Process (javac tool)



Parameters used in First Java Program

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create an object to invoke the main method. So it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.

Parameters used in First Java Program (cont.)

- **main** represents the starting point of the program.
- **String[] args** is used for command line argument. We will learn it later.
- **System.out.println()** is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class.

Java Architecture and It’s components

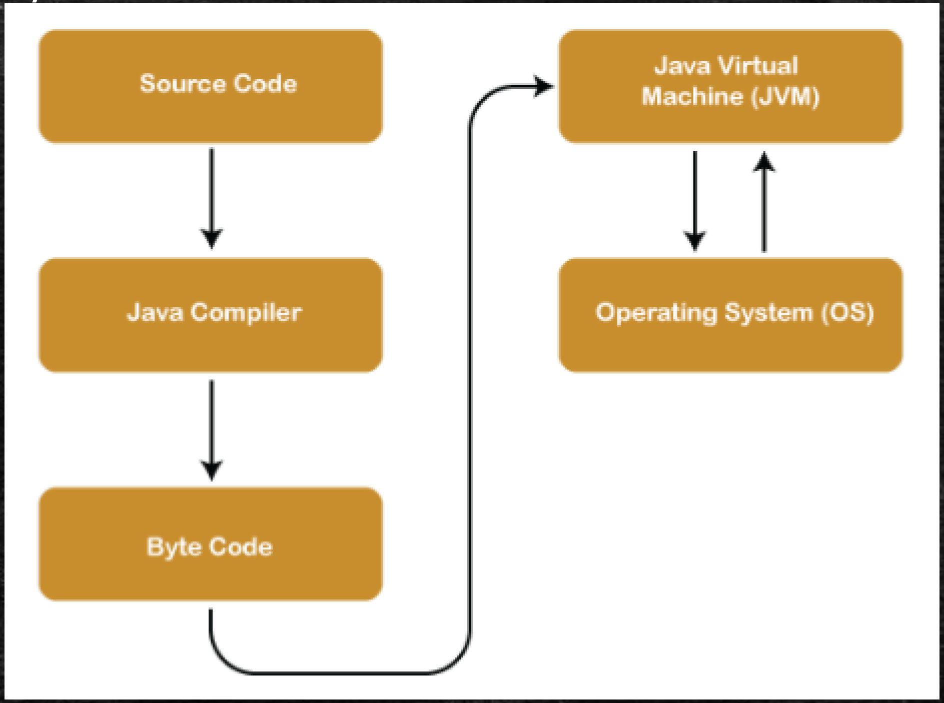
Java Architecture

- Java Architecture is a collection of components, i.e., JVM, JRE, and JDK. It integrates the process of interpretation and compilation. It defines all the processes involved in creating a Java program. Java Architecture explains each and every step of how a program is compiled and executed.
- Java Architecture can be explained by using the following steps:

1. There is a process of compilation and interpretation in Java.
2. Java compiler converts the Java code into byte code.
3. After that, the JVM converts the byte code into machine code.
4. The machine code is then executed by the machine.

Java Architecture

- The following figure represents the Java Architecture in which each step is elaborate graphically.



Components of Java Architecture

- The Java architecture includes the three main components:

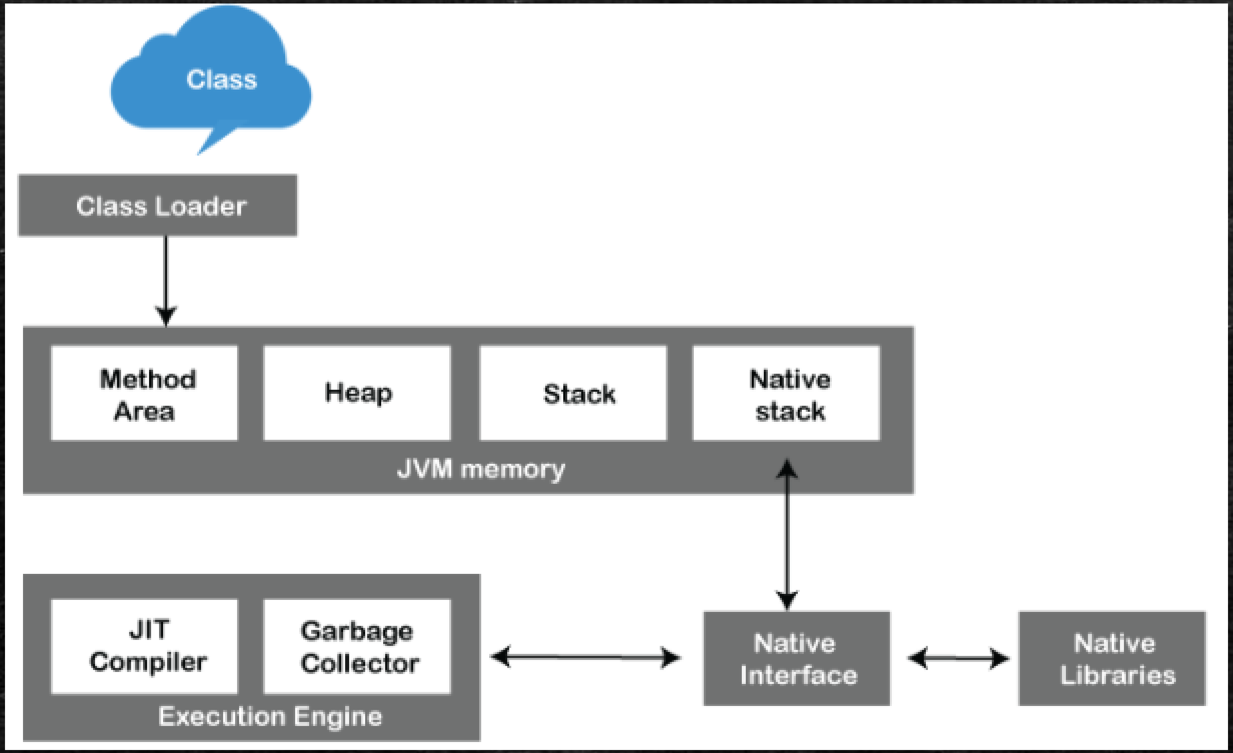
1. Java Virtual Machine (JVM)
2. Java Runtime Environment (JRE)
3. Java Development Kit (JDK)

Java Virtual Machine

- The main feature of Java is WORA. WORA stands for Write Once Run Anywhere. The feature states that we can write our code once and use it anywhere or on any operating system.
- Our Java program can run any of the platforms only because of the Java Virtual Machine. It is a Java platform component that gives us an environment to execute java programs. JVM's main task is to convert byte code into machine code.
- JVM, first of all, loads the code into memory and verifies it. After that, it executes the code and provides a runtime environment. Java Virtual Machine (JVM) has its own architecture, which is given below:

JVM Architecture

- JVM is an abstract machine that provides the environment in which Java bytecode is executed. The falling figure represents the architecture of the JVM.



JVM Architecture

- ClassLoader:
  - ClassLoader is a subsystem used to load class files. ClassLoader first loads the Java code whenever we run it.
- Class Method Area:
  - In the memory, there is an area where the class data is stored during the code's execution. Class method area holds the information of static variables, static methods, static blocks, and instance methods.
- Heap:
  - The heap area is a part of the JVM memory and is created when the JVM starts up. Its size cannot be static because it increase or decrease during the application runs.

JVM Architecture

- Stack:
  - It is also referred to as thread stack. It is created for a single execution thread. The thread uses this area to store the elements like the partial result, local variable, data used for calling method and returns etc.
- Native Stack:
  - It contains the information of all the native methods used in our application.
- Execution Engine:
  - It is the central part of the JVM. Its main task is to execute the byte code and execute the Java classes. The execution engine has three main components used for executing Java classes.

JVM Architecture

- Interpreter:
  - It converts the byte code into native code and executes. It sequentially executes the code. The interpreter interprets continuously and even the same method multiple times. This reduces the performance of the system, and to solve this, the JIT compiler is introduced.
- JIT Compiler:
  - The Just-In-Time (JIT) compiler is a key component of the OpenJ9 VM that improves the performance of Java applications by compiling platform-neutral Java bytecode into native machine code at run time.
  - Without the JIT, the VM has to interpret the bytecodes itself - a process that requires extra CPU and memory

JVM Architecture

- Garbage Collector:
  - The garbage collector is used to manage the memory, and it is a program written in Java. It works in two phases, i.e., Mark and Sweep. Mark is an area where the garbage collector identifies the used and unused chunks of memory. The Sweep removes the identified object from the Mark
- Java Native Interface
  - Java Native Interface works as a mediator between Java method calls and native libraries.

Components of Java Architecture

- Java Runtime Environment
  - It provides an environment in which Java programs are executed. JRE takes our Java code, integrates it with the required libraries, and then starts the JVM to execute it.
- Java Development Kit
  - It is a software development environment used in the development of Java applications and applets. Java Development Kit holds JRE, a compiler, an interpreter or loader, and several development tools in it.
- These are three main components of Java Architecture. The execution of a program is done with all these three components.

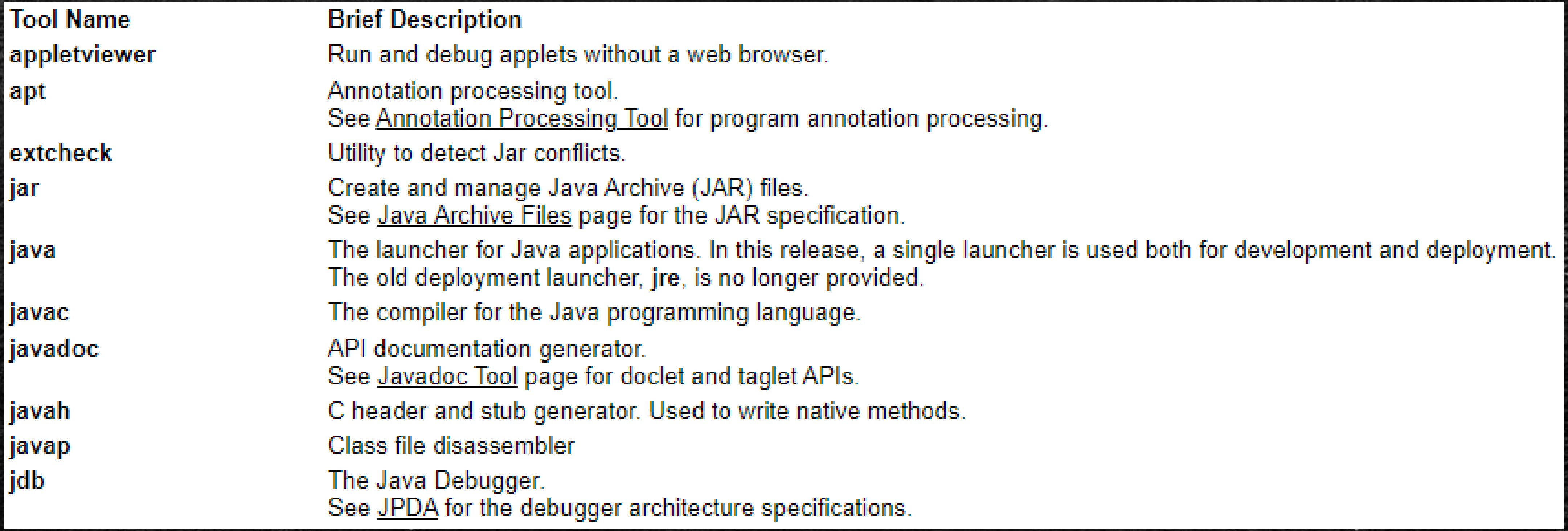
Tools available in JDK

Standard JDK Tools and Utilities

- Basic Tools (appletviewer, apt, extcheck, jar, java, javac, javadoc, javah, javap, jdb)
- Security Tools (keytool, jarsigner, policytool, kinit, klist, ktab)
- Internationalization Tools (native2ascii)
- Remote Method Invocation (RMI) Tools (rmic, rmiregistry, rmid, serialver)
- Java IDL and RMI-IIOP Tools (tnameserv, idlj, orbd, servertool)
- Java Deployment Tools (javafxpackager, pack200, unpack200)
- Java Web Start Tools (javaws)
- Java Troubleshooting, Profiling, Monitoring and Management Tools (jcmd, jconsole, jmc, jvisualvm)
- Java Web Services Tools (schemagen, wsgen, wsimport, xjc)

Tools available in JDK

- These tools are the foundation of the JDK. They are the tools you use to create and build applications.



Tool Name	Brief Description
appletviewer	Run and debug applets without a web browser.
apt	Annotation processing tool. See Annotation Processing Tool for program annotation processing.
extcheck	Utility to detect Jar conflicts.
jar	Create and manage Java Archive (JAR) files. See Java Archive Files page for the JAR specification.
java	The launcher for Java applications. In this release, a single launcher is used both for development and deployment. The old deployment launcher, **jre**, is no longer provided.
javac	The compiler for the Java programming language.
javadoc	API documentation generator. See Javadoc Tool page for doclet and taglet APIs.
javah	C header and stub generator. Used to write native methods.
javap	Class file disassembler
jdb	The Java Debugger. See JPDA for the debugger architecture specifications.

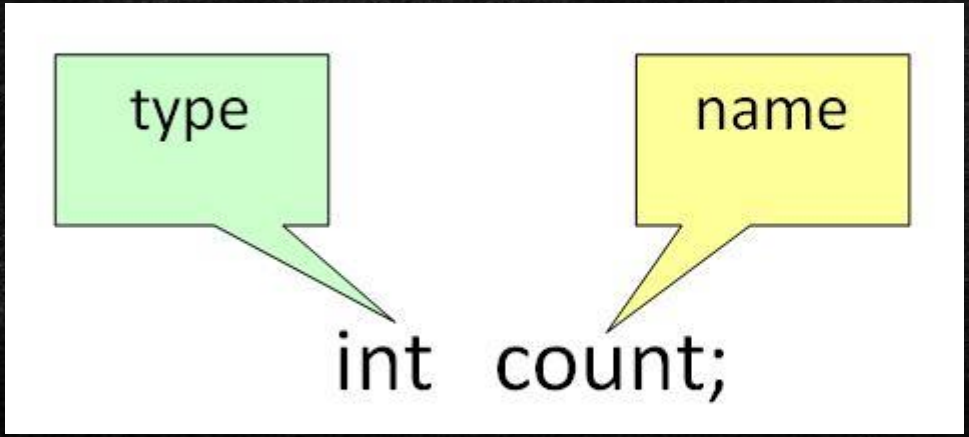
- Click here to know more, https://docs.oracle.com/javase/7/docs/technotes/tools/

Java Variables

- A variable is a container which holds the value while the Java program is executed.
- A variable is assigned with a data type.
- Variable is a name of memory location.
- There are three types of variables in java: local, instance and static.
- There are two types of data types in Java: primitive and non-primitive.
- It is a combination of "vary + able" which means its value can be changed.

Variable Declaration

- You must declare all variables before they can be used. Following is the basic form of a variable declaration
- **Syntax:**
  - **data type variable [= value] [,variable [= value] ...] ;**
- To declare a variable, you must specify the data type & give the variable a unique name.

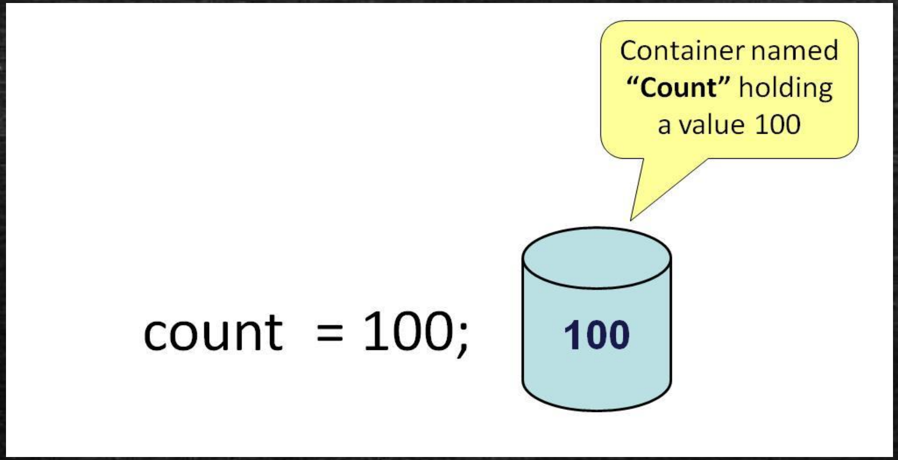


Examples of other Valid Declarations

- int num1,num2,num3;
- float pi;
- double d;
- char a;

Variable Initialization

- To initialize a variable, you must assign it a valid value.
- pi=3.14f;
- do=15.11d;
- c=‘a’;
- byte b=10;



- You can combine variable declaration and initialization.
  - int age=50;

Types of Variables

- There are three types of variables in Java:

1. local variable
2. instance variable
3. static variable

1) Local Variable

- A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
- A local variable cannot be defined with "static" keyword.

2) Instance Variable

- A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.
- It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

- A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class.
- Memory allocation for static variables happens only once when the class is loaded in the memory.

```
//Example to understand the types of variables in java
class vardemo
{
        static int number = 1; //static variable
        int ivar = 2; //instance variable
        void method()
        {
                int a =3 ; //local variable
        }
}
```

More on Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

More on Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables can be declared in class level before or after use.

More on Instance Variables (Cont…)

- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level).
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.

Class/Static Variables (Cont…)

- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case.
- If the static variables are not public and final, the naming syntax is the same as instance and local variables.

Data Types in Java

There are majorly two types of languages.

1) **Statically typed language** where each variable and expression type is already known at compile time. Once a variable is declared to be of a certain data type, it cannot hold values of other data types.
Example: C, C++, Java.

2) The other is **Dynamically typed languages.** These languages can receive different data types over time.
Example: Ruby, Python

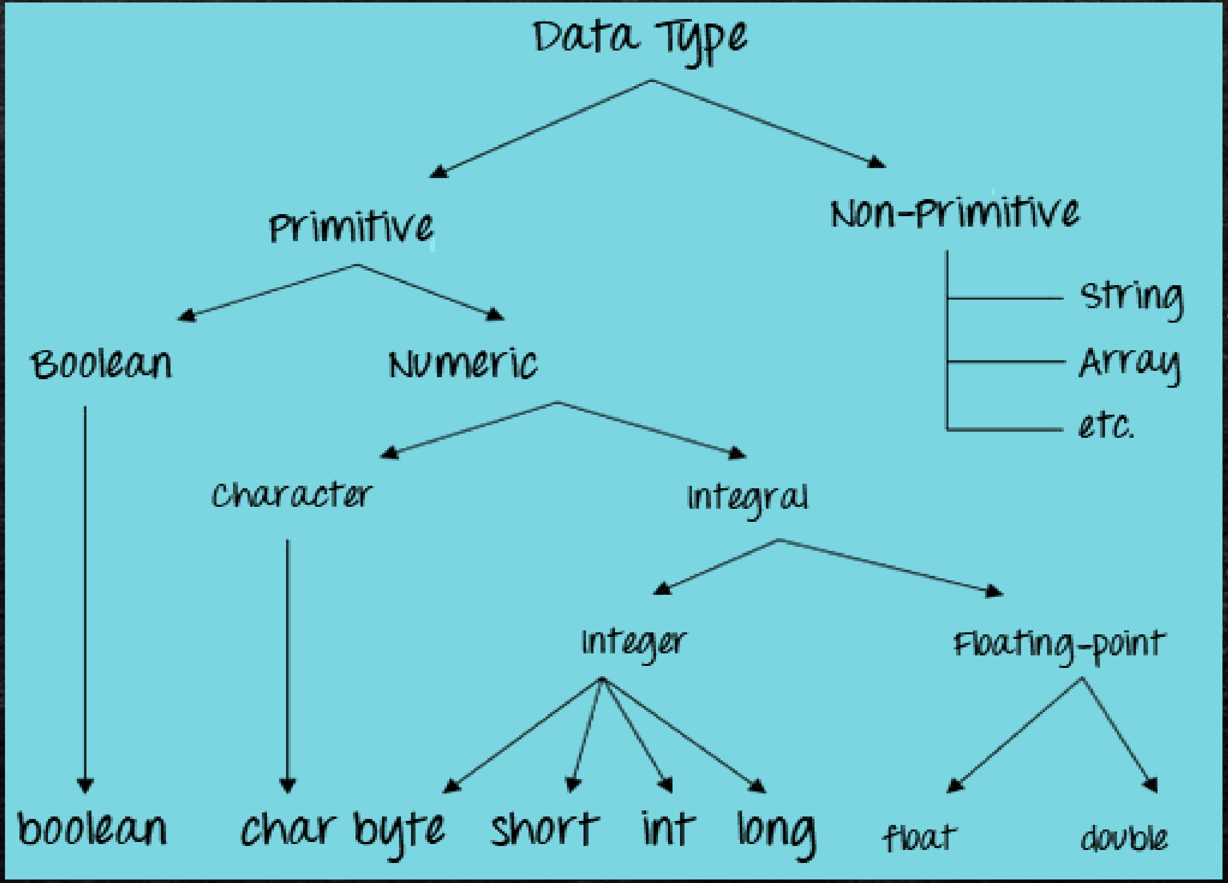
Data types in Java

- Java is **statically typed and also a strongly typed language** because:
  - In Java, each type of data (such as integer, character, hexadecimal, packed decimal, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types.

Data Types in Java

- Data types classify the different values to be stored in the variable. In java, there are two types of data types:

1) Primitive Data Types
2) Non-primitive Data Types



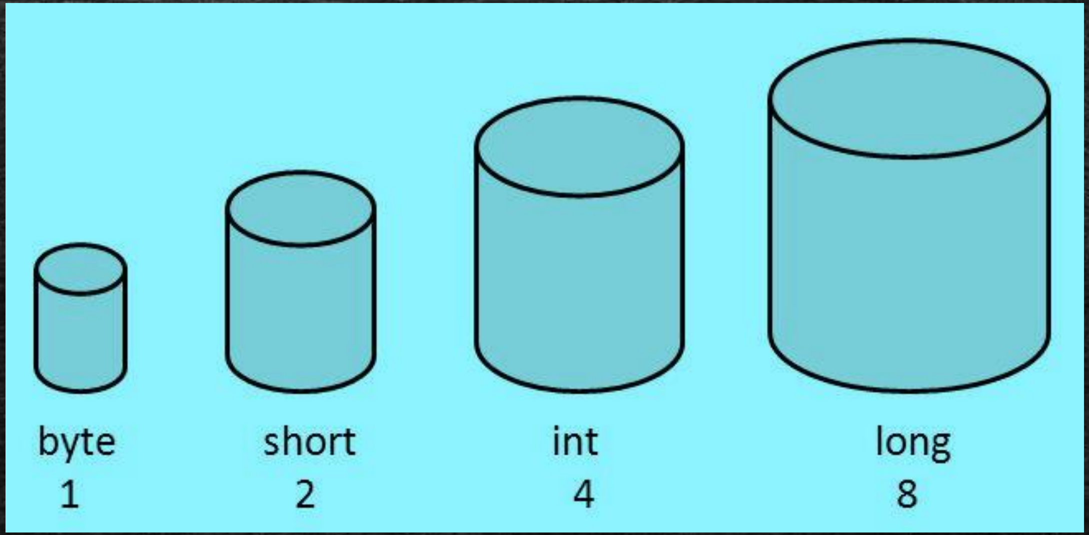
Primitive Data Types

- Primitive Data Types are predefined and available within the Java language. Primitive values do not share state with other primitive values.
- There are 8 primitive types:
  - byte
  - short
  - int
  - long
  - char
  - float
  - double and
  - boolean

Primitive Data Types

- **Integer Data types:**
  - byte (1 byte)
  - short (2 bytes)
  - int (4 bytes)
  - long (8 bytes)

- **Floating Data type:**
  - float (4 bytes)
  - double (8 bytes)



Primitive Data Types

- **Textual Data type:**
  - char (2 bytes)

- **Logical Data type:**
  - boolean (1 bit) (true/false)

Default size of primitive datatypes

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

Points to Remember

- All numeric data types are signed(+/-).
- The size of data types remain the same on all platforms (standardized)
- char data type in Java is 2 bytes because it uses **UNICODE** character set. By virtue of it, Java supports internationalization. UNICODE is a character set which covers all known scripts and language in the world

Java Keywords

- Java keywords are also known as reserved words.
- Keywords are particular words that act as a key to a code.
- These are predefined words by Java so they cannot be used as a variable or object name or class name.

Keywords in Java

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	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

List of Java Keywords

Primitive Types and void	Modifiers	Declarations	Control Flow	Miscellaneous
1. boolean	1. public	1. class	1. if	1. this
2. byte	2. protected	2. interface	2. else	2. new
3. char	3. private	3. enum	3. try	3. super
4. short	4. abstract	4. extends	4. catch	4. import
5. int	5. static	5. implements	5. finally	5. instanceof
6. long	6. final	6. package	6. do	6. null
7. float	7. transient	7. throws	7. while	7. true
8. double	8. volatile		8. for	8. false
9. void	9. synchronized		9. continue	9. strictfp
	10. native		10. break	10. assert
			11. switch	11. _ (underscore)
			12. case	12. goto
			13. default	13. const
			14. throw	
			15. return	

Java Operators

- Operators are special symbols (characters) that carry out operations on operands (variables and values).
- There are many types of operators in Java which are given below:
  - Arithmetic Operators,
  - Relational Operators,
  - Logical Operators,
  - Assignment Operators,
  - Increment and Decrement Operators,
  - Conditional Operators,
  - Bitwise Operators and
  - Special Operators.

Arithmetic Operators

- Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.
- Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators –

Assume integer variable A holds 10 and variable B holds 20, then – i.e. A=10, B=20

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

Relational Operator

- **Relational operator** that tests or defines some kind of relation between two entities.
- These include numerical equality (e.g., 5 = 5) and inequalities (e.g., 4 ≥ 3).

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

- Assume Boolean variables A holds true and variable B holds false, then –

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

Assignment Operators

- Assignment operators are used to assign values to variables.

In the example below, we use the **assignment** operator (=) to assign the value **10** to a variable called **x**:

  - int x=10;

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Increment/Decrement Operators

1. It is one of the variation of "Arithmetic Operator".
2. Increment and Decrement Operators are Unary Operators.
3. Unary Operator Operates on One Operand.
4. Increment Operator is Used to Increment Value Stored inside Variable on which it is operating.
5. Decrement Operator is used to decrement value of Variable by 1 (default).

Increment/Decrement Operators

```
a = 5
++a;            // a becomes 6
a++;            // a becomes 7
--a;            // a becomes 6
a--;            // a becomes 5
```

Conditional Operator(? :)

- Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as –
- variable x = (expression) ? value if true : value if false

Bitwise Operator

- Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.
- Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows –
  - a = 0011 1100
  - b = 0000 1101
  - -----------------
  - a&b = 0000 1100
  - a|b = 0011 1101
  - a^b = 0011 0001
  - ~a = 1100 0011

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Special Operators

- The **dot (.)** operator is used to select members of a class or object instance
- The new operator is used to create objects out of a class.
- The instanceof operator is used to test whether an object is an instance of a class.

Control Flow Statements

Selection Statements/ Decision Making in Java (if, if-else, switch, break, continue, jump):

- Decision Making in programming is similar to decision making in real life. In programming also we face some situations where we want a certain block of code to be executed when some condition is fulfilled.
- A programming language uses control statements to control the flow of execution of program based on certain conditions.
- These are used to cause the flow of execution to advance and branch based on changes to the state of a program.

Selection statements

- Java If-else Statement
- The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in Java.
  - if
  - If-else
  - Nested-if
  - if-else-if ladder
  - switch-case
  - jump: break, continue, return

  These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

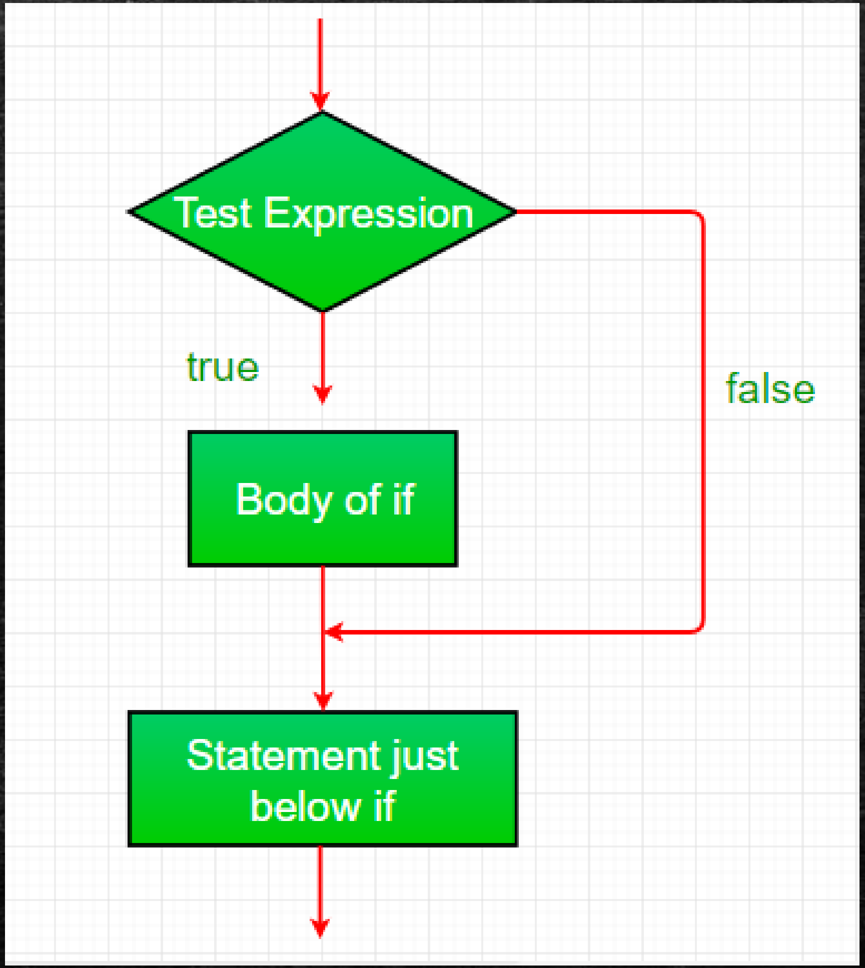
Java if Statement

- The Java if statement tests the condition. It executes the *if block* if condition is true.
- Syntax:

```
    if(condition)
    {
        //code to be executed  if stmt true
    }
```

- Here, **condition** after evaluation will be either true or false. if statement accepts boolean values – if the value is true then it will execute the block of statements under it.

Flowchart of if Statement



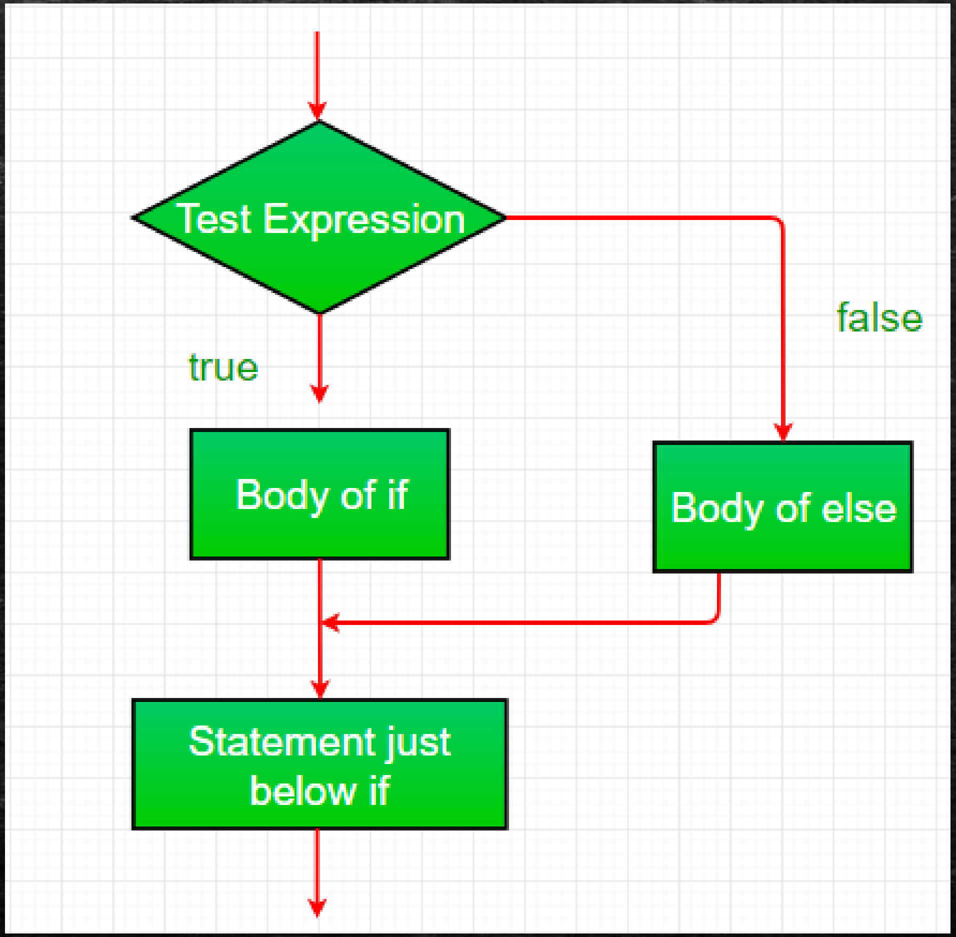
```
//the use of if statement
public class IfExample
{
        public static void main(String[] args)
        {
                int age=20;  //defining an 'age' variable
                if(age>18)    //checking the age
                {
                        System.out.print("Age is greater than 18");
                }
        }
}
```

if-else Statement

- The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.
- **Syntax:**

```
if(condition)
{
        //code if condition is true
}
else
{
        //code if condition is false
}
```

Flowchart of if-else statement



```
//Program to demonstrate if-else statement.
public class IfElseExample
 {
        public static void main(String[] args)
        {
                int number=13;
                if(number%2==0) //Check if the number is divisible by 2 or not
                {
                        System.out.println("even number");
                }
                else
                {
                        System.out.println("odd number");
                }
        }
}
```

Using Ternary Operator

- We can also use ternary operator (? :) to perform the task of if...else statement. It is a shorthand way to check the condition. If the condition is true, the result of ? is returned. But, if the condition is false, the result of : is returned.
- Syntax:

```
result = testStatement ? value1 : value2;
```

```
//Program to demonstrate Ternary Operator

public class IfElseTernaryExample
{
        public static void main(String[] args)
        {
                int number=13;
                //Using ternary operator
                String output=(number%2==0)?"even number":"odd number";
                System.out.println(output);
        }
}
```

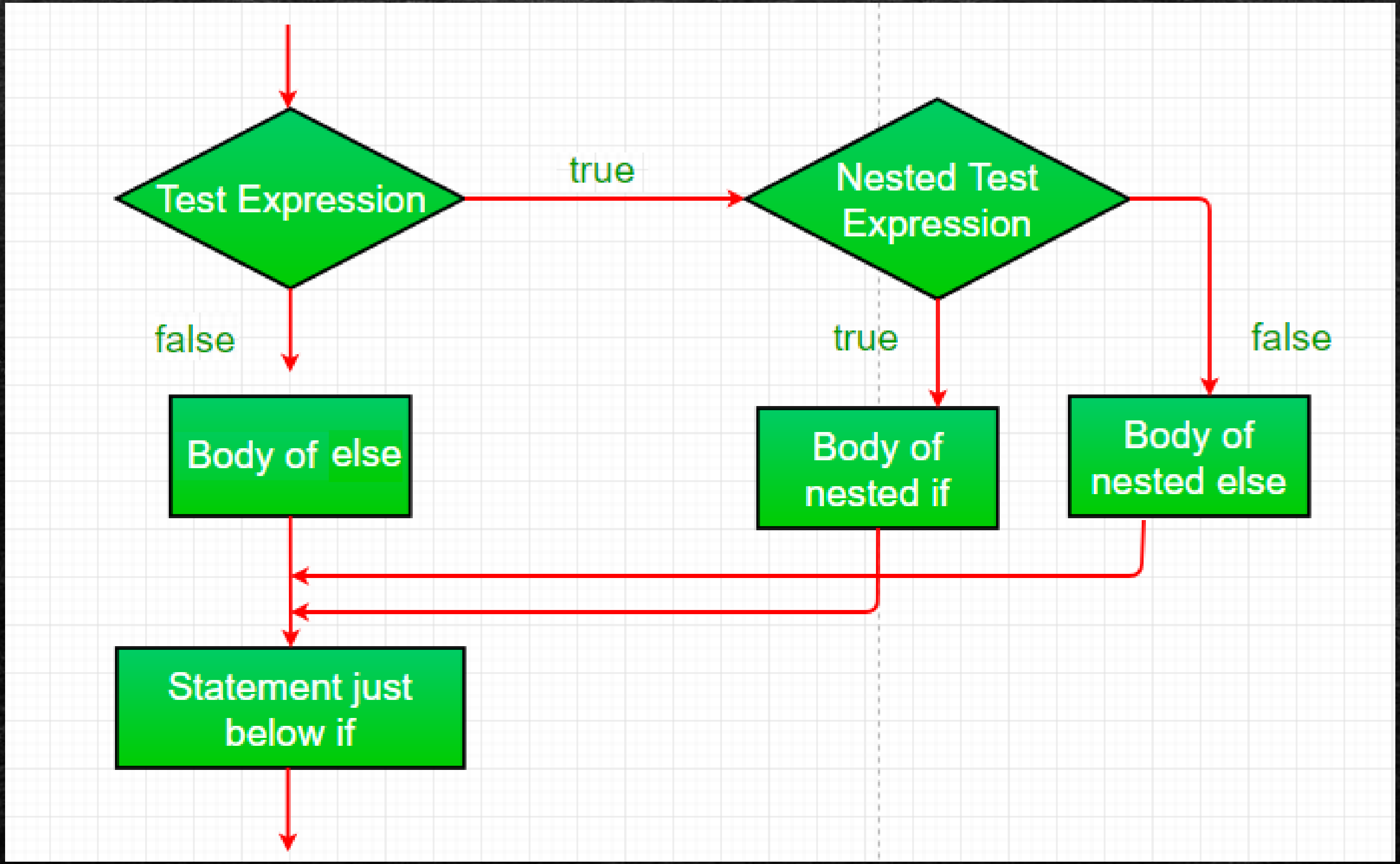
nested-if:

- A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement. Yes, java allows us to nest if statements within if statements.
- i.e, we can place an if statement inside another if statement.

Syntax:

```
if (condition1)
{
        // Executes when condition1 is true
        if (condition2)
        {
                // Executes when condition2 is true
        }
}
```

Flowchart of nested-if



```
//Program to demonstrate the use of nested-if statement.
public class nestedIfDemo
{
	public static void main(String[] args)
	{
		//Creating two variables for age and weight
		int age=20;
		int weight=80;
		//applying condition on age and weight
		if(age>=18)
		{
			if(weight>50)
			{
				System.out.println("You are eligible to donate blood");
			}
		}
	}
}
```

if-else-if ladder Statement

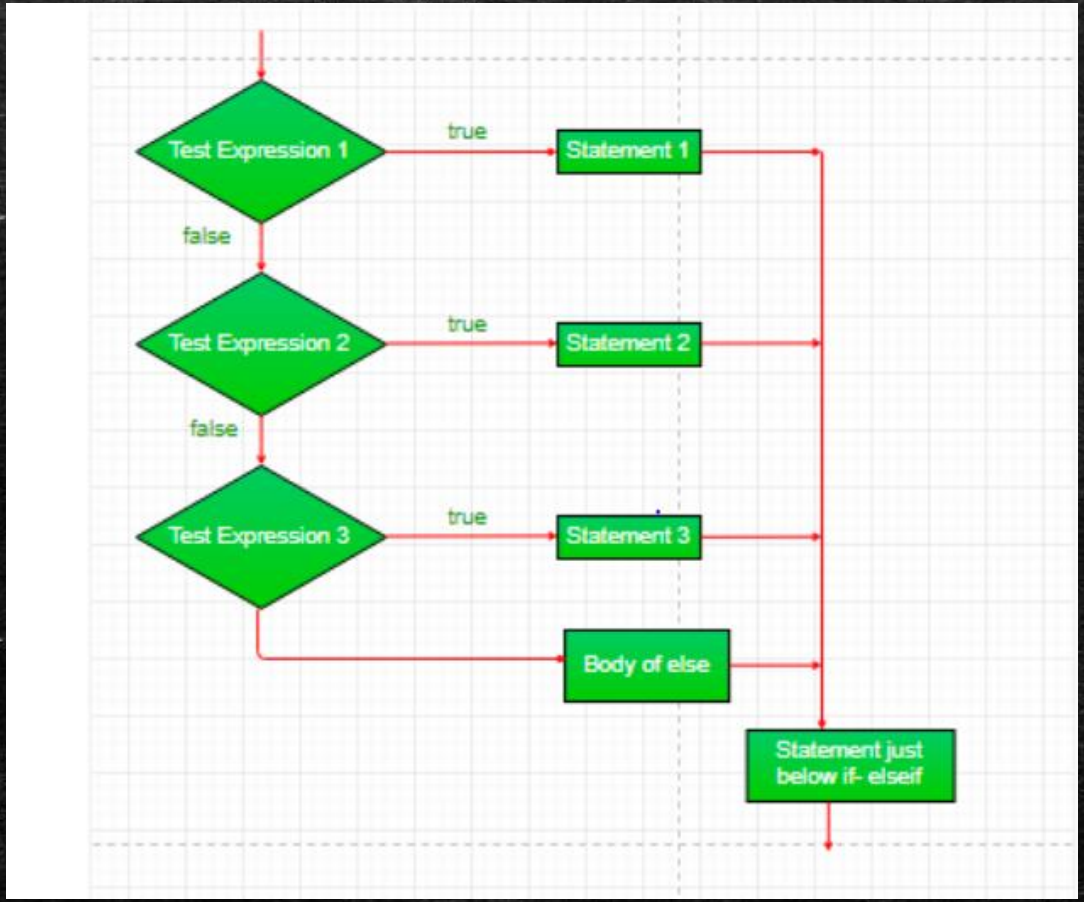
- The if-else-if ladder statement executes one condition from multiple statements.
- Here, a user can decide among multiple options.The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

if-else-if ladder Statement:

Syntax:

```
if(condition1)
{
        //code to be executed if condition1 is true
}
else if(condition2)
{
        //code to be executed if condition2 is true
}
else if(condition3)
{
        //code to be executed if condition3 is true
}
...
else
{
        //code to be executed if all the conditions are false
}
```

Flowchart of if-else-if ladder



```
// Java program to illustrate if-else-if ladder
class ifelseifDemo
{
		public static void main(String args[])
		{
				int i = 20;
				if (i == 10)
				{
						System.out.println("i is 10");
				}
				else if (i == 15)
				{
						System.out.println("i is 15");
				}
				else if (i == 20)
				{
						System.out.println("i is 20");
				}
				else
				{
						System.out.println("i is not present");
				}
		}
}
```

Switch statement

- The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.
- The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use strings in the switch statement.
- In other words, the switch statement tests the equality of a variable against multiple values.

Syntax of switch-case:

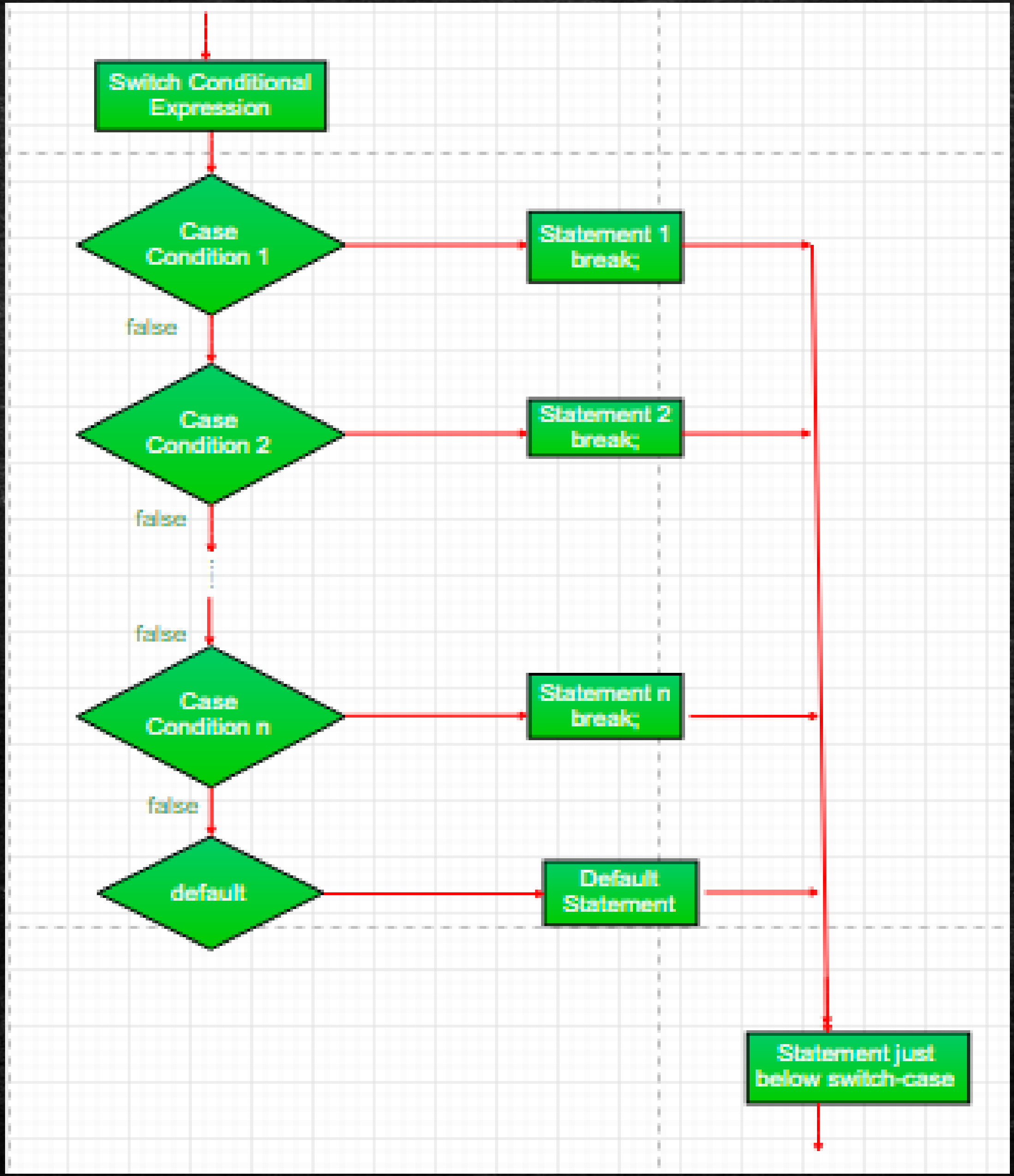
```
switch(expression)
{
        case value1:
                //code to be executed;
                break;  //optional
        case value2:
                 //code to be executed;
                 break;  //optional
                ……
        default:
                code to be executed if all cases are not matched;
}
```

Points to Remember

- There can be *one or N number of case values* for a switch expression.
- The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
- The case values must be *unique*. In case of duplicate value, it renders compile-time error.
- The Java switch expression must be of *byte, short, int, long (with its Wrapper type), enums and string*.

Points to Remember

- Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a *default label* which is optional.


Switch Conditional Expression
Case Condition 1
Statement 1 break;
false
Case Condition 2
Statement 2 break;
false
false
Case Condition n
Statement n break;
false
default
Default Statement
Statement just below switch-case

```
public class SwitchExample
{
        public static void main(String[] args)
        {
                //Declaring a variable for switch expression
                int number=20;
                //Switch expression
                switch(number)
                {
                        //Case statements
                        case 10:
                                System.out.println("10");
                                break;
                        case 20:
                                System.out.println("20");
                                break;
                        case 30:
                                System.out.println("30");
                                break;
                        //Default case statement
                        default:
                                System.out.println("Not in 10, 20 or 30");
                }
        }
}
```

Jump Statements

- Java supports three jump statement:
  - **break,**
  - **continue** and
  - **return**.
- These three statements transfer control to other part of the program.

Break Statement

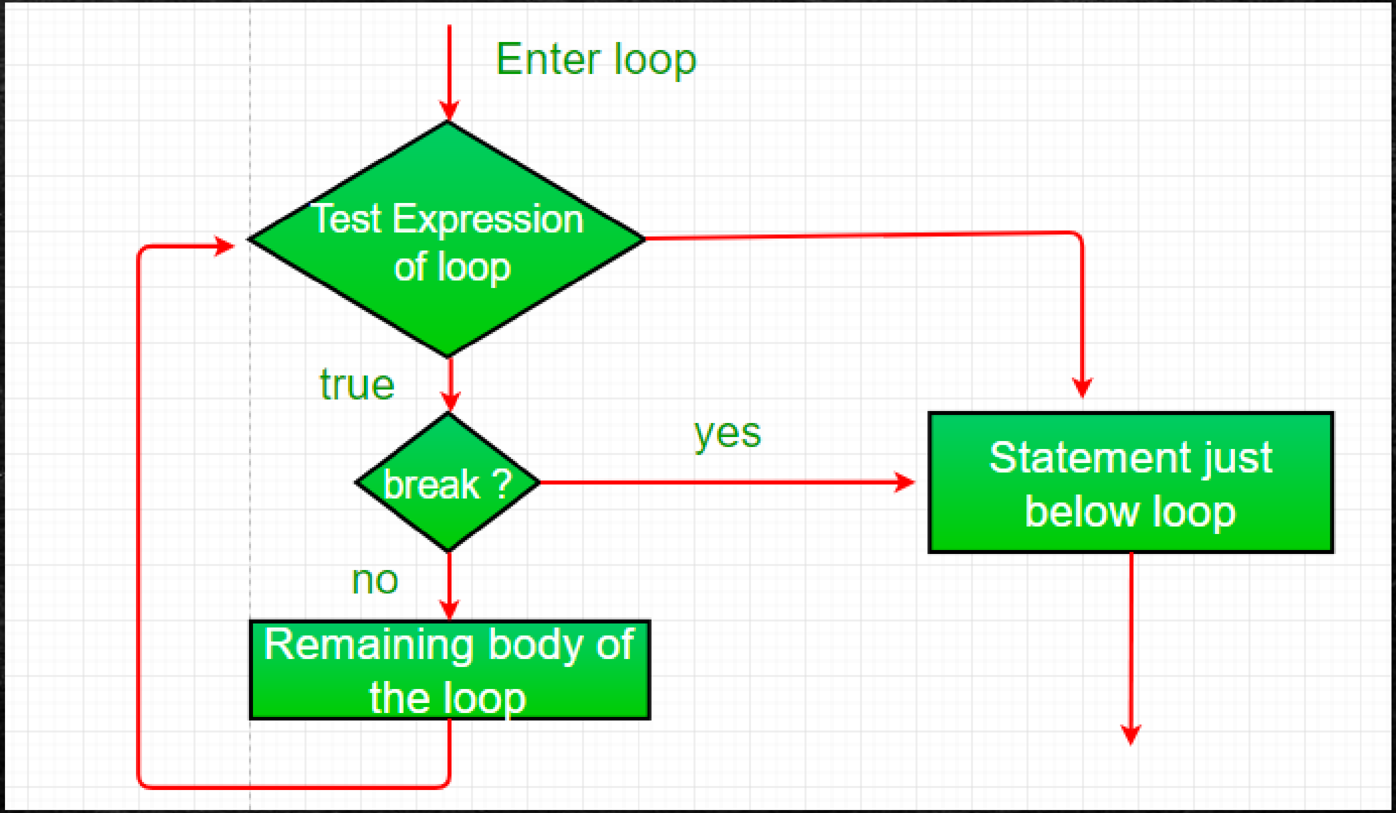
- In Java, break is majorly used to:
  - Terminate a sequence in a switch statement (discussed above).
  - To exit a loop.
- When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- The Java *break* statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.
- We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Break Statement

- Syntax and Flowchart:

  jump-statement;

  break;



```
//Java Program to demonstrate the use of break statement
//inside the for loop.
public class BreakExample
{
        public static void main(String[] args)
        {
                //using for loop
                for(int i=1;i<=10;i++)
                {
                        if(i==5)
                        {
                        //breaking the loop
                        break;
                        }
                        System.out.println(i);
                }
        }
}
```

Continue Statement

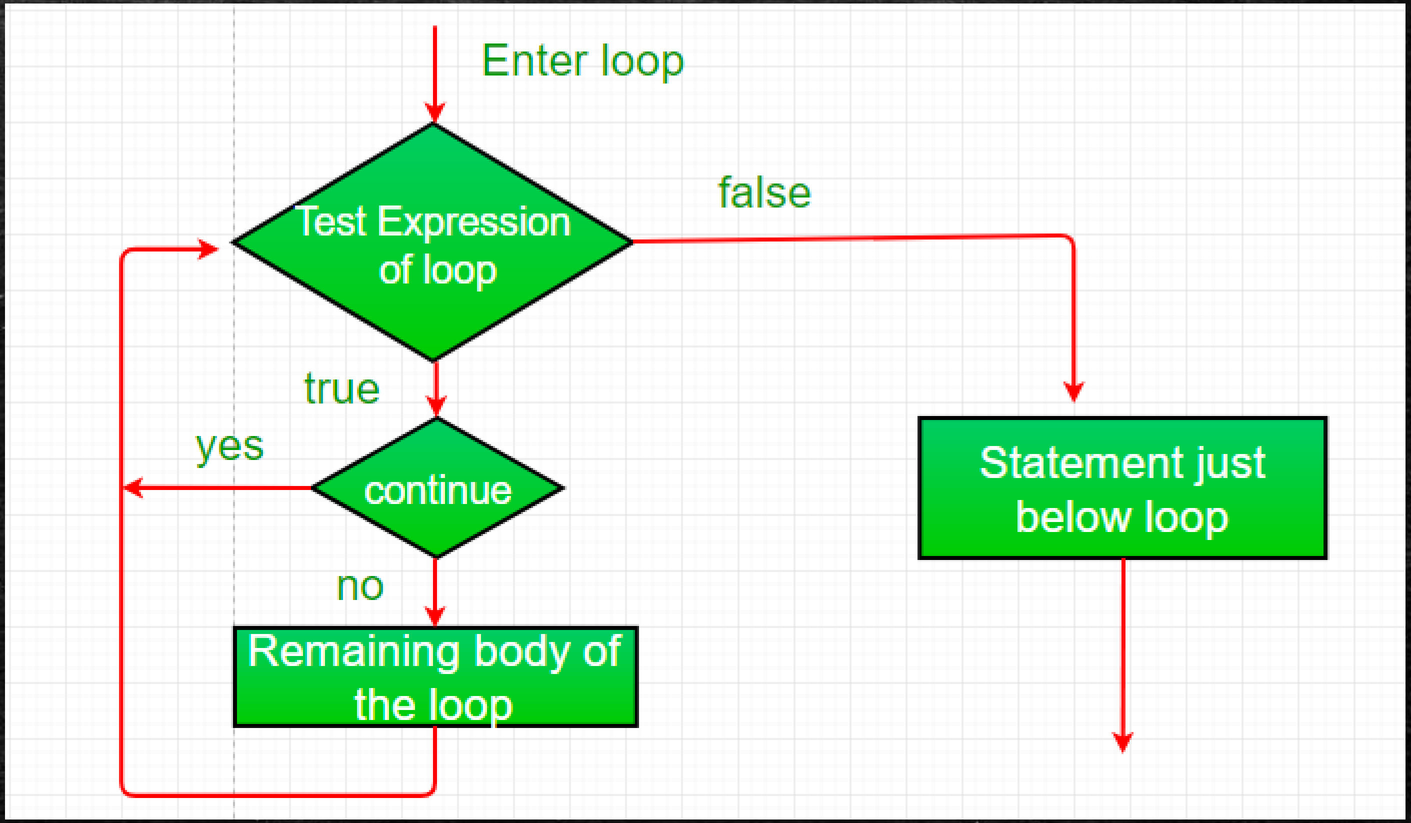
- The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately.
- The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.
- We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Continue Statement

- Syntax and Flowchart:

  jump-statement;

  continue;



```
//Java Program to demonstrate the use of continue statement
//inside the for loop.
public class ContinueExample
{
		public static void main(String[] args)
		{
				//for loop
				for(int i=1;i<=10;i++)
				{
						if(i==5)
						{
								//using continue statement
								continue;//it will skip the rest statement
						}
						System.out.println(i);
				}
		}
}
```

```
//Program to demonstrate the use of continue statement
//inside the while loop.
public class ContinueWhileExample
{
        public static void main(String[] args)
        {
                //while loop
                int i=1;
                while(i<=10)
                {
                        if(i==5)
                        {
                        //using continue statement
                        i++;
                        continue;//it will skip the rest statement
                        }
                        System.out.println(i);
                        i++;
                }
        }
}
```

return statement

- A **return statement** causes the program control to transfer back to the caller of a method.
- Every method in Java is declared with a return type and it is mandatory for all java methods.
- A return type may be a **primitive type** like i**nt, float, double,** a **reference type** or **void type**(returns nothing).

```
// 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.");
        }
}
```

```
public class ReturnTypeTest
{
        public int add() // without arguments
        {
                int x = 30;
                int y = 70;
                int z = x+y;
                return z;
        }
        public static void main(String args[])
        {
                ReturnTypeTest1 test = new ReturnTypeTest1();
                int add = test.add();
                System.out.println("The sum of x and y is: " + add);
        }
}
```

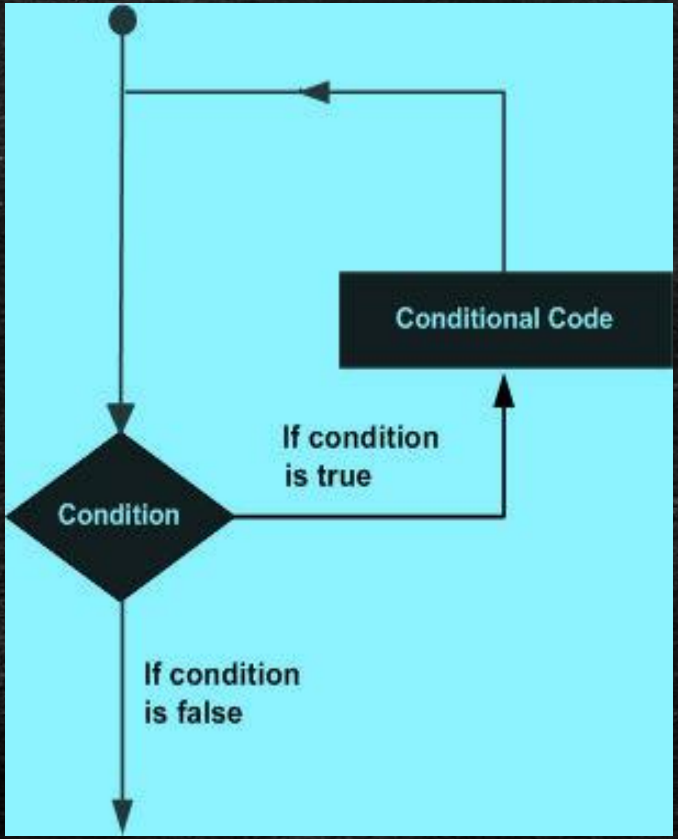
Loops in Java

- loops are used to execute a set of instructions/functions repeatedly when some conditions become true.
- Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.
- Following are the three types of Loops:

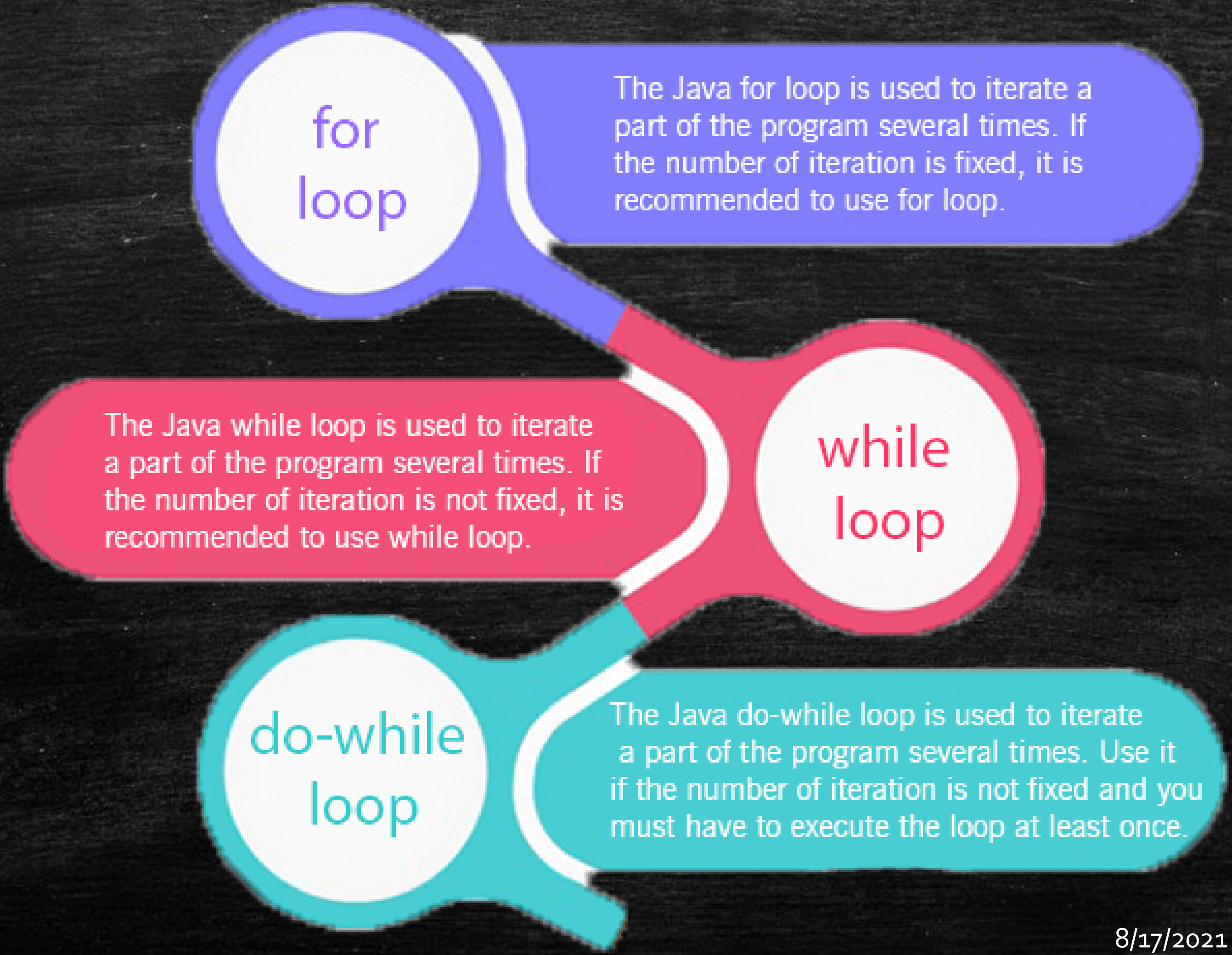
1) for loop
2) while loop
3) do-while loop

General Flow

- A **loop** statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages



Small Notable Difference



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.

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.

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.

`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.
- There are three types of for loops in java.
  - Simple For Loop
  - for-each or Enhanced For Loop
  - Labeled For Loop

Simple for Loop

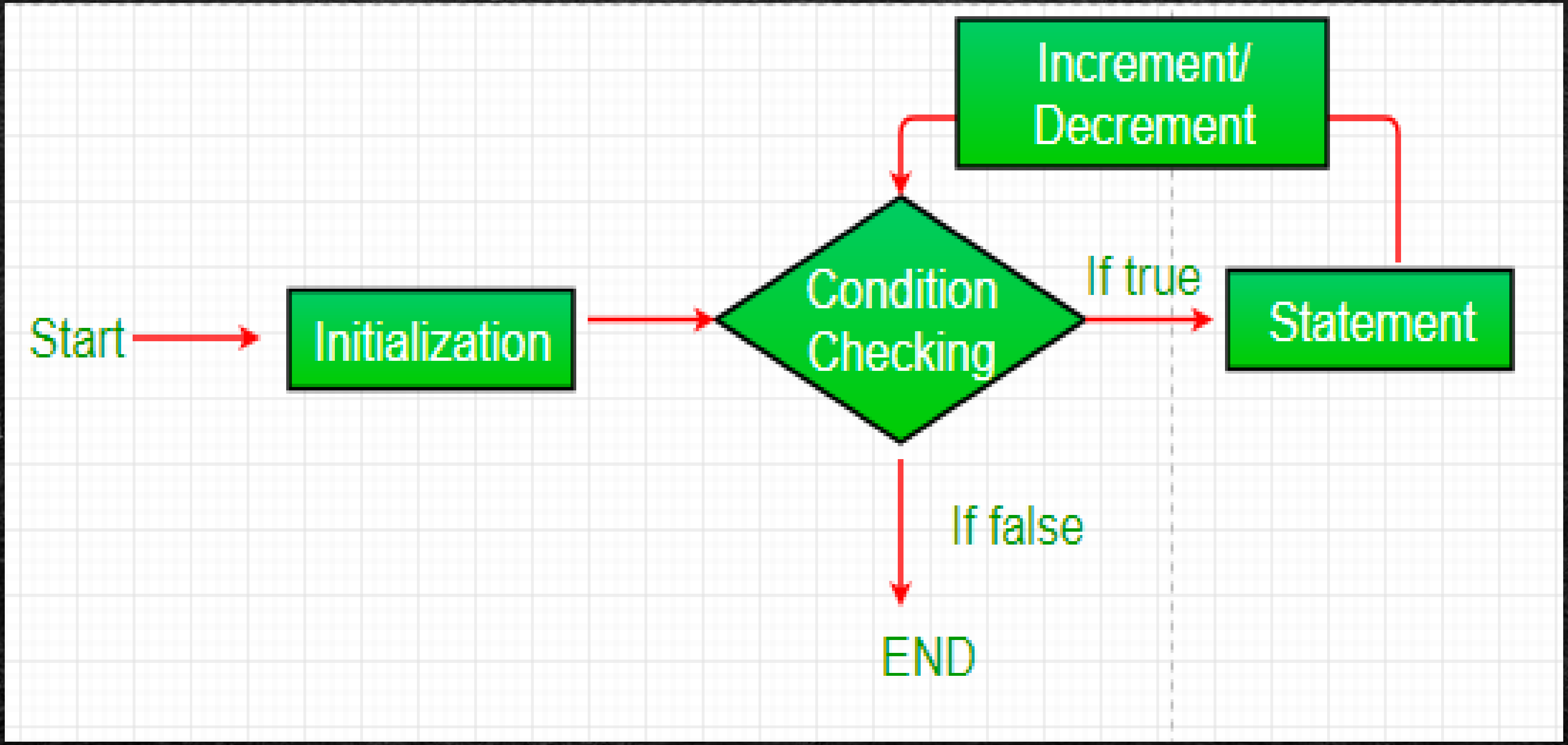
- A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:
  - **Initialization:** It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable.
  - **Condition:** It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false.
  - **Statement:** The statement of the loop is executed each time until the second condition is false.
  - **Increment/Decrement:** It increments or decrements the variable value.

Syntax of for Loop

- Syntax:

```
for(initialization;condition;update_expression)
{
        //statement or code to be executed
}
```

Flow chart of for Loop



```
//for loop demo
public class forDemo
{
        public static void main(String args[])
        {
                for(int x = 10; x < 20; x = x + 1)
                {
                        System.out.println("value of x : " + x );
                }
        }
}
```

Java for-each Loop

- The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.
- It works on elements basis not index. It returns element one by one in the defined variable.
- Syntax:

```
for(Type var:array)
{
        //code to be executed
}
```

```
//Java For-each loop example which prints the
//elements of the array
public class ForEachExample
{
        public static void main(String[] args)
        {
                //Declaring an array
                int arr[]={12,23,44,56,78};
                //Printing array using for-each loop
                for(int i:arr)
                {
                        System.out.println(i);
                }
        }
}
```

Java Labeled For Loop

- We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.
- Usually, break and continue keywords breaks/continues the innermost for loop only.
- Syntax

```
labelname:
for(initialization;condition;incr/decr)
{
    //code to be executed
}
```

```
//A Java program to demonstrate the use of labeled for loop
public class LabeledForExample
{
        public static void main(String[] args)
        {
                //Using Label for outer and for loop
                aa:
                        for(int i=1;i<=3;i++)
                        {
                        bb:
                                for(int j=1;j<=3;j++)
                                        {
                                                if(i==2&&j==2)
                                                {
                                                break aa;
                                                }
                                                System.out.println(i+" "+j);
                                }
                        }
        }
}
```

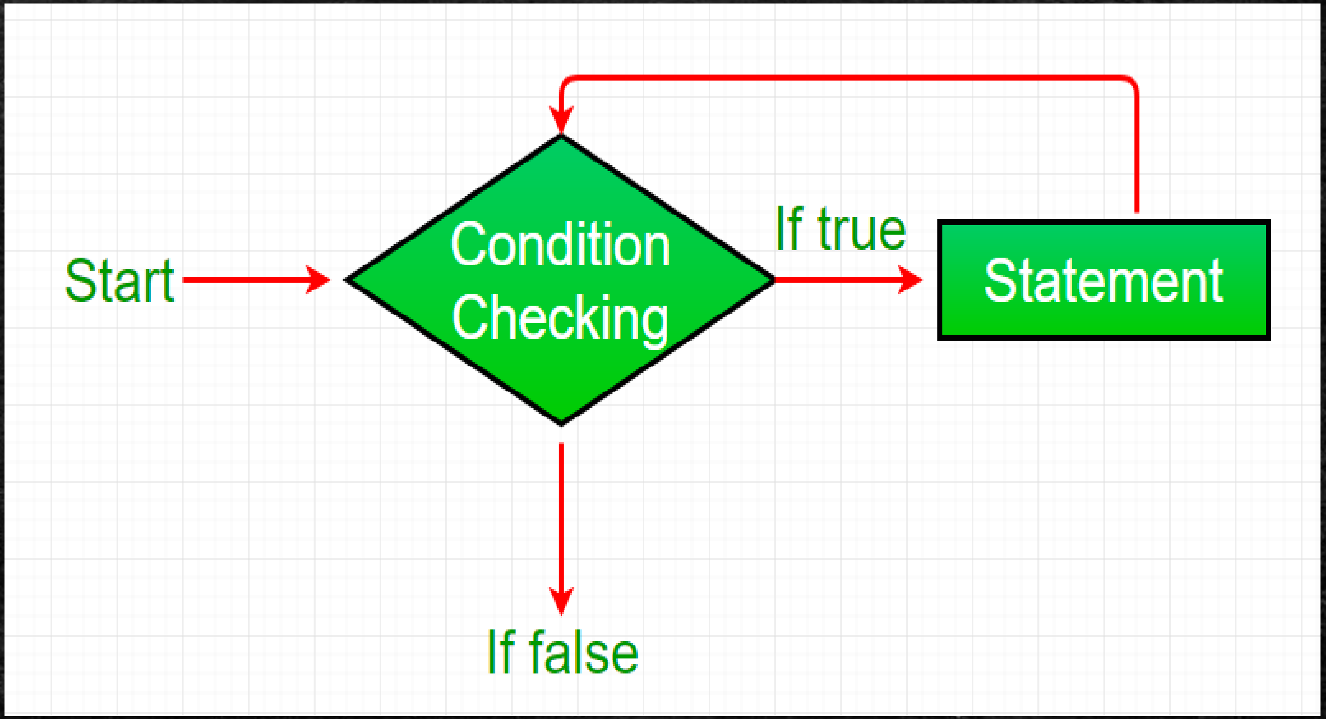
`while loop`

- 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.
- A **while** loop statement in Java programming language repeatedly executes a target statement as long as a given condition is true.
- The while loop can be thought of as a repeating if statement.

Syntax and flow chart of while loop

Syntax:

```
while (test_condition)
{
        loop statements...
}
```



More on while loop

- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop**
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

```
//while loop demo
public class WhileExample
{
        public static void main(String[] args)
        {
                int i=1;
                while(i<=10)
                {
                        System.out.println(i);
                        i++;
                }
        }
}
```

```
//while example 2
public class whileDemo
{
		public static void main(String args[])
		{
				int x = 10;
				while( x < 20 )
				{
						System.out.println("value of x : " + x );
						x++;
				}
		}
}
```

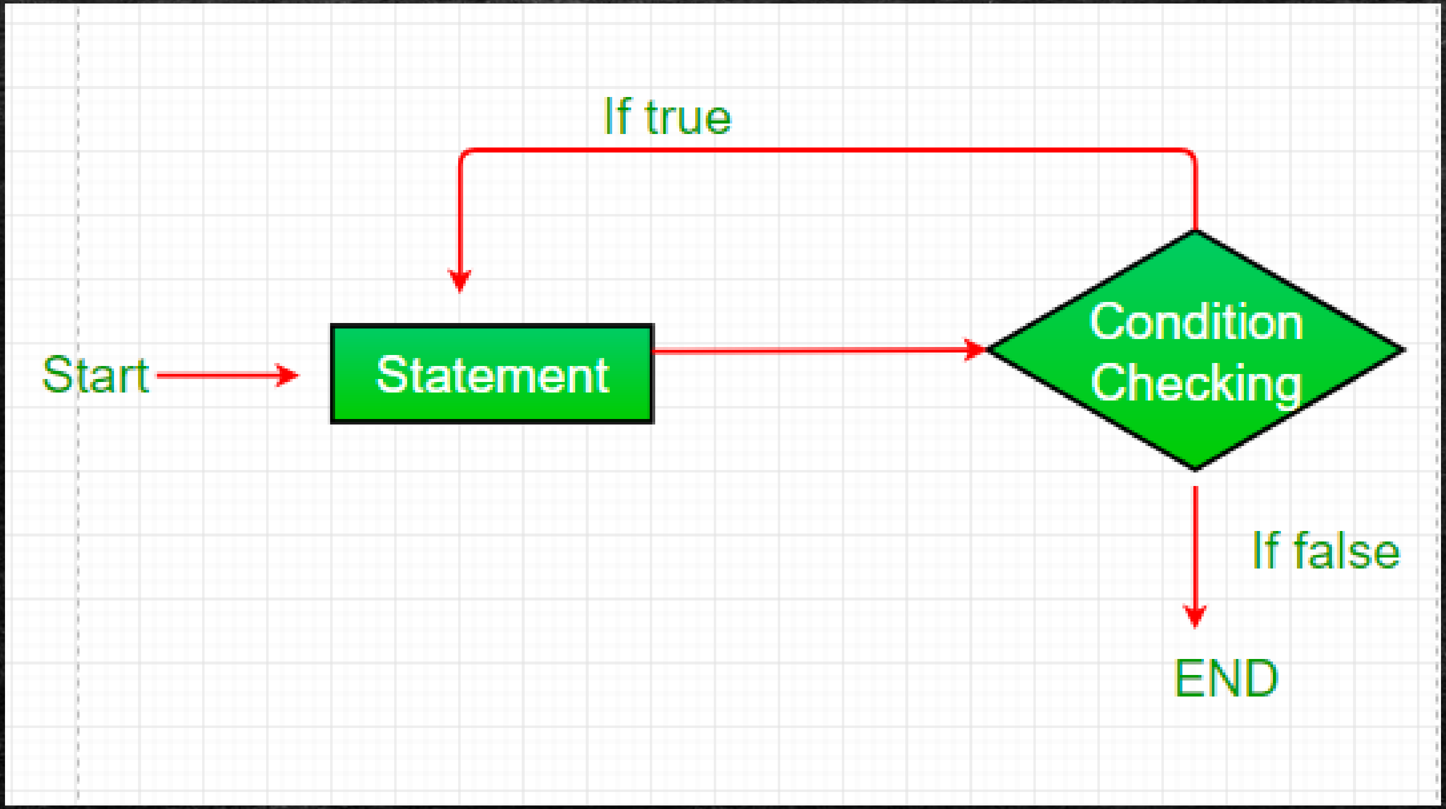
do-while loop

- It is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.
- The Java *do-while loop* is executed at least once because condition is checked after loop body.
- It checks for condition after executing the statements, and therefore is an example of **Exit Control Loop.**

Syntax of do-while loop

```
do
{
        statements..
}
while (condition);
```

Flow chart of do-while loop



```
//do-while loop example

public class DoWhileExample
{
        public static void main(String[] args)
        {
                int i=1;
                do
                {
                        System.out.println(i);
                        i++;
                }
                while(i<=10);
        }
}
```

```
// Java program to illustrate do-while loop
class dowhileloopDemo
{
        public static void main(String args[])
        {
                int x = 21;
                do
                {
                        // The line will be printed even
                        // if the condition is false
                        System.out.println("Value of x:" + x);
                        x++;
                }
                while (x < 20);
        }
}
```

More on do-while loop

- do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements at least once before any condition is checked, and therefore is an example of exit control loop.

Comparison	for loop	while loop	do while loop
Introduction	The Java for loop is a control flow statement that iterates a part of the programs multiple times.	The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.	The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.
When to use	If the number of iteration is fixed, it is recommended to use for loop.	If the number of iteration is not fixed, it is recommended to use while loop.	If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.
Syntax	for(init;condition;incr/decr){ // code to be executed }	while(condition){ //code to be executed }	do{ //code to be executed }while(condition);
Example	//for loop for(int i=1;i<=10;i++){ System.out.println(i); }	//while loop int i=1; while(i<=10){ System.out.println(i); i++; }	//do-while loop int i=1; do{ System.out.println(i); i++; }while(i<=10);
Syntax for infinitive loop	for(;;){ //code to be executed }	while(true){ //code to be executed }	do{ //code to be executed }while(true);

Java String

- In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string.
- For example:

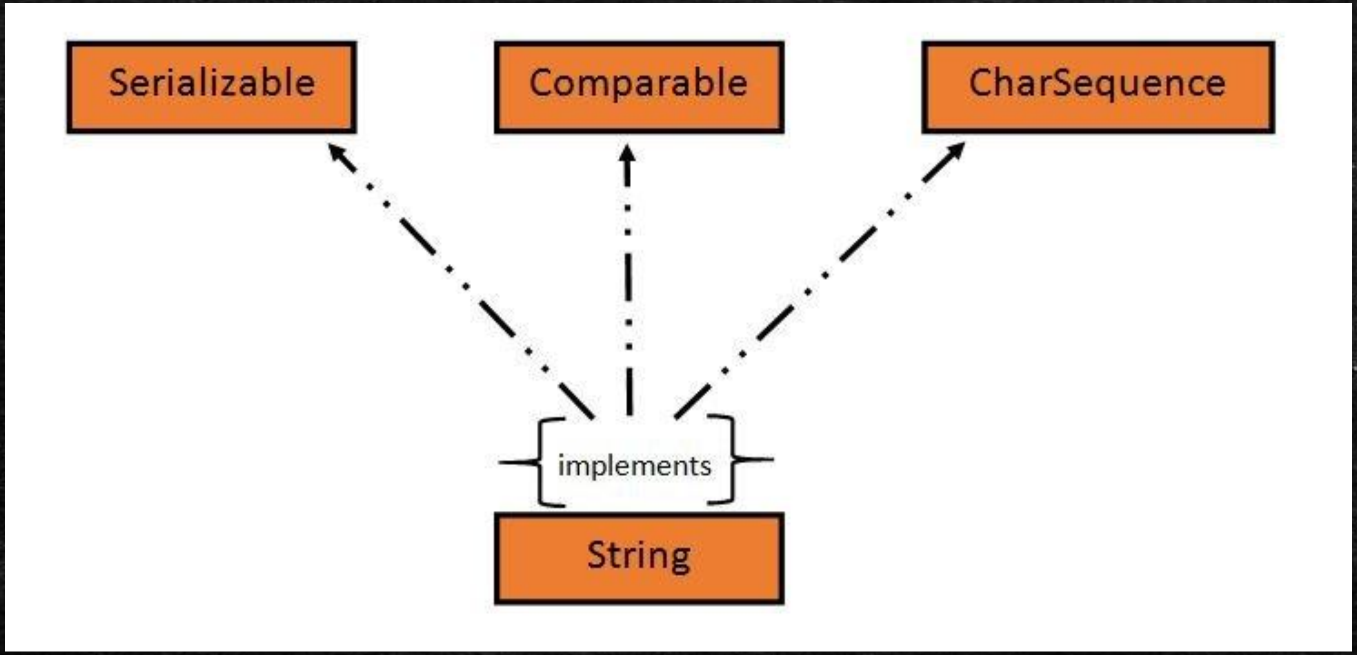
```
char[] ch={'j','a','v','a'};
String s=new String(ch);
```

- is same as:

```
String s="java";
```

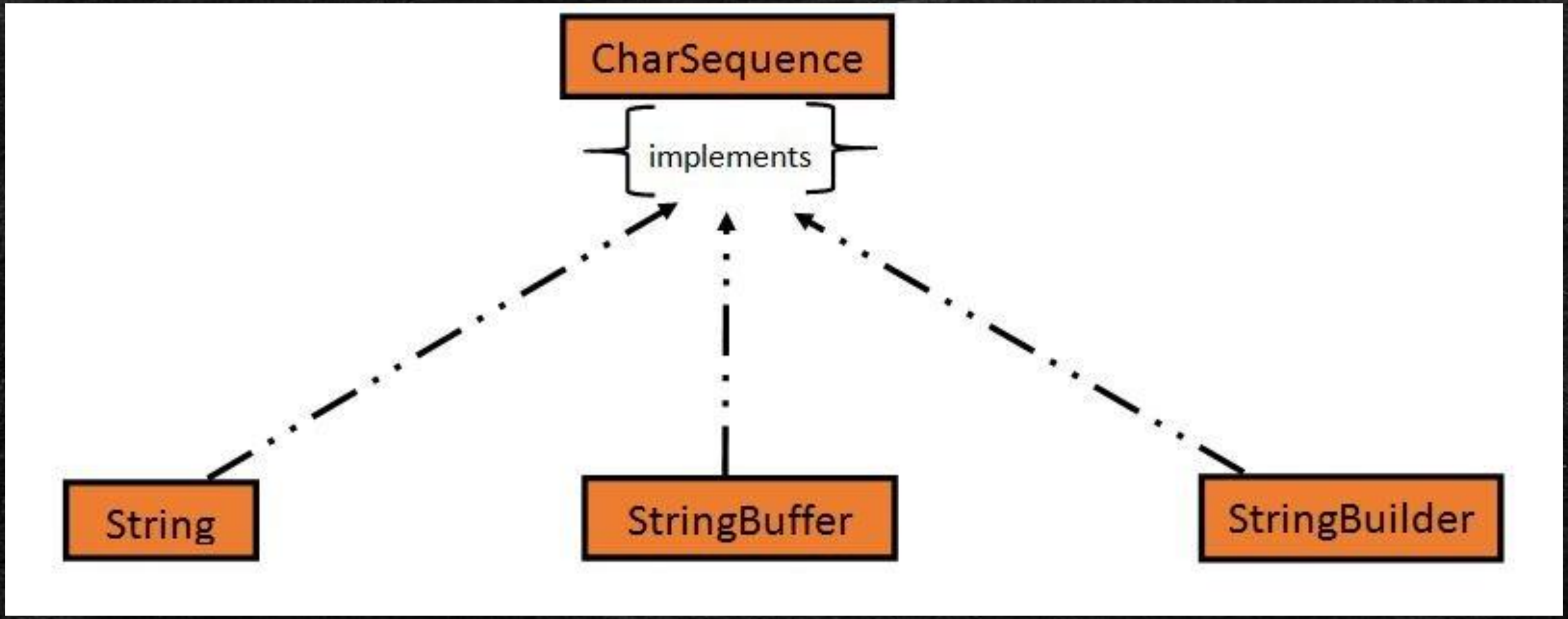
Java String

- Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.
- The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.



CharSequence Interface

- The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it.
- It means, we can create strings in Java by using these three classes.



String is immutable

- The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created.
- For mutable strings, you can use StringBuffer and StringBuilder classes.

How to create a string object?

- There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

- Java String literal is created by using double quotes. For Example:
  - String s="welcome";
  - String s2="Welcome";
- Why Java uses the concept of String literal?
  - To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

2) By new keyword

- String s=new String("Welcome");
- //creates two objects and one reference variable

Java String Example

```
//StringExample.java
public class StringExample
{
        public static void main(String args[])
           {
                String s1="java";//creating string by Java string literal
                char ch[]={'s','t','r','i','n','g','s'};
                String s2=new String(ch);//converting char array to string
                String s3=new String("example");//creating Java string by new keyword
                System.out.println(s1);
                System.out.println(s2);
                System.out.println(s3);
           }
}
```

Java Arrays

- An array is a collection of similar data types. Array is a container object that hold values of homogeneous type. It is also known as static data structure because size of an array must be specified at the time of its declaration.
- Array starts from zero index and goes to n-1 where n is length of the array.
- In Java, array is treated as an object and stores into heap memory. It allows to store primitive values or reference values.
- Array can be single dimensional or multidimensional in Java

Features of Array

- It is always indexed. Index begins from 0.
- It is a collection of similar data types.
- It occupies a contiguous memory location.
- It allows to access elements randomly.

Single Dimensional Array

- Single dimensional array use single index to store elements. You can get all the elements of array by just increment its index by one.

Array Declaration

- Syntax :
  - datatype[] arrayName;;

    or
  - datatype arrayName[];
- Java allows to declare array by using both declaration syntax, both are valid.
- The arrayName can be any valid array name and datatype can be any like: int, float, byte etc.

Example

```
int[ ] arr;

char[ ] arr;

short[ ] arr;

long[ ] arr;

int[ ][ ] arr;   // two dimensional array.
```

Initialization of Array

- Initialization is a process of allocating memory to an array. At the time of initialization, we specify the size of array to reserve memory area.

Initialization Syntax

- arrayName = new datatype[size]

- new operator is used to initialize an array.
- The arrayName is the name of array, new is a keyword used to allocate memory and size is length of array.
- We can combine both declaration and initialization in a single statement.
  - Datatype[] arrayName = new datatype[size]

Example : Create An Array

```
//Lets create a single dimensional array.
class Demo
{
public static void main(String[] args)
  {
    int[] arr = new int[5];
        for(int x : arr)
        {
                System.out.println(x);
        }
   }
}
```

Set Array Elements

We can set array elements either at the time of initialization or by assigning direct to its index.

```
                int[] arr = {10,20,30,40,50};
```

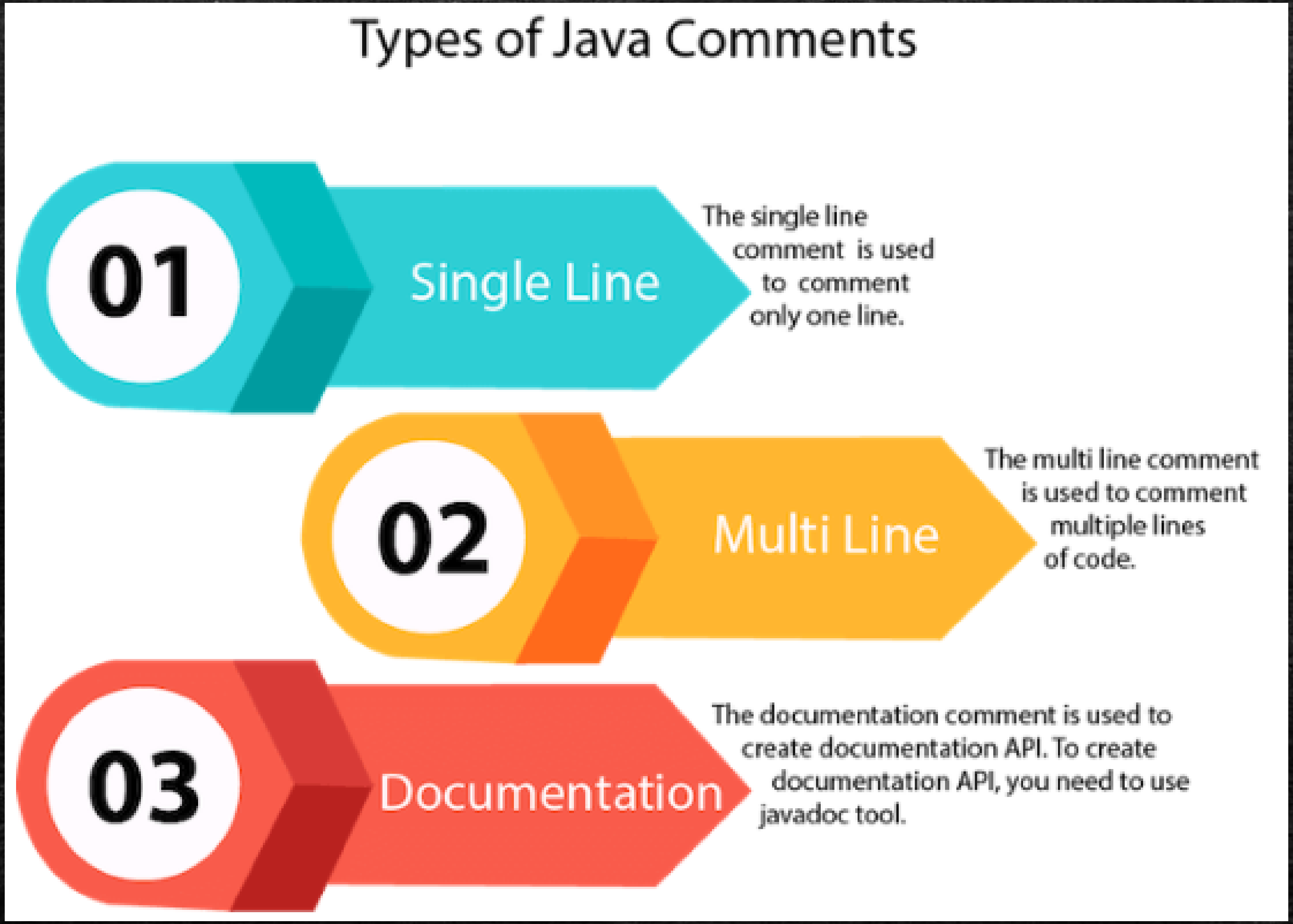
Example:

```
class Demo
{
public static void main(String[] args)
  {
                int[] arr = {10,20,30,40,50};
       for(int x : arr)
       {
              System.out.println(x);
       }

    // assigning a value

       arr[1] = 105;
       System.out.println("element at first index: " +arr[1]);
  }
}
```

Java Comments

- The Java comments are the statements in a program that are not executed by the compiler and interpreter.
- Why do we use comments in a code?
  - Comments are used to make the program more readable by adding the details of the code.
  - It makes easy to maintain the code and to find the errors easily.
  - The comments can be used to provide information or explanation about the variable, method, class, or any statement.
  - It can also be used to prevent the execution of program code while testing the alternative code.

Types of Java Comments


Types of Java Comments
01
Single Line
The single line comment is used to comment only one line.
02
Multi Line
The multi line comment is used to comment multiple lines of code.
03
Documentation
The documentation comment is used to create documentation API. To create documentation API, you need to use javadoc tool.

1) Java Single Line Comment

- The single-line comment is used to comment only one line of the code. It is the widely used and easiest way of commenting the statements.
- Single line comments starts with two forward slashes (//). Any text in front of // is not executed by Java.
- Syntax:
  - //This is single line comment
- Let's use single line comment in a Java program.

2) Java Multi Line Comment

- The multi-line comment is used to comment multiple lines of code. It can be used to explain a complex code snippet or to comment multiple lines of code at a time (as it will be difficult to use single-line comments there).
- Multi-line comments are placed between /* and */. Any text between /* and */ is not executed by Java.
- Syntax:

```
/*
    This
    is
    multi line
    comment
*/
```

- Let's use multi-line comment in a Java program.

3) Java Documentation Comment

- Documentation comments are usually used to write large programs for a project or software application as it helps to create documentation API. These APIs are needed for reference, i.e., which classes, methods, arguments, etc., are used in the code.
- To create documentation API, we need to use the javadoc tool. The documentation comments are placed between /** and */.

Syntax

- Syntax:

```
/**
*
*We can use various tags to depict the parameter
*or heading or author name
*We can also use HTML tags
*
*/
```

Javadoc tags

- Some of the commonly used tags in documentation comments:

Tag	Syntax	Description
{@docRoot}	{@docRoot}	to depict relative path to root directory of generated document from any page.
@author	@author name - text	To add the author of the class.
@code	{@code text}	To show the text in code font without interpreting it as html markup or nested javadoc tag.
@version	@version version-text	To specify "Version" subheading and version-text when -version option is used.
@since	@since release	To add "Since" heading with since text to generated documentation.
@param	@param parameter-name description	To add a parameter with given name and description to 'Parameters' section.
@return	@return description	Required for every method that returns something (except void)

References

- Text book:
  - Core Java 8 for Beginners, by Sharanam Shah and Vaishali Shah, Shroff Publishers & Distributors PVT. LTD. First Edition, July 2015.
  - Java The Complete Reference, Ninth Edition, Herbert Schildt, McGraw-Hill Education, 2014.
- Websites:
  - https://www.javatpoint.com/java-tutorial
  - https://www.geeksforgeeks.org/java/
  - https://www.tutorialspoint.com/java/index.html

Google Classroom code for “Programming in Java”

- Join the Google Classroom by Using following Code:

x76ov4y

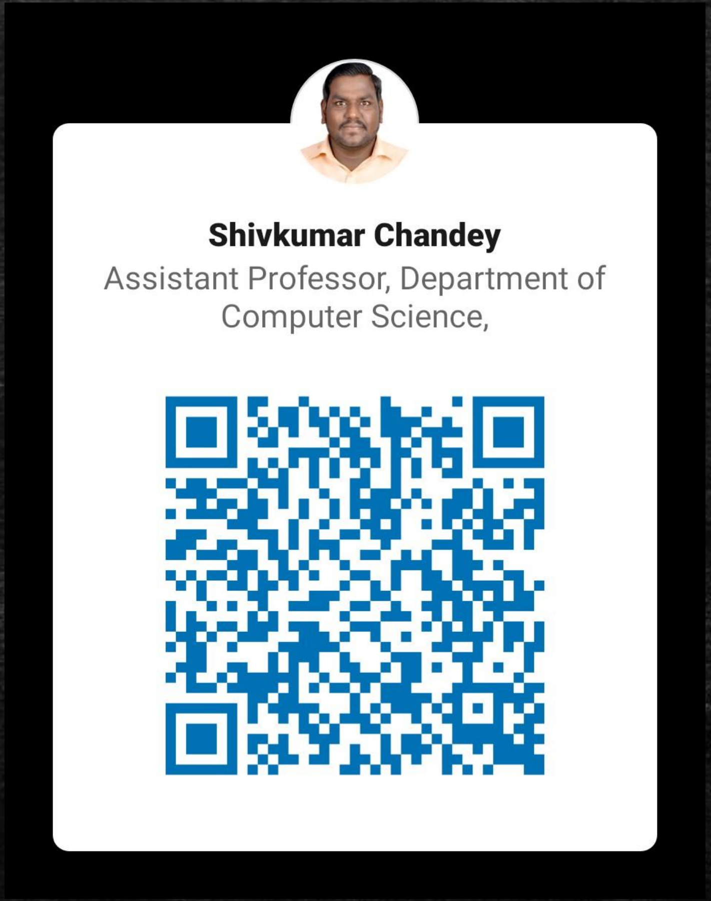
Thank You!!!Any Query?

asktoshivsir@gmail.com

Shivkumar Chandey

(+91 9987389441)

Scan QR Code to connect on LinkedIn


Shivkumar Chandey
Assistant Professor, Department of Computer Science,