JAVA BASIC AND OOPS CONCEPT

INTRODUCTION TO JAVA

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 name from Oak 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.

JAVA EXAMPLE

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

o/p

Hello java

APPLICATION

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

TYPES OF JAVA APPLICATIONS

Parul® University

- There are mainly 4 types of applications that can be created using Java programming:
  1) Standalone Application
  2) Web Application
  3) Enterprise Application
  4) Mobile Application

HISTORY OF JAVA

- 1) James Gosling, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.
- Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.
- Firstly, it was called "**Greentalk**" by James Gosling, and the file extension was .gt.
- After that, it was called **Oak** and was developed as a part of the Green project.
- In 1995, Oak was renamed as "**Java**" because it was already a trademark by Oak Technologies.
- Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.
- JDK 1.0 was released on January 23, 1996.

PARADIGMS OF PROGRAMMING LANGUAGES IN JAVA

Java is a versatile programming language that supports multiple programming paradigms, allowing developers to approach problems in different ways.

1. Object-Oriented Programming (OOP)

Object-Oriented Programming is the primary paradigm in Java. OOP is based on the concept of objects, which are instances of classes.

- **Encapsulation**: Bundling the data (attributes) and methods (functions) that operate on the data into a single unit called a class. Encapsulation also involves restricting access to certain details of an object (through access modifiers like private, protected, and public).

OBJECT-ORIENTED PROGRAMMING (OOP) CONT.

- **Inheritance**: The mechanism by which one class (the subclass) can inherit fields and methods from another class (the superclass). This promotes code reuse and establishes a hierarchical relationship between classes.
- **Polymorphism**: The ability of a single interface to represent different underlying forms (data types). In Java, polymorphism is typically achieved through method overloading and method overriding.
- **Abstraction**: The concept of hiding the complex implementation details and showing only the essential features of the object. In Java, abstraction is achieved using abstract classes and interfaces.

2. PROCEDURAL PROGRAMMING

Procedural Programming is a paradigm derived from structured programming, based on the concept of procedure calls, where statements are structured into procedures (also known as routines, functions, or subroutines). Java supports procedural programming, though it's not the primary paradigm.

3. Functional Programming

Functional Programming is a paradigm that treats computation as the evaluation of mathematical functions and avoids changing state or mutable data. Java supports functional programming to some extent, especially with the introduction of lambdas and the Stream API in Java 8.

4. Concurrent Programming

Concurrent Programming is a paradigm used to handle multiple tasks at the same time, often by dividing them into subtasks that can run in parallel. Java provides strong support for concurrency through the java.util.concurrent package, threads, and other synchronization mechanisms.

5. EVENT-DRIVEN PROGRAMMING

- **Event-Driven Programming** is a paradigm where the flow of the program is determined by events such as user actions (clicks, key presses), sensor outputs, or messages from other programs. Java supports event-driven programming, particularly in GUI applications with the Swing and JavaFX libraries.

EVOLUTION OF OO METHODOLOGY IN JAVA

- Object-Oriented (OO) methodology in Java has evolved significantly since the language's inception. Java itself was designed with object-oriented principles at its core, but the methodologies and best practices for applying these principles have evolved over time. Here's a look at the evolution of OO methodology in Java:

1. Early OO Concepts (Pre-Java)

- **Simula and Smalltalk:** The OO methodology began with languages like Simula (1960s) and Smalltalk (1970s), which introduced the core concepts of objects, classes, inheritance, and polymorphism.
- **C++ Influence:** C++, an extension of C, added OO features and heavily influenced Java's development, although Java aimed to be simpler and more secure.

2. JAVA 1.0 (1995): THE BEGINNING

- **Core OO Principles:** Java was released by Sun Microsystems in 1995 with a strong focus on OO principles like encapsulation, inheritance, and polymorphism.
- **Platform Independence:** The "Write Once, Run Anywhere" philosophy emphasized platform independence, which was achieved through the use of the Java Virtual Machine (JVM).
- **Basic OO Features:** Early Java versions supported basic OO features—classes, objects, inheritance, interfaces, and simple exception handling.
- **Applets:** Java initially gained popularity through applets, which were small, secure applications running within web browsers.

3. JAVA 2 (1998-2004): STRENGTHENING OO CONCEPTS

- **Swing and AWT:** The introduction of Swing and the Abstract Window Toolkit (AWT) provided more robust and flexible GUI components, reinforcing the OO nature of Java applications.
- **Java Collections Framework:** The introduction of the Collections Framework (Java 2) standardized data structures like lists, sets, and maps, promoting better OO design by encouraging the use of interfaces and polymorphism.
- **Enterprise JavaBeans (EJB):** EJB and the broader Java Enterprise Edition (J2EE) framework promoted component-based development and the use of design patterns, particularly in enterprise-level applications.
- **JSP and Servlets:** JavaServer Pages (JSP) and Servlets encouraged the development of web applications using Java, blending OO principles with web technologies.

4. Java 5 (2004): Enhanced OO Capabilities

- **Generics:** Java 5 introduced generics, allowing for type-safe collections and reducing the need for type casting. This improved code reusability and robustness.
- **Enhanced for Loop:** The enhanced for loop simplified iteration over collections, aligning with OO principles of abstraction and encapsulation.
- **Annotations:** The introduction of annotations provided a way to add metadata to classes, methods, and fields, influencing how OO frameworks (like Spring) could automate certain processes.
- **Concurrency API:** New concurrency utilities were introduced, promoting better design in multi-threaded applications.

5. JAVA 6 AND 7 (2006-2011): REFINEMENTS AND NEW PATTERNS

- **Java SE 6:** Focused on performance improvements and enhancements to existing APIs, while continuing to support OO principles.
- **Java SE 7:** Introduced try-with-resources, simplifying resource management and adhering to OO best practices like encapsulation and code clarity.
- **NIO.2:** Enhanced the NIO package, improving how Java handled file systems and IO, encouraging more modular and maintainable code.

6. Java 8 (2014): Functional Programming Meets OO

- **Lambda Expressions:** Introduced functional programming features, allowing functions to be treated as first-class citizens. This led to more concise and flexible code while maintaining OO principles.
- **Streams API:** The Streams API enabled functional-style operations on collections, promoting immutability and encouraging a more declarative approach to processing data.
- **Default Methods in Interfaces:** Interfaces could now have default methods, allowing for backward-compatible evolution of APIs and promoting better interface design.
- **Optional Class:** Introduced to help avoid NullPointerException, promoting more robust and safe OO code.

7. JAVA 9 AND BEYOND (2017-PRESENT): MODULARITY AND NEW FEATURES

- **Java Platform Module System (JPMS):** Java 9 introduced the module system (Project Jigsaw), which redefined how large applications were structured, improving encapsulation and maintainability.
- **Reactive Streams:** Introduced in Java 9, enabling reactive programming, which complements OO by handling asynchronous data streams more effectively.
- **Enhanced APIs:** Continuous improvement in APIs, including Collections, Streams, and Concurrency utilities, refined OO practices in Java.
- **Records (Java 14):** Introduced as a preview feature and officially in Java 16, records are a new kind of class in Java that succinctly defines immutable data carriers, reducing boilerplate code while adhering to OO principles.

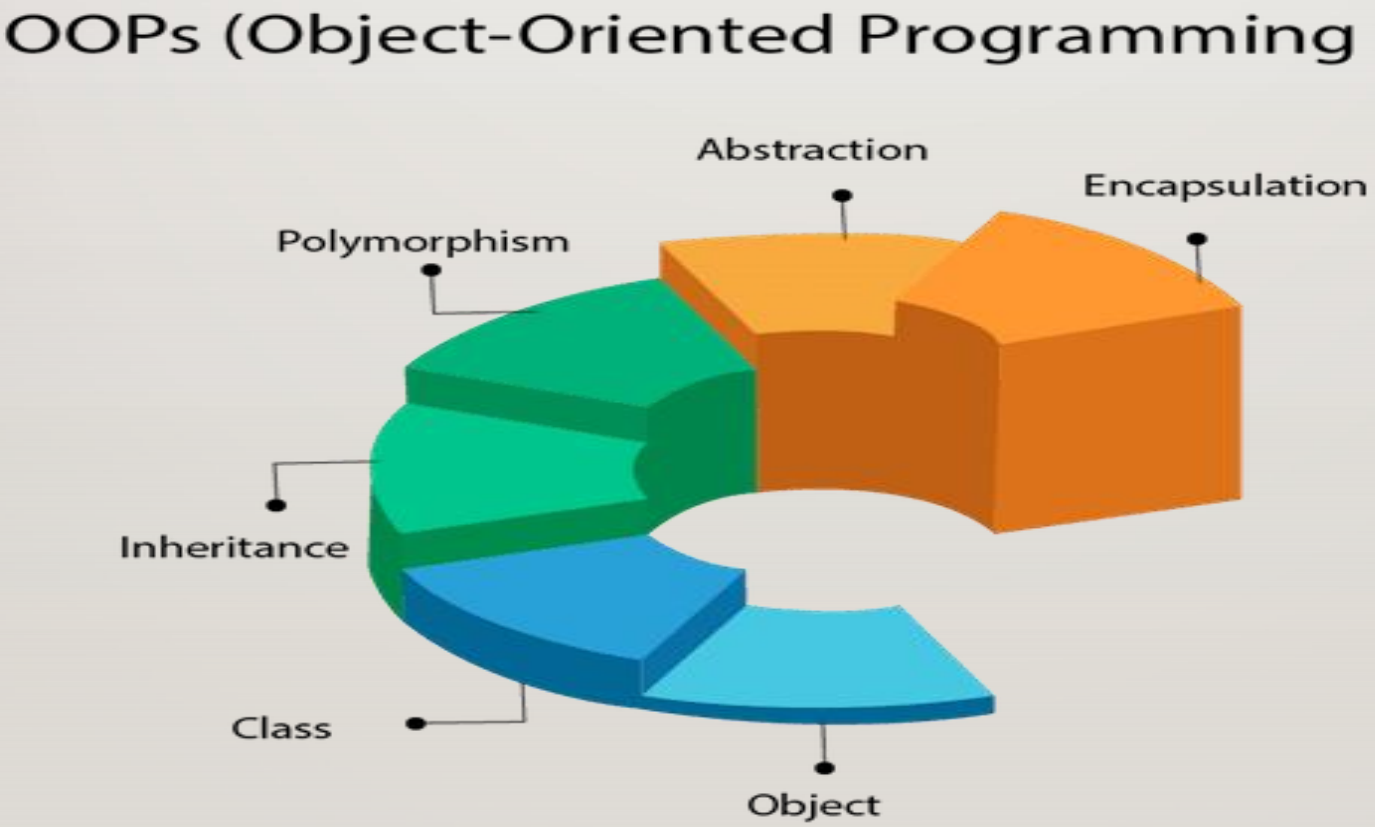
8. MODERN TRENDS:

- **Microservices:** Java's OO principles have adapted to new architectural paradigms like microservices, where modularity and encapsulation are crucial.
- **Cloud-Native Java:** With the rise of cloud computing, Java frameworks like Spring Boot have evolved to embrace OO principles in distributed, cloud-native environments.
- **Kotlin and Scala Influence:** While Java remains dominant, languages like Kotlin and Scala (which run on the JVM) have influenced modern Java practices, bringing in new OO and functional programming paradigms.

9. BEST PRACTICES AND DESIGN PATTERNS:

- **Design Patterns:** The adoption of design patterns like Singleton, Factory, and Observer in Java applications became widespread, promoting best practices in OO design.
- **SOLID Principles:** These principles (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion) became widely adopted, guiding OO design in Java.

OOPS


OOPs (Object-Oriented Programming System)
Abstraction
Encapsulation
Polymorphism
Inheritance
Class
Object

JAVA OOPS CONCEPTS

What is an object in Java

- An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical.
- An object has three characteristics:
- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

OBJECT

- **An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- An object is a real-world entity.
- An object is a runtime entity.
- The object is an entity which has state and behavior.
- The object is an instance of a class.

CLASS

What is a class in Java

- A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface

INHERITANCE IN JAVA

Parul® University

- **nheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object.

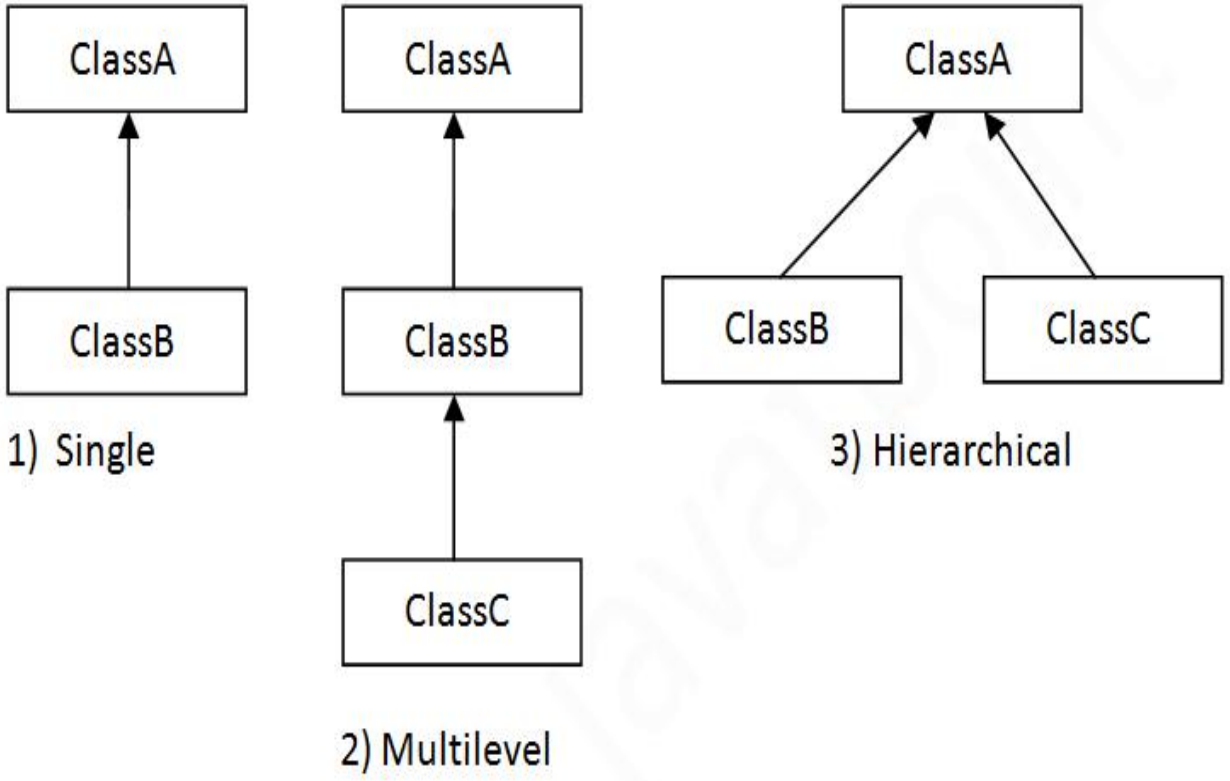
Why use inheritance in java

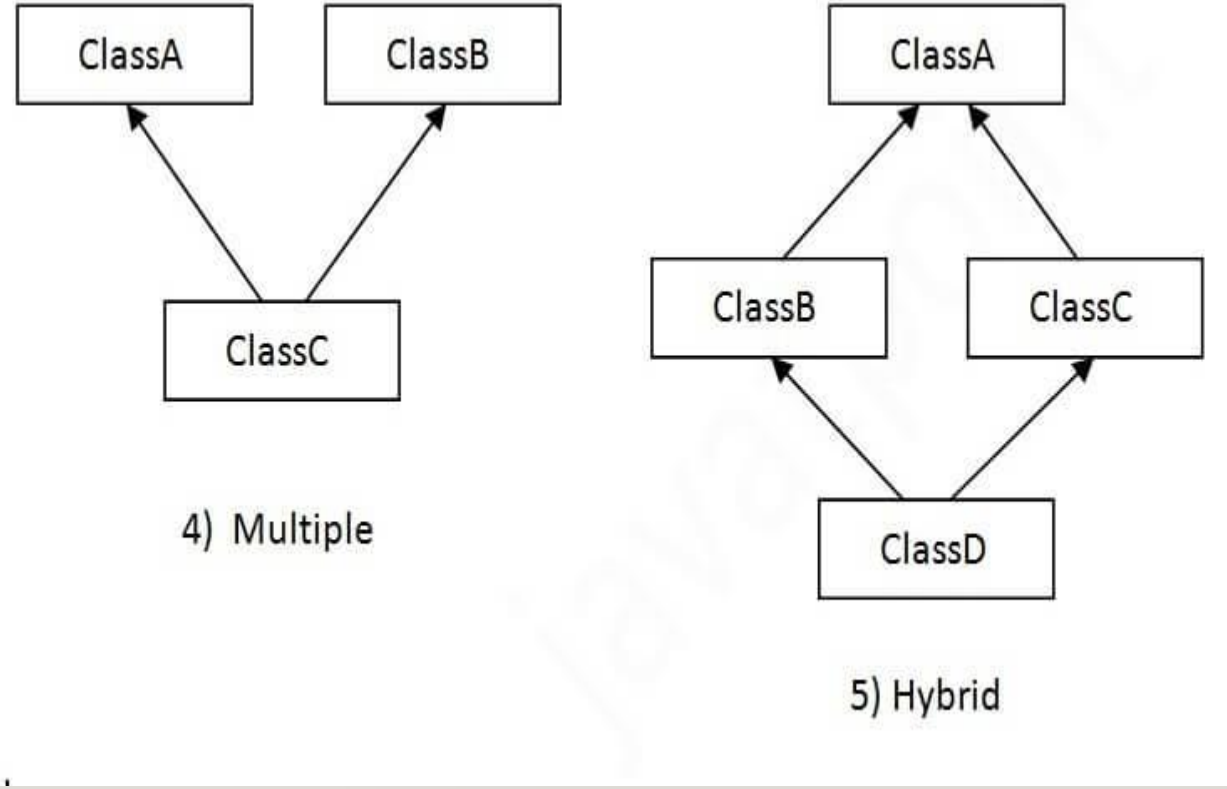
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

The syntax of Java Inheritance

```
1.class Subclass-name extends Superclass-name
2.{
3.   //methods and fields
4.}
```

TYPES OF INHERITANCE IN JAVA





JAVA POLYMORPHISM

- If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.
- In Java, we use method overloading and method overriding to achieve polymorphism.
- Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

ABSTRACTION

- Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.
- In Java, we use abstract class and interface to achieve abstraction.

ENCAPSULATION

- Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.
- A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

COMPARISON OF OBJECT ORIENTED AND PROCEDURE ORIENTED APPROACHES.

- **1. Basic Concept:**
- **Object-Oriented Programming (OOP):**
  - Organizes code around objects, which are instances of classes.
  - Focuses on representing real-world entities through objects that encapsulate data and behavior.
- **Procedural Programming:**
  - Organizes code into procedures or functions.
  - Focuses on the sequence of tasks or steps to be performed, typically using functions or subroutines.

CONT.

- **2. Structure:**
- **OOP:**
  - Programs are divided into objects, each containing both data (attributes) and methods (functions).
  - Emphasizes the relationships between objects and how they interact.
- **Procedural:**
  - Programs are divided into functions, each performing a specific task.
  - Emphasizes the flow of control and the order in which functions are called.

CONT.

- **3. Data Handling:**
- **OOP:**
  - Data is encapsulated within objects, and access to it is controlled through methods.
  - Promotes data hiding and abstraction, ensuring that data is only accessible in ways that are intended.
- **Procedural:**
  - Data is often passed between functions using parameters or global variables.
  - Less emphasis on data encapsulation, which can lead to unintended side effects if data is modified unexpectedly.

CONT.

- **4. Reusability:**
- **OOP:**
  - High reusability through inheritance and polymorphism.
  - Allows new classes to reuse and extend existing ones, promoting code reuse and reducing redundancy.
- **Procedural:**
  - Reusability is achieved through function libraries, but it is less flexible compared to OOP.
  - Functions can be reused, but extending or modifying functionality often requires rewriting or duplicating code.

CONT.

- **5. Maintainability:**
- **OOP:**
  - Easier to maintain and modify because objects are self-contained and modular.
  - Changes to one part of the system often have minimal impact on other parts.
- **Procedural:**
  - Can be harder to maintain, especially in large programs, as changes in one function may affect many other functions.
  - Tightly coupled code can lead to cascading changes when modifications are required.

CONT.

- **6. Scalability:**
- **OOP:**
  - More scalable due to its modular nature. New features can be added by creating new classes or extending existing ones.
  - Better suited for large, complex systems.
- **Procedural:**
  - Less scalable for large projects, as the code can become unwieldy with many interdependent functions.
  - More suitable for smaller, simpler programs.

CONT.

- **7. Approach to Problem-Solving:**
- **OOP:**
  - Models problems in terms of objects, focusing on the entities involved and their interactions.
  - Encourages thinking about the system as a collection of interacting objects.
- **Procedural:**
  - Models problems in terms of a sequence of steps or operations.
  - Encourages thinking about the problem in terms of tasks and procedures to be executed.

CONT.

8. Examples:

- **OOP:**
  - Languages like Java, C++, Python (with OOP features), Ruby.
- **Procedural:**
  - Languages like C, Pascal, Fortran, and Python (when used in a procedural style).

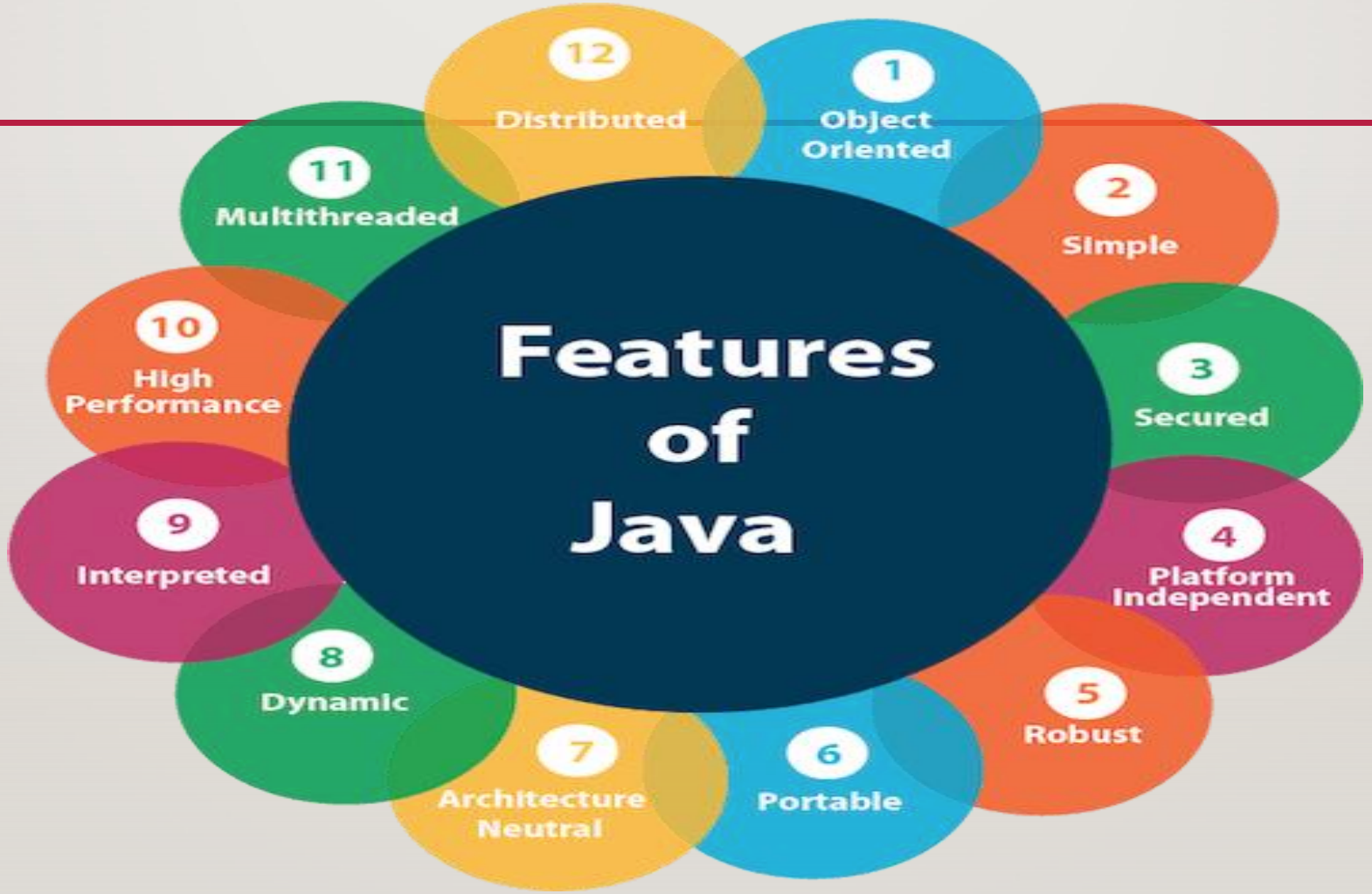
CONT.

- **9. Performance:**
- **OOP:**
  - May have a slight overhead due to object creation and method calls, but modern optimizations reduce this impact.
  - Performance can be optimized through proper design patterns and techniques.
- **Procedural:**
  - Generally more straightforward and can be more efficient for small, simple programs.
  - Directly executes a series of commands, often leading to faster execution in certain cases.

CONT.

- **10. Examples of Usage:**
- **OOP:**
  - Suitable for complex applications like GUIs, simulations, and games, where modeling real-world entities is beneficial.
- **Procedural:**
  - Suitable for tasks like scripting, system programming, and small utilities where straightforward execution of a sequence of tasks is needed

FEATURES OF JAVA



FEATURES (CONT.)

- **Simple**

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

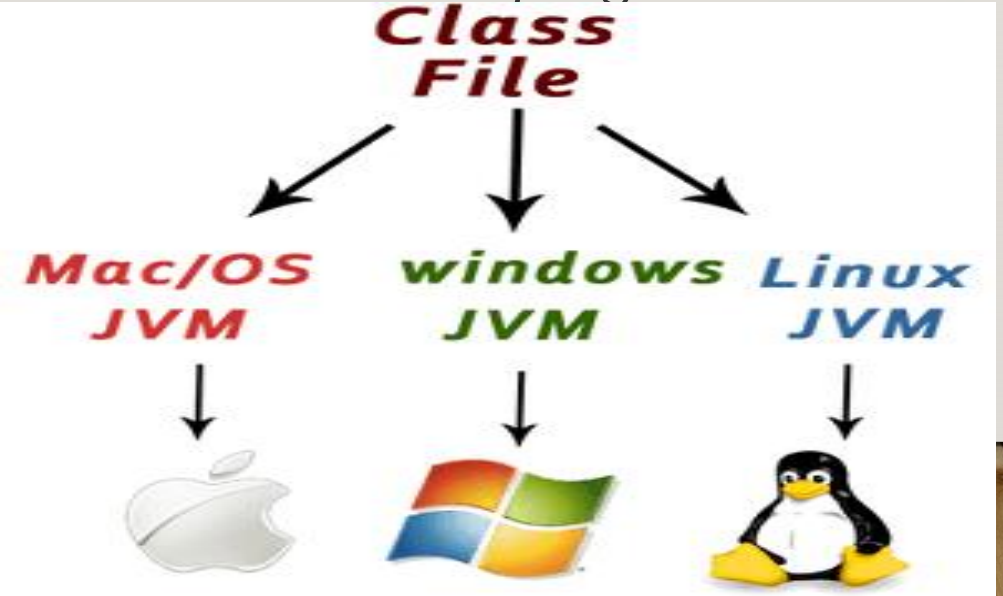
- **Object-oriented**

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 incorporate both data and behavior.

FEATURES(CONT.)

- 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. A platform is the hardware or software environment in which a program runs.



FEATURES(CONT.)

- Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

1 No explicit pointer

2 Java Programs run inside a virtual machine sandbox

FEATURES (CONT.)

- Robust

The English mining of Robust is strong. Java is robust because:

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

FEATURES (CONT.)

- Portable

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

- 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.

FEATURES (CONT.)

- 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.

- Multi-threaded

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.

FIRST JAVA PROGRAM | HELLO WORLD EXAMPLE

- The requirement for Java Hello World Example

1. Install the JDK if you don't have installed it, download the JDK and install it.
2. Set path of the jdk/bin directory. http://www.javatpoint.com/how-to-set-path-in-java
3. Create the Java program
4. Compile and run the Java program

CREATING HELLO WORLD EXAMPLE

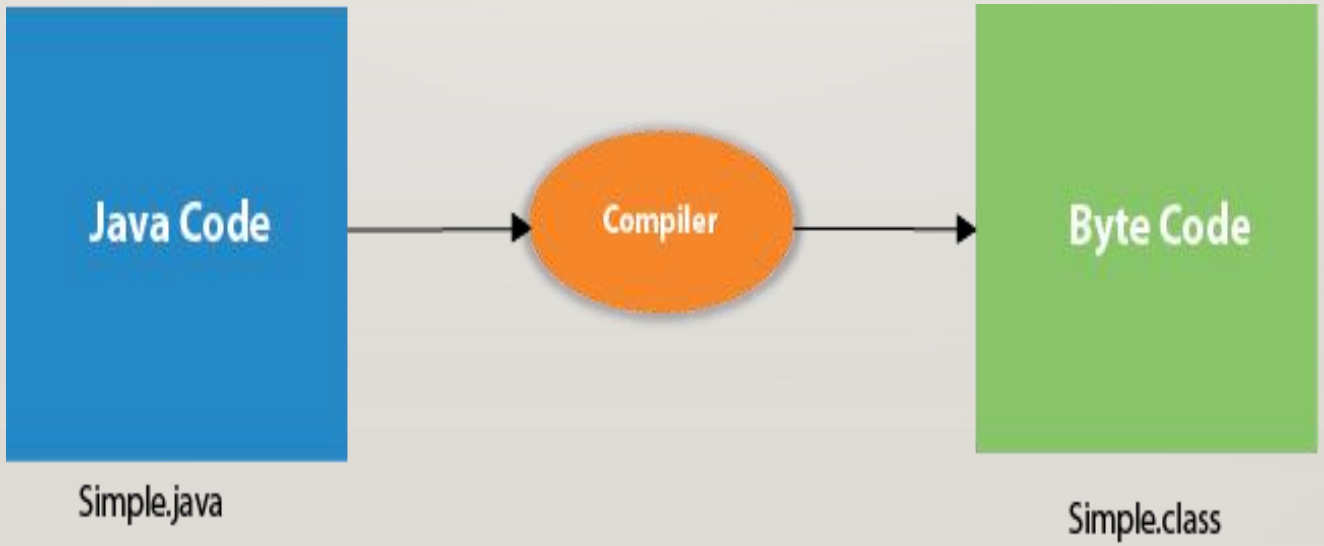
```
class Simple{
    public static void main(String args[]){
     System.out.println("Hello Java");
    }
}
```

To compile:	javac Simple.java
To execute:	java Simple

O/P Hello Java

COMPILATION FLOW:

- When we compile Java program using javac tool, the Java compiler converts the source code into byte code.



PARAMETERS USED IN FIRST JAVA PROGRAM

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that 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 creating 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.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class.

IN HOW MANY WAYS WE CAN WRITE A JAVA PROGRAM?

Parul® University

1) By changing the sequence of the modifiers, method prototype is not changed in Java.

- **static public void** main(String args[])

2) The subscript notation in the Java array can be used after type, before the variable or after the variable.

- **public static void** main(String[] args)
- **public static void** main(String []args)
- **public static void** main(String args[])
- 3) You can provide var-args support to the main() method by passing 3 ellipses (dots)

public static void main(String... args)

IN HOW MANY WAYS WE CAN WRITE A JAVA PROGRAM?

4) Having a semicolon at the end of class is optional in Java.

```
class A{
static public void main(String... args){
System.out.println("hello java4");
}
};
```

VALID JAVA MAIN() METHOD SIGNATURE

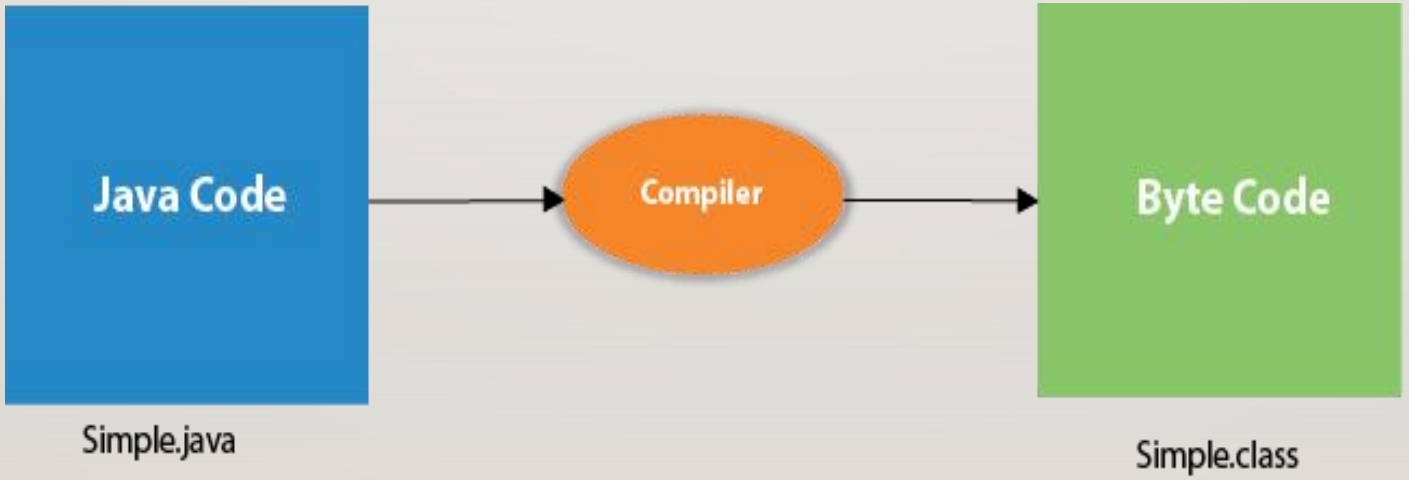
1. public static void main(String[] args)
2. public static void main(String []args)
3. public static void main(String args[])
4. public static void main(String... args)
5. static public void main(String[] args)
6. public static final void main(String[] args)
7. final public static void main(String[] args)
8. final strictfp public static void main(String[] args)

INVALID JAVA MAIN() METHOD SIGNATURE

1. `public void main(String[] args)`
2. `static void main(String[] args)`
3. `public void static main(String[] args)`
4. `abstract public static void main(String[] args)`

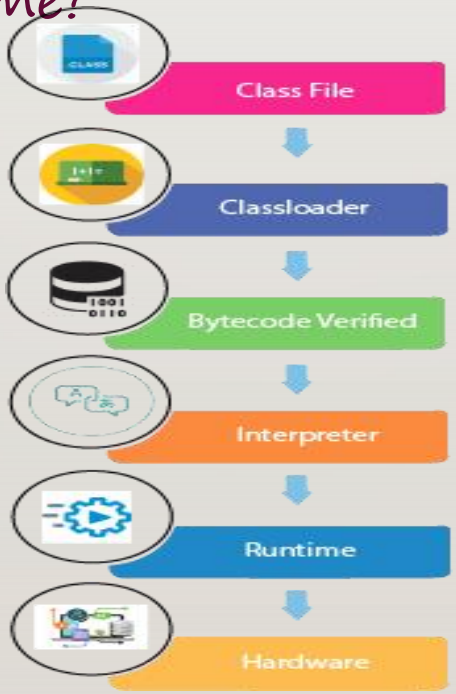
INTERNAL DETAILS OF HELLO JAVA PROGRAM

- What happens at compile time?



INTERNAL DETAILS OF HELLO JAVA PROGRAM

- What happens at runtime?



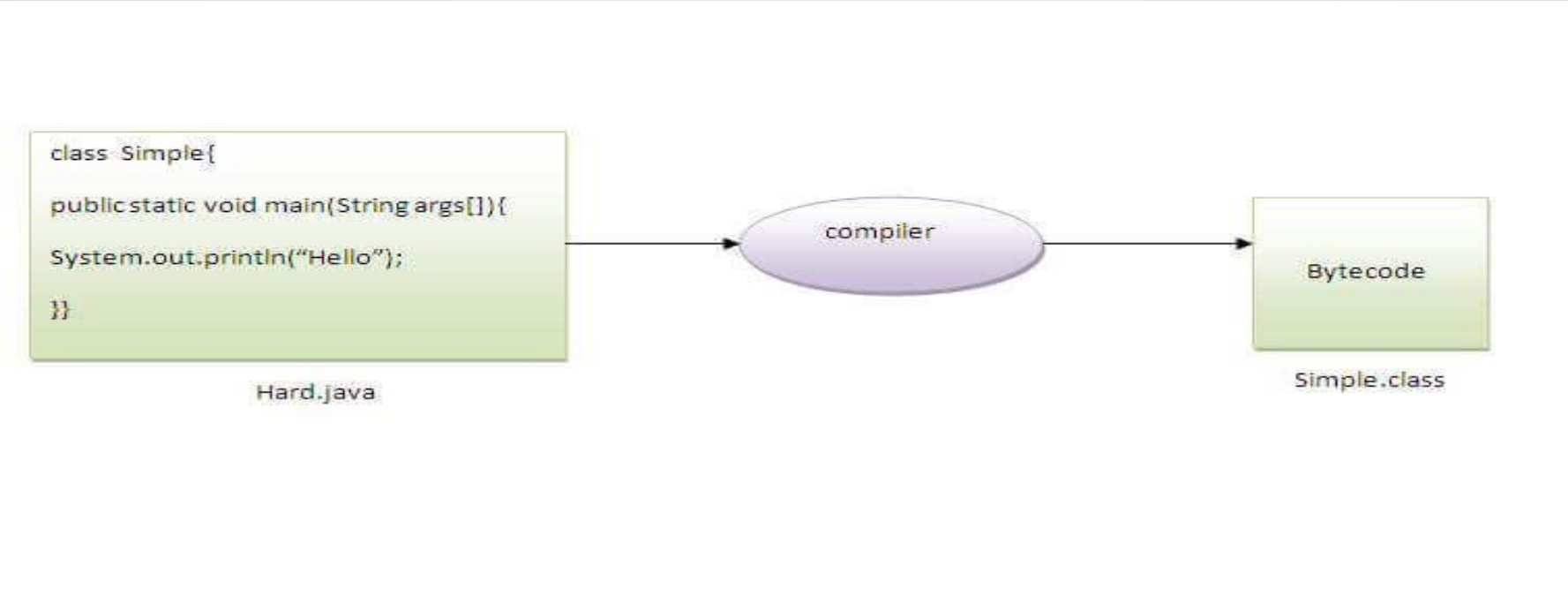
INTERNAL DETAILS OF HELLO JAVA PROGRAM

- **Classloader:** 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 rights to objects.
- **Interpreter:** Read bytecode stream then execute the instructions.

Q) Can you save a Java source file by another name than the class name?

Q) CAN YOU SAVE A JAVA SOURCE FILE BY ANOTHER NAME THAN THE CLASS NAME?

- Yes, if the class is not public.



JDK, JRE, AND JVM

Parul® University

JVM

- JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist.
- It is a specification that provides a runtime environment in which Java bytecode can be executed.
- It can also run those programs which are written in other languages and compiled to Java bytecode.

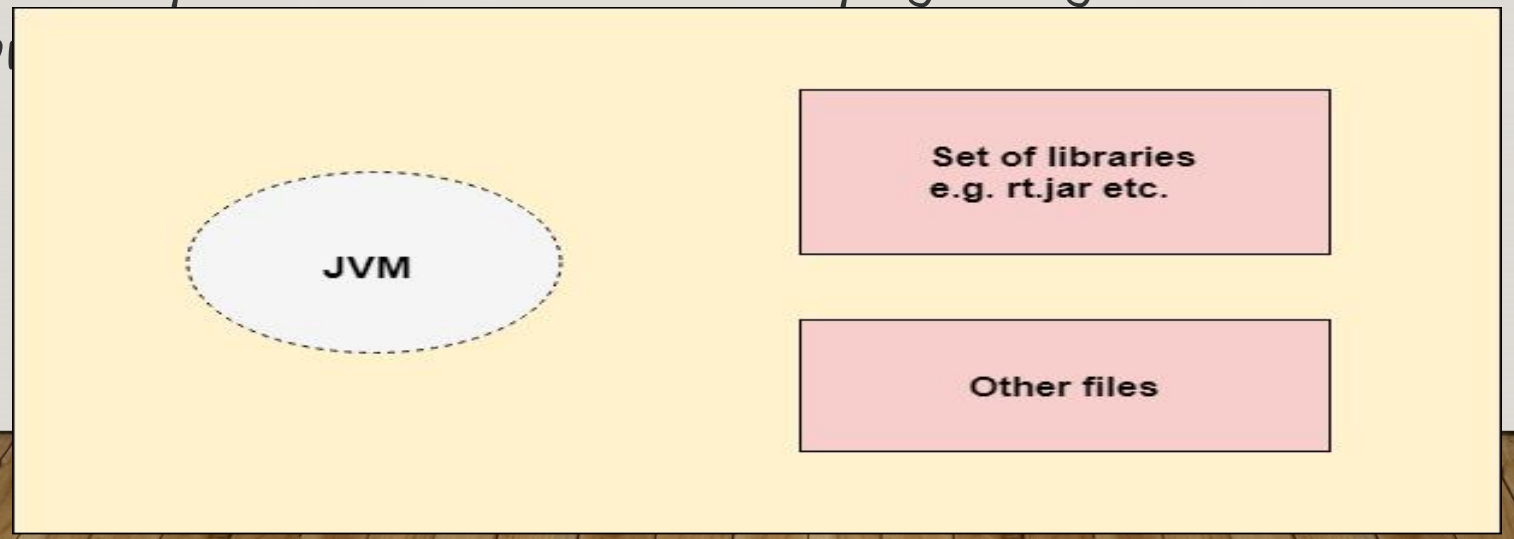
The JVM performs the following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JAVA RUNTIME ENVIRONMENT

- JRE

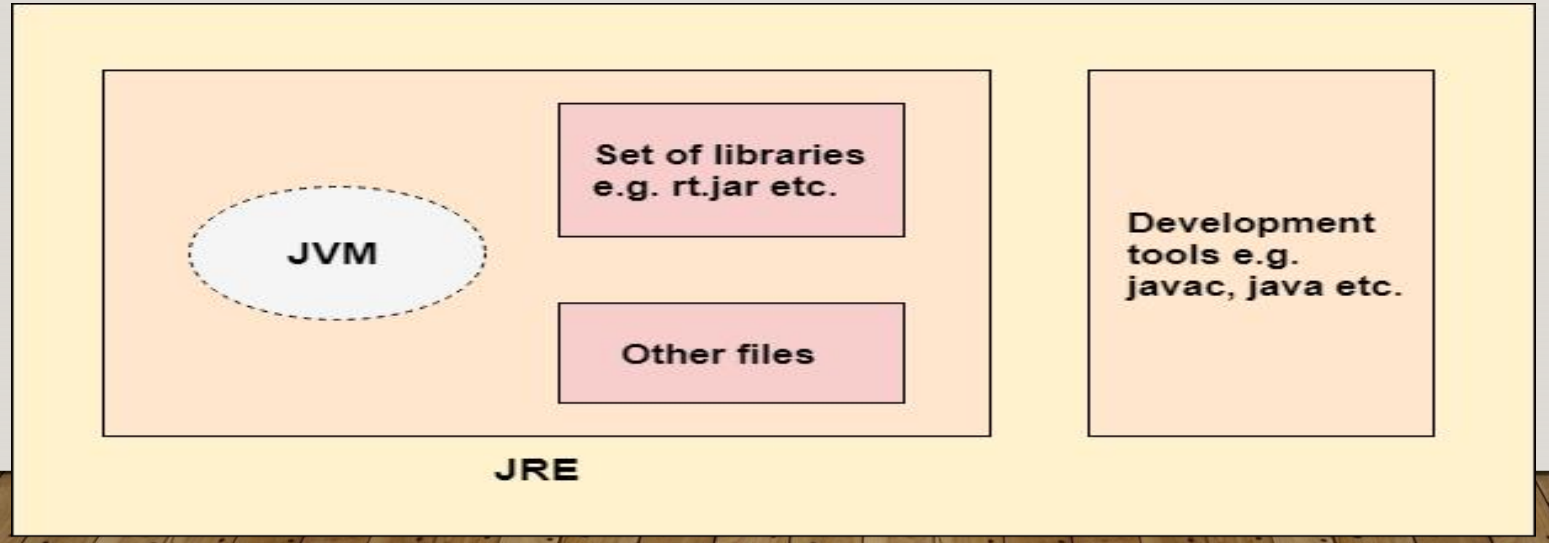
- JRE is an acronym for Java Runtime Environment. The Java Runtime Environment is a set of software tools which are used for developing Java applications.
- It is used to provide the runtime environment.
- It is the implementation of JVM. It physically exists. It contains a set of libr



JRE

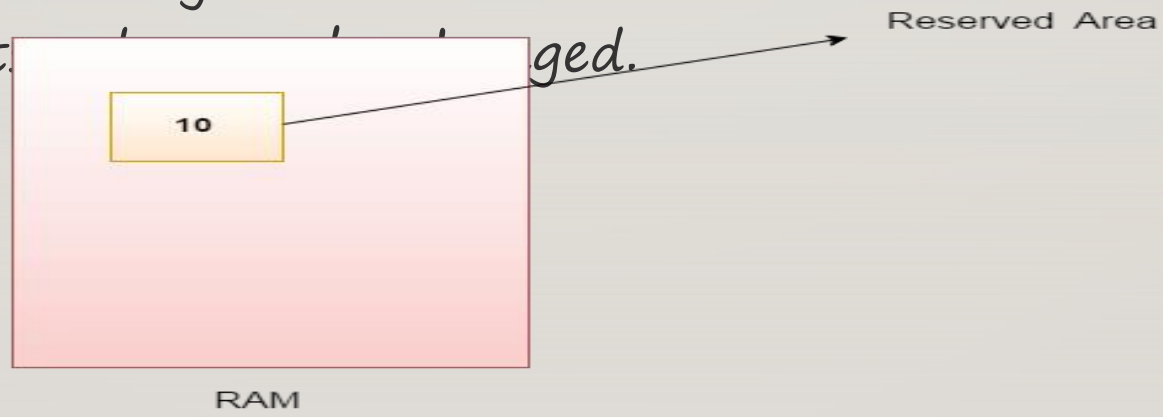
JAVA DEVELOPMENT KIT

- JDK is an acronym for Java Development Kit.
- The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets.
- It physically exists. It contains JRE + development tools.



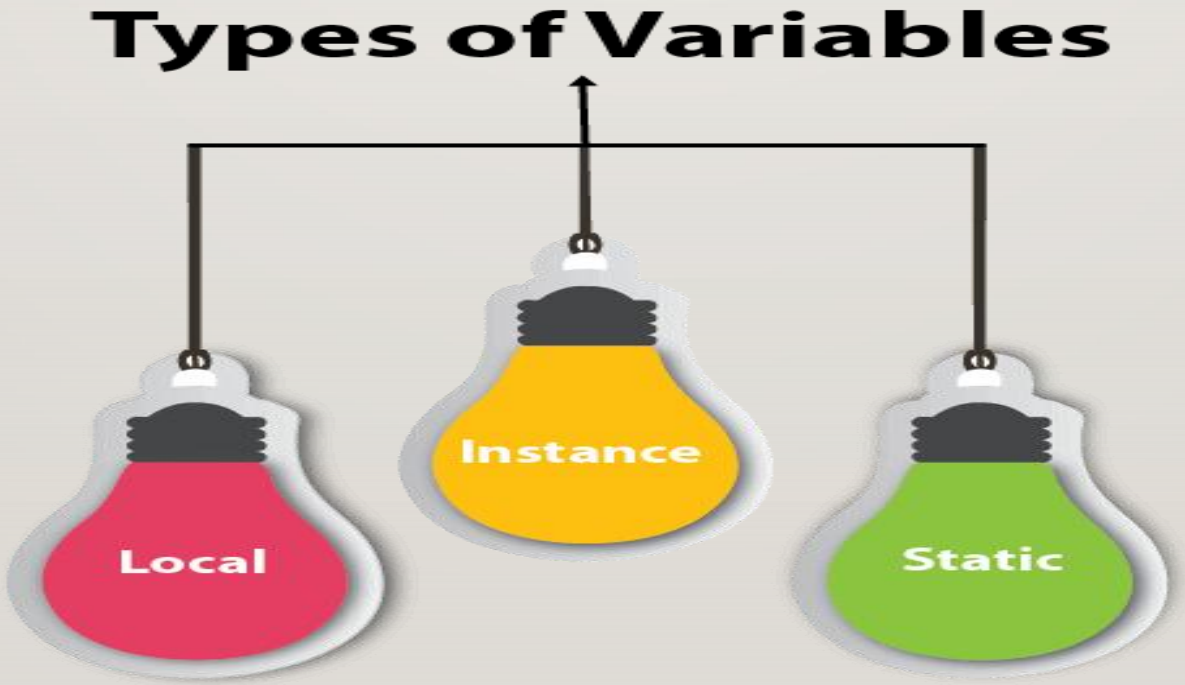
JAVA VARIABLES

- 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.
- A variable is the name of a reserved area allocated in memory. it is a name of the memory location. It is a combination of "vary + able" which means it [illegible] ged.



TYPES OF VARIABLES

- There are three types of variables in Java:



TYPES

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

EXAMPLE TO UNDERSTAND THE TYPES OF VARIABLES IN JAVA

```
1. public class A
2. {
3.     static int m=100;//static variable
4.     void method()
5.     {
6.         int n=90;//local variable
7.     }
8.     public static void main(String args[])
9.     {
10.         int data=50;//instance variable
11.     }
12.}//end of class
```

JAVA VARIABLE EXAMPLE: ADD TWO NUMBERS

```
public class Simple{
public static void main(String[] args){
int a=10;
int b=10;
int c=a+b;
System.out.println(c);
}
}
```

DATA TYPES IN JAVA

Parul® University

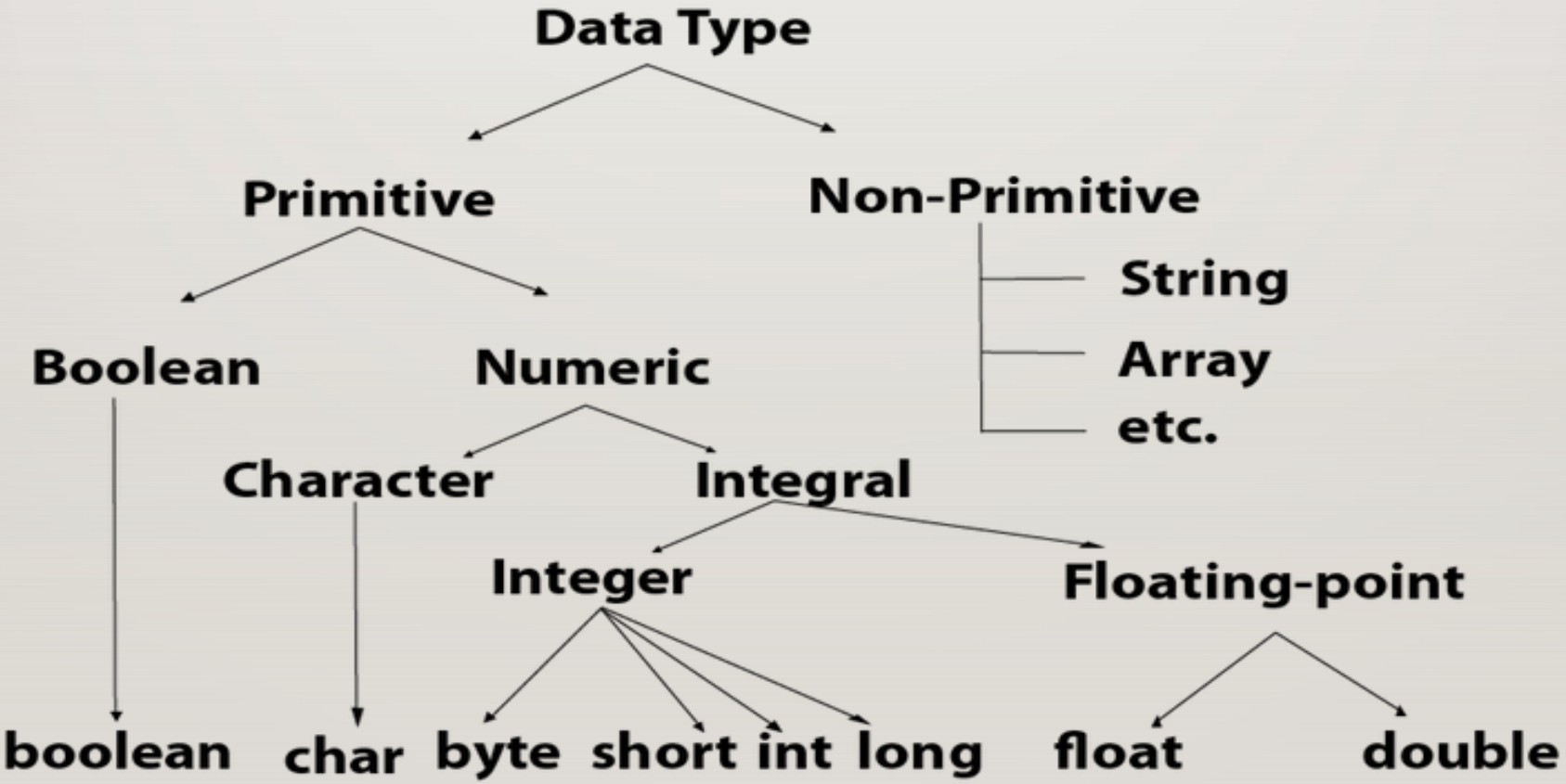
- Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

- In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.
- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type

DATA TYPE


Data Type
Primitive
Non-Primitive
String
Array
etc.
Boolean
Numeric
Character
Integral
Integer
Floating-point
boolean
char
byte
short
int
long
float
double

DATA TYPE VALUE AND SIZE

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

UNICODE SYSTEM

- Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

Why java uses Unicode System?

Before Unicode, there were many language standards:

- **ASCII** (American Standard Code for Information Interchange) for the United States.
- **ISO 8859-1** for Western European Language.
- **KOI-8** for Russian.
- **GB18030 and BIG-5** for chinese, and so on.

PROBLEM

Parul® University

This caused two problems:

1. A particular code value corresponds to different letters in the various language standards.
2. The encodings for languages with large character sets have variable length.Some common characters are encoded as single bytes, other require two or more byte

- S

To solve these problems, a new language standard was developed i.e. Unicode System.

In unicode, character holds 2 byte, so java also uses 2 byte for

lowest value:\u0000

highest value:\uFFFF

OPERATORS IN JAVA

- **Operator** in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

1. Unary Operator,
2. Arithmetic Operator,
3. Shift Operator,
4. Relational Operator,
5. Bitwise Operator,
6. Logical Operator,
7. Ternary Operator and
8. Assignment Operator.

JAVA OPERATOR PRECEDENCE

Parul® University

Operator Type	Category	Precedence
Unary	postfix	expr++ expr--
	prefix	++expr --expr +expr -expr ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

JAVA UNARY OPERATOR

- The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

```
1.public class OperatorExample{
2.public static void main(String args[]){
3.int x=10;
4.System.out.println(x++);//10 (11)
5.System.out.println(++x);//12
6.System.out.println(x--);//12 (11)
7.System.out.println(--x);//10
8.}}
```

Output:
10 12 12 10

JAVA UNARY OPERATOR EXAMPLE: ~ AND !

```
1.public class OperatorExample{
2.public static void main(String args[]){
3.int a=10;
4.int b=-10;
5.boolean c=true;
6.boolean d=false;
7.System.out.println(~a);//-11 (minus of total positive value which starts from 0)
8.System.out.println(~b);//9 (positive of total minus, positive starts from 0)
9.System.out.println(!c);//false (opposite of boolean value)
10.System.out.println(!d);//true
11.}}
```

Output:
-11 9 false true

JAVA ARITHMETIC OPERATORS

- Java Arithmetic Operator Example

```
1.public class OperatorExample{
2.public static void main(String args[]){
3.int a=10;
4.int b=5;
5.System.out.println(a+b);//15
6.System.out.println(a-b);//5
7.System.out.println(a*b);//50
8.System.out.println(a/b);//2
9.System.out.println(a%b);//0
10.}}
```

Output:
15 5 50 2 0

JAVA LEFT SHIFT OPERATOR

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java Left Shift Operator Example

```
1.public class OperatorExample{
2.public static void main(String args[]){
3.System.out.println(10<<2);//10*2^2=10*4=40
4.System.out.println(10<<3);//10*2^3=10*8=80
5.System.out.println(20<<2);//20*2^2=20*4=80
6.System.out.println(15<<4);//15*2^4=15*16=240
7.}}
```

Output:

```
40 80 80 240
```

JAVA RIGHT SHIFT OPERATOR

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

Java Right Shift Operator Example

```
1.public OperatorExample{
2.public static void main(String args[]){
3.System.out.println(10>>2);//10/2^2=10/4=2
4.System.out.println(20>>2);//20/2^2=20/4=5
5.System.out.println(20>>3);//20/2^3=20/8=2
6.}}
```

Output:

2 5 2

JAVA AND OPERATOR EXAMPLE: LOGICAL && AND BITWISE &

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.
The bitwise & operator always checks both conditions whether first condition is true or false.

```
1.public class OperatorExample{
2.public static void main(String args[]){
3.int a=10;
4.int b=5;
5.int c=20;
6.System.out.println(a<b&&a<c);//false && true = false
7.System.out.println(a<b&a<c);//false & true = false
8.}}
```

Output:

```
false false
```

JAVA OR OPERATOR EXAMPLE: LOGICAL ||

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.
The bitwise | operator always checks both conditions whether first condition is true or false.

```
1.public class OperatorExample{
2.public static void main(String args[]){
3.int a=10;
4.int b=5;
5.int c=20;
6.System.out.println(a>b||a<c);//true || true = true
7.System.out.println(a>b|a<c);//true | true = true
8.//|| vs |
9.System.out.println(a>b||a++<c);//true || true = true
10.System.out.println(a);//10 because second condition is not checked
11.System.out.println(a>b|a++<c);//true | true = true
12.System.out.println(a);//11 because second condition is checked
13.}}
```

Output:

true true true 10 true 11

JAVA TERNARY OPERATOR

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

Java Ternary Operator Example

```
1.public class OperatorExample{
2.public static void main(String args[]){
3.int a=2;
4.int b=5;
5.int min=(a<b)?a:b;
6.System.out.println(min);
7.}}
```

Output:

2

JAVA ASSIGNMENT OPERATOR

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

Java Assignment Operator Example

```
1.public class OperatorExample{
2.public static void main(String args[]){
3.int a=10;
4.int b=20;
5.a+=4;//a=a+4 (a=10+4)
6.b-=4;//b=b-4 (b=20-4)
7.System.out.println(a);
8.System.out.println(b);
9.}}
```

Output:

14 16

CONTROL STATEMENTS IN JAVA

- java compiler executes the code from top to bottom.
- The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code.
- Such statements are called control flow statements.
- It is one of the fundamental features of Java, which provides a smooth flow of program.

JAVA PROVIDES THREE TYPES OF CONTROL FLOW STATEMENTS.

1. Decision Making statements
    1. if statements
    2. switch statement
2. Loop statements
    1. do while loop
    2. while loop
    3. for loop
    4. for-each loop
3. Jump statements
    1. break statement
    2. continue statement

1 -DECISION-MAKING STATEMENTS:

- Decision-making statements decide which statement to execute and when.
- Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided.
- There are two types of decision-making statements in Java, i.e., **If statement and switch statement.**

A) IF STATEMENT:

- The "if" statement is used to evaluate a condition.
- The control of the program is diverted depending upon the specific condition.
- The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

1. Simple if statement
2. if-else statement
3. if-else-if ladder
4. Nested if-statement

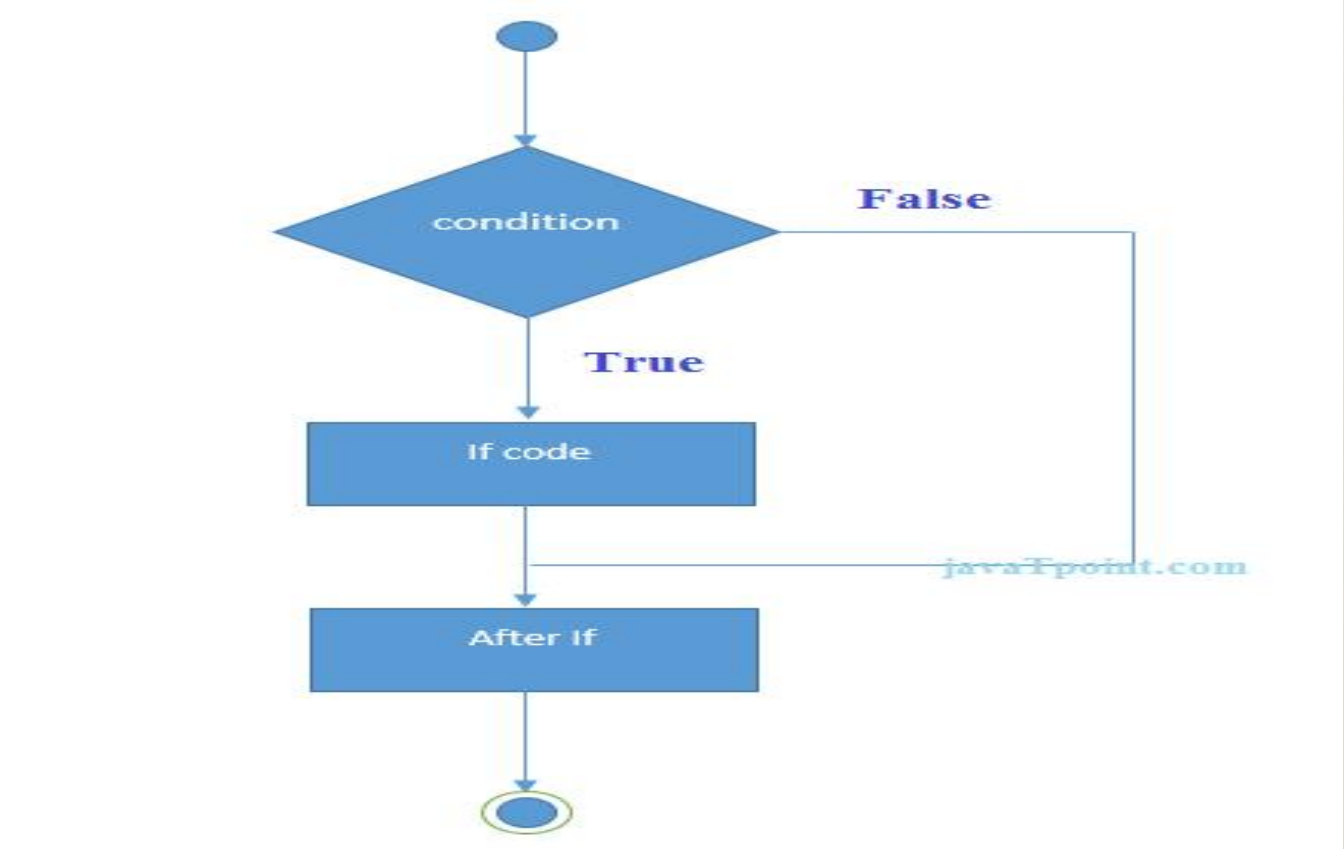
1) SIMPLE IF STATEMENT:

- It is the most basic statement among all control flow statements in Java.
- It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
if(condition) {
statement 1; //executes when condition is true
}
```

FLOW CHART



EXAMPLE SIMPLE IF STATEMENT

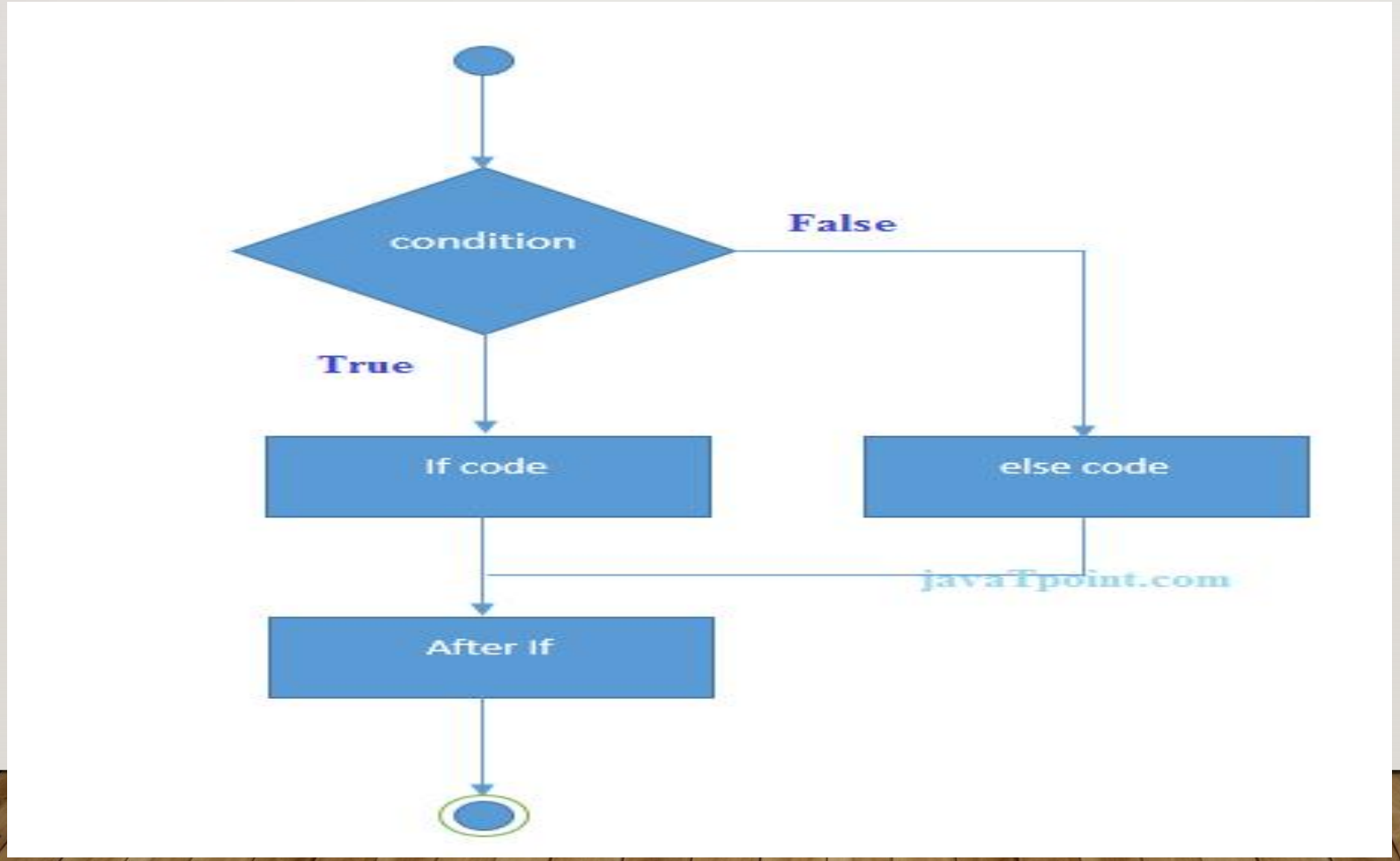
```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y > 20) {
System.out.println("x + y is greater than 20");
}
}
}
```

2) IF-ELSE STATEMENT

- The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block.
- The else block is executed if the condition of the if-block is evaluated as false.
- **Syntax:**

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
```

FLOW CHART


condition
False
True
If code
else code
javaTpoint.com
After If

STUDENT.JAVA

```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y < 10) {
System.out.println("x + y is less than      10");
}   else {
System.out.println("x + y is greater than 20");
}
}
}
```

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.

EXAMPLE

```
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);
}
}
```

3) IF-ELSE-IF LADDER:

- The if-else-if statement contains the if-statement followed by multiple else-if statements.

Syntax of if-else-if statement is given below.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

FLOW CHART

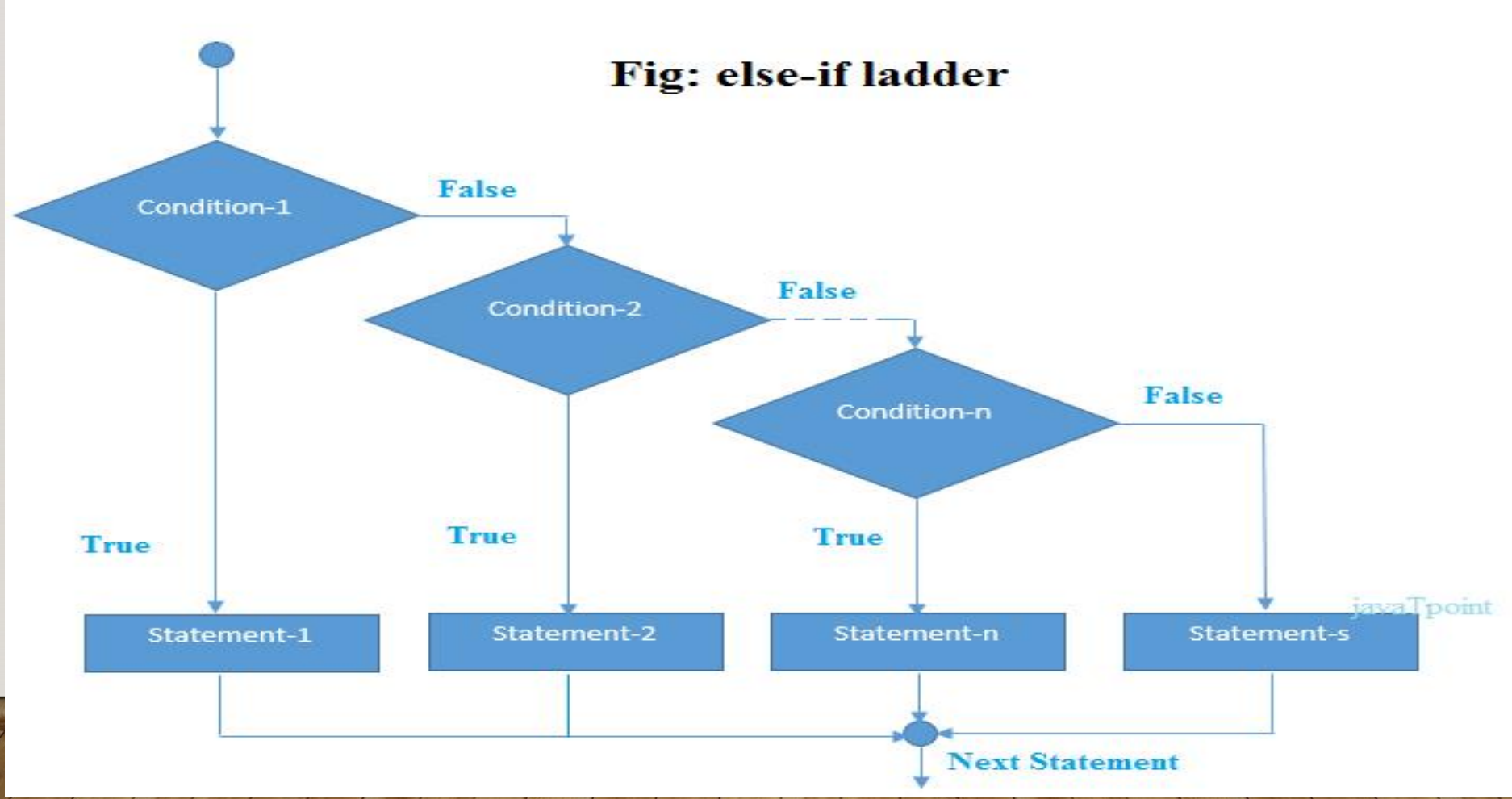


Fig: else-if ladder
Condition-1
False
Condition-2
False
Condition-n
False
True
True
True
Statement-1
Statement-2
Statement-n
Statement-s
javaTpoint
Next Statement

EXAMPLE

```
public class Student {
public static void main(String[] args) {
String city = "Delhi";
if(city == "Meerut") {
System.out.println("city is meerut");
}else if (city == "Noida") {
System.out.println("city is noida");
}else if(city == "Agra") {
System.out.println("city is agra");
}else {
System.out.println(city);
}
}
}
```

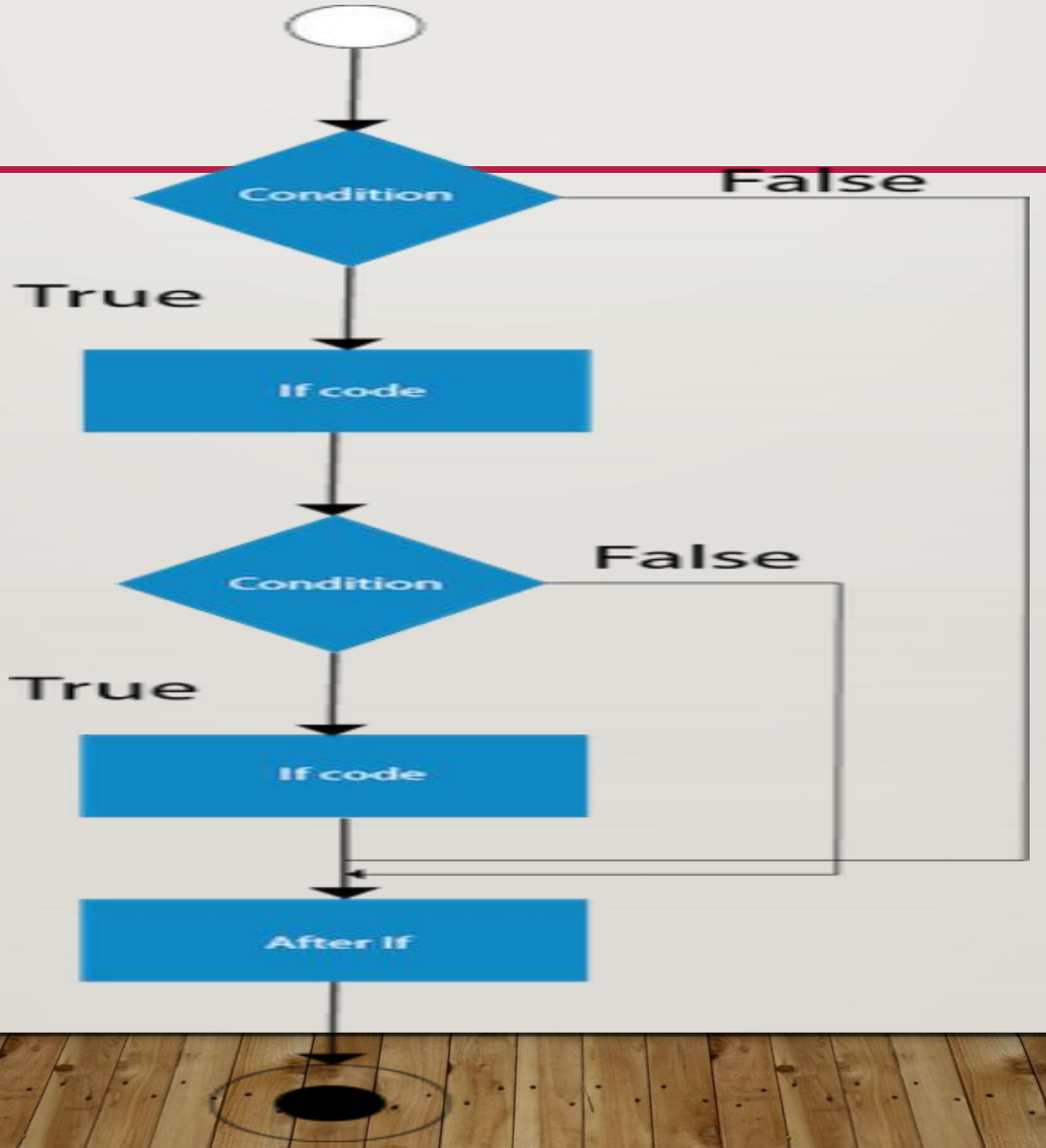
4. NESTED IF-STATEMENT

- In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
if(condition 2) {
statement 2; //executes when condition 2 is true
}
else{
statement 2; //executes when condition 2 is false
}
}
```

FLOW CHART


Condition
False
True
If code
Condition
False
True
If code
After If

EXAMPLE

```
public class Student {
public static void main(String[] args) {
String address = "Delhi, India";

if(address.endsWith("India")) {
if(address.contains("Meerut")) {
System.out.println("Your city is Meerut
");
}else if(address.contains("Noida")) {
System.out.println("Your city is Noida");
}else {
System.out.println(address.split(",")[0]);
}
}else {
System.out.println("You are not living in India");
}
}
}
```

SWITCH STATEMENT:

- In Java, Switch statements are similar to if-else-if statements.
- The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched
- The switch statement is easier to use instead of if-else-if statements.
- It also enhances the readability of the program.

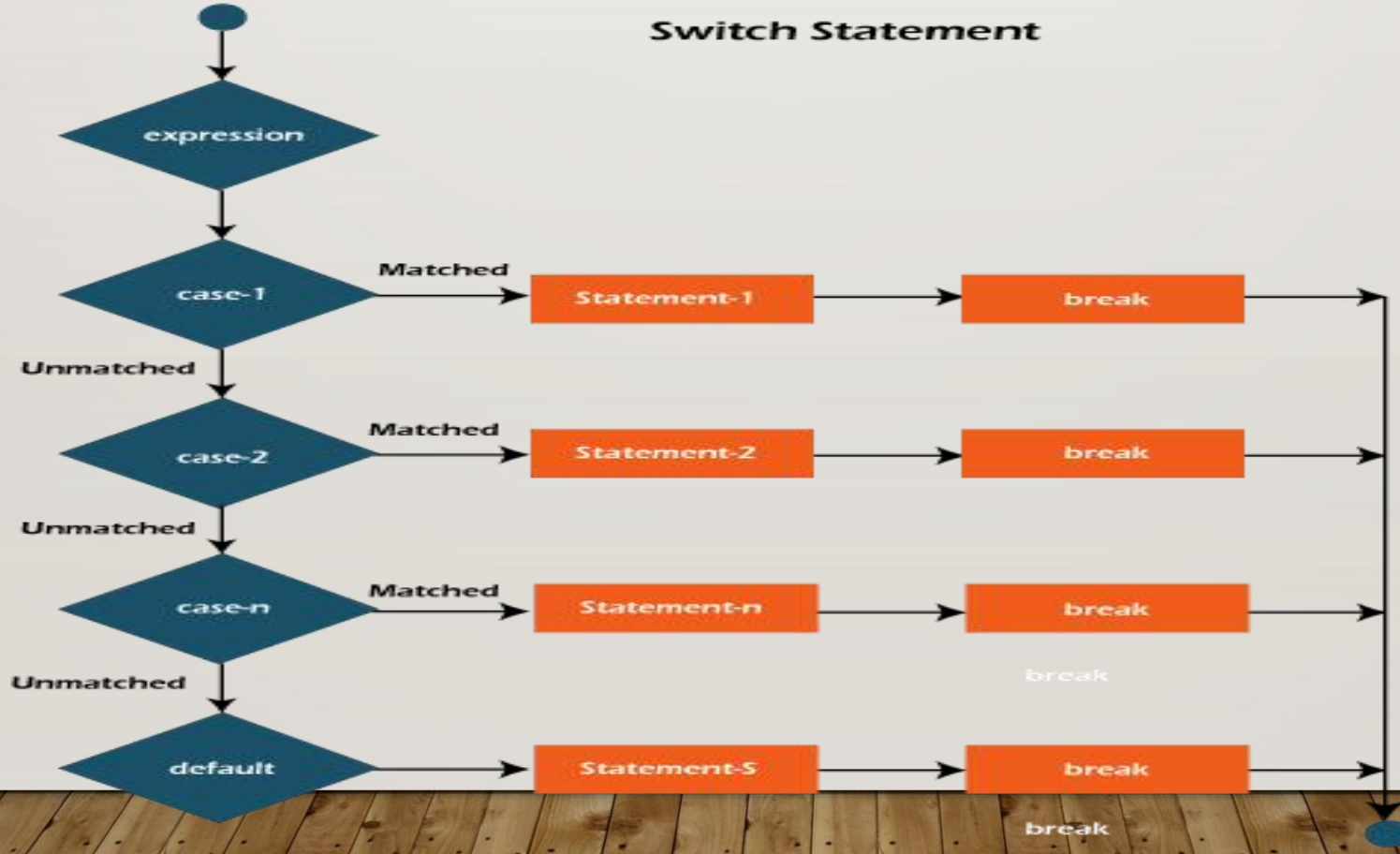
POINTS TO BE NOTED ABOUT SWITCH STATEMENT

- The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
- Cases cannot be duplicate
- Default statement is executed when any of the case doesn't match the value of expression. It is optional.
- Break statement terminates the switch block when the condition is satisfied. It is optional, if not used, next case is executed.
- While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

THE SYNTAX TO USE THE SWITCH STATEMENT IS GIVEN BELOW.

```
switch (expression){
    case value1:
     statement1;
     break;
    .
    .
    .
    case valueN:
     statementN;
     break;
    default:
     default statement;
}
```

FLOW CHART OF SWITCH CASE



EXAMPLE

```
public class Student implements Cloneable {
public static void main(String[] args) {
int num = 2;
switch (num){
case 0:
System.out.println("number is 0");
break;
case 1:
System.out.println("number is 1");
break;
default:
System.out.println(num);
}}}
```

JAVA NESTED SWITCH STATEMENT

Parul® University

- In Java, a **nested switch-case** is a switch statement inside another switch statement. This allows you to further refine the decision-making process based on more conditions.

```
switch (expression1) {
    case value1:
        // some code for outer switch case
        switch (expression2) {
            case value2:
                // some code for inner switch case
                break;
            case value3:
                // some code for inner switch case
                break;
            default:
                // default code for inner switch
        }
        break;
    case value4:
        // code for another outer switch case
        break;
    default:
        // default code for outer switch
}
```

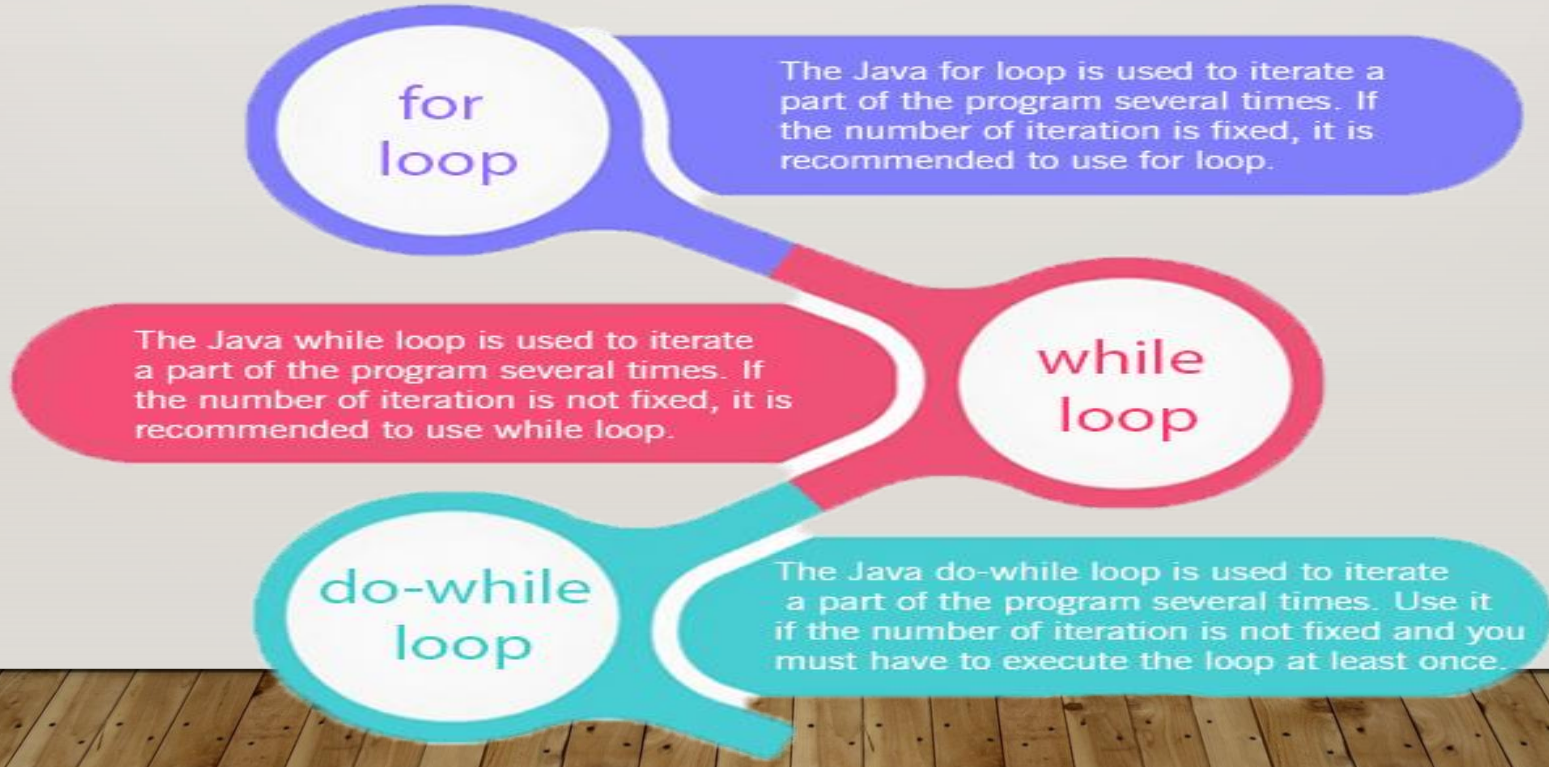
EXAMPLE

```
public class NestedSwitchExample {
    public static void main(String[] args) {
        int day = 2;
        String mealTime = "Lunch";

        switch (day) {
            case 1:
                System.out.println("Monday");
                break;
            case 2:
                System.out.println("Tuesday");
// Nested switch for meal times
                switch (mealTime) {
                    case "Breakfast":
                        System.out.println("Pancakes for breakfast.");
                        break;
                    case "Lunch":
                        System.out.println("Sandwich for lunch.");
                        break;
                    case "Dinner":
                        System.out.println("Pasta for dinner.");
                        break;
                    default:
                        System.out.println("Invalid meal time.");
                }
                break;
            default:
                System.out.println("Invalid day.");
        }
    }
}
```

LOOPS IN JAVA

- 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.



JAVA SIMPLE FOR LOOP

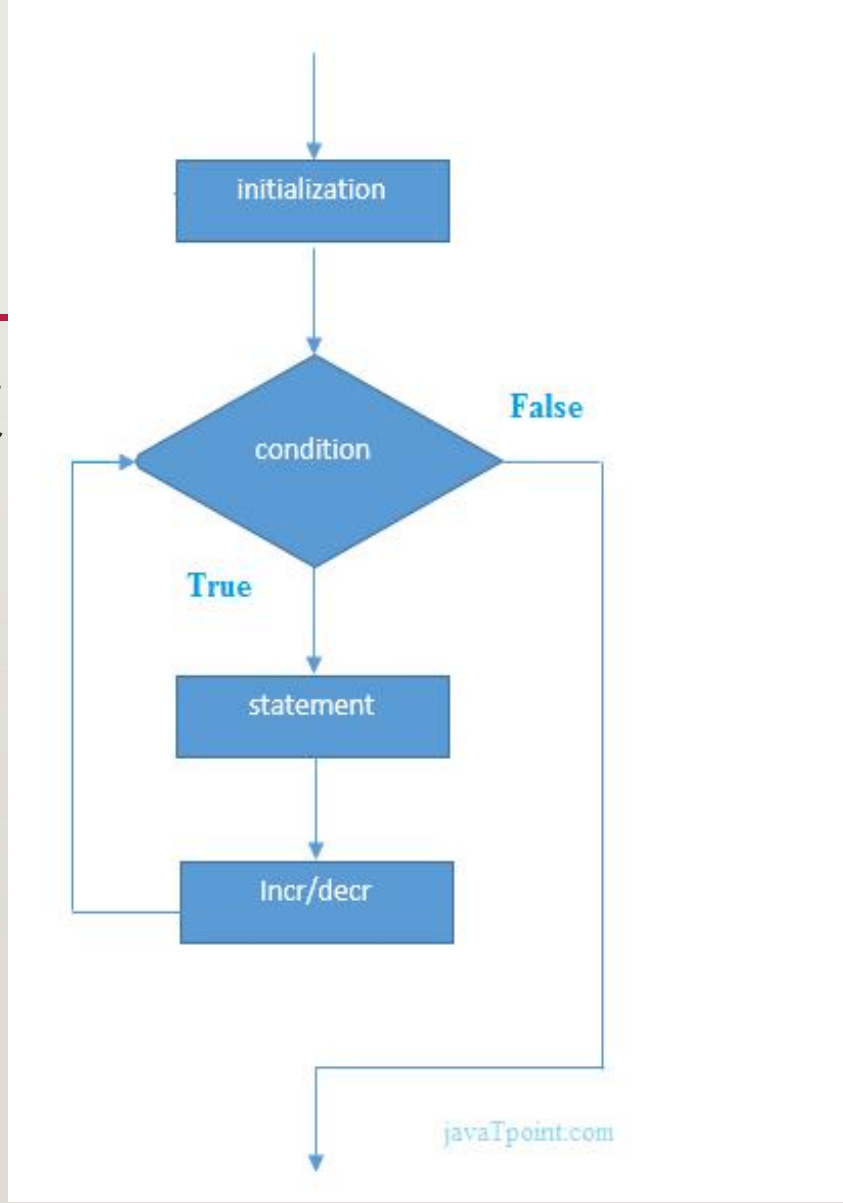
- A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **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. It is an optional condition.
2. **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. It is an optional condition.
3. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.
4. **Statement**: The statement of the loop is executed each time until the second condition is false.

SYNTAX

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

Flow chart



EXAMPLE OF FOR LOOP

```
//Java Program to demonstrate the example of for loop
//which prints table of 1
public class ForExample {
public static void main(String[] args) {
    //Code of Java for loop
    for(int i=1;i<=10;i++){
        System.out.println(i);
    }
}
}
```

JAVA NESTED FOR LOOP

- If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

```
public class NestedForExample {
public static void main(String[] args) {
//loop of i
for(int i=1;i<=3;i++){
//loop of j
for(int j=1;j<=3;j++){
        System.out.println(i+" "+j);
}//end of i
}//end of j
} }
```

JAVA FOR-EACH LOOP

- he 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 the basis of elements and not the index. It returns element one by one in the defined variable.
- **Syntax:**

```
for(data_type variable : array_name){
//code to be executed
}
```

EXAMPLE

```
//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 WHILE LOOP

- The Java *while loop* is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.
- The while loop is considered as a repeating if statement. If the number of iteration is not fixed, it is recommended to use the while loop
- **syntax**

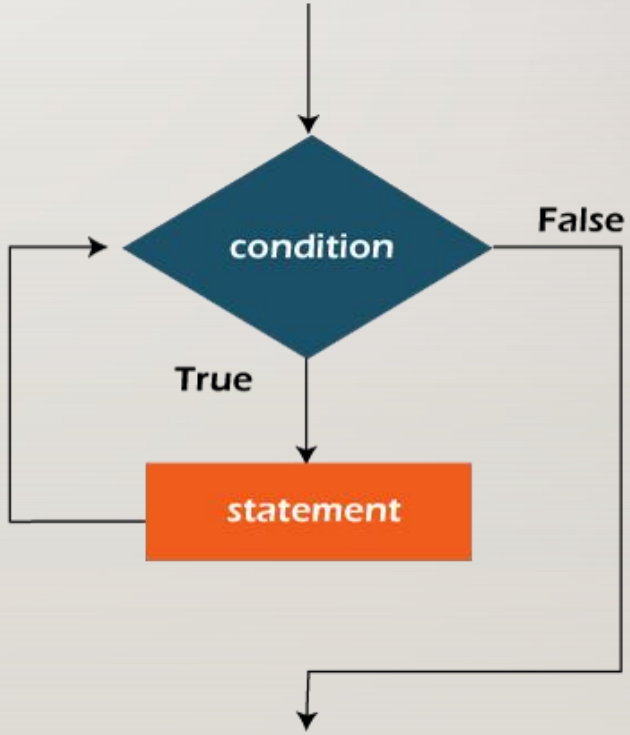
```
while (condition){
//code to be executed
I ncrement / decrement statement
}
```

THE DIFFERENT PARTS OF DO-WHILE LOOP:

- 1. Condition: It is an expression which is tested. If the condition is true, the loop body is executed and control goes to update expression. When the condition becomes false, we exit the while loop.
- **Example**:
- i <=100
- 2. Update expression: Every time the loop body is executed, this expression increments or decrements loop variable.
- **Example**:
- i++;

FLOWCHART OF JAVA WHILE LOOP

- the important thing about while loop is that, sometimes it may not even execute. If the condition to be tested results into false, the loop body is skipped and first statement after the while loop will be executed.



EXAMPLE OF WHILE LOOP

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

JAVA INFINITIVE WHILE LOOP

- If you pass **true** in the while loop, it will be infinitive while loop.
- **Syntax:**

```
while(true){

//code to be executed

}
```

example

```
public class WhileExample2 {
public static void main(String[] args) {
 // setting the infinite while loop by passing true to the conditi
on
  while(true){
        System.out.println("infinitive while loop");
    }
}
}
```

JAVA DO-WHILE LOOP

- The Java *do-while loop* is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop.
- Java do-while loop is called an **exit control loop**. Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body. The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

```
do{
//code to be executed / loop body
//update statement
}while (conditi
```

THE DIFFERENT PARTS OF DO-WHILE LOOP:

- 1. Condition: It is an expression which is tested. If the condition is true, the loop body is executed and control goes to update expression. As soon as the condition becomes false, loop breaks automatically.

Example:

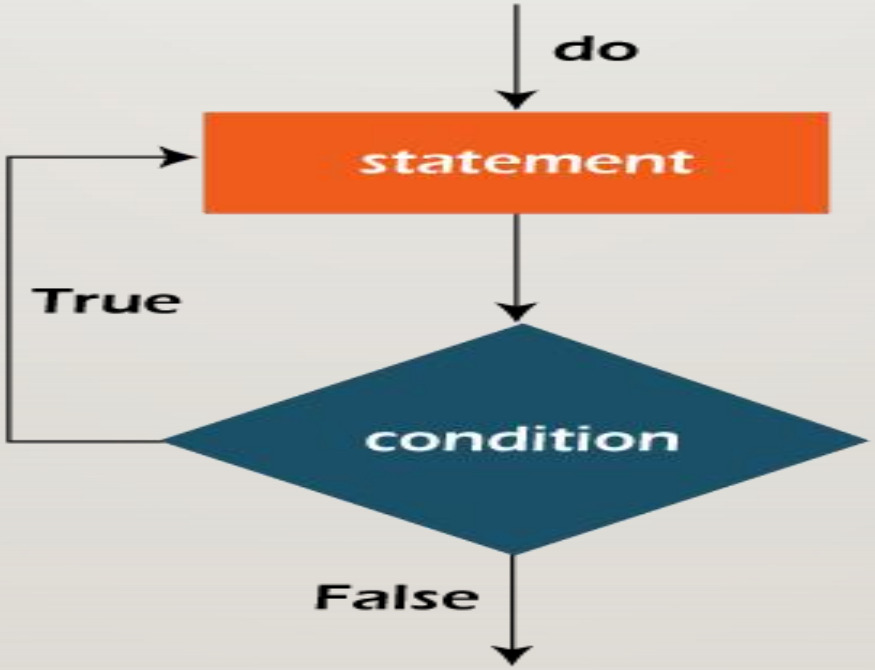
i <=100

2. Update expression: Every time the loop body is executed, the this expression increments or decrements loop variable.

Example:

i++;

FLOWCHART OF 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);
}
}
```

Output:

```
1 2 3 4 5 6 7 8 9
10
```

Java Infinitive do-while Loop

If you pass **true** in the do-while loop, it will be infinitive do-while loop.

Syntax:

```
1.do{
2.//code to be executed
3.}while(true
```

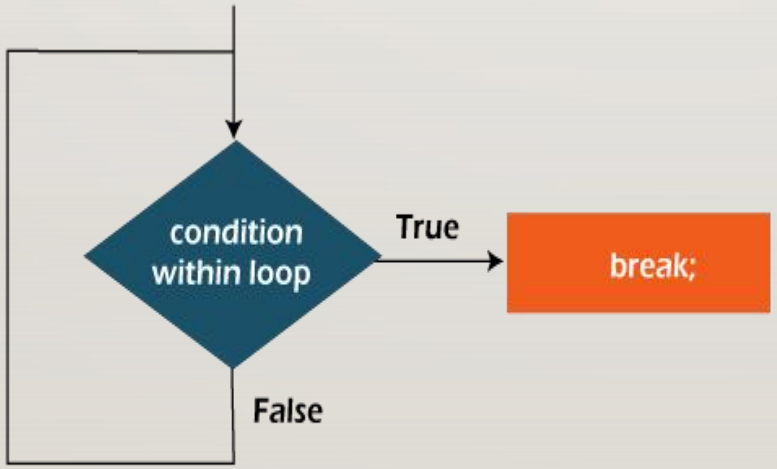
JAVA BREAK STATEMENT

- 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.

FLOW CHART

- Syntax

```
jump-statement;
break;
```



Flowchart of break statement

```
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);
    }
}
}
```

JAVA CONTINUE STATEMENT

- The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.
- 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.

CONT.

- Syntax

```
jump-statement;

continue;
```

```
//Java Program to demonstrate the use of contin
ue 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 statemen
t
        }        System.out.println(i);
  }
}
}
```

JAVA RETURN KEYWORD

The `return` keyword finishes the execution of a method, and can be used to return a value from a method.

```
public class Main {
static int myMethod(int x) {
return 5 + x; }
public static void main(String[] args)
{ System.out.println(myMethod(3));
}
}
// Outputs 8 (5 + 3)
```

JAVA ARRAYS

- Normally, an array is a collection of similar type of elements which has 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.
- 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.

ADVANTAGE AND DISADVANTAGE

Advantage

- **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 ARRAY IN JAVA

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax to Declare an Array in Java

1.dataType[] arr; (or)

2.dataType []arr; (or)

3.dataType arr[];

Instantiation of an Array in Java

arrayRefVar=new datatype[size];

EXAMPLE

```
//Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
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++)//length is the property of array
System.out.println(a[i]);
}}
```

DECLARATION, INSTANTIATION AND INITIALIZATION OF JAVA ARRAY

- We can declare, instantiate and initialize the java array together by:
- int a[]={33,3,4,5};//declaration, instantiation and initialization

example

```
//Java Program to illustrate the use of declaration, instantiation
//and initialization of Java array in a single line
class Testarray1{
public static void main(String args[]){
int a[]={33,3,4,5};//declaration, instantiation and initialization
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

MULTIDIMENSIONAL ARRAY IN JAVA

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

Syntax to Declare Multidimensional Array in Java

1.dataType[][] arrayRefVar; (or)

2.dataType [][]arrayRefVar; (or)

3.dataType arrayRefVar[][]; (or)

4.dataType []arrayRefVar[];

CONT.

- Example to instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

- Example to initialize Multidimensional Array in Java

```
1.arr[0][0]=1;
2.arr[0][1]=2;
3.arr[0][2]=3;
4.arr[1][0]=4;
5.arr[1][1]=5;
6.arr[1][2]=6;
7.arr[2][0]=7;
8.arr[2][1]=8;
9.arr[2][2]=9;
```

EXAMPLE OF MULTIDIMENSIONAL JAVA ARRAY

```
//Java Program to illustrate the use of multidimensional array
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();
}
}}
```

Parul® University