REVA UNIVERSITY
Bengaluru, India

M23DE0202 – Object Oriented Programming using JAVA

II Semester MCA Academic Year : 2024 - 2025

School of Computer Science and Applications

Pinaka Pani. R
Assistant Professor

www.reva.edu.in

Unit I

Object Oriented Programming using JAVA

Lecture – 1

Unit-01: Introduction to the Fundamentals of Java Programming:

An overview of Java, Internal Details of JVM, Difference between JDK, JRE and JVM, Features of Java, Data types, Tokens, Type Conversion, Casting, Arrays, Operators and Precedence, Branching and Looping statements. Classes, objects, methods, Constructors, this, Lambda Expressions, Java Memory Management - Garbage collector.

Java is a versatile, high-level, object-oriented programming language designed for creating platform-independent applications.

It is widely used for web development, mobile apps, enterprise applications, cloud computing, and more.

Why Learn Java in MCA?

- **Industry Relevance:** Java is one of the most in-demand programming languages in the software industry.
- **Platform Independence:** Java's "Write Once, Run Anywhere" (WORA) feature allows programs to run on any platform with a Java Virtual Machine (JVM).
- **Versatility:** Java is used in desktop applications, web applications, mobile applications (Android), cloud computing, big data, and more.
- **Strong Community Support:** Java has a vast and active developer community, making it easier to find help and resources.

Java Project Ideas:

1. Student Management System
2. Library Management System
3. Online Examination System
4. E-commerce Application
5. Chat Application using Sockets

LECTURE 1: Topics to be learned from this class

- History of JAVA
- Java Version History
- Features of JAVA

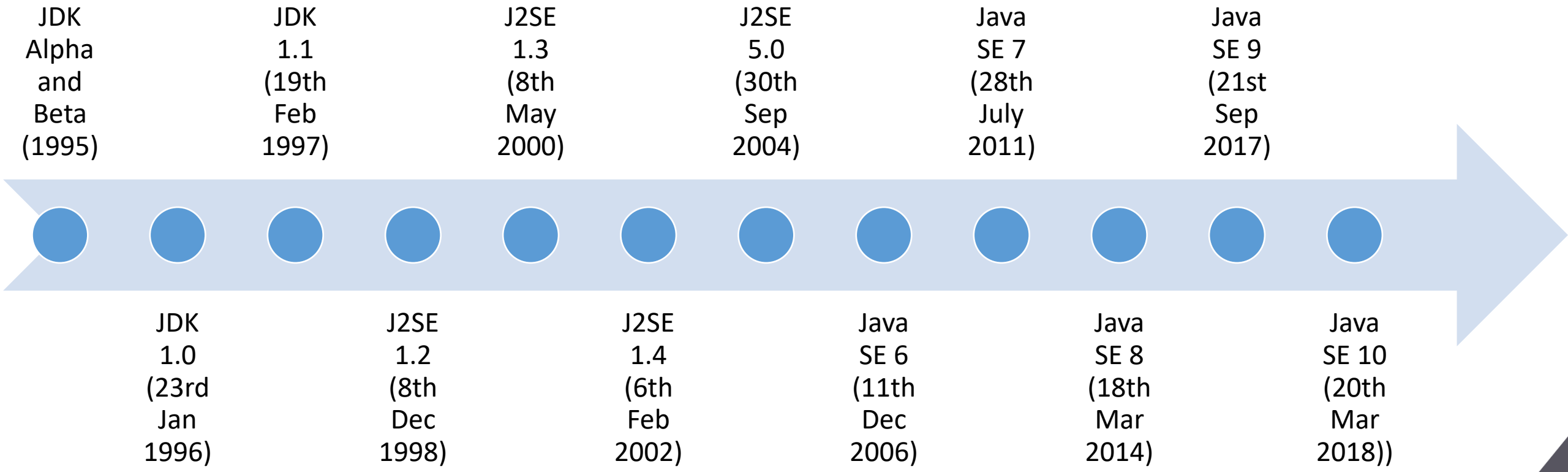
- **HISTORY Of JAVA**
  - The history of Java starts with the Green Team.
  - Designed for interactive television(iTV).
  - In 1991, **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** of **Sun Microsystems** began working on the "**Green Project**", aiming to create a programming language for consumer **electronic devices**.
  - Their initial focus was on **interactive devices**, particularly interactive **television** (iTV).
  - The team initially considered using C**++,** but its **complexity** and **platform dependency** made it unsuitable for iTV devices.
  - Principles for creating Java programming were "**Simple**, **Robust**, **Portable**, **Platform-independent**, **Secured**, **High Performance**, **Multithreaded**, **Architecture Neutral**, **Object-Oriented**, **Interpreted**, and **Dynamic**".
  - Java is used in **internet programming**, **mobile devices**, **games**, **e-business solutions**, etc.

Why Java Programming named "Java"?

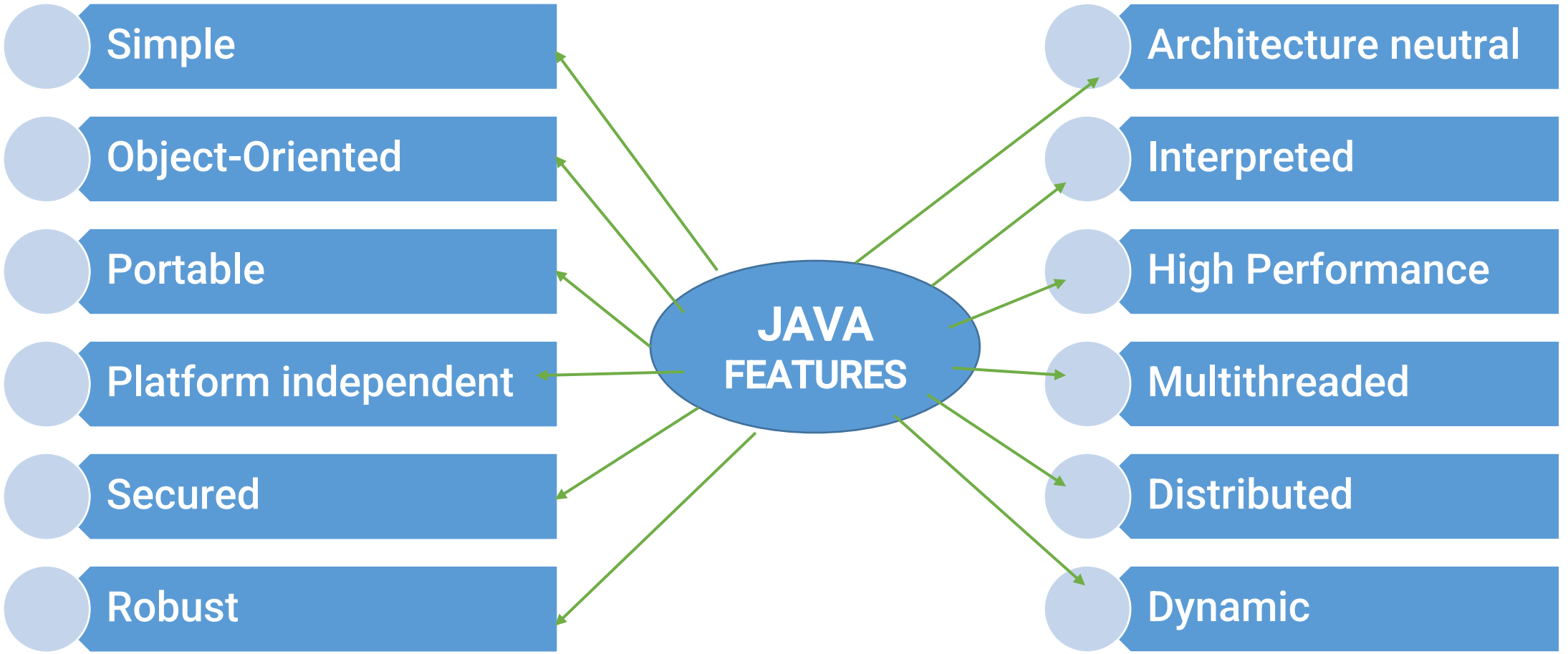
- The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc.
- Java is an island of Indonesia where the first coffee was produced (called java coffee).
- Java is just a name, not an acronym.
- Initially developed by **James Gosling** at **Sun Microsystems**.
- **JDK 1.0 released in(January** 23, **1996**).


Green Team
1991
Patrick Naughton
Mike Sheridan
James Gosling

Java version History



Features of JAVA



1. Simple and Easy to Learn

It is because of following reasons:

- Java comprises the same syntax as C, and C++.
- It holds automatic garbage collection features.
- Java eliminated its unused features.
- It keeps bringing regular updates for better performance.
- Java has its own community to make learning and using Java easy.

2. Object-Oriented Programming

Almost everything written in Java is object and class, making it a true object-oriented programming (OOP) language.

The basic concept of OOP is:

- **Object:** Object is a real-world entity in Java that encompasses **state**, **functionality**, and **identity**.
- **Class:** Class is a **logical entity** which includes a group of objects with common properties. It contains **fields**, **methods**, **constructors**, **blocks**, **nested classes** and **interfaces**.
- **Inheritance:** It's a concept in Java through which developers can **create new classes** built upon existing classes to **achieve runtime polymorphism**.
- **Polymorphism:** A Mechanism in Java through which you can **perform** a **single action** in **multiple ways**. Polymorphism can be of two types- Compile time and runtime.
- **Abstraction:** It's a method to **hide internal processing** and show only **essential** things to the users.

3. Platform Independence:

Java is not limited to any specific machine and dependent on other factors to run.

The **Java platform** is **independent** because:

- It uses a **runtime environment** of its own, i.e. JVM.
- It is a software-based platform that runs on top of other hardware-based platforms.
- Its code can be executed on multiple platforms, including Windows, Linux, Sun Solaris, and Mac/OS.
- The Java code is compiled by the **compiler** and **converted** into **bytecode**.

4. Automatic Memory Management:

- **Automatic memory management** is a crucial feature of Java programming.
- It helps in **Create** high-performance system.
- Automatically **allocate** and **free up space for objects**.
- Eliminating the worries about **memory management**.
- Issues like **object destruction** don't occur and there is no need to add memory management logic.

5. Security

Java programming language is known for its **security**. We can create virus-free systems.

- It uses its own runtime environment- JVM.
- Java includes a security manager, which determines what resources a class can access, such as reading and writing to the local disk.
- In Java run time, a class loader separates the package for the classes of the local file system from the files imported from network sources.
- Java also consists of Byte code Verifier, which checks the code fragments for illegal code.

6. Rich API

- Java Advanced Imaging (JAI)
- Java Data Objects (JDO)
- Java Media Frameworks (JMF)
- Java Persistence API (JPA)
- Java 3D (J3D)

7. Multithreading

- It lets Java developers execute multiple threads at the same time.
- It's used to achieve multitasking.
- It saves time.
- It's mostly used in games and animation.
- Threads work independently and don't impact other threads, even if created simultaneously.

8. High Performance

Java is a programming language with a high-performance rate.

- Java uses bytecode that can be easily translated into native machine code.
- It has multiple easy-to-use frameworks.
- It is compatible with multiple platforms, including Windows, Linux, Sun Solaris, and Mac/OS.
- It is a write-once run-anywhere language (WORA).
- Java also automatically clears the garbage to enhance its performance.

9. Scalability

Java offers its users scalability, which means it can deal with more and more users and works

- Java is an object-oriented programming language.
- It has the ability to handle large databases.
- Java doesn’t require multiple resources while running.
- It uses multithreading and multiprocessing.
- Java includes a higher volume of code.

10. Simplified Syntax

Java is one of the most used programming languages with **simplified** syntax.

- Java programming language is **easily understood** by the **compiler** and computer.
- It is **case-sensitive**, which means “hello world” or “HelloWorld” are two different things.
- It includes a bunch of **objects** and **classes**.
- The Java code statements end with **semicolons (;).**
- The delimiters **“{…}”** that denote a block of code.

SUMMARY

- History of JAVA
- Java Version History
- Features of JAVA

- **QUIZ FOR THE CLASS**

1. Applications of JAVA Programming…….

A. Desktop Applications

B. Web Applications

B. Enterprise Applications

D. All of the above

2. Features of JAVA Programming……

A. Platform independent

B. Secured

C. Multithreaded

D. All of the above

Topics to be learned from this class

- How JAVA Differ from C and C++
- JAVA Environment

How JAVA Differ from C

Feature	C	Java
Memory Management	• C has manual memory management, requiring the programmer to allocate and free memory explicitly.	• Java has automatic memory management through garbage collection.
Type of Language	• C is a procedural programming language.	• Java is an object-oriented programming language.
Compilation	• C code is compiled to machine code for a specific platform.	• Java code is compiled to bytecode, which is then interpreted by the Java Virtual Machine (JVM) on any platform.
Portability	• C code is not portable across different platforms without modification.	• Java code is portable across different platforms.
Syntax	• C has a syntax that is close to machine language and can be difficult to read and write.	• Java has a syntax that is closer to English and is designed to be easier to read and write.
Security	• C has no built-in security mechanisms.	• Java has built-in security mechanisms to prevent malicious code from being executed.
Standard Library	• C has a limited standard library.	• Java has a large standard library that includes a wide range of functionality.

How JAVA Differ from C++

C++	Java
C++ is designed to work with compiler only	Java can support both compiler and interpreter
Platform dependent.	Platform independent.
C++ uses “cin” and “cout”.	Complex in/out methods (System.in and System.out)
Incorporates backward compatibility with C .	No backward compatibility support.
C++ is a combination of OOPs and Procedural type of programming.	Supports only Object-Oriented Programming style .
Memory management is manual, and the user is responsible for the memory consumed.	JVM manages memory without user intervention.
C++ can provide multiple inheritances.	Java cannot support multiple inheritances.
C++ supports both method overloading and operator overloading.	Java supports only method overloading.
Source code is not portable between different operating systems.	Source code is portable to any operating system.

JAVA Environment

Local Environment Setup

- Setting Up the Path for Windows
- Setting Up the Path for Linux, UNIX, Solaris, FreeBSD
- Popular Java Editors

SUMMARY

- How JAVA Differ from C and C++
- JAVA Environment

JAVA is used for……

A. System Programming

B. Application Programming

Overview of JAVA Language

- Why JAVA?
- Introduction to Java Technology
- The Java Platform
- Where Java is used?
- **Types of Java Applications**
- **History of Java**
- **Versions**
- **Features of JAVA**
- **Drawbacks of JAVA**

JAVA PROGRAM STRUCTURE

- **The requirement for Java Simple Example**
  - ➢Install the JDK
  - ➢Set path of the jdk/bin directory.
  - ➢Create the java program
  - ➢Compile and run the java program

JAVA PROGRAM STRUCTURE

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

To compile: javac Simple.java

To execute: java Simple

JAVA PROGRAM STRUCTURE

- **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.
  - **static** is a keyword
  - **void** is the return type of the method.
  - **main** represents the starting point of the program.
  - **String[] args** is used for command line argument.
  - **System.out.println()** is used to print statement.

PATH Setting in JAVA:

The **path** allows the system to locate and use the Java Development Kit (JDK) tools such as javac (the compiler), java (the interpreter), and others from any command prompt location.

Without setting the path - Navigate to the bin folder of the JDK every time to compile or run Java programs, which is inefficient.

The PATH environment variable shows the directories where your system looks for executable files like java, javac, etc.
java –version
javac -version

echo %PATH%

Java_path.bat

```
@echo off
set "JAVA_HOME=C:\Program Files\Java\jdk-21“
set "PATH=%JAVA_HOME%\bin;%PATH%“


echo Java PATH has been set temporarily.

java -version
```

Tips for Effective Path Management :

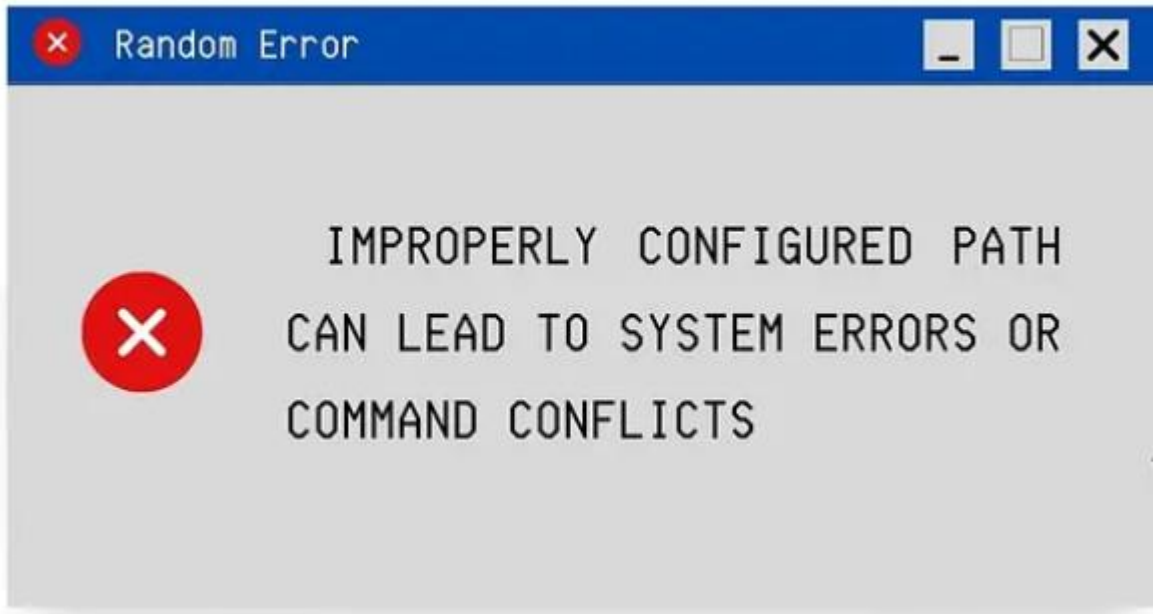
Keep Path Backup: Before altering your PATH, it’s wise to put into in a safe harbor—a text file or cloud storage.

Meaningful Additions Only: Adding only necessary directories reduces the risk of overflowing your PATH and encountering haunted system errors. It’s the quality, not the quantity, that counts.

Cohesive Grouping: Arrange associated paths together. This organizes your collection of paths so that they’re more readable and manageable, like a well-organized chart that guides you to your destination without unnecessary detours.

Leverage Environment Variables: Incorporate other environment variables within your PATH. For example, referring to %USERPROFILE% instead of C:UsersYourName makes your PATH dynamically adjust to different users and system configurations, like a compass that automatically adjusts to true north.

Regular Audits: Every so often, take a voyage through your PATH variables and clear out any that no longer serve a purpose, like removing barnacles from a ship's hull to improve its speed and handling.

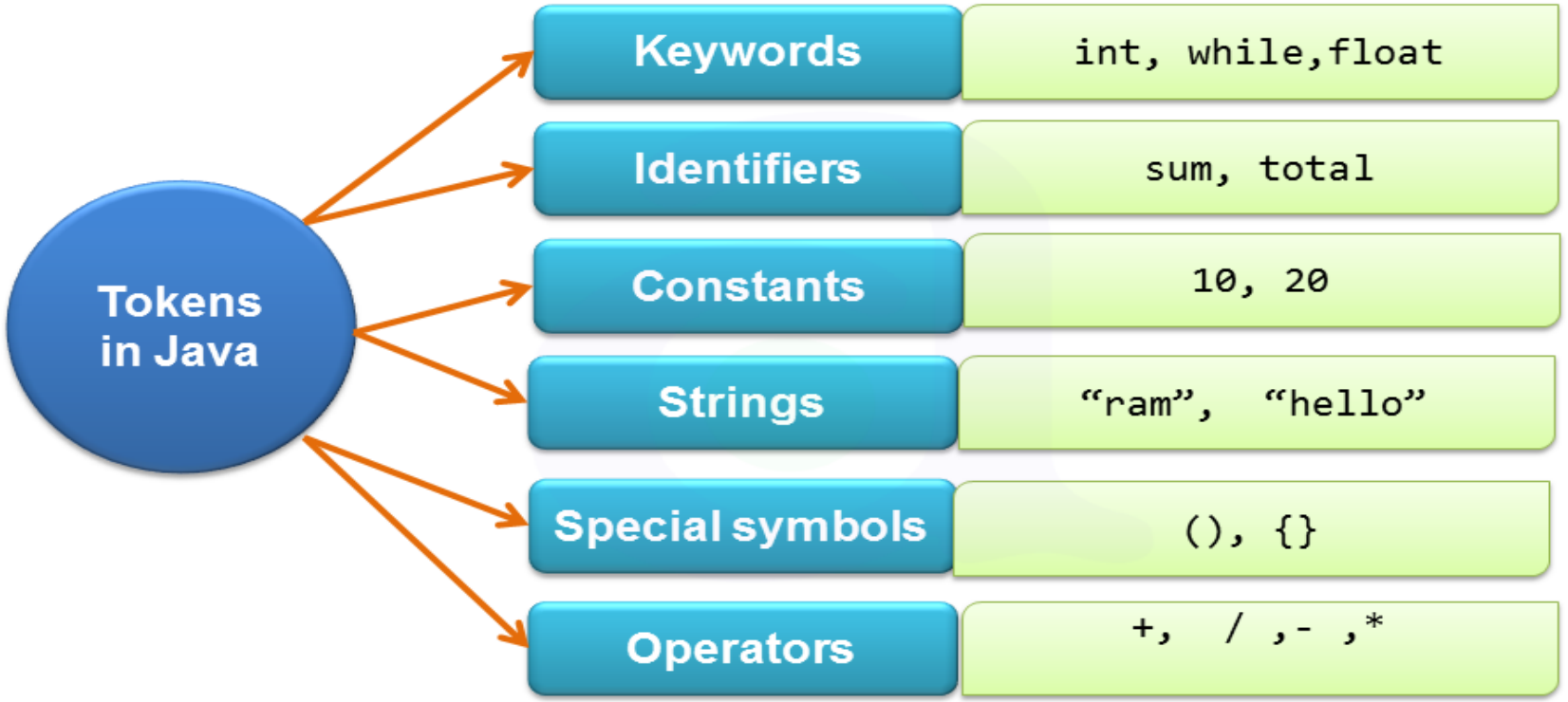

Random Error
IMPROPERLY CONFIGURED PATH
CAN LEAD TO SYSTEM ERRORS OR
COMMAND CONFLICTS

JAVA Tokens

JAVA Statements

JAVA Tokens

- The Smallest Individual units in a program are called “**Tokens**”



JAVA TOKENS: Keywords

- Keywords are pre-defined or reserved words in a programming language.
- Each keyword is meant to perform a specific function in a program.
- Keywords are referred names for a compiler.
- These can’t be used as variable names
- **Java** language supports 60 keywords.
- true, false, and null are not keywords. They are literals and reserved words that cannot be used as identifiers.

abstract
assert
boolean
break
byte
case
catch
char
class
continue
const - Not in use
default
do
double
else
enum
exports
extends
final
finally
float
for
Goto - Not in use
if
implements
import
instanceof
int
interface
long
module
native
new
package
private
protected
public
requires
return
short
static
strictfp
super
switch
synchronized
this
throw
throws
transient
try
var
void
volatile
while

JAVA TOKENS: Identifiers

- Identifiers are used as the **general terminology** for **naming** of **variables**, **functions** and **arrays**.
- Identifier names must **differ in spelling and case** from any keywords.
- **cannot** use **keywords** as **identifiers**.
- **valid identifiers:** myvariable, MYVariable, MYVARIABLE**.**
- **invalid identifiers:** my variable, M YVariable, MYV ARIABLE.

JAVA TOKENS: Constants/Literals

- Constants are also like **normal variables**.
- The values can **not be modified** by the program once they are defined.
- Constants refer to **fixed values**.
- They are also called as **literals**.
- Constants **may belong** to any of the **data type**.
- Syntax: **final data_type variable_name;**
- **Ex: final int PI=3.14;**

JAVA STATEMENTS

- **Statements** are **equivalent** to **sentences** in natural languages.
- A statement forms a **complete unit** of **execution**.
- The following types of expressions can be made into a statement by **terminating** the expression with a **semicolon** (;).
  - Assignment expressions
  - use of ++ or --
  - Method invocations
  - Object creation expressions
  - declaration statements
  - control flow statements

- JAVA Virtual Machine
- Command Line Arguments

JAVA Virtual Machine

- JVM (Java Virtual Machine) is an abstract machine.
- It is a specification that provides runtime environment in which java bytecode can be executed.

What is JVM

- A specification
- An implementation
- Runtime Instance

JAVA Virtual Machine

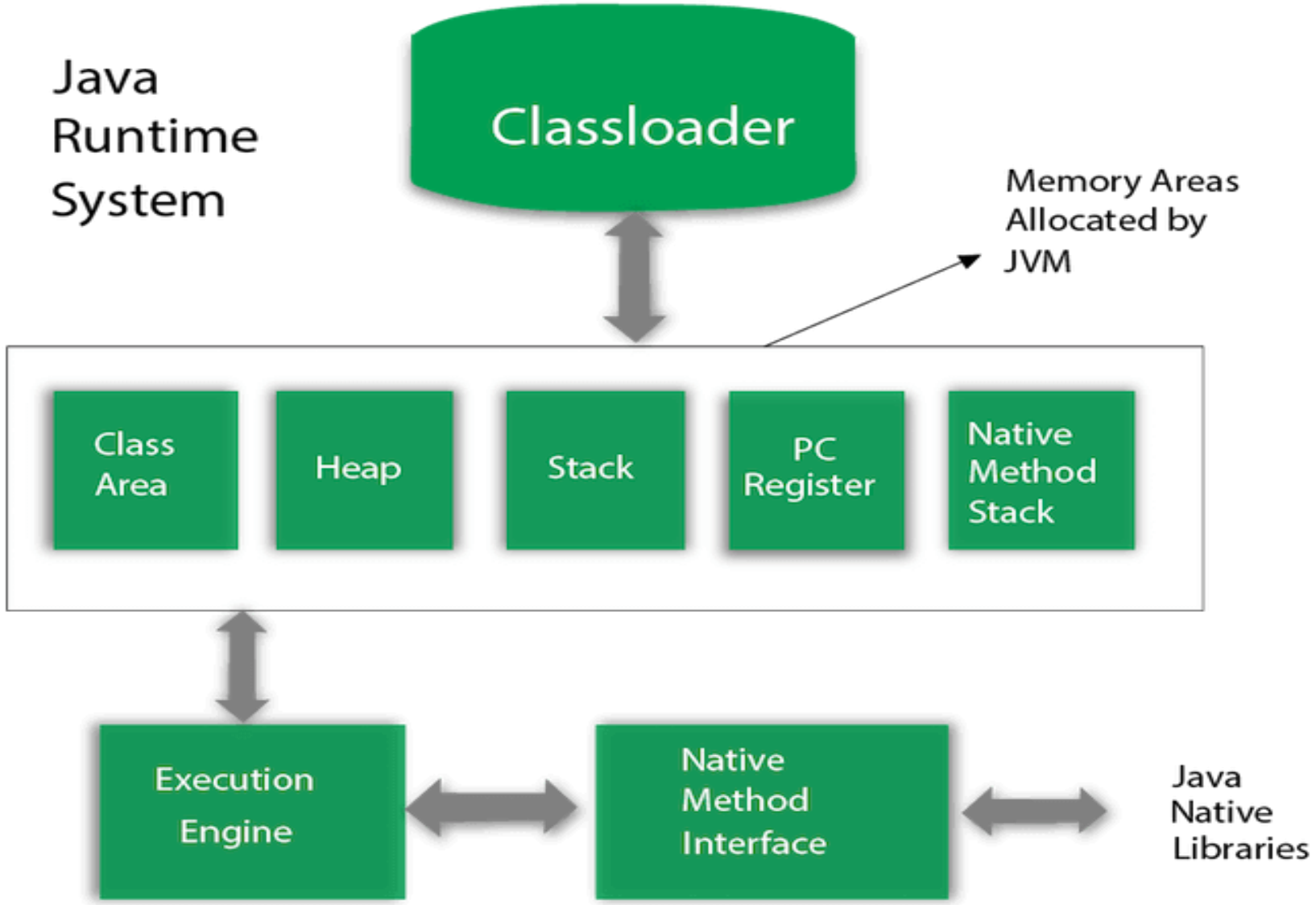
What it does?

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

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap

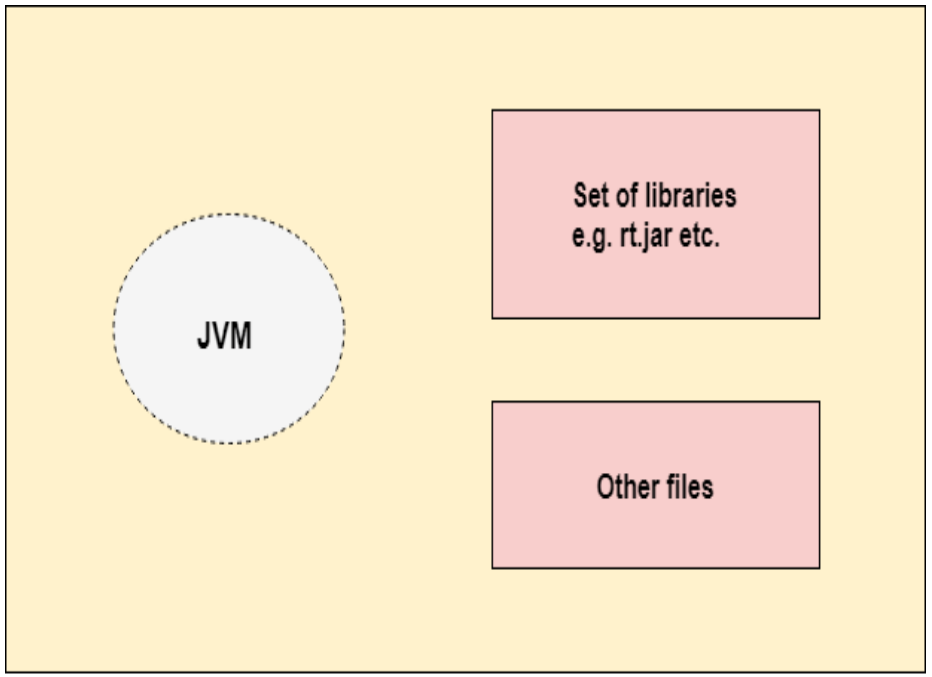
JVM Architecture



JVM Architecture

- **Classloader**
  - Extension ClassLoader
  - System/Application ClassLoader
- **Class(Method) Area**
- **Heap**
- **Stack**
- **Program Counter** Register
- **Native Method** Stack
- **Execution Engine**
  - A virtual processor
  - Interpreter
  - Just-In-Time(JIT) compiler
- Java Native Interface

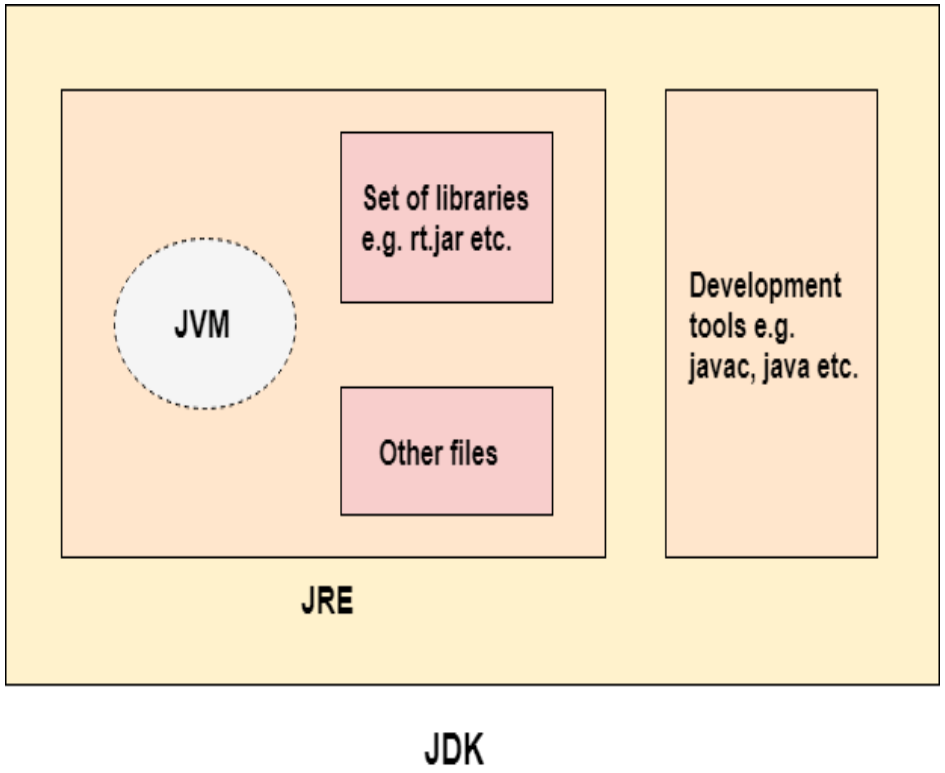
JRE



JRE

- The Java Runtime Environment is a set of software tools which are used to **run Java applications**.
- It is used to provide the **runtime environment.**
- It is the implementation of JVM.
- It contains a **set** of **libraries + other files** that JVM uses at runtime.

JDK



- The Java Development Kit (JDK) is a **software development environment** which is used to **develop Java** applications and applets.
- It contains **JRE + development** tools.
- JDK is an **implementation** of any one of the below given Java Platforms released by Oracle Corporation:
  - **Standard Edition** Java Platform
  - **Enterprise Edition** Java Platform
  - **Micro Edition** Java Platform

Java Command Line Arguments

- **command line arguments** is a way to pass inputs to the java program during application execution from the terminal or command prompt.
- These arguments are passed to the **main method** as an array of **String** values.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- it provides a convenient way to check the behavior of the program for the different values.
- Can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

- Java command-line arguments are very useful **to create parameterized java programs** which can accept the parameters **dynamically**.

  Users have **runtime control** over the **behavior** of the **program** as arguments can be passed to the **main() method**.

  To **configure** the **application** behavior by passing the arguments **before start** of the **application**.

  This mechanism enables **dynamic parameterization** of Java programs through console inputs, enhancing **versatility** and **interactivity**.

Java Command Line Arguments

```
class CommandLineExample
{
  public static void main(String args[])
    {
          System.out.println("Your first argument is: "+args[0]);
    }
}
```

Data Types in Java

- Data types specify the different sizes and values that can be stored in the variable.
- There are two types of data types in Java:
  - **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
  - **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

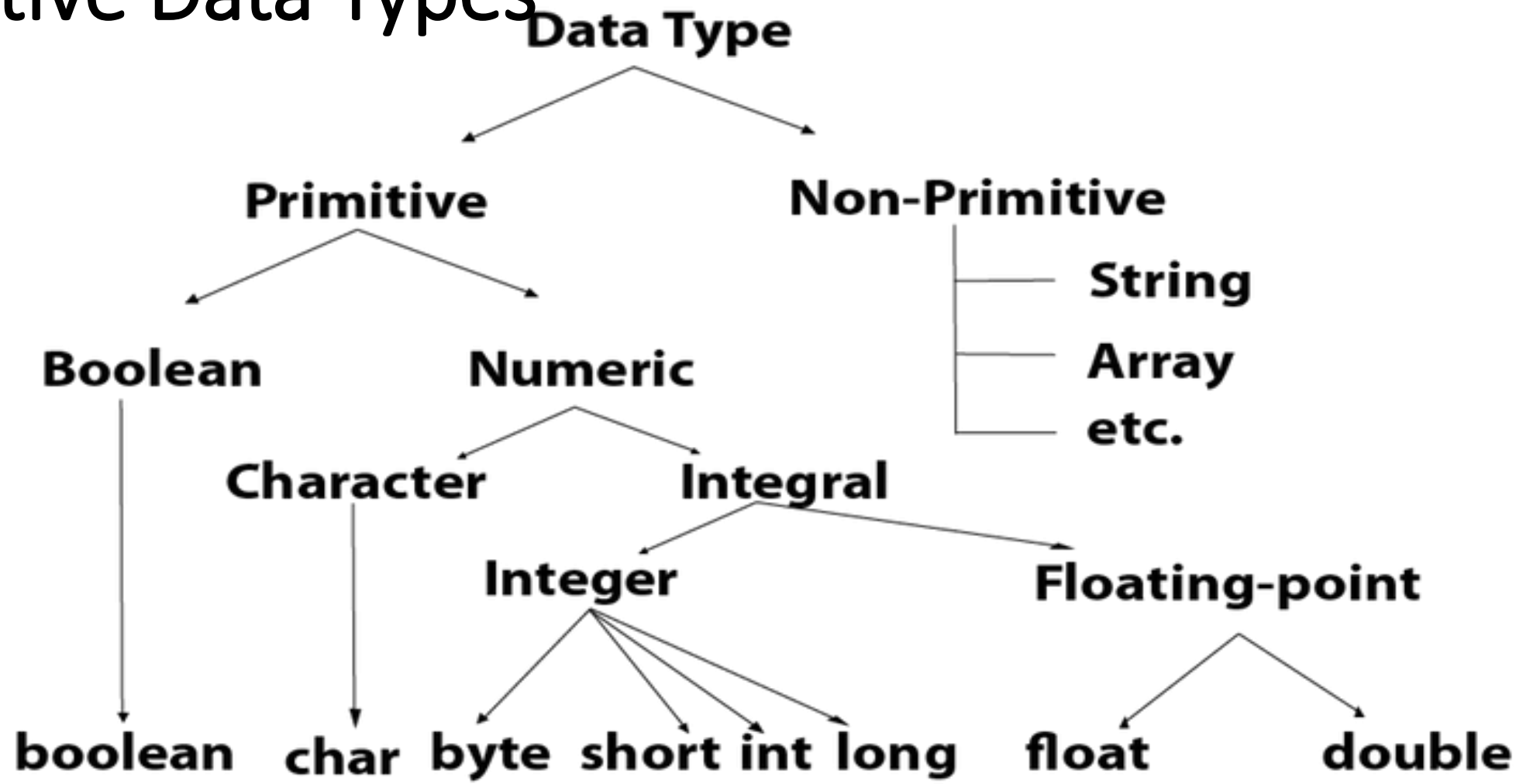
Aspect	Primitive Data Structures	Non-Primitive Data Structures
Definition	Basic types provided by a programming language as building blocks.	Complex data structures that are built using primitive **data types**.
Examples	int, float, char, double, boolean, byte, short, long.	Arrays, lists, **stacks**, **queues**, trees, graphs, sets, maps, classes
Memory Allocation	Allocated on the stack.	Typically allocated on the heap.
Size	Fixed and predefined by the language.	Dynamic and can vary during runtime.
Storage	Store single values.	It can store multiple and complex sets of data.

Operations	Limited to basic arithmetic and logical operations.	Support a wide range of operations specific to the data structure.
Default Values	Have default values (like 0 for int, false for boolean).	Usually, initialize to null or require explicit initialization.
Efficiency	Generally more efficient in terms of memory and performance.	It can be less efficient due to additional memory and overhead.
Direct Access	It can be accessed directly through their variable name.	It may require special methods or operations to access or manipulate.
Usage	Used for representing simple values and performing basic operations.	Used for organizing and managing data in more complex ways.

Primitive Data Types

- Primitive data types are the building blocks of data manipulation.
- There are 8 types of primitive data types:
  - boolean data type
  - byte data type
  - char data type
  - short data type
  - int data type
  - long data type
  - float data type
  - double data type

Primitive Data Types



Primitive Data Types

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

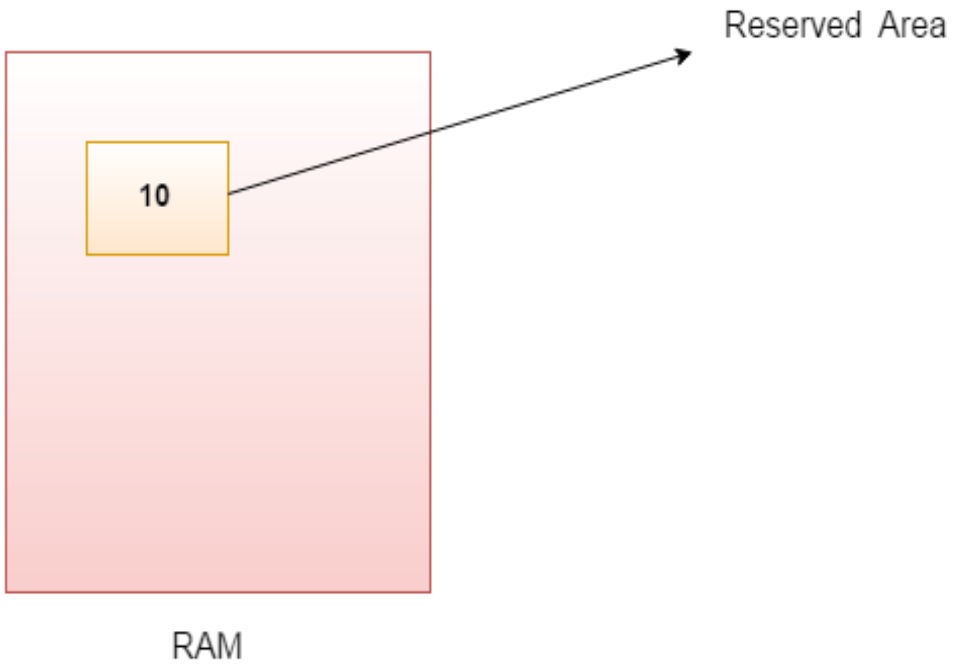
JAVA VARIABLES

- 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, static.

VARIABLE

- **Variable** is name of *reserved area allocated in memory*.
- It is a combination of "vary + able" that means its value can be changed.



TYPES OF VARIABLES

- There are three types of variables in Java

  - **local variable:** The variables declared inside the body of the method are termed local variables.
  - **instance variable:** Instance variables are **fields declared within a class but outside any method**. They are used to store unique data for each instance of the class.
  - **static variable:** static variables can be used to refer to the common property of all objects (which is not unique for each object).

Ex: college name of students, company name of employees, CEO of a company, etc.

It makes the program memory efficient (i.e., it saves memory).

CONSTANTS IN JAVA

- A constant is a variable whose value **cannot change once it has been assigned**.
- Java doesn't have built-in support for constants.
- To define a variable as a constant, we just need to add the keyword "**final**" in front of the variable declaration.
- Syntax: **final float PI= 3.14f;**

- **QUIZ FOR THE CLASS**

1. What is the range of short data type in Java?

A. -128 to 127

B. -32768 to 32767

C. -2147483648 to 2147483647

D. None

2. Which data type value is returned by all transcendental math functions?

A. int

B. float

C. double

D. long

Topics to be learned from this class

- Scope of variables
- Type Casting

• Scope of Variables

Scope of a variable is the part of the program where the variable is accessible.

- Java programs are organized in the form of classes.

Member Variables (Class Level Scope)

- These variables must be declared inside class (outside any function).
- They can be directly accessed anywhere in class.

Example

```
public class Test
{
    // All variables defined directly
inside a class
    // are member variables
    int a;
    private String b
    void method1() {....}
    int method2() {....}
    char c;
}
```

Scope of Variables

- We can declare class variables anywhere in class, but outside methods.
- Access specified of member variables doesn't effect scope of them within a class.
- Member variables can be accessed outside a class with following rules

Modifier	Package	Subclass	World
public	Yes	Yes	Yes
protected	Yes	Yes	No
Default (no modifier)	Yes	No	No
private	No	No	No

Scope of Variables

Local Variables (Method Level Scope)

- Variables declared inside a method have method level scope and can't be accessed outside the method.
- Local variables don't exist after method's execution is over.

Loop Variables (Block Scope)

- A variable declared inside pair of brackets "{" and "}" in a method has scope within the brackets only.

```
Example
public class Test
{
 void method1()
 {
// Local variable (Method
level scope)
 int x;
 }
 }
```

Type Casting

- Casting is a process of changing one type value to another type.

Type conversion in Java

- Widening or Automatic Type Conversion
- Narrowing or Explicit Conversion

Type casting

Widening or Automatic Type Conversion

- Widening conversion takes place when two data types are automatically converted.
- This happens when:
  - The two data types are compatible.
  - When we assign value of a smaller data type to a bigger data type.

Byte –> Short –> Int –> Long – > Float –> Double

Widening or Automatic Conversion

Example

```
class AutomaticTypeConversion
{
   public static void main(String[] args)
   {
      int i = 100;
            // automatic type conversion
      long l = i;
           // automatic type conversion
      float f = l;
      System.out.println("Int value "+i);
      System.out.println("Long value "+l);
      System.out.println("Float value "+f);
   }
}
```

Type casting

- **Narrowing or Explicit Conversion**
  - Assign a value of larger data type to a smaller data type then perform explicit type casting or narrowing.
    - This is useful for incompatible data types where automatic conversion cannot be done.
    - Target-type specifies the desired type to convert the specified value to.

Double –> Float –> Long –> Int –> Short –> Byte

Narrowing or Explicit Conversion

```
class Test
{
   public static void main(String[] args)
   {
      double d = 100.04;

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

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

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

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

summary

- Scope of variables
- Type Casting

- **QUIZ FOR THE CLASS**

1. Which of this method is given parameter via command line arguments?

A. main()

B. recursive() method

C. Any method

D. System defined methods

2. Which of these data types is used to store command line arguments?

A. Array

B. Stack

C. String

D. Integer

LESSON 8: Topics discussed in the previous class

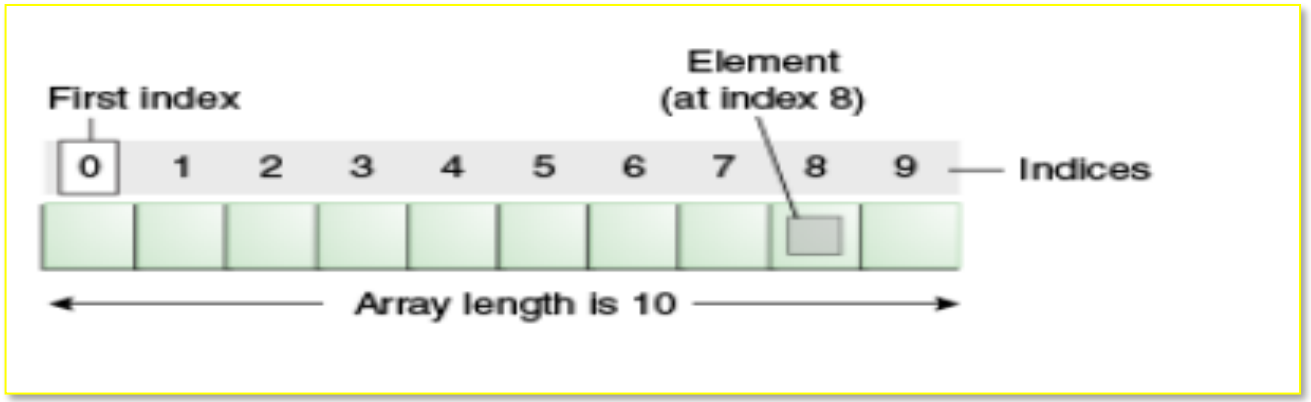
- Scope of variables
- Type Casting

Topics to be learned from this class

- Java Arrays
- One Dimensional Arrays

Arrays

- Array is a collection of similar type of elements
- Array is an object in java which contains elements of a similar data type.
- The elements of an array are stored in a contiguous memory location.
- We can store only a fixed set of elements in an array.
- Array is index-based



Arrays

- It is a data structure where we store similar elements.
- Unlike C/C++, we can get the length of the array using the length member.
- In Java, array is an object of a dynamically generated class.
- Java array inherits the Object class
- Java array implements the Serializable as well as Cloneable interfaces.
- We can store primitive values or objects in an array.
- We can create single dimensional or multi dimensional arrays.

Pros and cons

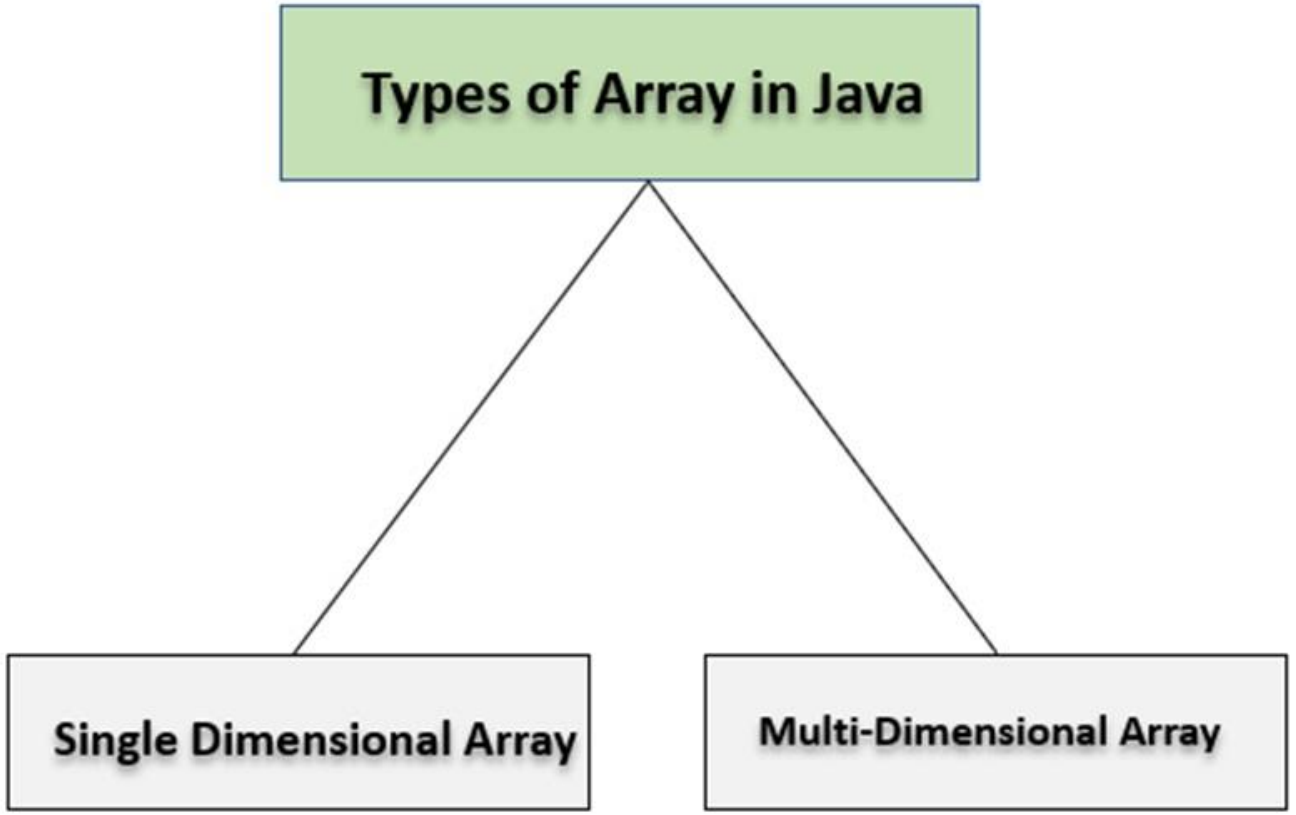
Pros

- **Code Optimization**
  - **We can retrieve or sort the data efficiently.**
- **Random access**
  - **We can get any data located at an index position.**

Cons

- **Size Limit**
  - **We can store only the fixed size of elements in the array.**
  - **It doesn't grow its size at runtime.**

Types of arrays



One Dimensional Array

- **One dimensional array will have only one index point to store values**
- **If the data is linear, we can use the One Dimensional Array.**
- **Syntax:**

```
dataType[] arr;
    (or)
dataType arr[];
```

- **Instantiation:**

```
arrayRefVar=new datatype[size];
```

One Dimensional Array

```
class TestArray{
        public static void main(String args[]){
                int a[]=new int[5];
                a[0]=10;
                a[1]=20;
                a[2]=70;
                a[3]=40;
                for(int i=0;i<a.length;i++)
                System.out.println(a[i]);
        }
}
```

One Dimensional Array

```
class TestArray1{
        public static void main(String args[]){
                int a[]={80,30,40,50};
                for(int i=0;i<a.length;i++)
                System.out.println(a[i]);
        }
}
```

Array using For-each Loop

- We can also print the Java array using for-each loop.
- The Java for-each loop prints the array elements one by one.
- It holds an array element in a variable, then executes the body of the loop.
- Syntax:

```
for(data type variable : array){
        //body of the loop
}
```

- Example

```
int a[]={33,3,4,5};
for(int x:a){
System.out.println(x);
}
```

Passing Array to a Method

```
class PassArray_example{
static void min(int arr[]){
int min=arr[0];
for(int i=1;i<arr.length;i++)
{
if(min>arr[i])
min=arr[i];
}
System.out.println(min);
}
public static void main(String args[])
{int a[]={8,2,4,5,0};
min(a); }
}
```

Array PRACTICE Programs

- **Write a java program to sort list of elements in ascending and descending order and show the exception handling.**
- **Write a java program to find smallest number in an array.**
- **Write a java program to find the element in an array using linear search.**
- **Write a java program to print the elements of an array present at even position only.**

Two Dimensional Arrays

- The data stored in rows and columns.
- We can access the value using row and column index.
- Multi-Dimensional Array is used to store multi-level data.
- Two Dimensional Array is the simplest form of Multi-Dimensional Array.
- Java uses zero-based indexing.
- The indexing of arrays will starts with 0.
- Syntax:

```
a[row_index][column_index]
```

- Example:

```
int[][] a = new int[3][4];
```

Two Dimensional Arrays

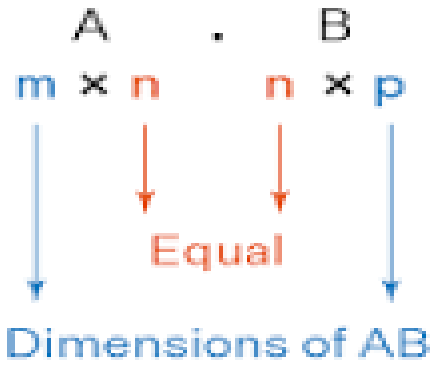
```
class TwoDimArray{
        public static void main(String args[]){
            int a[][]={{1,2,3},{2,4,5},{4,4,5}};
            for(int i=0;i<3;i++){
                    for(int j=0;j<3;j++){
                     System.out.print(a[i][j]+" ");
                }
                System.out.println();
            }
        }
}
```

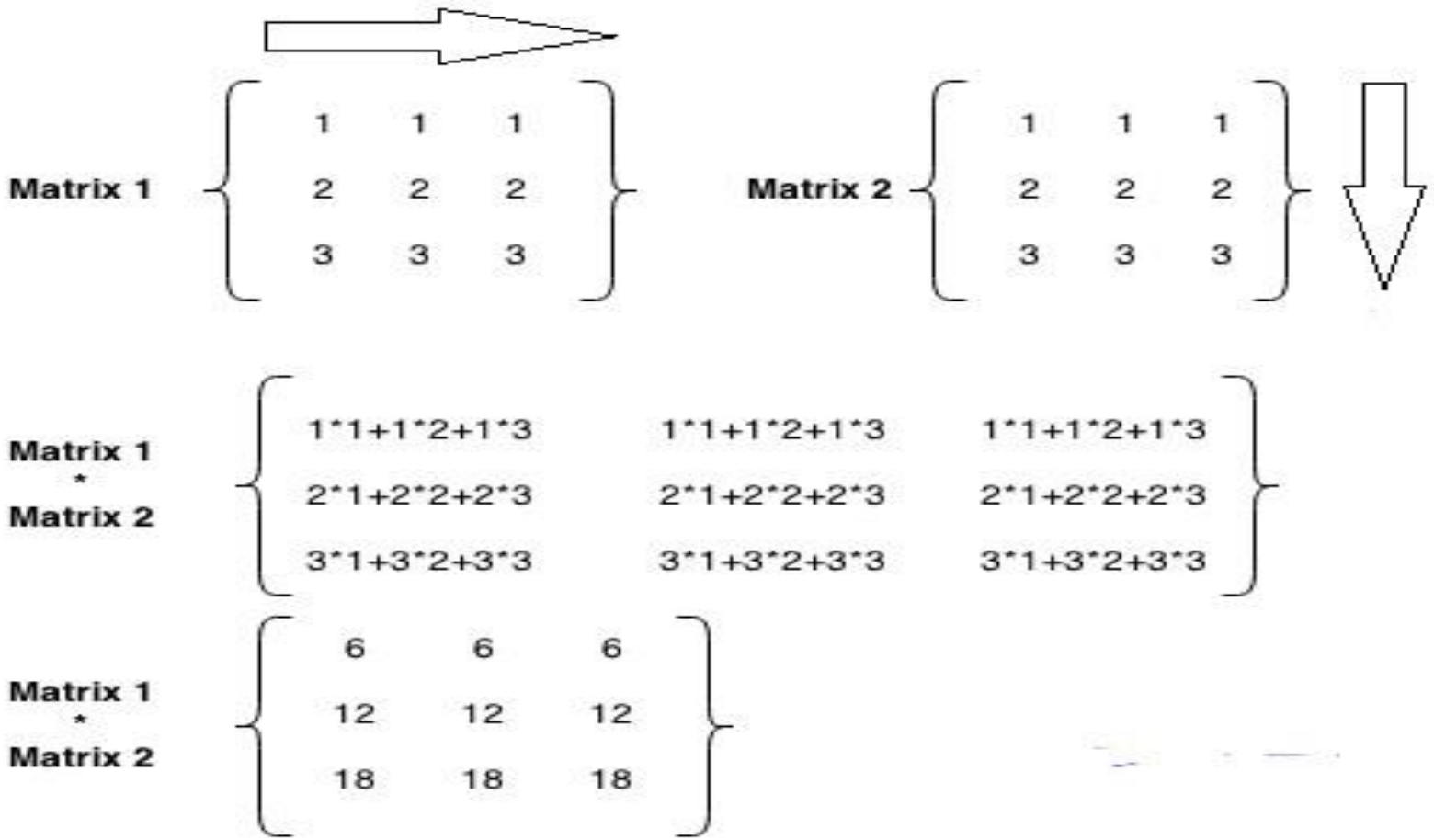
	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

2-dimensional Array

Multiply two matrices: The number of columns in the first matrix must be equal to the number of rows in the second matrix.

Multiplication of Matrices





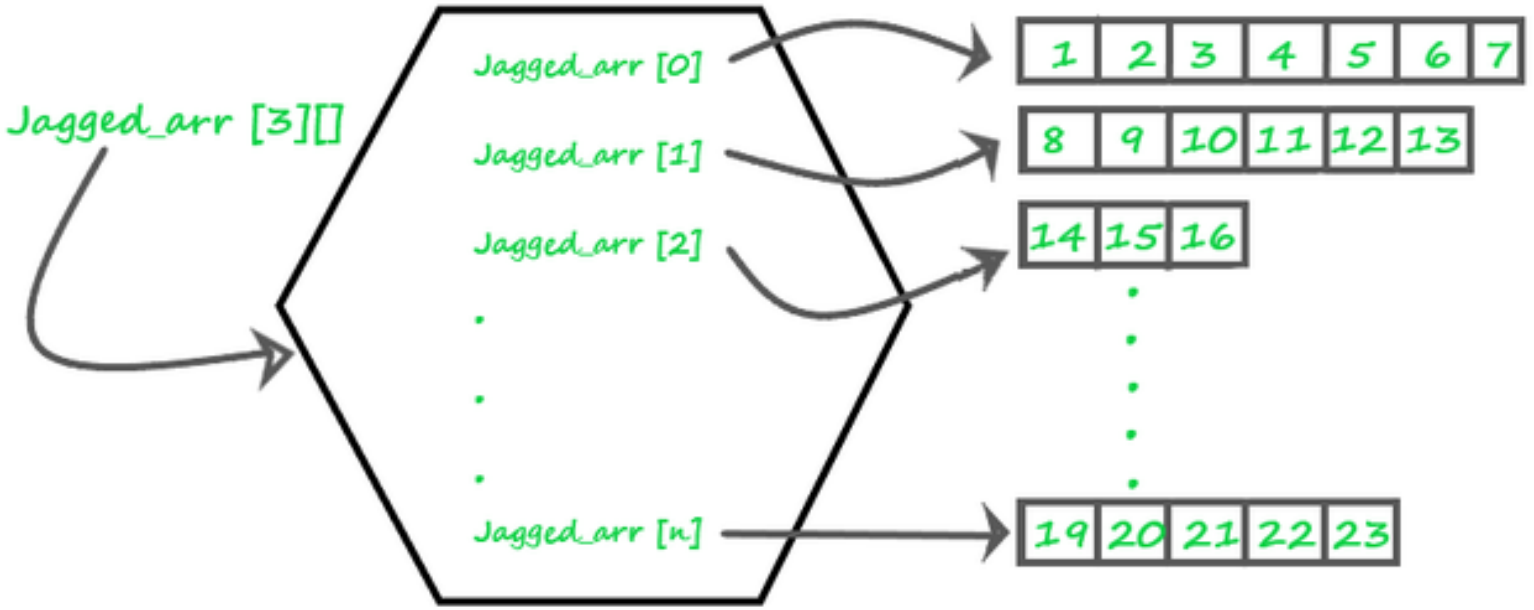
Jagged array

- A jagged Array is an array of arrays where each element is an array.
- It is a special type of Multidimensional array where there are a variable number of columns in each row.
- An array with different number of columns.

```
class Array2D {
        public static void main(String[] args) {
                int[][] a = { {1, 2, 3}, {4, 5, 6, 9}, {7} };
                System.out.println("Length of row 1: " + a[0].length);
                System.out.println("Length of row 2: " + a[1].length);
                System.out.println("Length of row 3: " + a[2].length);
        }
}
```

Jagged array applications

- Allows for flexibility in storing data when the number of elements in each sub-array is different.
- This is particularly useful in scenarios where the number of columns may vary, such as when dealing with irregular data or sparse matrices.
- Pictorial representation in Heap memory:



Advantages of Jagged Array

- Memory Efficiency
- Flexibility
- Easy to Initialize
- Enhanced Performance
- More natural representation of data
- Easier to manipulate

summary

- Two dimensional array
- Matrix Addition
- Matrix Multiplication
- Jagged Arrays

Quiz for this class

1. Java uses _____based indexing?

   A. 0

   B. 1

   C. 2

   D. 3

0

2. _____Array is used to store multi-level data?

   A. Two Dimensional

   B. One Dimensional

   C. Three Dimensional

   D. Multi-Dimensional

Multi-Dimensional

summary

- Arrays in java
- One dimensional array
- Passing array to a method
- Example Programs

Topics to be learned from this class

- Operators in JAVA

Operators in Java

- **Operator** in Java is a symbol which is used to perform operations.

  For example: +, -, *, / etc.

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

- Unary Operator
- Arithmetic Operator
- Shift Operator
- Relational Operator
- **Bitwise Operator**
- **Logical Operator**
- **Ternary Operator**
- **Assignment Operator**

Java Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	expr++ expr--
	prefix	++expr --expr +expr -expr ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >=
	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.:
  - incrementing/decrementing a value by one
  - negating an expression
  - inverting the value of a boolean

```
class OperatorExample{
public static void main(String args[]
){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

Java Arithmetic Operators

- Java arithmetic operators are used to perform addition, subtraction, multiplication, and division.
- Basic mathematical operations.

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//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.

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=24
0
}}
```

Java Right Shift Operator

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

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

>>> operator (Unsigned right shift operator)

- **Shifts bits to the right** just like >>
- **Fills the leftmost bits with zeros**, regardless of whether the number is positive or negative.
- **Used only for integer and long types**.

- int x = -8; // binary: 11111111 11111111 11111111 11111000

- System.out.println(x >> 2);   // Output: -2
- System.out.println(x >>> 2); // Output: 1073741822

Java Shift Operator Example: >> vs >>>

```
class OperatorExample{
public static void main(String args[]){
   //For positive number, >> and >>> works same
   System.out.println(20>>2);
   System.out.println(20>>>2);
   //For negative number, >>> changes parity bit (MSB) to 0
   System.out.println(-20>>2);
   System.out.println(-20>>>2);
}}
```

- **QUIZ FOR THE CLASS**

1. Which of the following can be operands of arithmetic operators?

A. Numeric

B. Boolean

C. Characters

D. Both Numeric & Characters

2. Can 8 byte long data type be automatically type cast to 4 byte float data type?

A. True

B. False

Logical && and Bitwise &

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

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);//false && tru
e = false
System.out.println(a<b&a<c);//false & true
= false
}}
```

Logical || and Bitwise |

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

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

Java Ternary Operator

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

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Java Assignment Operator

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

```
class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}
}
```

Expressions

- Operators may be used in building expressions, which compute values.
- Expressions are the core components of statements.
- Statements may be grouped into blocks.
- Regular **expression** is a sequence of pattern that defines a string.
- It is used to denote regular languages.
- It is also used to match character combinations in strings.
- In regular **expression**, x* means zero or more occurrence of x.
- In regular **expression**, x+ means one or more occurrence of x.

Expressions

- An *expression* is a construct made up of **variables, operators, and method invocations,** which are constructed according to the syntax of the language, that evaluates to a single value.
- The **data type of the value returned** by an expression depends on the elements used in the expression.
- The expression a = 0 returns an int because the assignment operator returns a value of the same data type as its left-hand operand.

Arithmetic Expressions

- An assignment statement or expression changes the value that is held in a variable

```
class example
{
      public static void main ( String[] args)
{
  long x ; //a declaration without an initial value
  x = 123; //an assignment statement
  System.out.println("The variable x contains: " + x );
}
}
```

Arithmetic Expressions

- Java **Arithmetic expressions** use arithmetic operators such as +, -, /, *, and %.
- The % operator is the remainder or modulo operator.
- Arithmetic expressions are used to assign arithmetic values to variables.
- An expression is a combination of literals, operators, variables, and parentheses used to calculate a value.

```
int x, y, z; x = 10;
y = 12;
z = y / x;
z = x + y;
x+y                    z
= y % x
```

Java operator associativity

Category	Operators	Associativity
Postfix	++ - -	Left to right
Unary	+ - ! ~ ++ - -	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left

Topics to be learned from this class

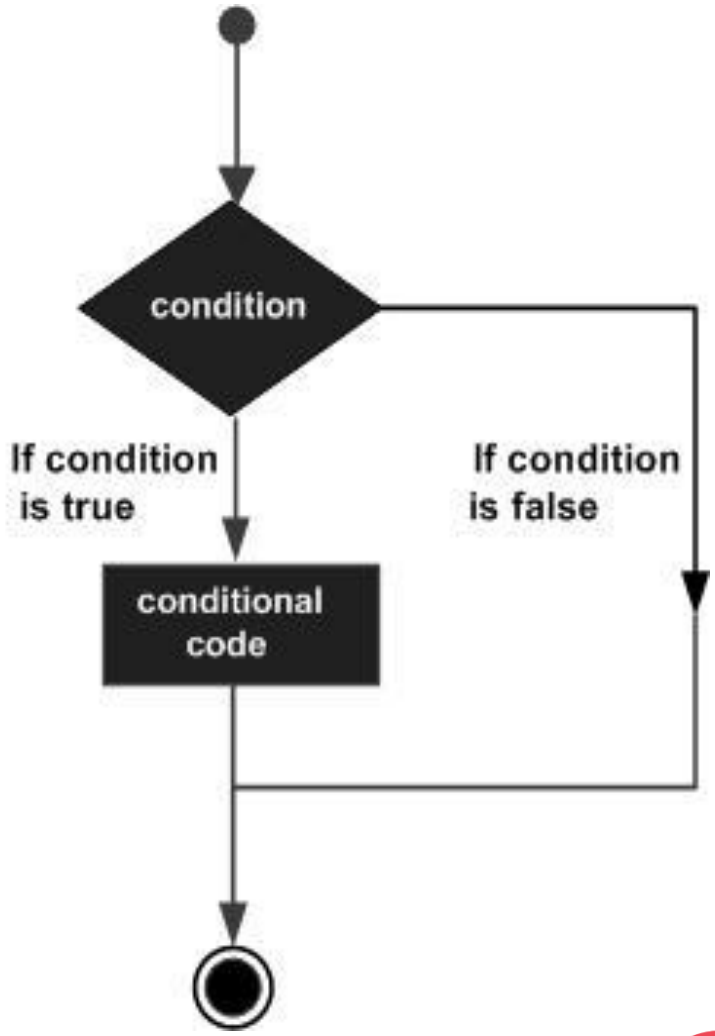
- Decision Making
- Branching

Control Flow Statements

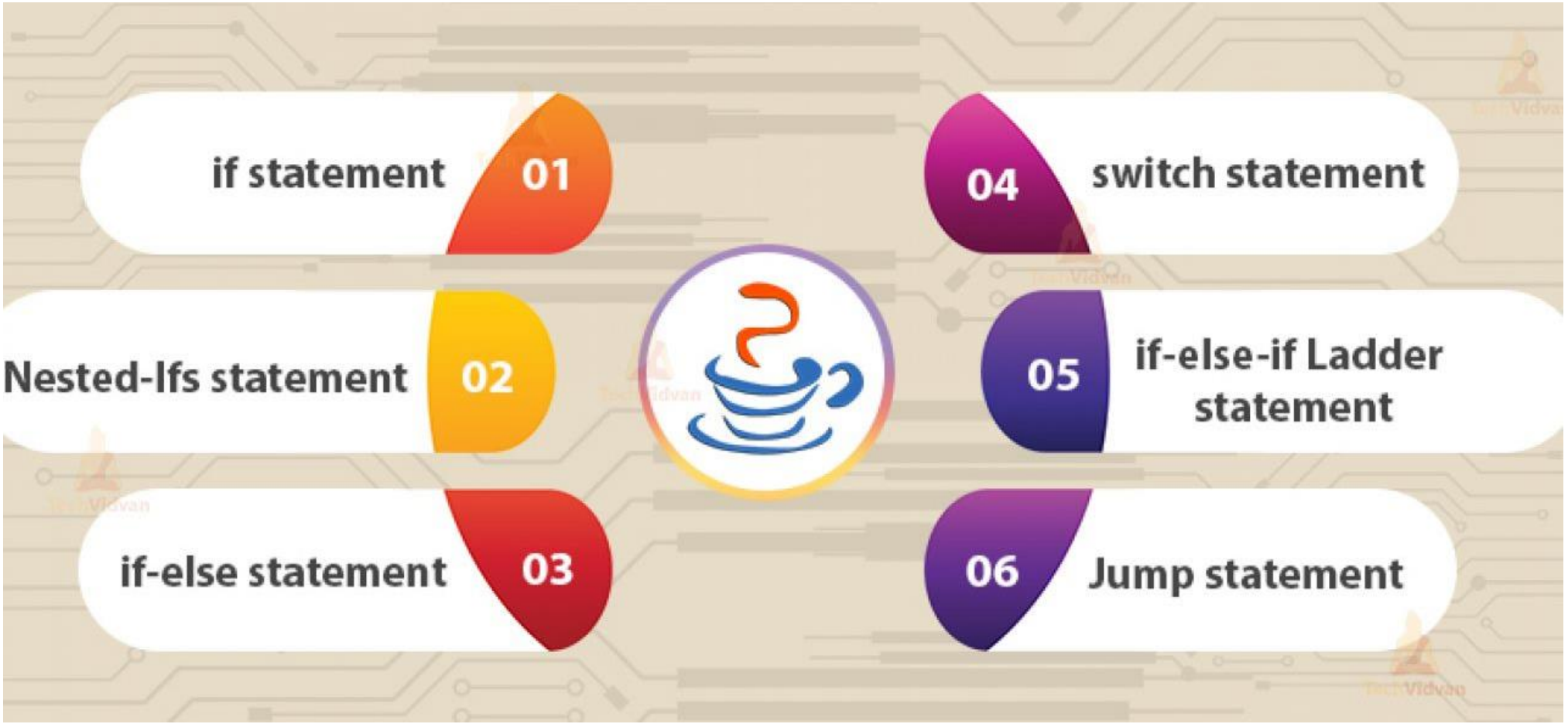
- ***Control flow statements: It*** break up the flow of execution by employing decision making, looping, and branching.
- Enabling the program to *conditionally* execute particular blocks of code.
- The decision-making statements (if-then, if-then-else, switch), the looping statements (for, while, do-while), and the branching statements (break, continue, return) supported by the Java programming language.

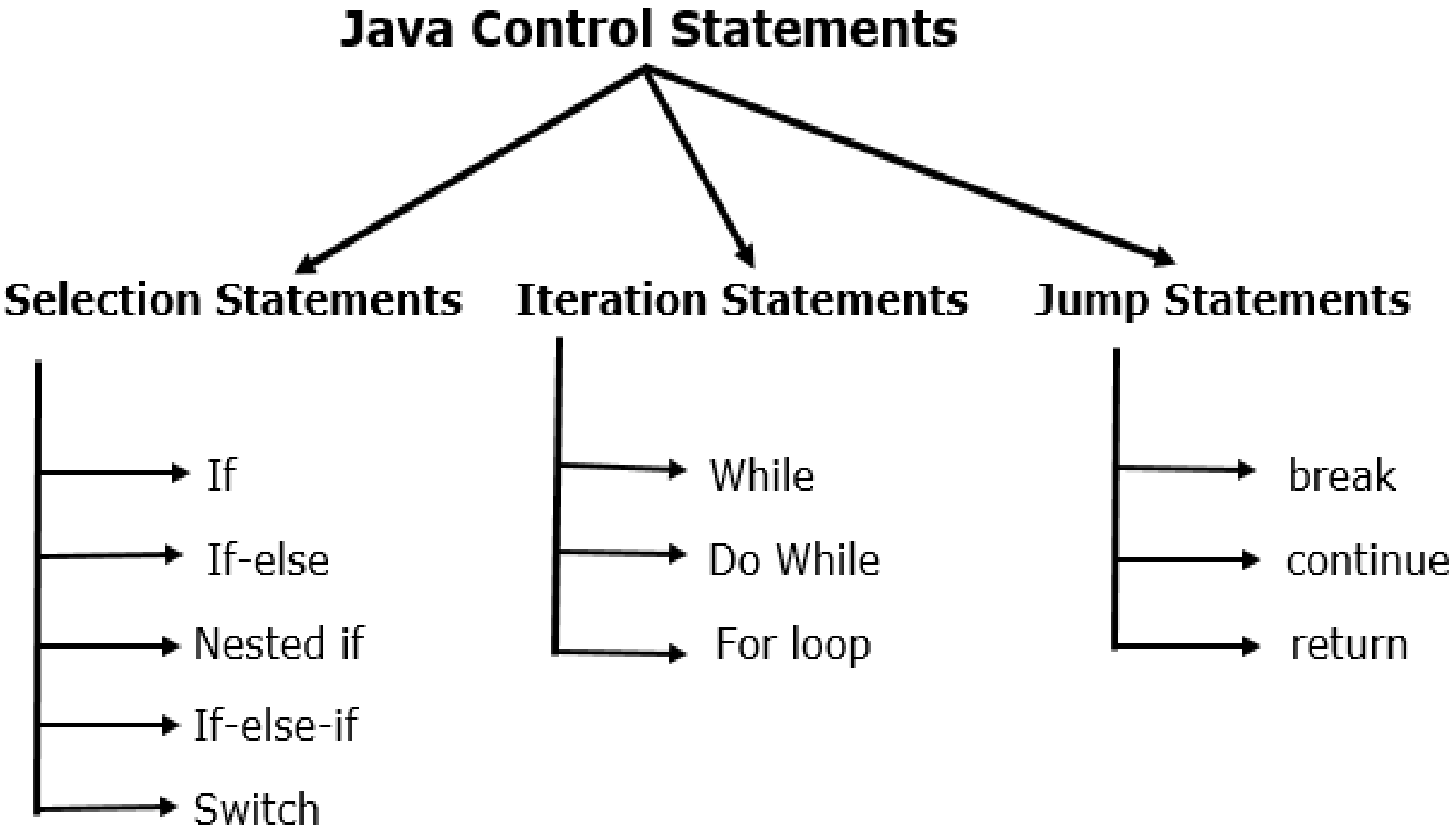
Decision Making

- Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be **true**.
- Other statements to be executed if the condition is determined to be **false**.



Decision Making in JAVA



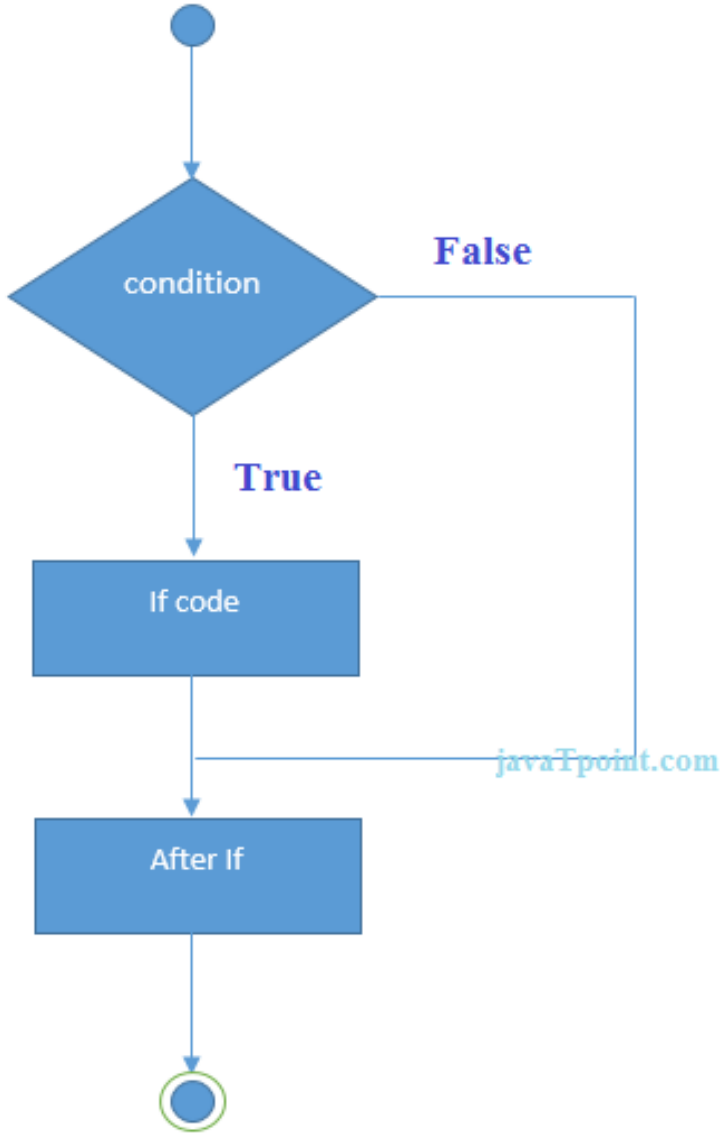

Java Control Statements
Selection Statements
Iteration Statements
Jump Statements
If
If-else
Nested if
If-else-if
Switch
While
Do While
For loop
break
continue
return

Java if Statement

- The Java *if statement* is used to test the condition. It checks boolean condition: *tru* or *false*. There are various types of if statement in Java.
  - if statement
  - if-else statement
  - if-else-if ladder
  - nested if statement
- The Java if statement tests the condition. executes the *if block* if condition is true.

Syntax:

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

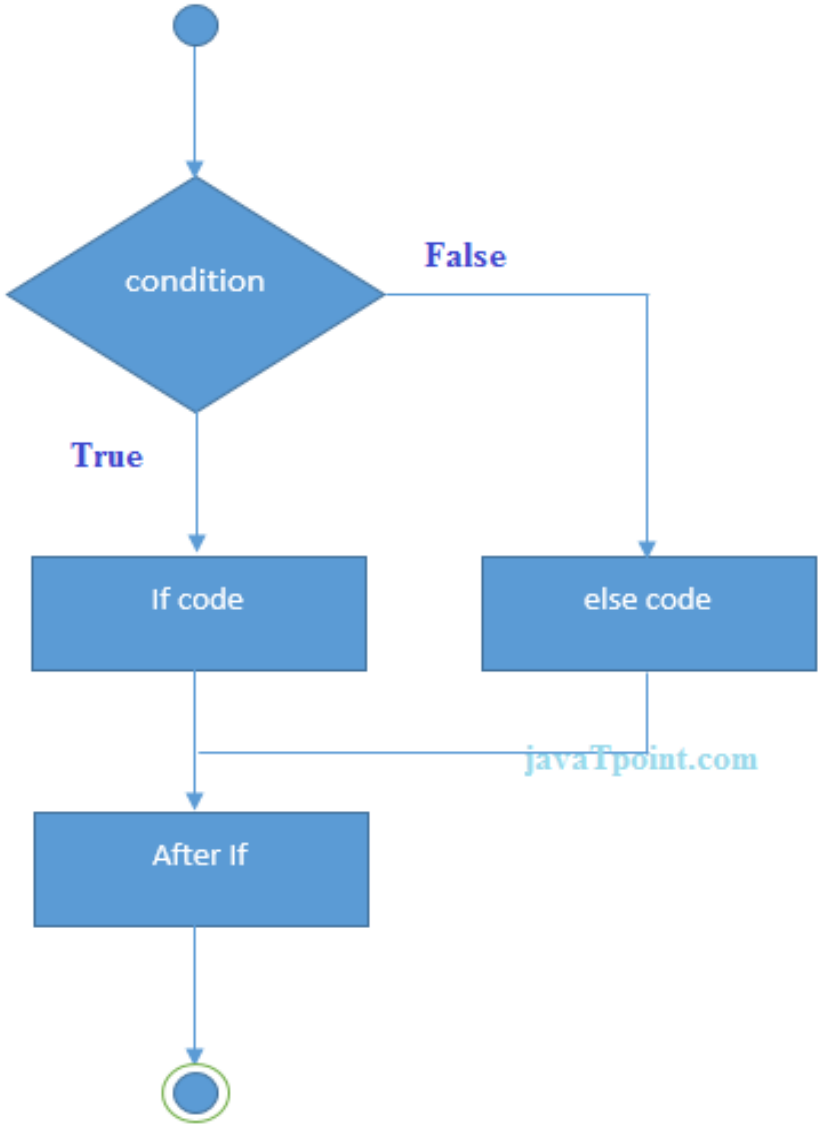


Java if-else Statement

- 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
}
```



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.

```
public class IfElseTernaryExample {

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

Java if-else-if ladder Statement

- The if-else-if ladder statement executes one condition from multiple statements.

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 f
alse
}
```

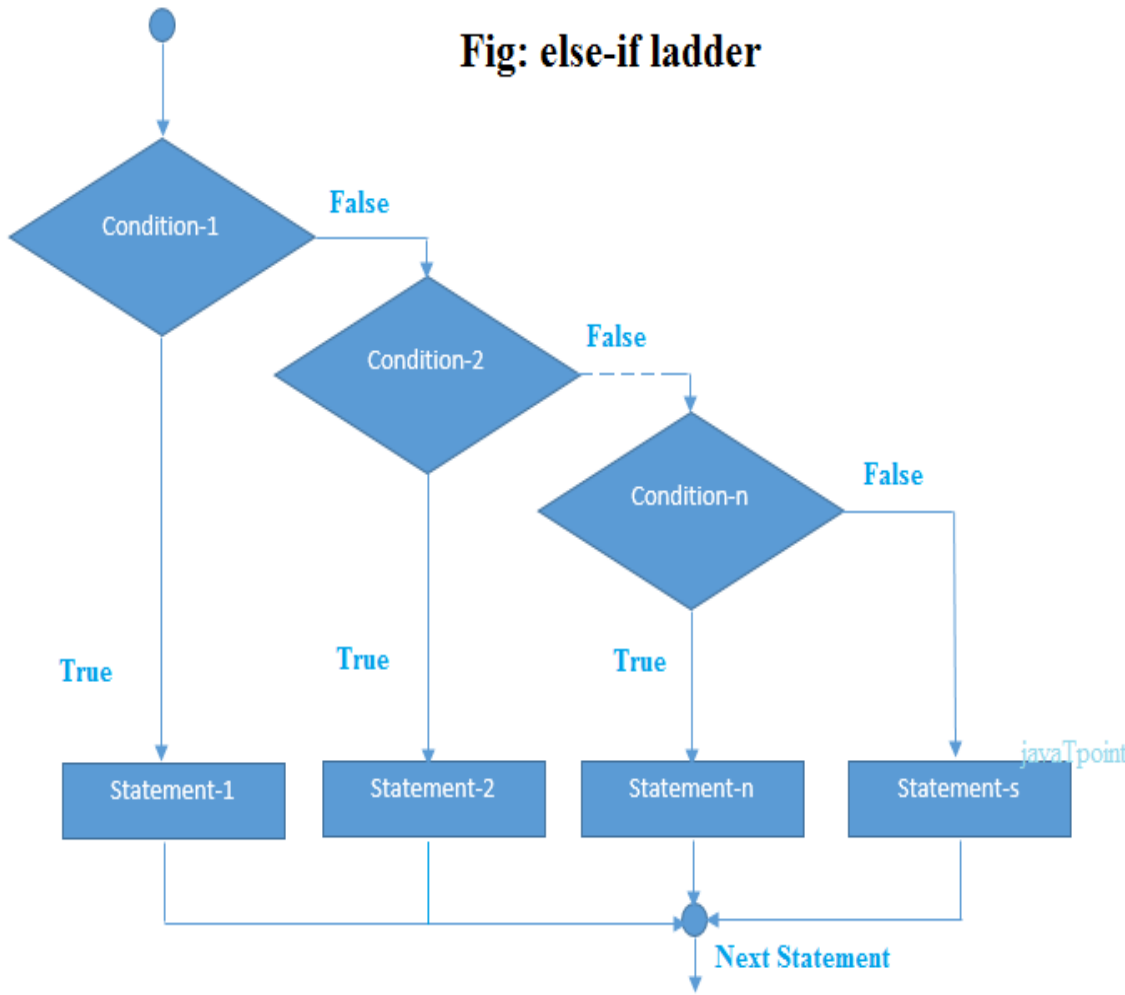


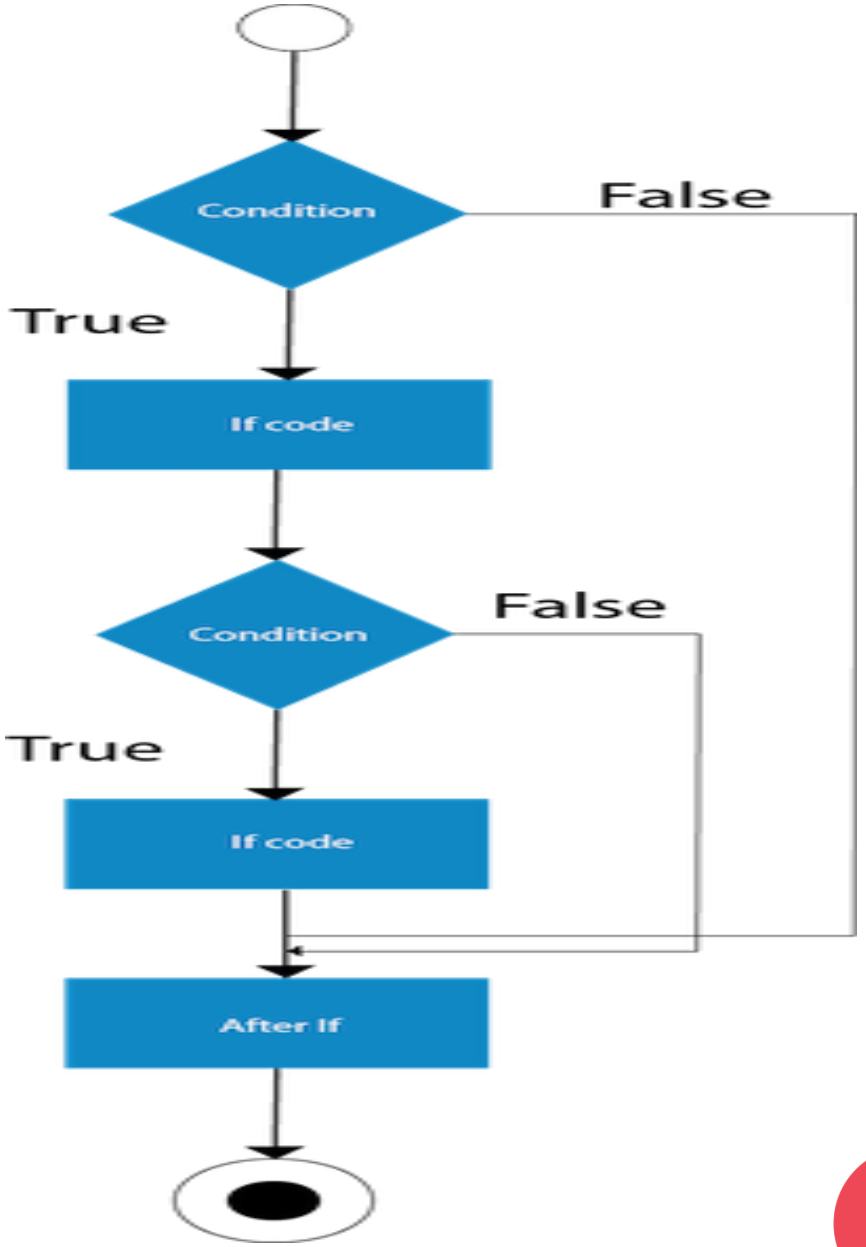
Fig: else-if ladder

Java Nested if statement

- The nested if statement represents the *if block within another if block*.
- The inner if block condition executes only when outer if block condition is true.

Syntax:

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



Java 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.

Java Switch Statement

- The switch statement tests the equality of a variable against multiple values.
  - 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*****.**
  - 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.

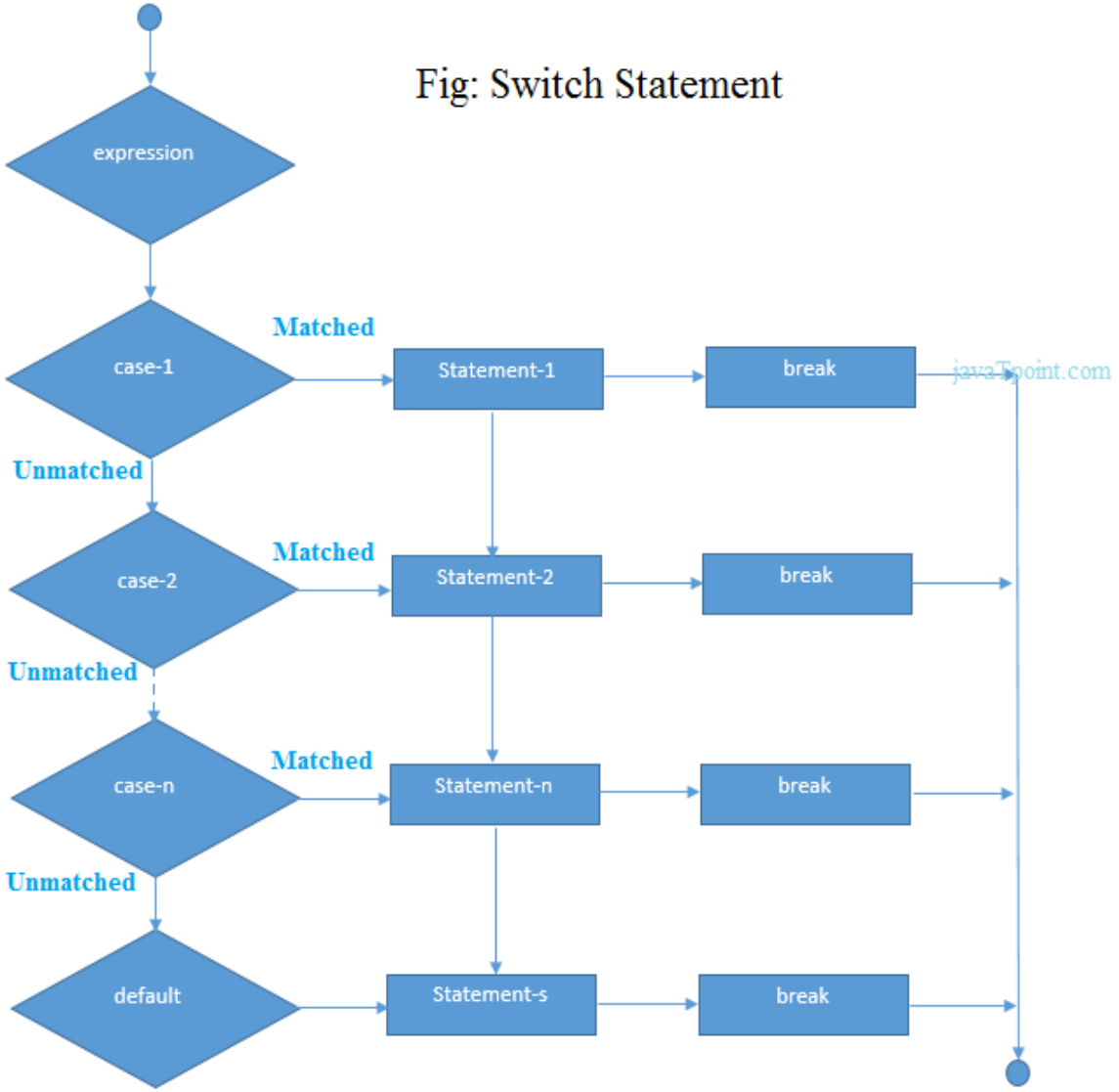
Java Switch Statement

Syntax:

```
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;

}
```

Fig: Switch Statement



- **QUIZ FOR THE CLASS**

1. The while loop repeats a set of code while the condition is not met?

A. True B. **False**

2. What is true about a break?

A. Break stops the execution of entire program

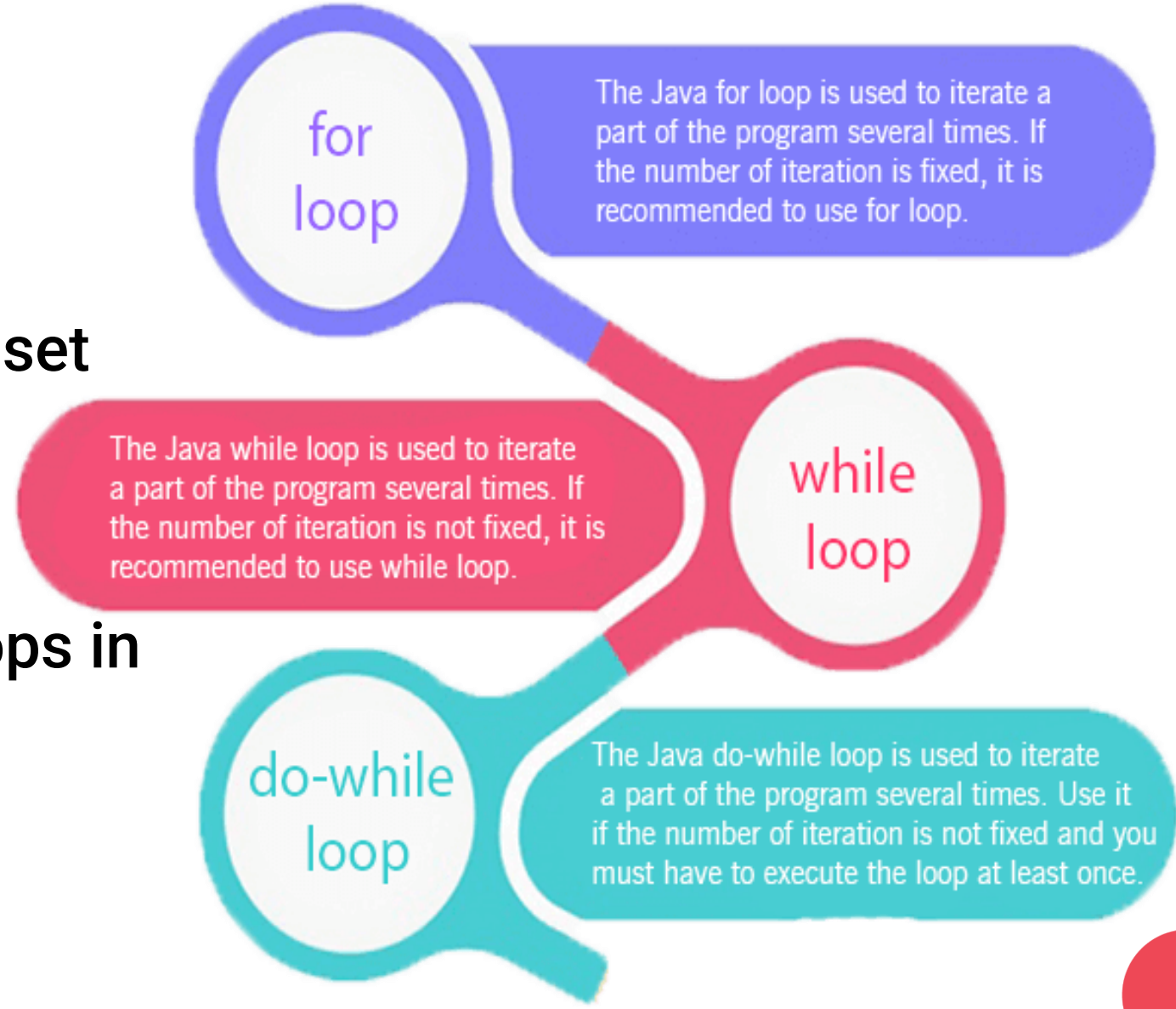
B. Break halts the execution and forces the control out of the loop

C. Break forces the control out of the loop and starts the execution of next iteration

D. Break halts the execution of the loop for certain time frame

Loops in Java

- In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true.
- There are three types of loops in Java.
  - for loop
  - while loop
  - do-while loop



Loops in Java

Java For Loop vs While Loop vs Do While Loop

Java 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

Loops in Java

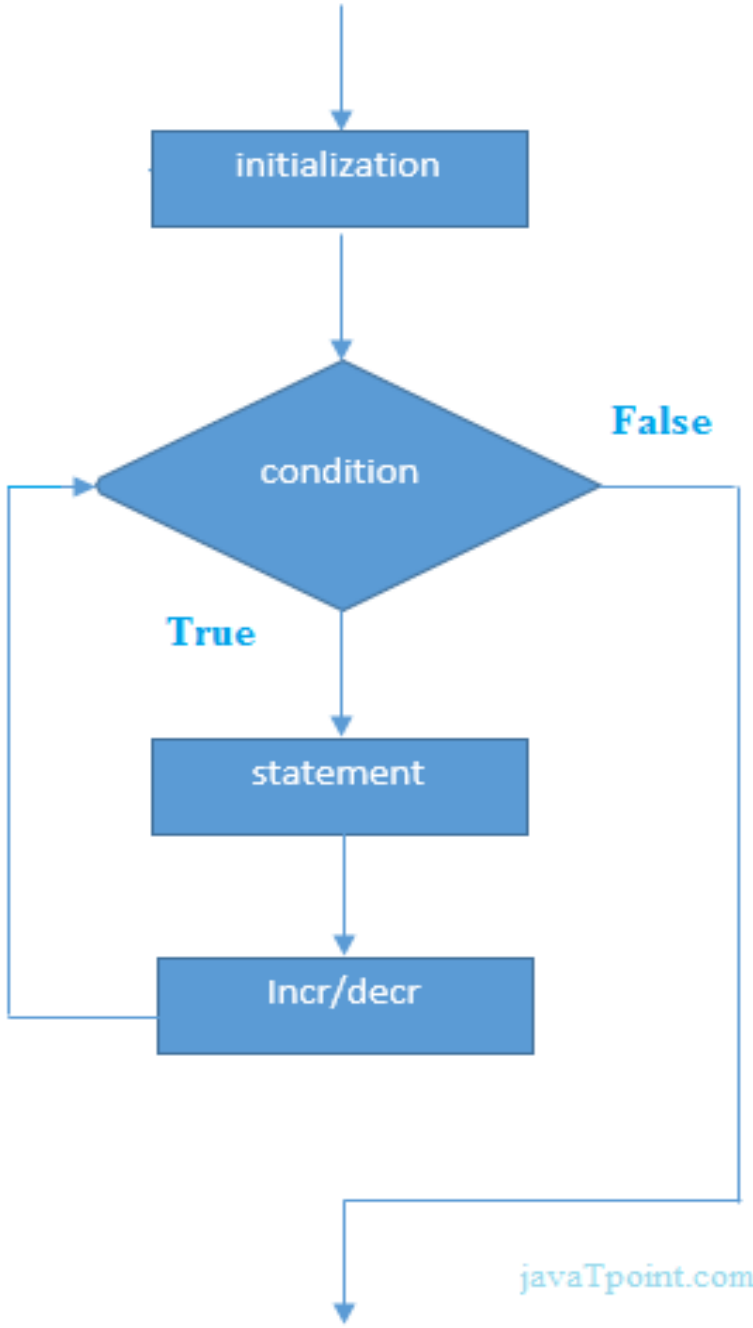
Java Simple For Loop

- 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:
    - **Initialization**
    - **Condition Statement**
    - **Increment/Decrement**

Loops in Java

Syntax:

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



Loops in Java

Java for-each Loop

- The for-each loop is used to traverse array or collection in java.
- It is easier to use the 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
}
```

Loops in Java

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.

Loops in Java

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
}
```

Loops in Java

Java Infinitive For Loop

- If you use two semicolons ;; in the for loop, it will be infinitive for loop.

Syntax:

```
for(;;){
//code to be executed
}
```

SUMMARY

- Decision Making
- Branching

- **QUIZ FOR THE CLASS**

1. **What will be the output of the following program?**

```
public class Test {
public static void main(String[] args)
   {
      int x = 10;
      if (x) {
         System.out.println("HELLO
REVA");
      } else {
         System.out.println("BYE");
      }
   }
}
```

Options:
1. HELLO REVA
2. Compile time error
3. Runtime Error
4. BYE

error: incompatible types: int cannot be converted to boolean

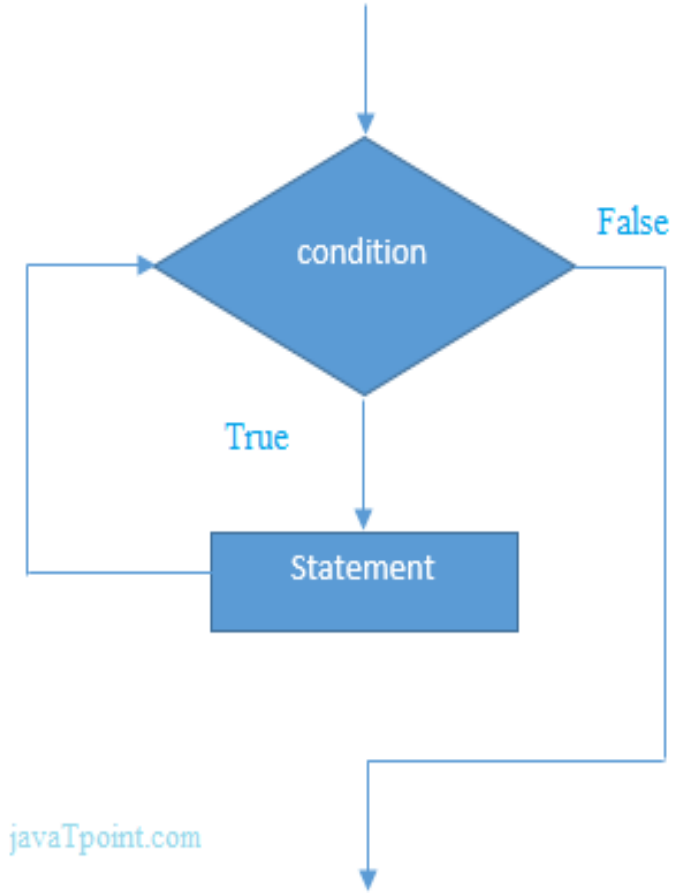
Loops in Java

Java 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.

Syntax:

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



Loops in Java

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
}
```

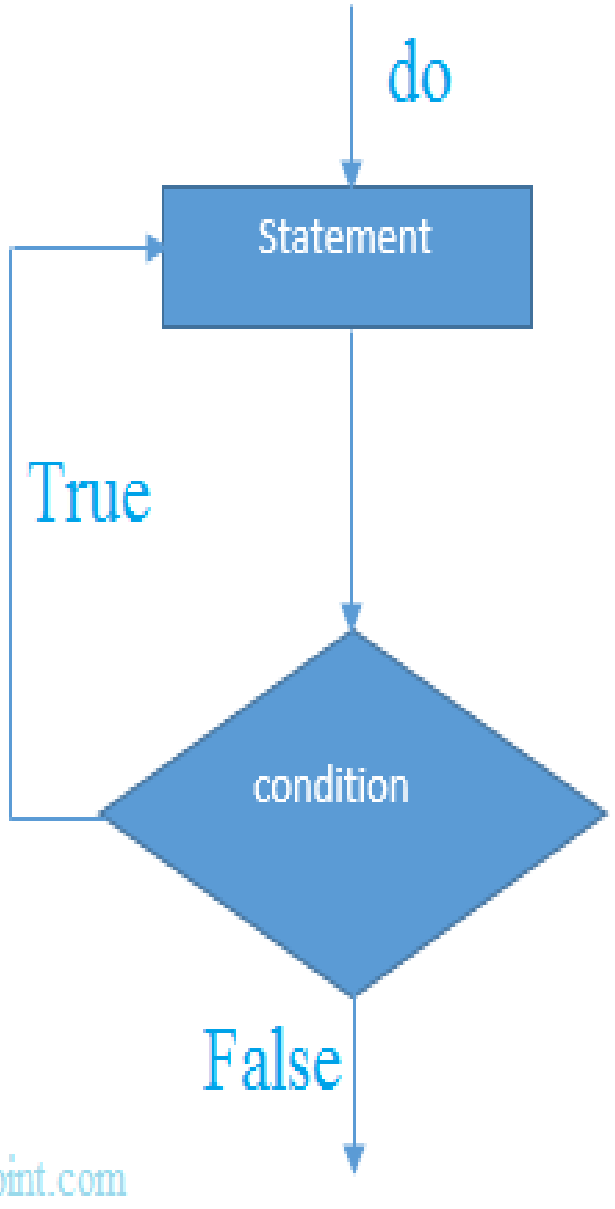
Loops in Java

Java do-while Loop

- The Java *do-while loop* is used to iterate a part of the program several times.
- The Java *do-while loop* is executed at least onc because condition is checked after loop body.

Syntax:

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



Loops in Java

Java Infinitive do-while Loop

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

Syntax:

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

Java Break and Continue

- The break and continue keywords in Java are used to **control** the **flow** of **loops** and **switch** statements.
- They provide a way to **alter** the **normal execution sequence** by either **exiting** a **loop** or **skipping** the **current** iteration.
- **break** Keyword
- The break keyword is used to **exit** a **loop** or **switch** statement prematurely.
- It **immediately terminates** the loop or switch and transfers control to the statement following the loop or switch.

Syntax:

```
break;
```

continue Keyword:

- The continue keyword is used to skip the current iteration of a loop and proceed to the next iteration. It does not terminate the loop; instead, it causes the loop to jump to the next iteration.

Syntax:

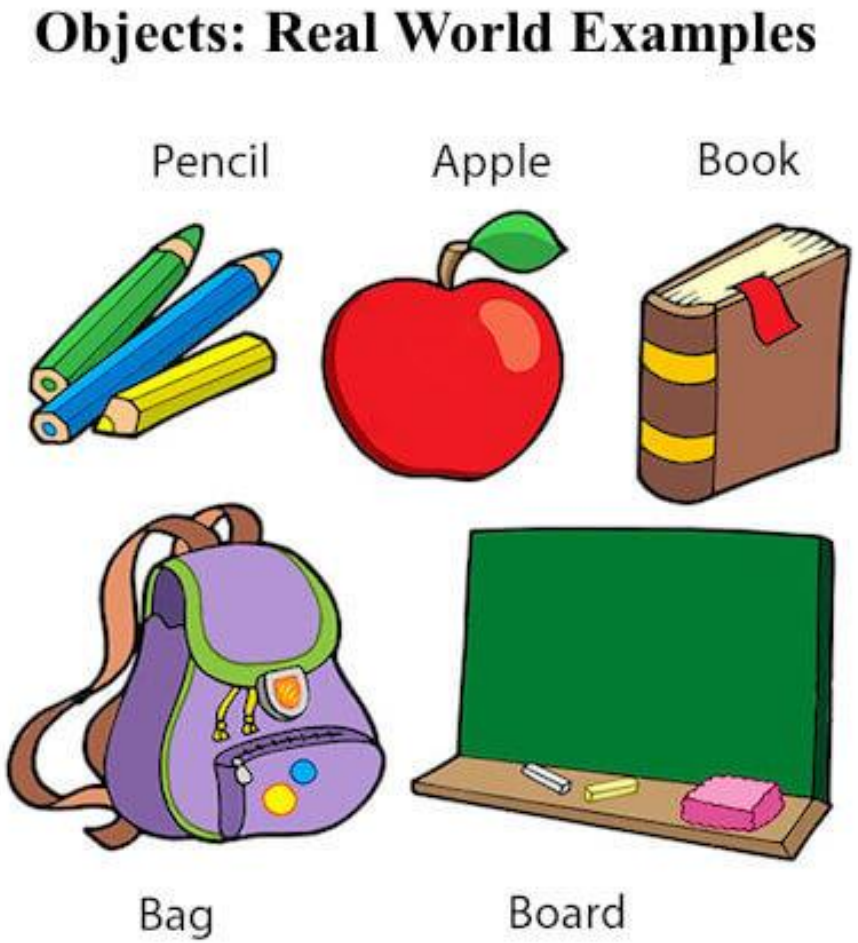
```
continue;
```

SUMMARY

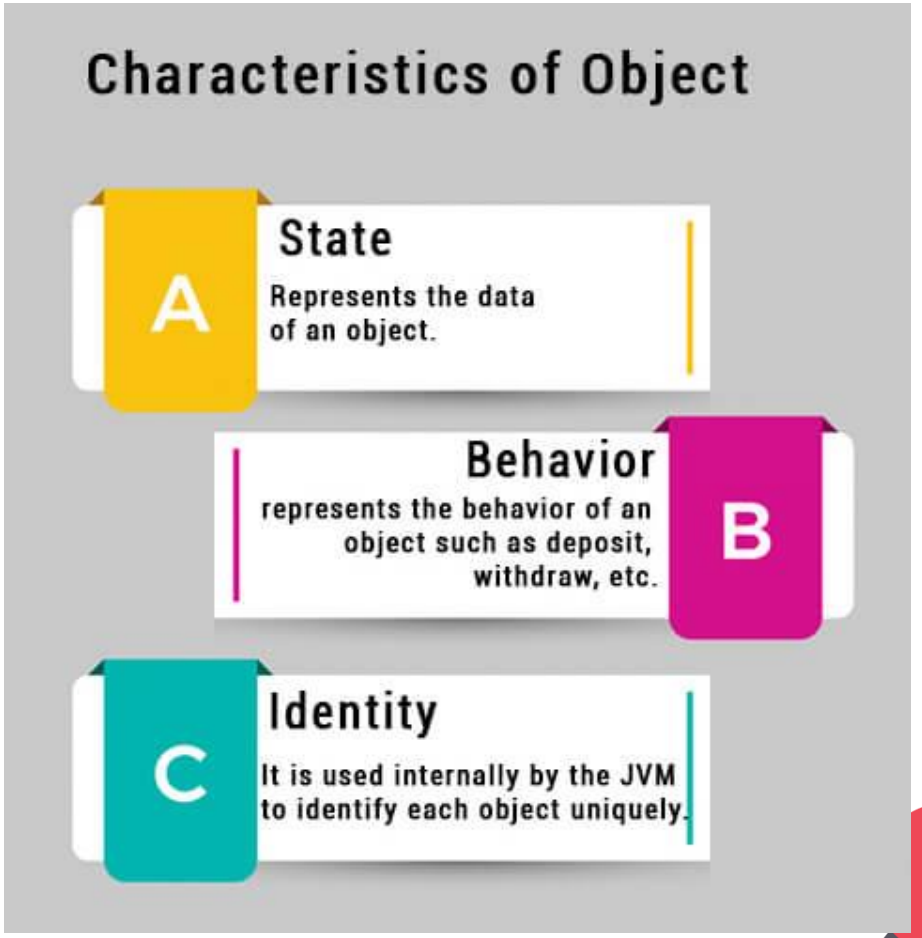
- Decision Making
- Branching

Classes and objects

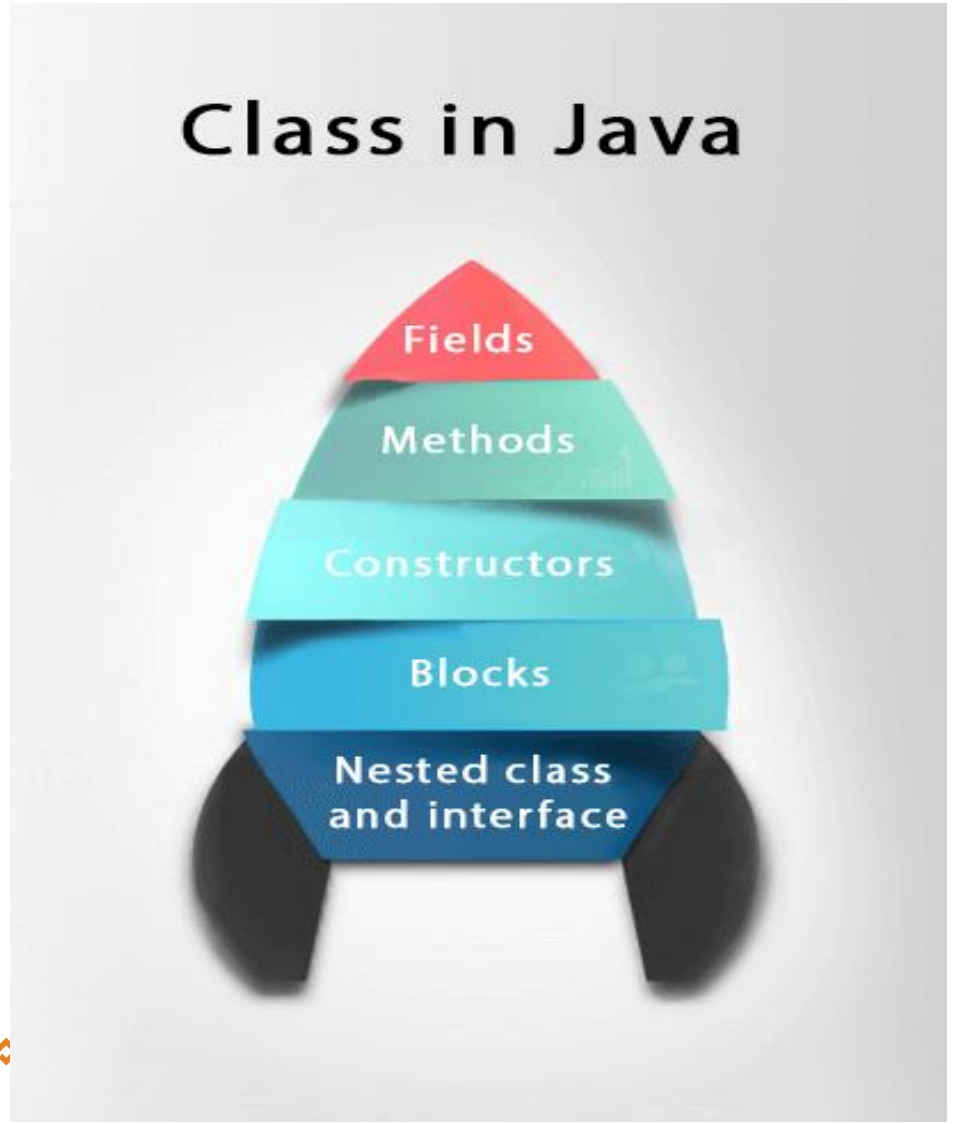
Object?



characteristics:



Members of the class

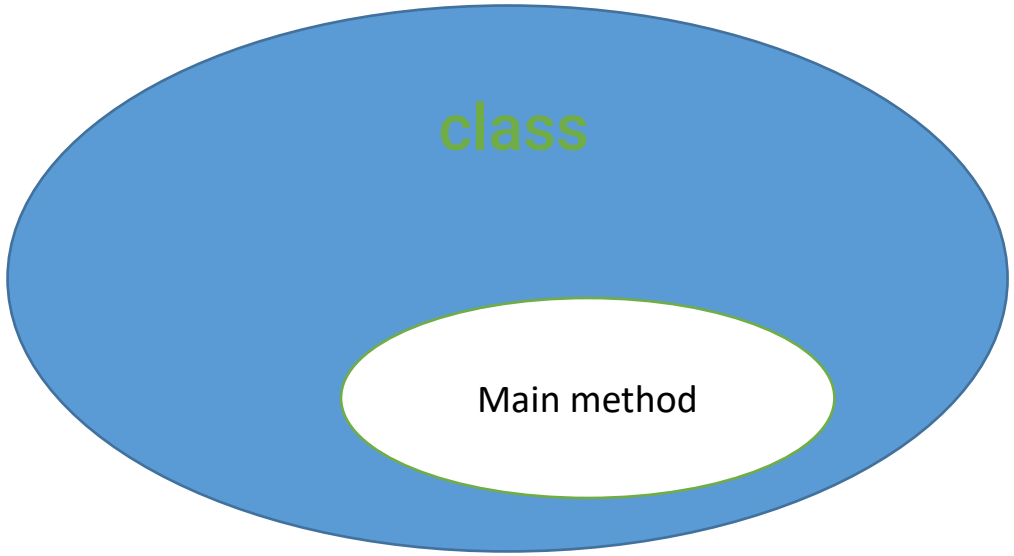


- A variable which is created inside the class but outside the method is known as an instance variable.
- Instance variable doesn't get memory at compile time.
- It gets memory at runtime when an object or instance is created.

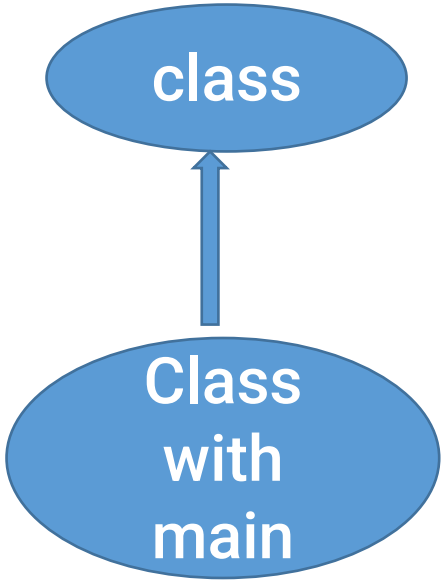
new keyword

- The new keyword is used to allocate memory at runtime.
- All objects get memory in Heap memory area of JVM.

main within the class



main outside the class



Quiz for this class

1. Identifiers can not starts with?

   A. Characters

   B. Digits

   C. underscore (_) symbols

   D. None of the above

Digits

2. Of the below, what is an invalid variable name?

   A. **reva_university@bangalore**

   B. **2004_reva_university**

   C. **reva@best_university**

   D. **_revauniversity**

2004_reva_universit
y

Topics to be learned from this class

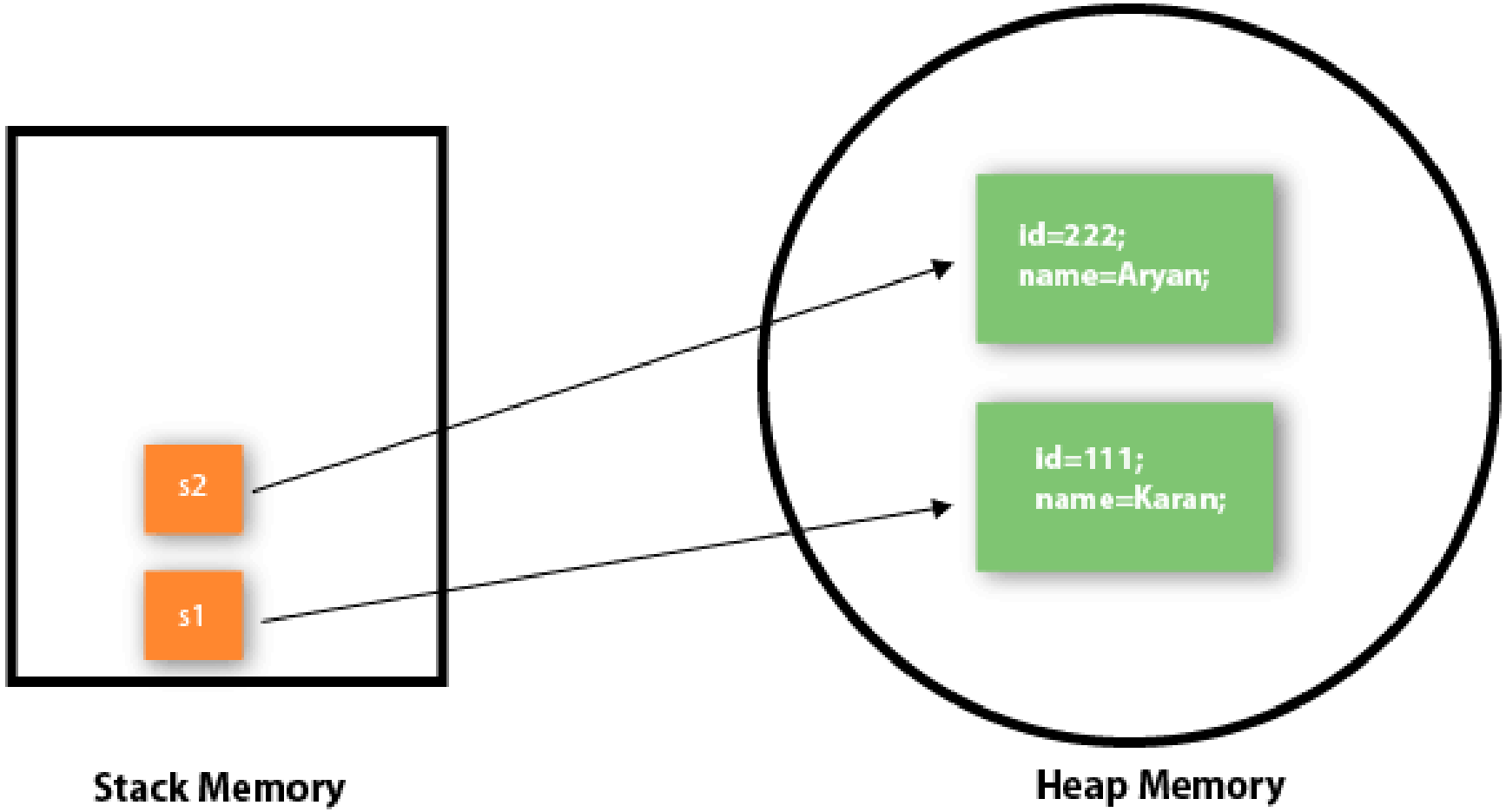
- Initialization of object in JAVA
- Creation of object in JAVA
- Constructors in JAVA

Initialization of object in JAVA

- There are 3 ways to initialize object in Java.

1. **By reference variable (multiple objects)**
2. **By method**
3. **By constructor**

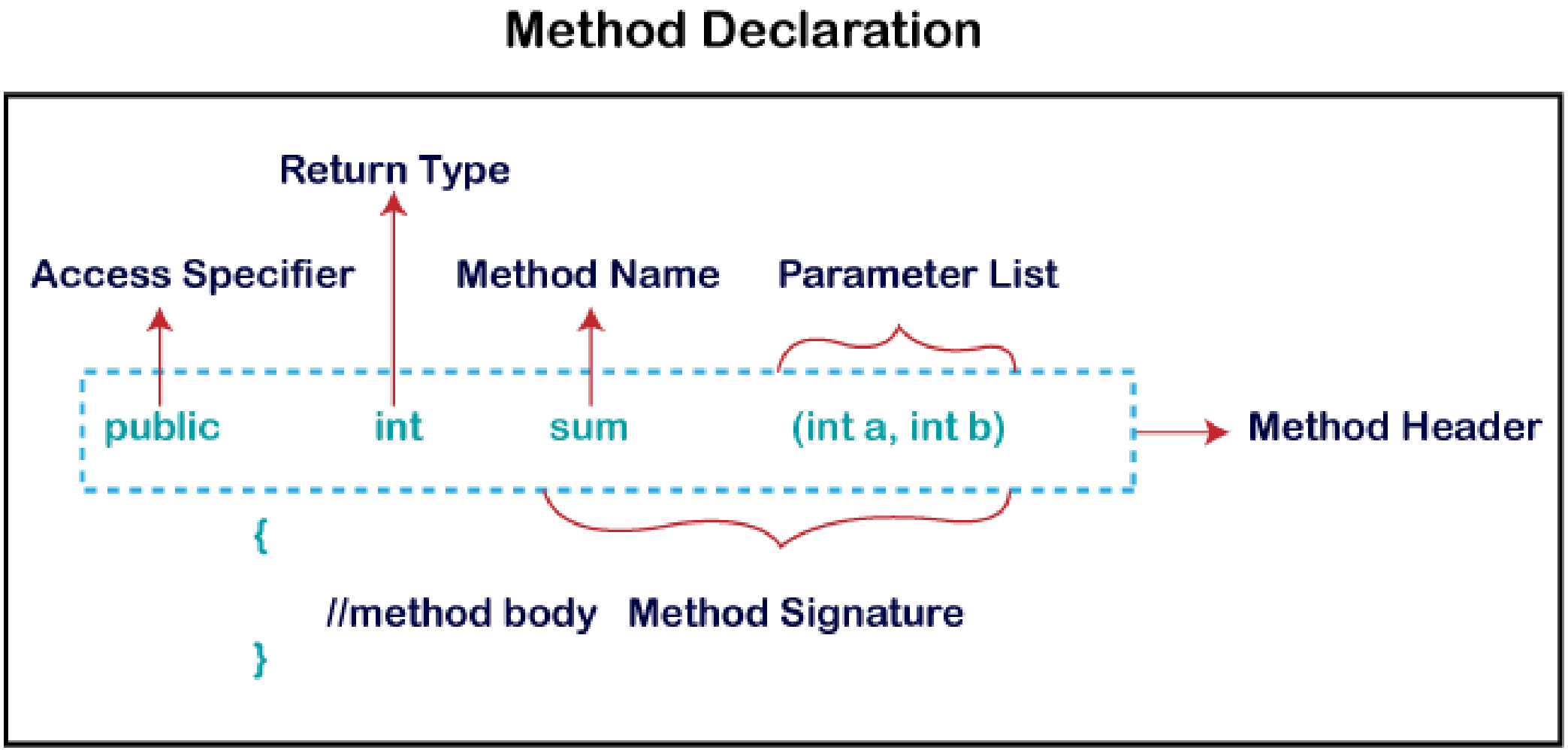
Initialization of object in JAVA



Methods in java

- A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.
- It is used to achieve the **reusability** of code.
- Write a method once and use it many times.
- We do not require to write code again and again.
- It also provides the **easy modification** and **readability** of code.
- The method is executed only when we call or invoke it.

Method Declaration



Types of Methods

There are two types of methods in Java:

- Predefined Method
- User-defined Method

- predefined methods are the method that is already defined in the Java class libraries is known as predefined methods.
- It is also known as the **standard library method** or **built-in method**.
- The method written by the user or programmer is known as **a user-defined** method.
- These methods are modified according to the requirement.

Abstract Method

- The method that does not has method body (without an implementation) is known as abstract method.
- It always declares in the **abstract class**.
- It means the class itself must be abstract if it has abstract method.
- To create an abstract method, we use the keyword **abstract**.
- Syntax:

```
abstract void method_name();
```

Constructor in JAVA

- Constructor in java is used to create the **instance of** the **class**.
- Constructors are almost similar to methods except for two things - its **name is the same as the class name** and it has **no return type**.
- Sometimes constructors are also referred to as **special methods to initialize an object.**
- '**new**' keyword to **create** an **instance** of a class, the constructor is invoked and the object of the class is returned.
- **Constructor** can only **return** the **object** to **class**, it's implicitly done by java runtime and we are **not supposed** to **add** a **return type** to it.
- If we add a return type to a constructor, then it will become a **method** of the class.

Constructor in JAVA

- A constructor in Java is a special method that is used to initialize objects.
- The constructor is called when an object of a class is created.
- It can be used to set initial values for object attributes.
- It is called when an instance of the class is created.
- At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the ‘new’ keyword, at least one constructor is called.

Types of Constructor in JAVA

- Three types of constructors in Java:
    - **No-arg constructor**
    - **Parameterized constructor.**
    - **Default constructor**
- **Note:** It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

The default constructor initializes any uninitialized instance variables with default values.

Type	Default Value
boolean	false
byte	**0**
short	**0**
int	**0**
long	**0L**
char	\u0000
float	**0.0f**
double	**0.0d**
object	Reference null

Important Notes on Java Constructors

- Constructors are invoked implicitly when you instantiate objects.

The rules for creating a constructor are:

1. The name of the constructor should be the same as the class.

2. A Java constructor must not have a return type.

3. Constructor should be declared in public section.

- If a class doesn't have a constructor, the Java compiler automatically creates a default constructor during run-time.
- The default constructor initializes instance variables with default values.

For example, the int variable will be initialized to 0

Constructor types:

- No-Arg Constructor - a constructor that does not accept any arguments
- Parameterized constructor - a constructor that accepts arguments
- Default Constructor - a constructor that is automatically created by the Java compiler if it is not explicitly defined.
- A constructor cannot be abstract or static or final.
- A constructor can be overloaded but can not be overridden.

summary

- Initialization of object in JAVA
- Creation of object in JAVA
- Introduction to Constructors in JAVA

Quiz for this class

1. Which of the following option leads to the portability and security of Java?

   A. Bytecode is executed by JVM

   B. The applet makes the Java code secure and portable

   C. Use of exception handling

   D. Dynamic binding between objects

2. Which of the following is not a Java features?

   A. Dynamic

   B. Architecture Neutral

   C. Use of pointers

Bytecode is executed by JVM

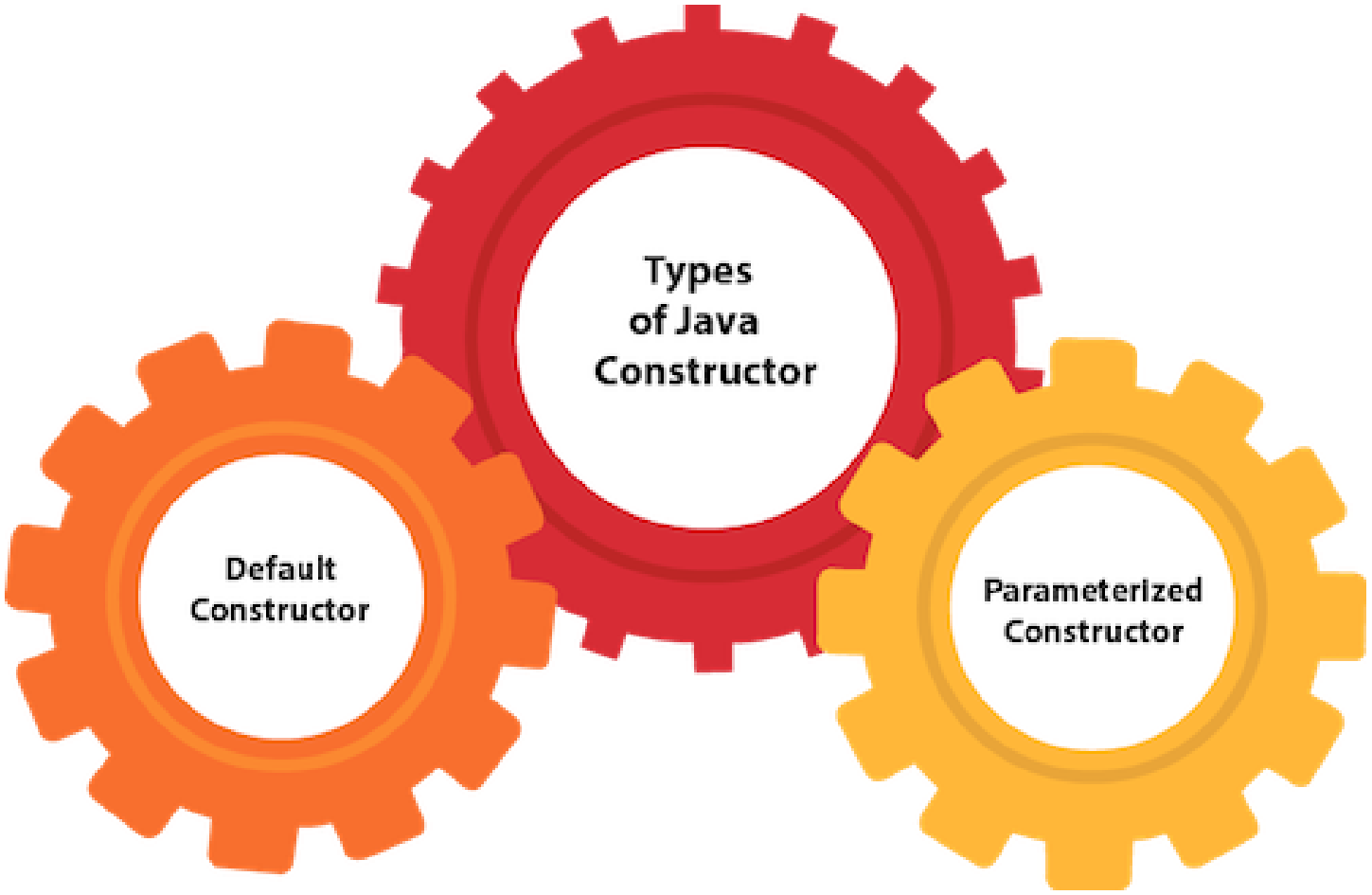
Use of pointers

Rules for creating Java constructor

Rules defined for the constructor.

- Constructor name must be the same as its class name.
- A Constructor must have no explicit return type.
- A Java constructor cannot be abstract, static, final, and synchronized.

Types of Java constructors



Types of Java constructors

- Default constructor (when it doesn't have any parameter).

Example

```
class Bike1{
Bike1()
{
System.out.println("Bike is created");
}
 public static void main(String args[])
{
Bike1 b=new Bike1();
}
}
```

Types of Constructor in JAVA

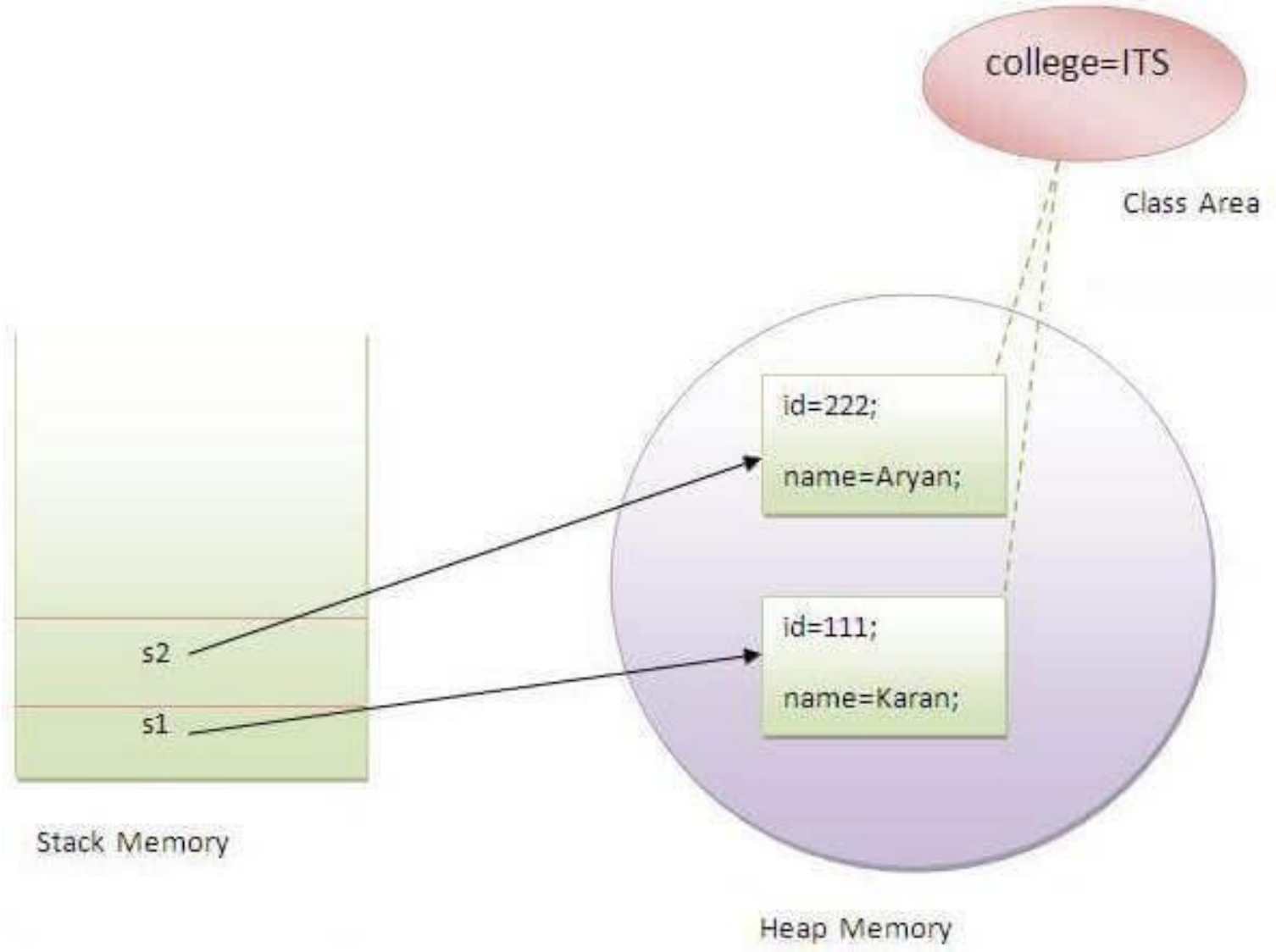
- which has a specific number of parameters is called a parameterized constructor.

Example

```
class Student4{
int id;
String name;
Student4(int i,String n)
{       id = i;
        name = n;
}
void display()
{
        System.out.println(id+" "+name);
}

public static void main(String args[])
{
   Student4 s1 = new Student4(111,"Karan");
   Student4 s2 = new Student4(222,"Aryan");

s1.display();
s2.display();
}
}
```


college=ITS
Class Area
id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

Constructor Overloading in Java

- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.
- They are arranged in a way that each constructor performs a different task.
- They are differentiated by the compiler by the number of parameters in the list and their types.

Java static keyword

- The static keyword applies to variables, methods, and nested classes, signifying that they belong to the class itself rather than to individual instances of the class.
- To access class members, we must first create an instance of the class.
- If we want to access class members without creating an instance of the class, we can use the **static** keyword in Java.
- The **Math** class in Java has almost all of its members static.
- The **static keyword** in Java is used for memory management mainly.

Static methods are also called class methods.

Static method belongs to the class rather than the object of a class.

Invoke static methods directly using the class name.

Static methods act upon static variables.

Static methods cannot read and act upon instance variables.

static variable whose single copy is shared by all objects.

classname.methodname().

static methods are stored in method area of JVM.

- A static method in a class can directly access other static members of the class. We do not need to create an object of class for accessing other static members. It can be called directly within the same class and outside the class using the class name.
- It cannot access instance (i.e. non-static) members of a class.
- We cannot declare a static method and instance method with the same signature in the same class hierarchy.

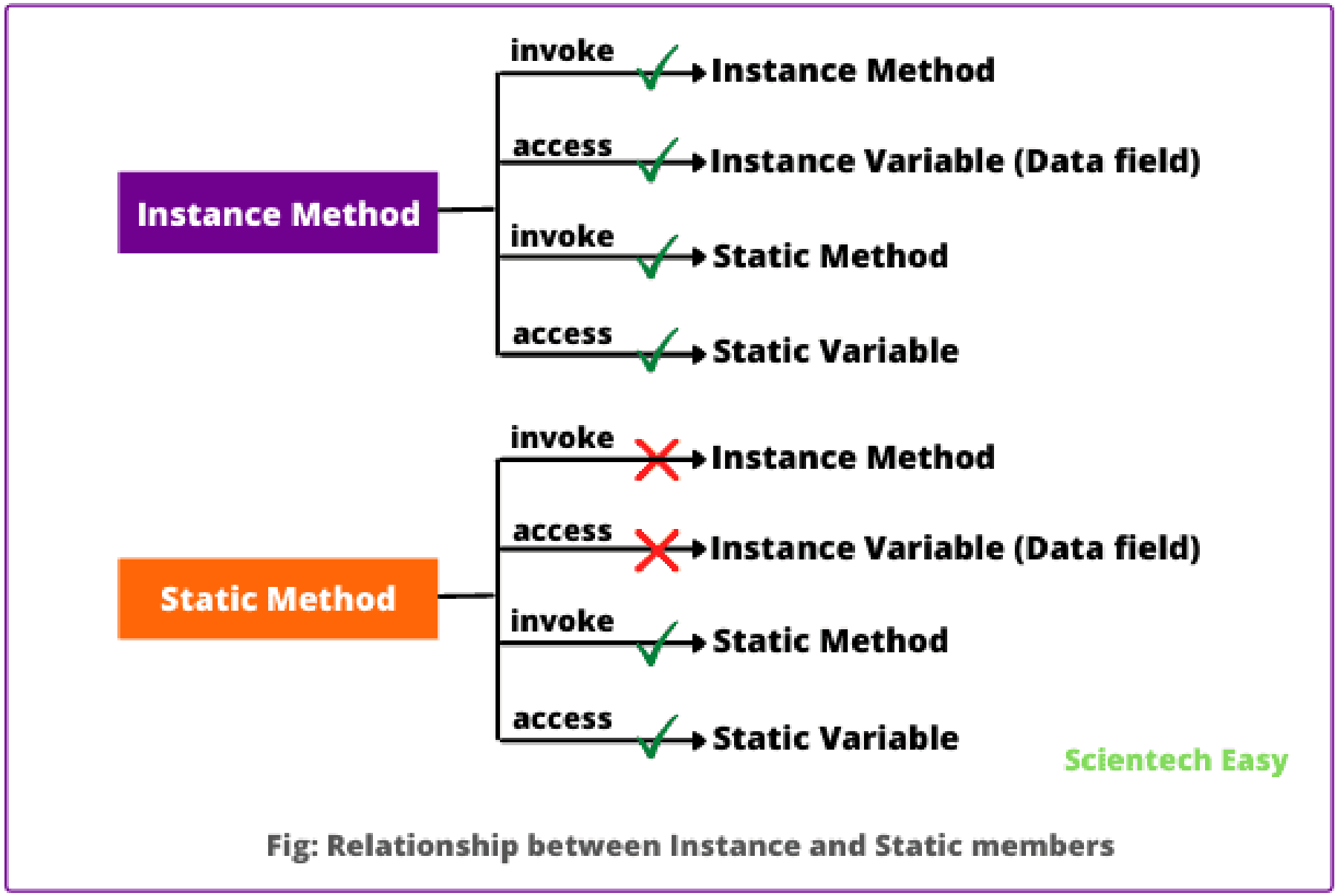


Fig: Relationship between Instance and Static members

- When we create a static method in the class, only one copy of the method is created in the memory and shared by all objects of the class. Whether you create 100 objects or 1 object.
- A static method in Java is also loaded into the memory before the object creation.
- The static method is always bound with compile time.
- Static methods can be overloaded in Java but cannot be overridden because they are bound with class, not instance.
- Static (variable, method, and inner class) are stored in Permanent generation memory (class memory).

Why Instance Variable is not Available to Static Method?

- When we declare a static method in Java program, JVM first executes the static method, and then it creates objects of the class. Since objects are not available at the time of calling the static method.
- Therefore, instance variables are also not available to a static method. Due to which a static method cannot access an instance variable in the class.

- Used in real-time applications for memory efficiency, shared data, utility operations, and structured code organization.

Counting number of users/sessions:

```
class User {
    static int userCount = 0;
    User() {
        userCount++;
    }
}
```

Static Methods: When behavior does not depend on object state.

To create utility/helper classes like Math, Arrays, or Collections.

Math.sqrt(25); // static method

```
class StaticMethodExample
{
    int a = 0;
    static int b = 0;

    void showData()
    {
        System.out.println("Value of a : "+ a);
        System.out.println("Value of b : "+ b);
    }

    static void displayData()
    {
        StaticMethodExample obj = new StaticMethodExample();
        System.out.println("Value of a : "+ obj.a);
        System.out.println("Value of b : "+ b);
    }


   public static void main(String arg[])
   {
      displayData();
      StaticMethodExample obj = new StaticMethodExample();
      obj.showData();
   }
}
```

A non-static method can call static and non-static variable directly within class

A static method can call static variable directly but non-static variable by use of object

A static method can directly call another static method but non-static method by use of object

Static Classes:

•Used to **group classes** logically and avoid memory overhead of enclosing class.

•Typically used in **Builders design patterns**, **Data Structures**, or **UI Components**.

```
class BinaryTree {
    static class Node {
        int data;
        Node left, right;
    }
}
```

- When an inner class is defined with a static modifier inside the body of another class, it is known as a **static nested class in Java**.
- A static nested class is also considered a top-level class, nested top-level class, or static member class in Java**.** But it is not considered as an inner class.

```
// A top-level class.
public class Outer  {
// Static member class.
 public static class Nested   {
 // Body for class Nested.
 }
}
```

- Since it is a static nested class, we do not need an instance of its outer class to create its object.

- An object of class Outer and an object of class Nested can exist independently because both are top-level classes.

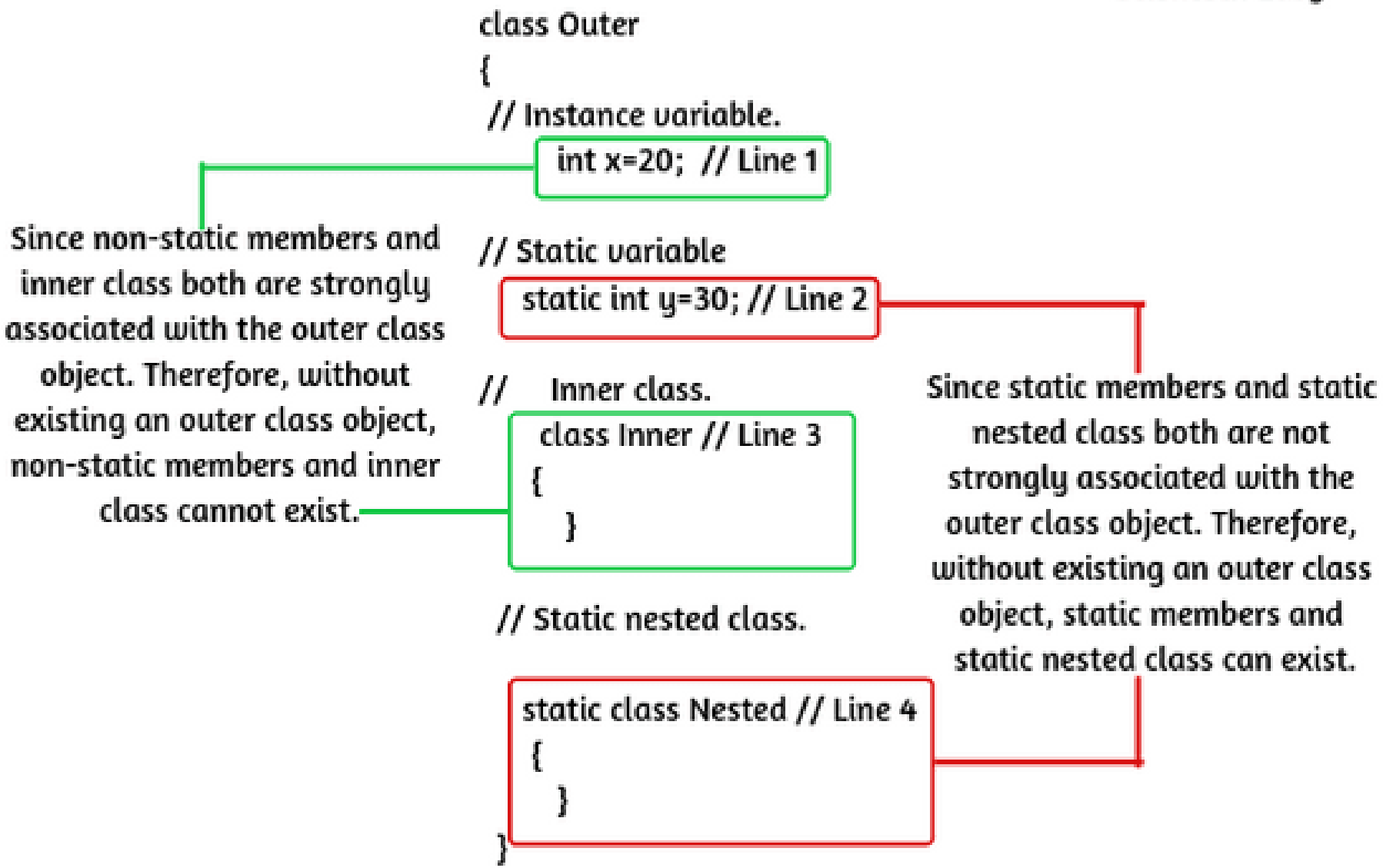


Fig: Association of static nested class and inner class with the outer class object.

‘this’ keyword in java

- Java will **not allow to declare two or more variables** having the **same name inside a class scope or method.**
- Instance variables and parameters may have the same name.

```
class Demo_Class {
    // instance variable
    int age=10;
    // parameter
    Demo_Class(int age){
        age = age;
    }
}
```

- the instance variable and the parameter have the same name: **age**.
- the Java compiler is confused due to name ambiguity.

```
class Main {
    int age;
    Main(int age){
        age = age;
    }
public static void main(String[] args) {
       Main obj = new Main(11);
        System.out.println("obj.age = " +
obj.age);
    }
}
```

Output: 0

Because Java compiler gets confused because of the ambiguity in names between instance the variable and the parameter.

```
class Main {
    int age;
    Main(int age){
        this.age = age;
    }
    public static void main(String[] args) {
        Main obj = new Main(8);
        System.out.println("obj.age = " + obj.age);
    }
}
```

Output: obj.age = 8

The constructor is called, **this** inside the constructor is replaced by the object **obj** that has called the constructor.

Hence the age variable is assigned value 8.

- If the name of the parameter and instance variable is different, the compiler automatically appends '**this**' keyword.

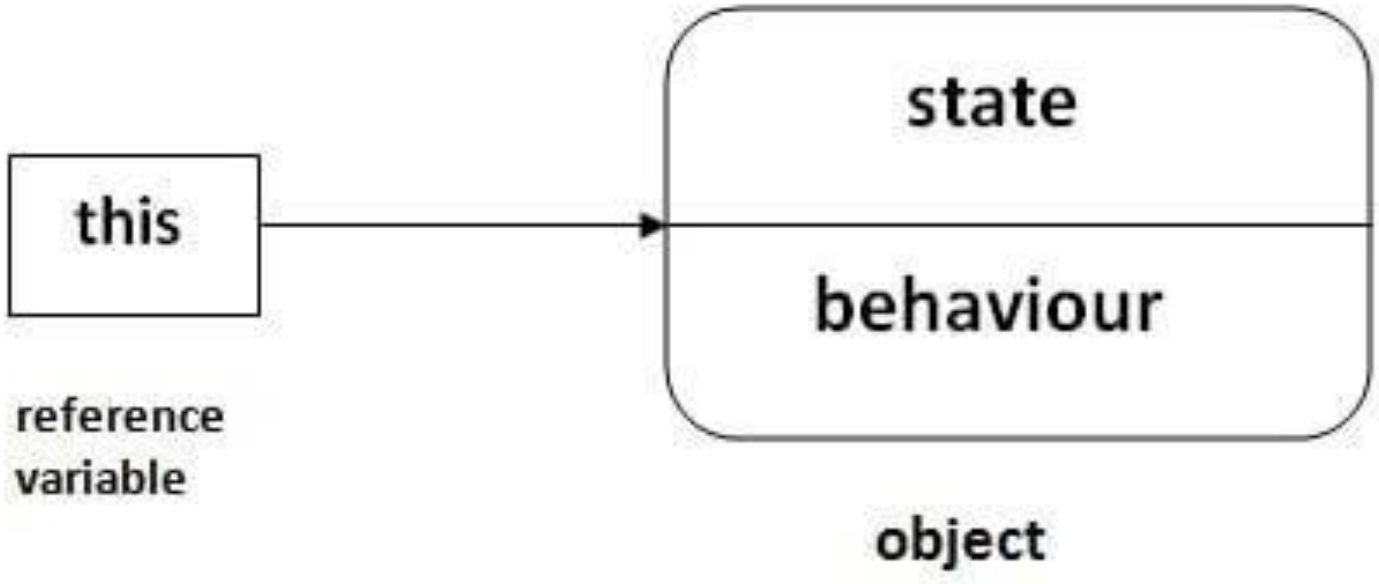
```
class Main {
    int age;
    Main(int i) {
        age = i;
    }
}
```

is equivalent to:

```
class Main {
    int age;
    Main(int i) {
        this.age = i;
    }
}
```

this keyword in java

- this is a **reference variable** that refers to the current object.



- The **this** keyword refers to the **current object** in a **method or constructor**.
- The most common use of the this keyword is to **eliminate the confusion** between **class attributes and parameters** with the same name.

***this** keyword is used:*

- To Invoke **current class** constructor.
- To Invoke current **class method**.
- To Return the **current class object**.
- To Pass an **argument** in the method call.
- To Pass an **argument** in the constructor call.
- advantages of this() is to **reduce the amount of duplicate code**.
- We should be always **careful** while using this() because **calling constructor from another constructor adds overhead** and it is a **slow process.**

- **'this' is used** with **Getters and Setters:**

  to assign value inside the setter method.

  to access value inside the getter method.

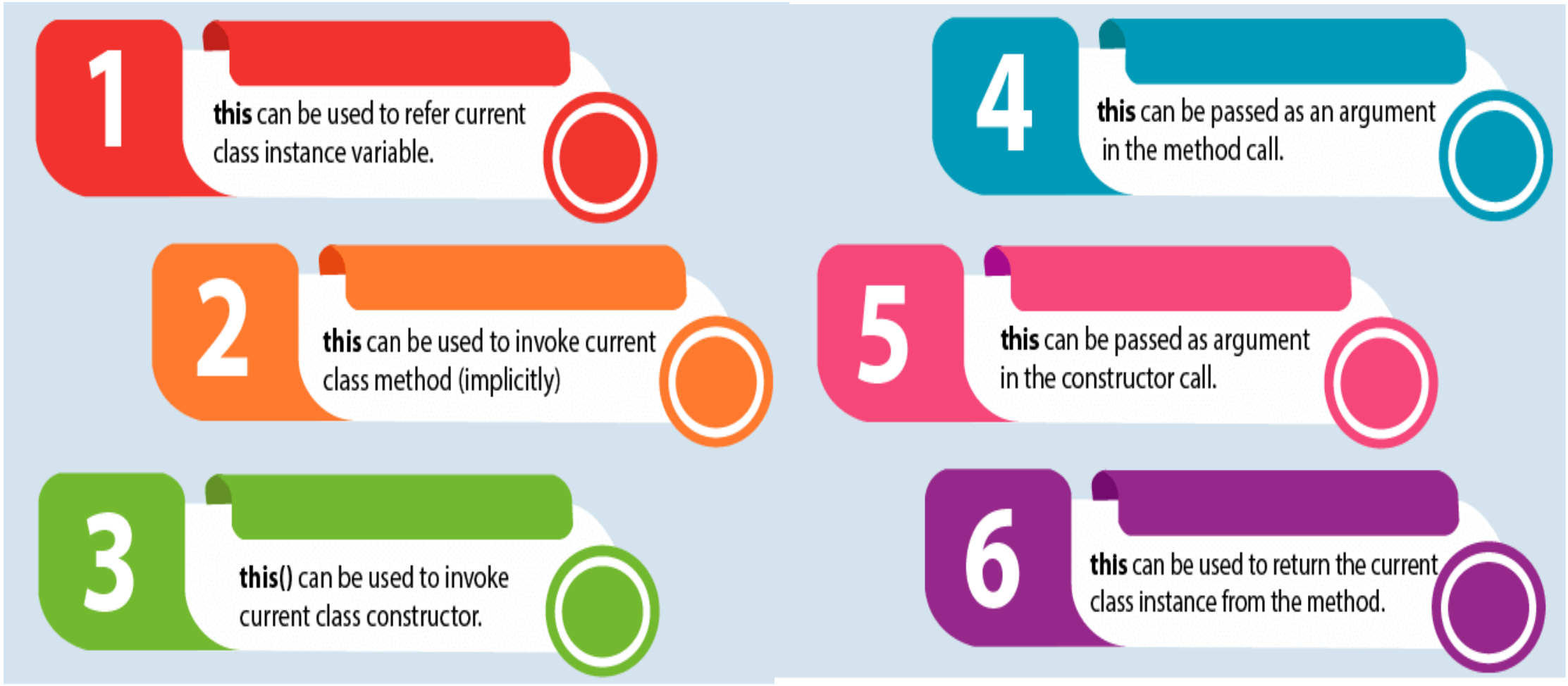
Using this in Constructor Overloading:

Invoking one constructor from another constructor is called explicit constructor invocation.

Passing 'this' as an Argument:

can use 'this' keyword to pass the current object as an argument to a method.

Usage of java this keyword



Quiz for this class

1. **What is true about constructor?**

   A. It can contain return type

   B. It can take any number of parameters

   C. It can have any non access modifiers

   D. Constructor cannot throw an exception

It can take any number of parameters

2. What is not the use of “this” keyword in Java?

   A. Passing itself to another method

   B. Calling another constructor in constructor chaining

   C. Referring to the instance variable when local variable has the same name

   D. Passing itself to method of the same class

Passing itself to method of the same class

Java memory management

- Java memory management is a fundamental concept that involves the automatic allocation and deallocation of objects, managed by the Java Virtual Machine (JVM).
- The JVM uses a garbage collector to automatically remove unused objects, freeing up memory in the background.
- This eliminates the need for developers to manually handle memory management.
- We will discuss on, how the heap works, reference types, garbage collection, and related concepts.

- Java automatically manages memory with the help of the garbage collector.
- Programmers do not need to manually destroy objects.
- Not all memory are managed equally by the garbage collector.
- Understanding memory management helps create efficient programs and debug crashes effectively.

- Some of the areas are created by the JVM, whereas some are created by the threads that are used in a program.
- The memory area created by the JVM is destroyed only when the JVM exits.
- The data areas of a thread are created during instantiation and destroyed when the thread exits.

1. Heap Area
2. Method Area
3. JVM Stacks
4. Native Method Stacks
5. Program Counter (PC) Registers

- **Heap Memory :** Managed by Garbage Collector (GC).
- Unreachable objects are automatically deleted.

- **Stack Memory:** Automatically cleared when method execution ends.

- **Method Area:** GC may clear unused classes and static data if no longer needed.

Heap Area

- Heap is a **shared runtime data area** where **objects** and **arrays** are **stored**.
- It is **created** when the **JVM starts**.
- The **memory** in the **heap** is **allocated** for all the **class instances** and **arrays**.
- Heap can be of **fixed** or **dynamic** size depending upon the **system's configuration**.
- JVM allows user to adjust the heap size.
- When the **new** keyword is used the **object** is **allocated** in the **heap** and its **reference** is **stored** in the **stack**.
- There exists **one** and **only one heap** for a running JVM process.

Scanner sc = new **Scanner(System.in)**

The **Scanner** object is stored in the **heap** and the **reference sc** is stored in the **stack**.

Note: **Garbage collection** in **heap area** is **mandatory**.

- **Method Area**
- Method area is a **logical** part of the heap and it is created when the JVM starts.
- Method area is used to **store class-level** information such as **class structures**, **Method bytecode**, **Static variables**, **Constant pool**, **Interfaces**.
- Method area can be of **fixed** or **dynamic** size depending on the **system's configuration**.
- Static variables are **stored** in the **Stack**.
- **Garbage collection** of the method area is not guaranteed and depends on JVM implementation.
- **Note**: Method area is logically a part of heap, many JVM like HotSpot uses a separate space known as **Metaspace** outside the heap to store it.

JVM Stacks

- A **stack** is created when a **thread** is **created**, and the JVM stack is used to store method execution data, including local variables, method **arguments**, and **return addresses**.
- Each **Thread** has its **own stack**, ensuring **thread safety**.
- Stacks size can be **either fixed** or **dynamic**, and it can be set when the **stack** is **created**.
- The memory for stack needs not to be **contiguous**.
- Once a method **completes execution**, its **associated stack frame** is removed **automatically**.

Native Method Stacks

- **Native method** stack is also **known** as **C stacks**.
- Native method stacks are **not written** in **Java language.**
- This memory is **allocated** for **each thread** when **it is created** and can have **either a fixed or dynamic size.**
- **Native method** stacks handle the **execution** of **native methods** that interact with the Java code.

Program Counter (PC) Registers

- Each **JVM thread** which carries out the task of a **specific method** has a **Program Counter (PC)** register associated with it.
- The **non native method** has a **PC** which **stores** the **address** of the **available JVM** instruction.
- **PC register is capable of storing** the **return address** or a **native pointer** on some **specific platform.**

Garbage Collector

- The Garbage collector in Java automatically removes the unused objects that are no longer needed.
- It runs in the background to free up memory.
- Garbage collector finds objects that are no longer needed by the program.
- It removes those unused objects to free up the memory and making space for new objects.
- Java uses generational garbage collection so that new objects are collected more frequently in the young generation than older objects which survive longer in the old generation, this helps improve efficiency.
- We can request garbage collection using System.gc(), but the JVM ultimately decides when it should run.

- System.gc() and Runtime.gc() are the methods which requests for Garbage collection to JVM explicitly.
- It doesn’t ensures garbage collection as the final decision of garbage collection is of JVM only.

Java final keyword

- The final keyword is used to denote constants.
- It can be used with variables, methods, and classes.
- Once any entity (variable, method or class) is declared final, it can be assigned only once.
  - the final variable cannot be reinitialized with another value
  - the final method cannot be overridden
  - the final class cannot be extended

https://www.scientecheasy.com/2021/05/labelled-loop-in-java.html/