# SDET Professional Course

By DevLabs Alliance

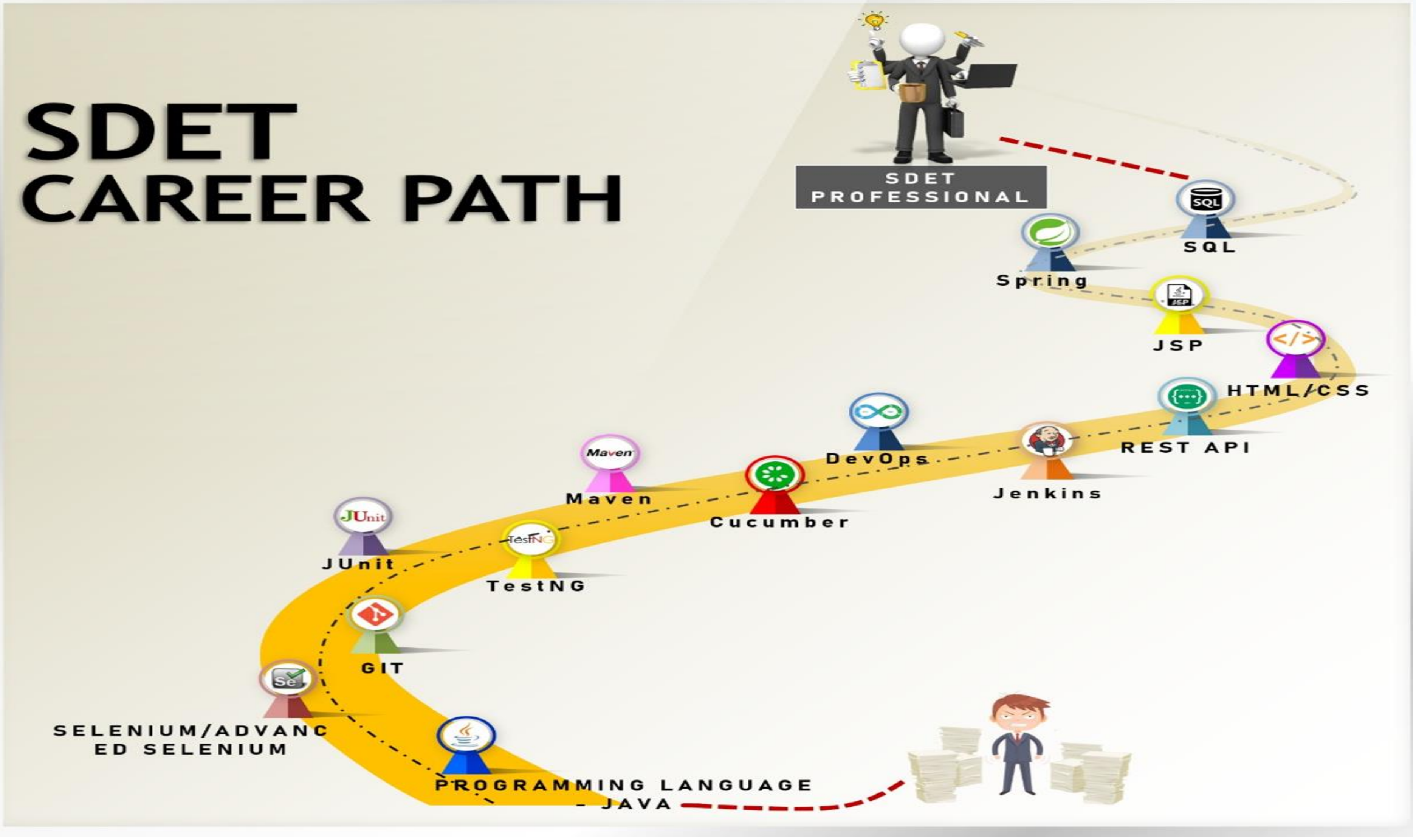
DEVLABS ALLIANCE

## Course Overview

- **Duration** – 12 weeks
- **Trainers** – 3 Industry practitioners
- **Lab** – Each session includes lab exercises/programs
- **Assignments** – Each session concludes with assignments/exercises
- **Certification** - on completion of assignments and lab exercises you will be awarded SDET Professional Certificate

DEVLABS ALLIANCE

# SDET Career Path



## SDET Foundation Curriculum

- Introduction to Programming
  - Core Java

- Automation Tools
  - Selenium
  - Cucumber
  - JUnit/TestNg
  - ATDD/BDD

DEVLABS ALLIANCE

## SDET Advanced Curriculum

- Automation Frameworks
- Rest API Automation
- DevOps
  - GIT
  - Jenkins
- Web Development
  - HTML
  - JSP
  - Servlets
  - JDBC
  - Spring Framework
  - Databases

DEVLABS ALLIANCE

# Core Java - Table of Contents

# Core Java - Table of Contents

DEVLABS ALLIANCE

# Core Java - Table of Contents

DEVLABS ALLIANCE

## What is Java?

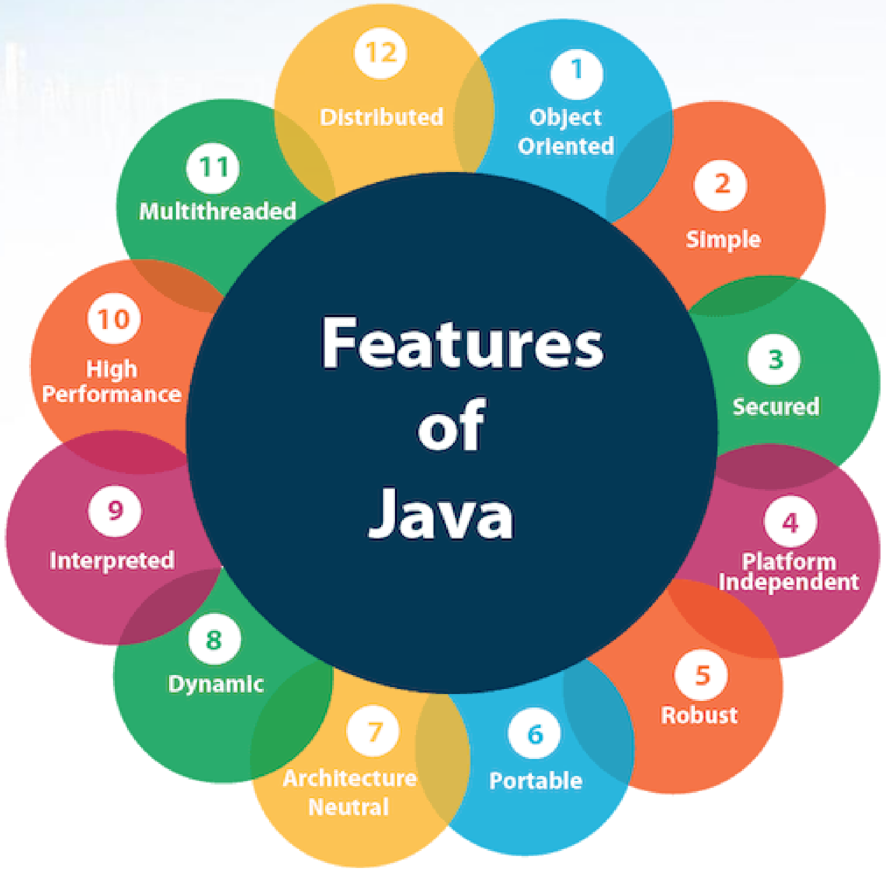
# Java

- **Java is an object-oriented programming language which produces software for multiple platforms.**

- **Java is platform independent.**
  - With application programs written in Java, you can change/upgrade OS and applications independently.

  - ***Write once, run anywhere!***
    For example, you can write and compile a Java program on UNIX and run it on Microsoft Windows.

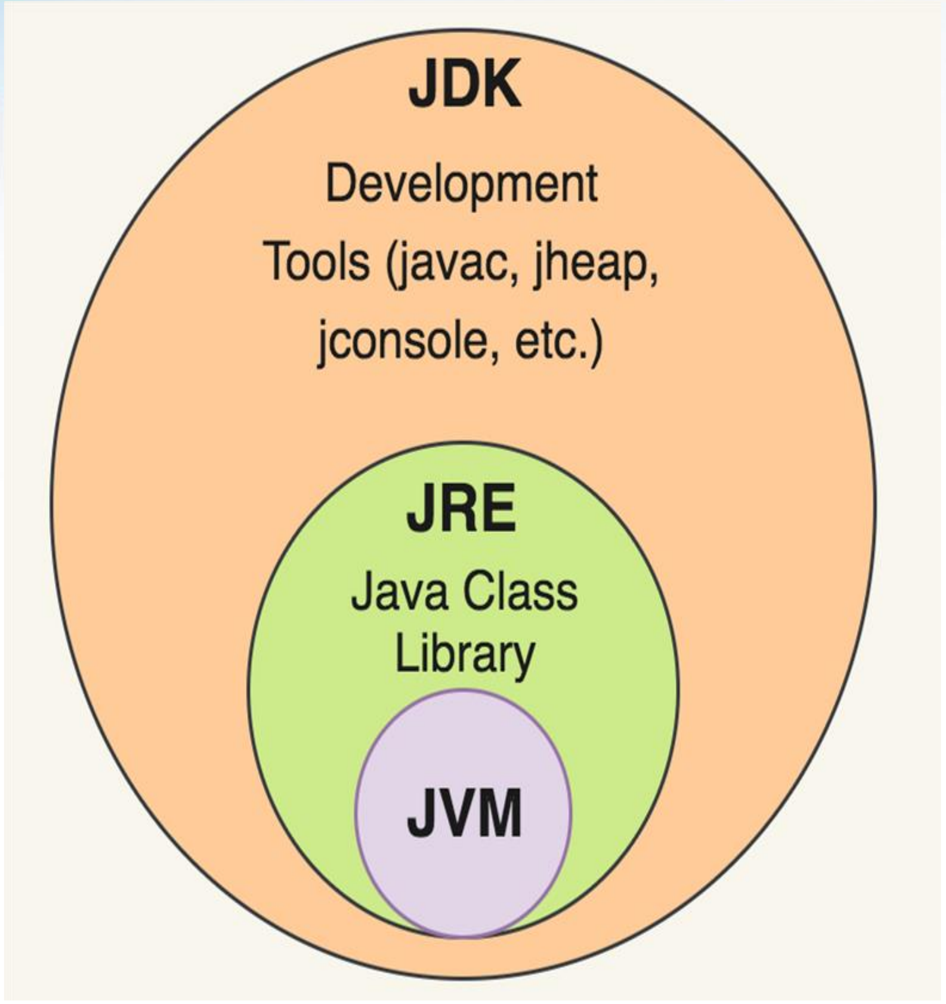
- **Java was developed by James Gosling of Sun Microsystems in 1995.**



DEVLABS ALLIANCE

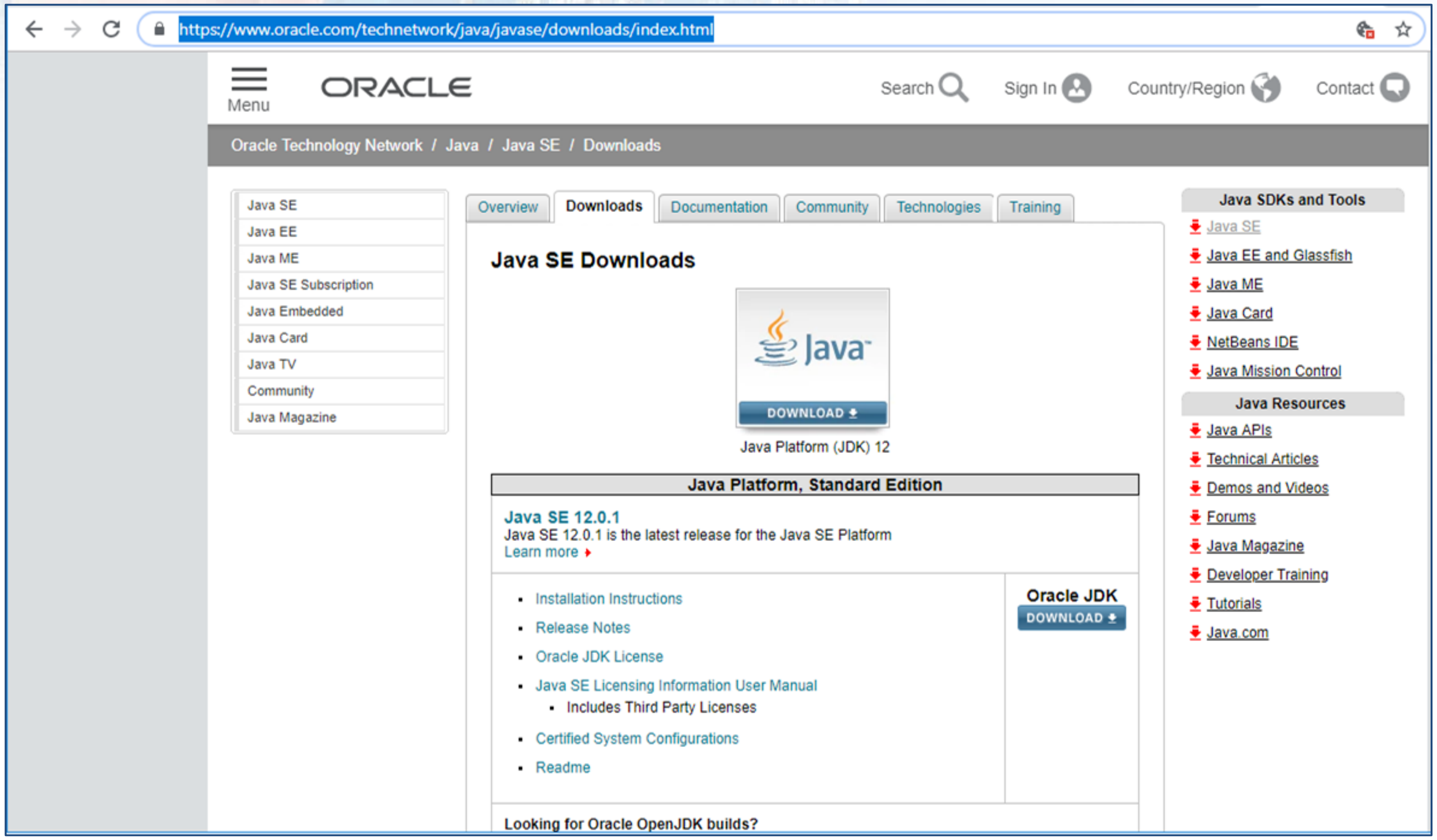
## Java Environment

- **JDK (Java Development Kit)** is the key platform component, includes JRE and development tools required for end-to-end execution of Java programs.
- **JRE (Java Runtime Environment)** is a runtime environment which provides all the libraries and files to JVM. It is the implementation of JVM.
- **JVM (Java Virtual Machine)** is an abstract machine. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line hence it is also known as interpreter.



# JDK Installation

- You can download JDK from https://www.oracle.com/technetwork/java/javase/downloads/index.html



Java SE Downloads

Java

DOWNLOAD

Java Platform (JDK) 12

Java Platform, Standard Edition

Java SE 12.0.1
Java SE 12.0.1 is the latest release for the Java SE Platform
Learn more

- Installation Instructions
- Release Notes
- Oracle JDK License
- Java SE Licensing Information User Manual
  - Includes Third Party Licenses
- Certified System Configurations
- Readme

Oracle JDK
DOWNLOAD

Looking for Oracle OpenJDK builds?

DEVLABS ALLIANCE

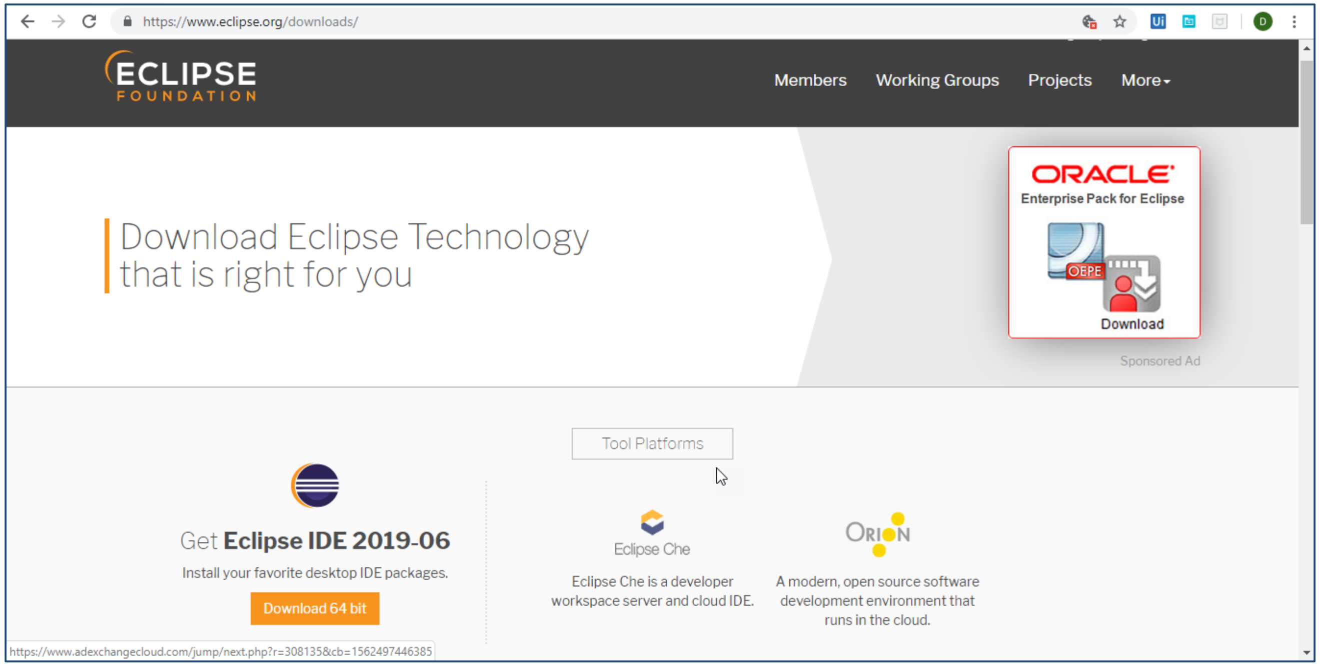
## JDK Setup

Follow these steps to set the permanent path of JDK on your system:

- Go to My Computer/ This PC -> Properties
- Click on Advanced system settings
- Then, click on Environment Variables button.
- Check the value of Path and Classpath variables.
  Path should be set to the bin folder of the jdk installation directory :
  **C:\Program Files\Java\jdk1.8.0_151\bin**
  **(**The path is used to locate the JDK packages that are used to convert the java source code into the machine-readable binary format.)
- Similarly, classpath should be set to the lib folder of the installation directory :
  **C:\Program Files\Java\jdk1.8.0_151\lib**

# IDE

- A software suite that consolidates basic tools to write and test software.
- An IDE typically contains a code editor, a compiler or interpreter, and a debugger, accessed through a single graphical user interface (GUI)
- Popular IDEs are Eclipse, NetBeans, Codenvy, IntelliJ and BlueJ.
- Download Eclipse from https://www.eclipse.org/downloads/


https://www.eclipse.org/downloads/
ECLIPSE FOUNDATION
Members
Working Groups
Projects
More
ORACLE
Enterprise Pack for Eclipse
OEPE
Download
Sponsored Ad
Tool Platforms
Download 64 bit
Eclipse Che
ORION
https://www.adexchangecloud.com/jump/next.php?r=308135&cb=1562497446385

Download Eclipse Technology that is right for you

Get **Eclipse IDE 2019-06**

Install your favorite desktop IDE packages.

Eclipse Che is a developer workspace server and cloud IDE.

A modern, open source software development environment that runs in the cloud.

DEVLABS ALLIANCE

## Getting Started

**File: HelloWorld.java**

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

**Compiling the file**
Eclipse: save the file
Command Prompt: javac HelloWorld.java

**Executing the program**
Eclipse: R-click, Run As Java Program
Command prompt: java HelloWorld

Main method

- public - accessible everywhere and to every object which may desire to use it for launching the application.
- static - No object needed to run static methods. A starting point for execution.
- void - There is no use of returning any value to JVM, who actually invokes this method.

DEVLABS ALLIANCE

# Java Program Structure

```
import java.util.*                                  Package details
public class HelloWorld                             Class
{
      int a = 5;                                    Data members
            void display()                          Member functions/ methods
            {
              System.out.println(“The value of a:” + a);
            }

      public static void main(String[] args)        Main method
      {                                              (processing start point)
          System.out.println("Hello, world!");
      }
}
```

## Language Basics

## Variables

- A variable is a container to hold a value while the java program is executed.

- A variable is assigned with a datatype.

  For example ***int x = 20***. Here, int is a data type and x is a variable.

- Variable is a name of memory location.

**Naming Conventions for variables:**

- ❏ Variable names are case-sensitive.

- ❏ A variable's name can be any legal identifier — a sequence of Unicode letters and digits, beginning with a letter, the dollar sign "$", or the underscore character "_“.

- ❏ String 1EmpName is an invalid name for a variable. (cannot begin with a number)

- ❏ The convention, is to always begin your variable names with a letter, not "$" or "_“. So, prefer String empName over String _EmpName.

DEVLABS ALLIANCE

# Variables

## Types of Variables

### Local Variables

Variables that are declared inside a particular method.

### Instance Variables

Variables that are declared inside a class but outside any method. They are not declared static.

### Static Variables

Variables that are declared as static. They cannot be local. They are called class variables also.

DEVLABS ALLIANCE

# Heap and Stack Memory

- **Stack** contains methods, local variables, and reference variables. Stack memory is always referenced in LIFO order.

- When a method is called, a frame for this method is created in the stack memory. If this method calls another method, then another frame will be created on top of the first frame, in the stack memory.

- Once a method has completed execution, the flow of control returns to the calling method and its corresponding stack frame is flushed.

- Stack Memory is not global, one stack data cannot access another stack's data.

- **Heap** is a section of memory which contains Objects. Instance variables are created in the heap & are part of the object they belong to.

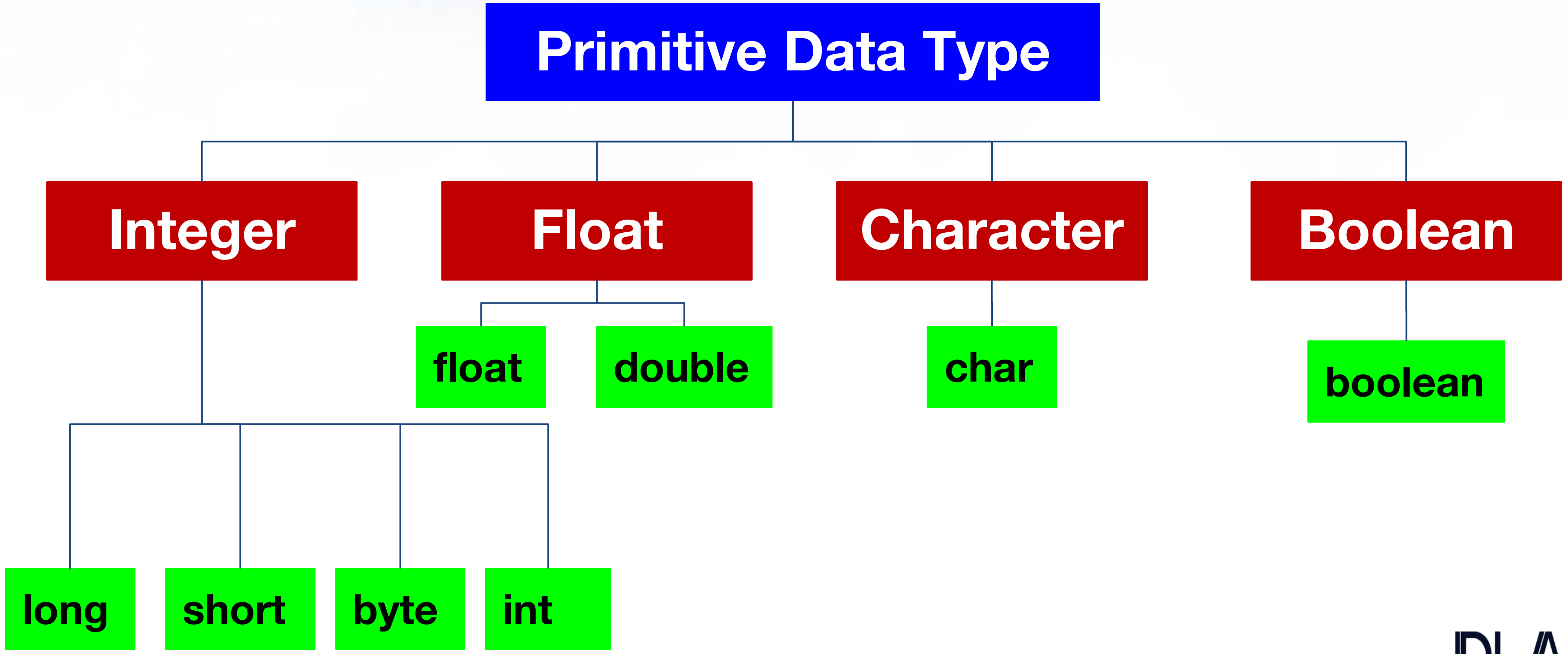
- In Java stack, memory allocation is done during execution, unlike stack which is at compile time.

- Heap is like a global memory pool. The objects you find here are accessible to all the functions.

## Data Types

Data types specify the different sizes and values that can be stored in the variable.

- **Primitive data types:** boolean, char, byte, short, int, long, float and double.
- **Non-primitive data types:** Strings, arrays etc.



## Data Types

| Data type | Range of values |
|---|---|
| `byte` | -128 .. 127 (8 bits) |
| `short` | -32,768 .. 32,767 (16 bits) |
| `int` | -2,147,483,648 .. 2,147,483,647 (32 bits) |
| `long` | -9,223,372,036,854,775,808 .. ... (64 bits) |
| `float` | +/-$10^{-38}$ to +/-$10^{+38}$ and 0, about 6 digits precision |
| `double` | +/-$10^{-308}$ to +/-$10^{+308}$ and 0, about 15 digits precision |
| `char` | Unicode characters (generally 16 bits per char) |
| `boolean` | True or false |

## Case Example

Suppose Priya, an employee at a Retail Store has to generate an invoice for a customer, Mac.
She needs to include the following fields in the invoice:

- Invoice No.
- Item No.
- Item Price
- Quantity of items purchased
- Discount
- Total Amount
- Comments

Can these fields be of same datatype? Priya is confused.
Let us help Priya decide the types of the fields.

DEVLABS ALLIANCE

# Keywords

**Java keywords** are also known as reserved words.

**Keywords** are particular words which acts as a key to a code.

These are **predefined words** by Java so it cannot be used as a variable or object name.

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

## Operators

**Operator** in java is a symbol that is used to perform operations.

There are mainly four categories of operators in Java.

- Arithmetic Operators
- Unary Operators
- Relational Operators
- Logical Operators

## Arithmetic Operators

| Operator | Meaning | Output | Example (a=6,b=2) |
|---|---|---|---|
| + | Adds two operands | Sum | a + b = 8 |
| - | Subtracts two operands | Difference | a - b = 4 |
| * | Multiplies two operands | Product | a * b = 12 |
| / | Divides two operands | Quotient | a / b = 3 |
| % | Modulus of two operands | Remainder | a % b = 0 |

DLA
DEVLABS ALLIANCE

## Unary Operators

| Operator | Meaning | Usage |
|---|---|---|
| **++** | Prefix increment | ++x |
| **++** | Postfix Increment | x++ |
| **--** | Prefix decrement | --x |
| **--** | Postfix Decrement | x-- |
| **!** | Logical negation | !x |

## Relational Operators

| Operator | Meaning | Usage Example | Meaning |
|---|---|---|---|
| < | Less than | x < y | Returns true if x is less than y |
| > | Greater than | x > y | Returns true if x is greater than y |
| <= | Less than or equal to | x <= y | Returns true if x is equal to or less than y |
| >= | Greater than or equal to | x >= y | Returns true if x is equal to or greater than y |
| == | Equal to | x == y | Returns true if x is equal to y |
| != | Not equal to | x != y | Returns true if x is not equal to y |

DEVLABS ALLIANCE

## Logical Operators

| Operator | Meaning | Usage | Output |
|---|---|---|---|
| **!** | **NOT** | ! x | Returns false if x is true |
| **&&** | **AND** | x && y | Returns true only if both x and y are true, otherwise false |
| **\|\|** | **OR** | x \|\| y | Returns true even if any one of x or y is true. Returns false only if both x and y are false. |

## Operators Example

Let us assume 4 variables with the values:

first = 6;
second = 8;
third = 11
flag = false;

| Expression | Value |
|---|---|
| **third - ((first+second)/2)** | **3** |
| **first++** | **7** |
| **third >= (first+second)** | **false** |
| **(second<third) && (first>second)** | **false** |
| **! flag** | **true** |

## Case Example

Priya needs help again! This time, she has to generate invoice for Alex who has bought three items from the store in specific quantities, with the prices as below:

| **Items** | **Price** | **Quantity bought** |
|---|---|---|
| Item A | 200 | 3 units |
| Item B | 80 | 5 units |
| Item C | 150 | 2 units |

She needs to give the 15% discount on all items as there is a sale on.
Also, she needs to add 2% service tax in the bill.

Lets help Priya compute the amount.

## Lab 1

1. Write a program to calculate the area and perimeter of a rectangle.

2. Write a program where you evaluate the result of an arithmetic expressions using unary operators.

3. Write a program to print the quotient and remainder of two numbers.

DEVLABS ALLIANCE

## Building Blocks of Java

## Classes

- A class is a group of objects that has common properties.
- It is a template or blueprint to create an object.
- Syntax of a class

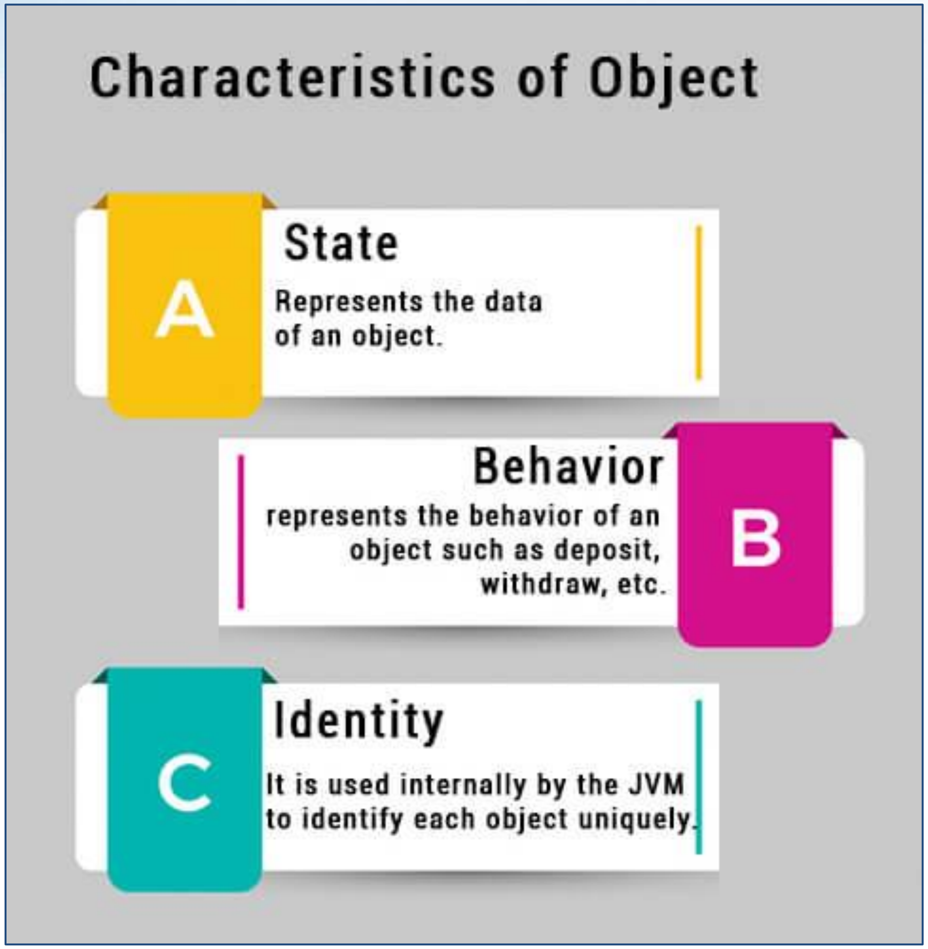
**<<modifiers>> class <<class name>>**
**{**
**// fields and members of the class**
**}**

- A class in java can contain:
  - variables
  - methods
  - constructors
  - blocks

```
class A
{
 int data=50;              //instance variable
 static int m=100;       //static variable
 public static void main(String args[])
 {
       int n=90;              //local variable
 }
} //end of class
```

## Objects

- Everything in Java is an object.
- Instance of a class.
- Containers with state and behavior.
- Actors in the application.
- Models real-world entities.
- In Java, an object is created using the keyword "new".
- Examples: Employee, Product, Animal

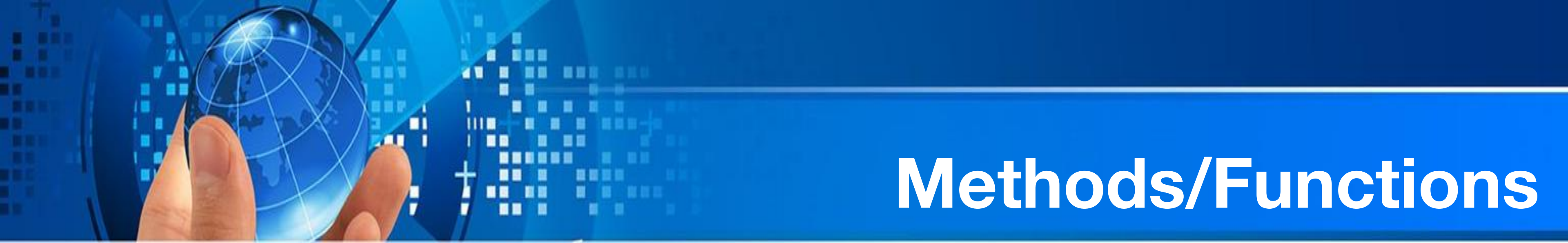


Characteristics of Object

A State
Represents the data of an object.

B Behavior
represents the behavior of an object such as deposit, withdraw, etc.

C Identity
It is used internally by the JVM to identify each object uniquely.



## Methods/Functions

A collection of statements that are grouped together to perform an operation.

Method is like a function which is used to expose the behavior of an object.

Example: When you call the System.out.println() method, for example, the system actually executes several statements in order to display a message on the console.

**Syntax:** **modifier returnType nameOfMethod (formal parameters)**

```
{
        // method body
 }
```

**Example:**

```
public static int getProduct (int a, int b)
{
  return (a*b);
}
```

**public static :** access modifier.
**int**: return type
**getProduct**: name of the method
**a, b**: formal parameters

## Call by Value /Call by Reference

Functions can be invoked in two ways:

- **Call by Value**
- **Call by Reference**

**Call by Value**: calling a method with a parameter as value. Here, argument's value is passed to the parameter. A **copy of the argument is created with the same value**.

**Call by Reference**: calling a method with a parameter as a reference. Through this, the reference (memory address) of the original argument is passed to the parameter.

**Java uses only call by value while passing object reference variables as well.**

**Call by value for primitives :** a copy of the variable with the same value

**Call by value for object references:** a copy of the reference of the original argument is created and passed as value to the method. This copied reference(method parameter) also points to the same underlying object in memory, that the original argument is pointing to.
As both the argument reference and the copied parameter reference, point to the same object in memory, the changes made to the copy are reflected in original also.

## Constructors

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, a constructor is called.

Constructors have no explicit return type and they always have the same name as the class.

All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to certain default values. However, once you define your own constructor, the default constructor is no longer used.

Example of Default Constructor

```
ClassName()

{
 System.out.println("Default called");
}
```

Example of Parameterized Constructor

```
ClassName(int a, String b)
{
 id = i;
 name = b;
 System.out.println("Parameterized Constructor");
}
```

## Packages

- A Java package is a mechanism for organizing Java classes into namespaces.
- A package provides a unique namespace for the types it contains.
- Classes in the same package can access each other's protected members.
- Packages are usually defined using a hierarchical naming pattern
    - Example – java.lang.*, com.uhg.*
- Classes need to be referred using its complete name (package + class name): for example, java.util.Calendar
- Packages can be “imported” to avoid writing package names every time you use * instead of class name.

  import java.util.*;
- Java.lang package is implicitly imported.

DEVLABS ALLIANCE

## Access Modifiers

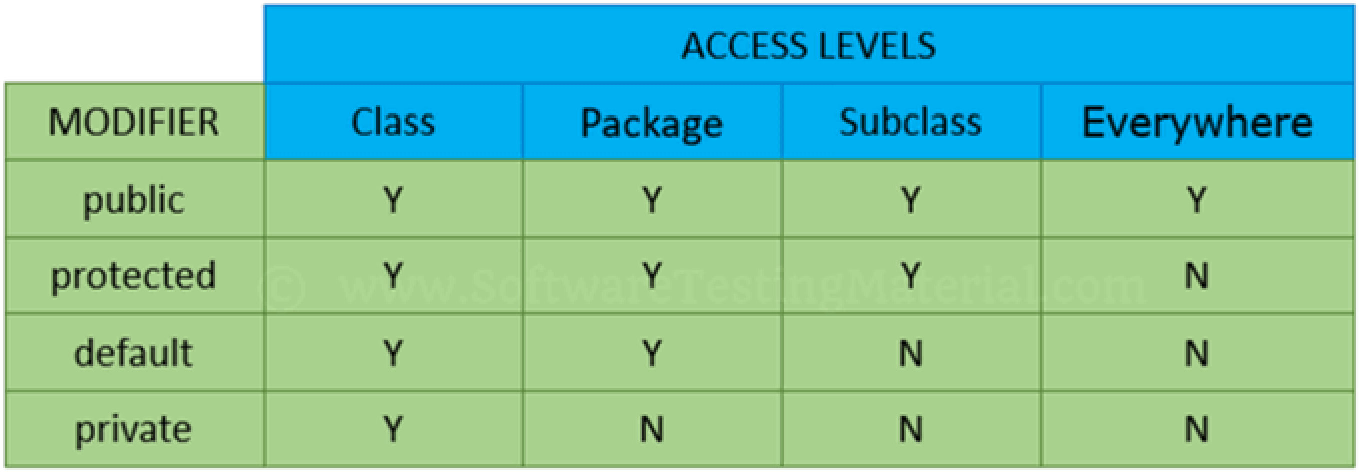
Modifiers are keywords that you add to those definitions to change their meanings.

The modifier precedes the rest of the statement.

The access modifiers in java specify accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

- private
- default
- protected
- public



| | ACCESS LEVELS | | | |
|---|---|---|---|---|
| MODIFIER | Class | Package | Subclass | Everywhere |
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| default | Y | Y | N | N |
| private | Y | N | N | N |

DEVLABS ALLIANCE

## Case Example

Let us define a Product class and instantiate it.

We will set its attributes and define its behaviour using constructors and member functions.

# Lab 2

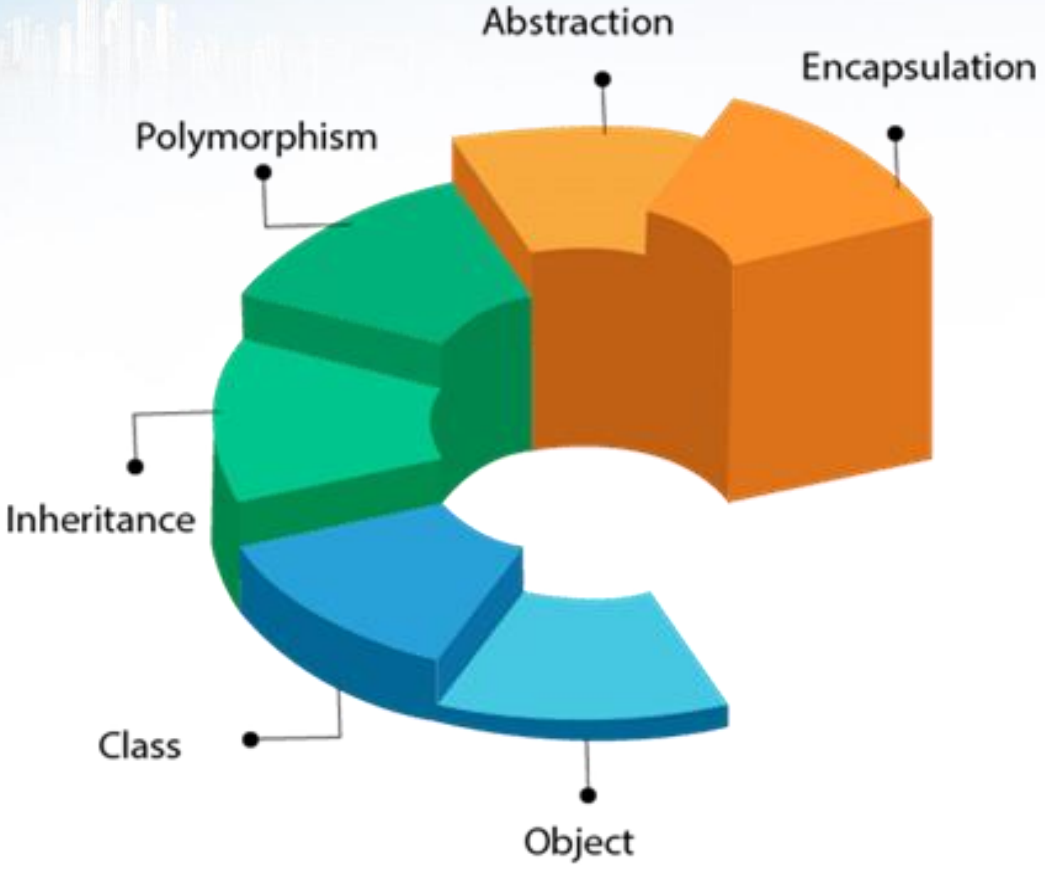
1. **Write a program where you declare all 3 different types of variables, a member function and print the value of each variable.**

2. **Write a program where to swap values of two float variables.**

3. **Define a class and create its objects, define its attributes, constructors and member functions.**

DEVLABS ALLIANCE

# OOPS Concepts



## Encapsulation

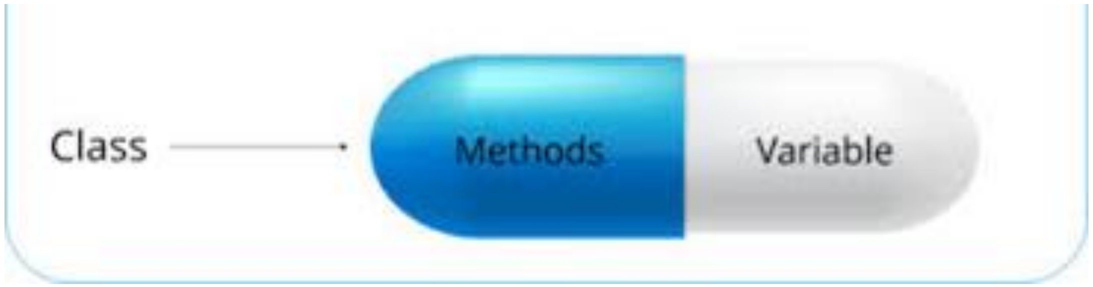
Encapsulation is one of the four fundamental OOPS concepts. The other three are Inheritance, Polymorphism, and Abstraction.

Encapsulation simply means binding object state (fields) and behaviour (methods) together. If you are creating a class, you are doing encapsulation.

To achieve encapsulation in Java -

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variable values.

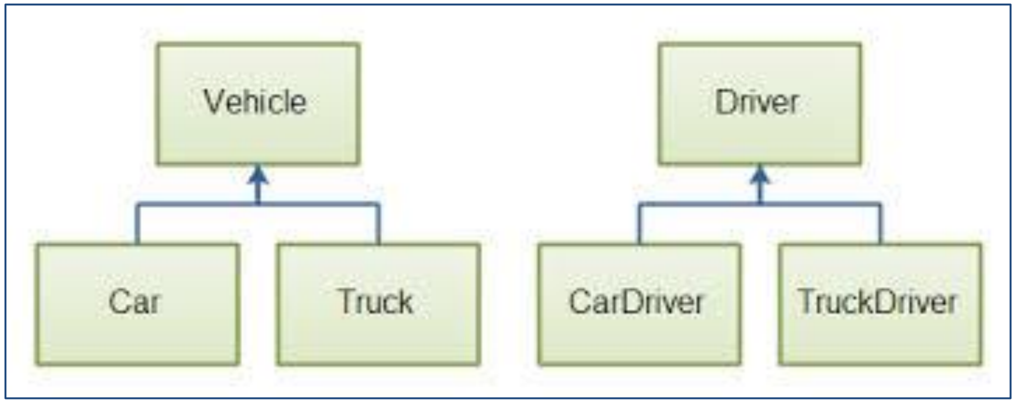
This way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes. That’s why encapsulation is known as **data hiding.**



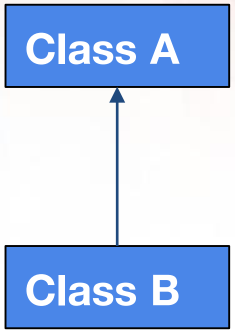
# Inheritance

- A mechanism in which one object acquires all the properties and behaviors of the parent object.
- The extends keyword indicates that you are making a new class that is built on top of an existing class.
- There is a parent-child relationship among the two classes, depicted by IS-A relation.
- A class A that is inherited is called the super class. The class B which extends A, is called a subclass, So, B is A.

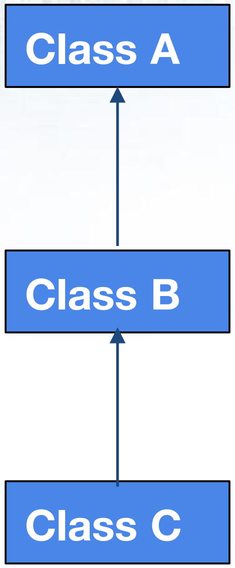
Car and Trucks are Vehicles.
Car Driver and Truck Driver are specific kinds of drivers.



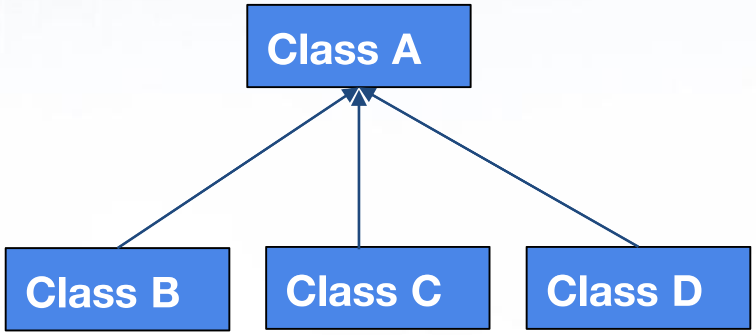
# Inheritance



Single level Inheritance



Multi-level Inheritance



Hierarchical Inheritance

## Abstraction

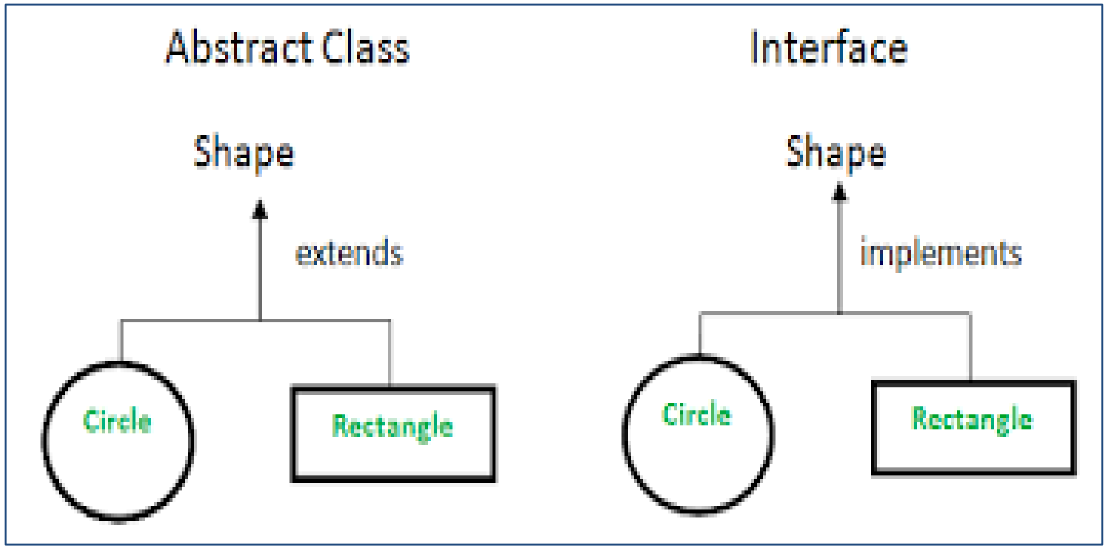
A process of hiding the implementation details and showing only the functionality to the user.

Abstraction lets you focus on what the object does instead of how it does it.

It lets you separate the behaviour and the implementation.

There are two ways to achieve abstraction in Java

- Abstract class
- Interfaces



## Abstract Classes

**SYNTAX:**
```
public abstract class <classname>
 {
//body
 }
```

- Abstract classes are used to achieve 0 to 100% abstraction.
- Must be declared with keyword abstract
- Can have abstract and non-abstract methods
- Abstract methods don't have an implementation (body), they just have method signature
- If a class has an abstract method it should be declared abstract, the vice versa is not true
- Instead of curly braces, an abstract method will have a semicolon (;) at the end
- If a class extends an abstract class, then the class must have to implement all the abstract methods of that class or it has to be declared abstract as well

# Interfaces

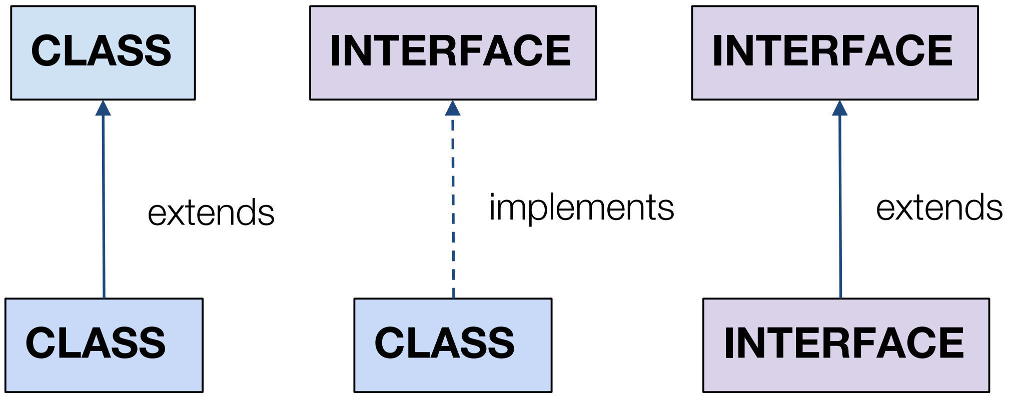
- Another way of achieving abstraction in Java is by using interfaces.

- An interface is a collection of abstract methods, it does not have any concrete methods, unlike an abstract class. But unlike abstract class, an interface provides full abstraction in Java.

- It can have both methods and variables just like a class. However, the methods declared in an interface are abstract by default.

- It enables multiple inheritance and loose coupling. One class can implement multiple interfaces.

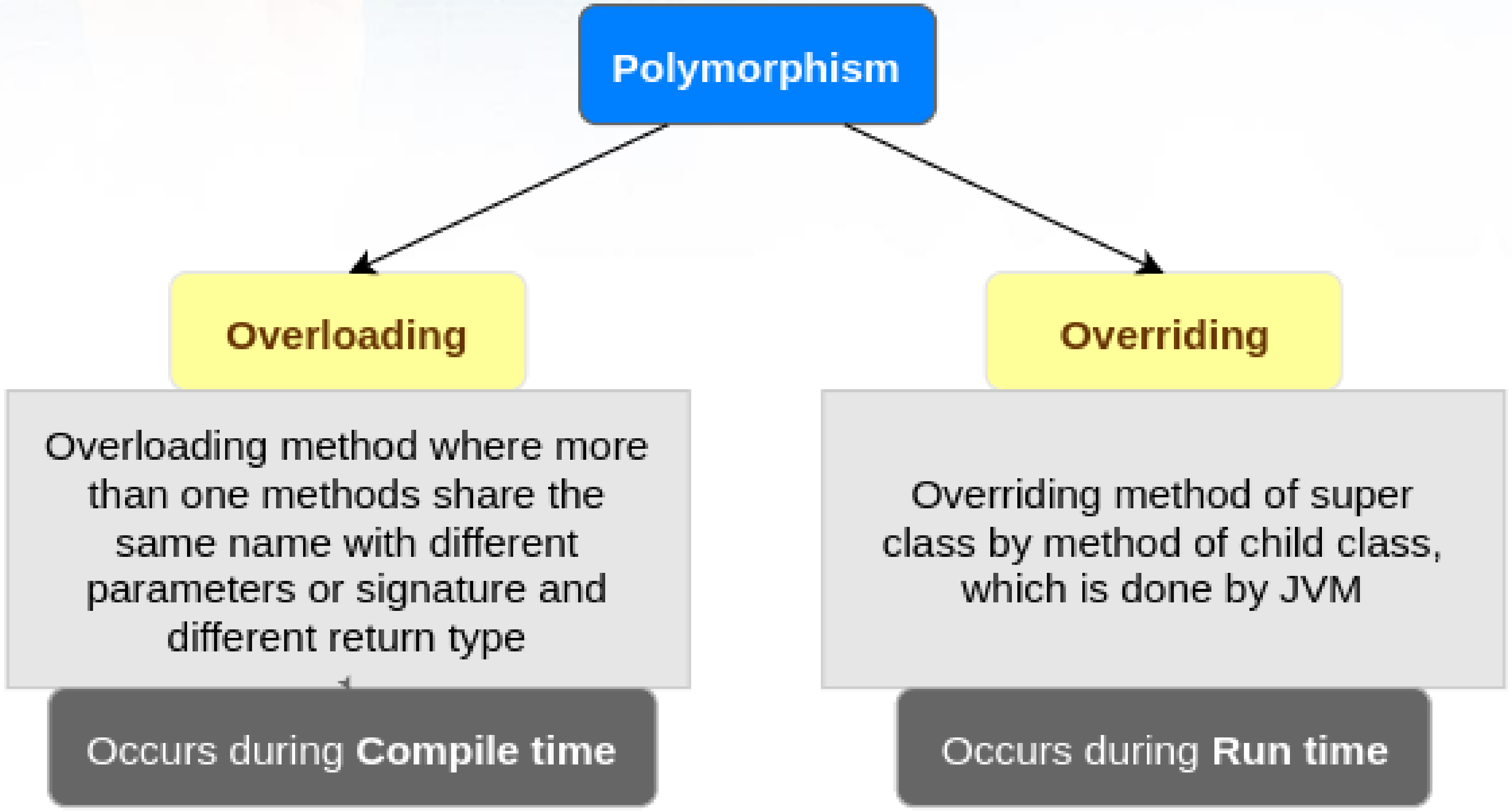
**SYNTAX:**
```
public interface <interfaceName>
{

        //body

}
```



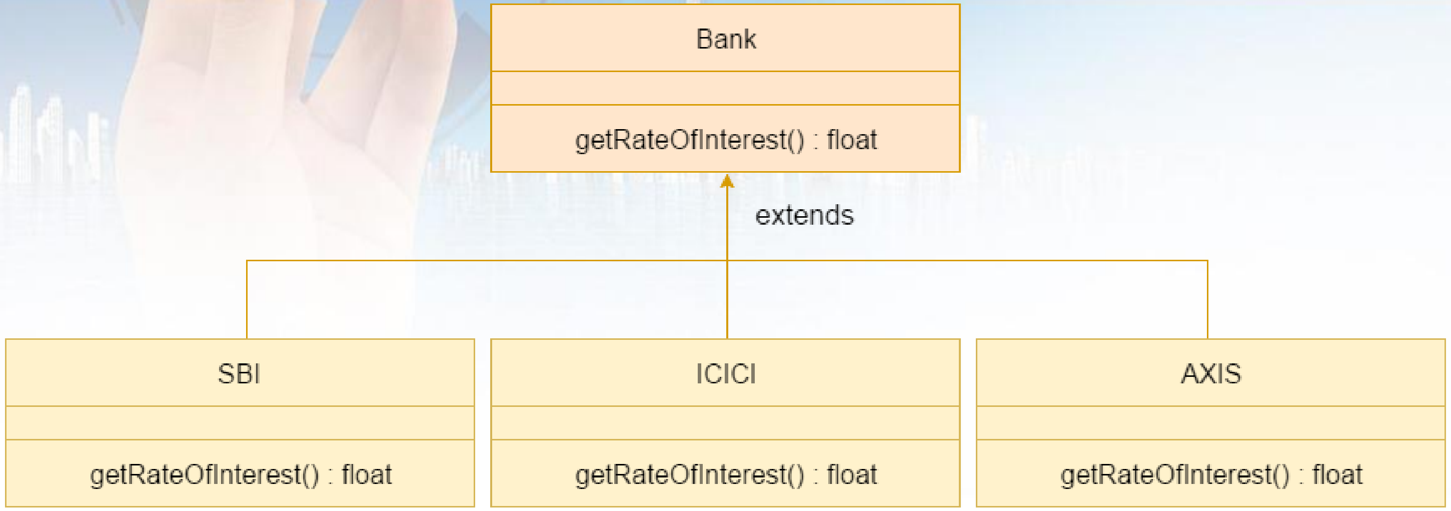
## Polymorphism

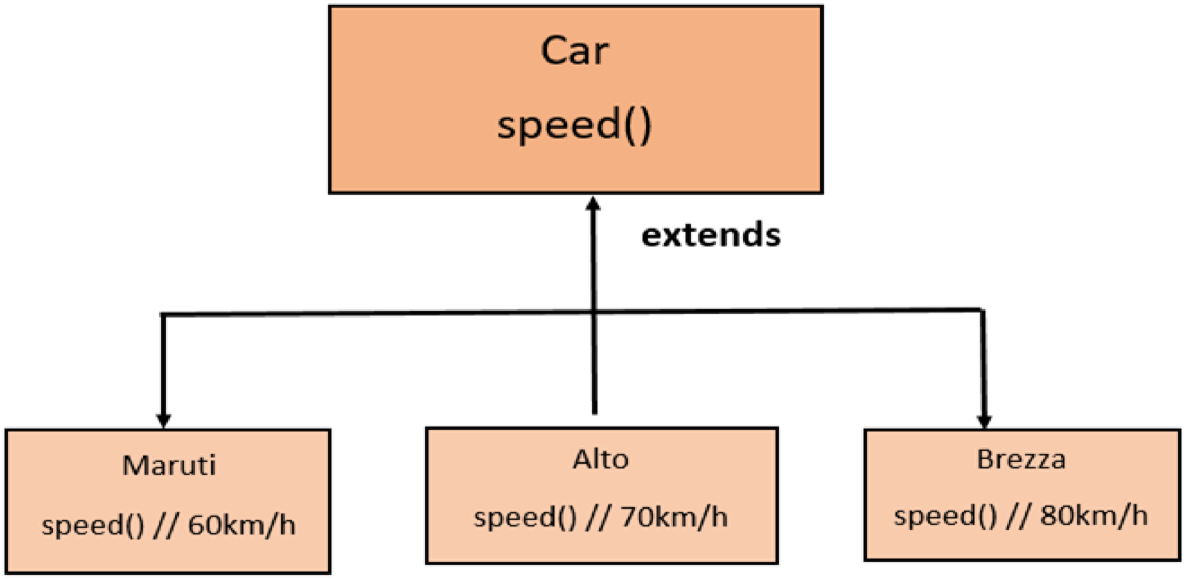
Polymorphism is the ability of an object to take on many forms.

Polymorphism allows us to perform a single action in different ways.


Polymorphism
Overloading
Overriding
Overloading method where more than one methods share the same name with different parameters or signature and different return type
Overriding method of super class by method of child class, which is done by JVM
Occurs during Compile time
Occurs during Run time

# Polymorphism


Bank
getRateOfInterest() : float
extends
SBI
getRateOfInterest() : float
ICICI
getRateOfInterest() : float
AXIS
getRateOfInterest() : float



## Case Example

**Let us define a Mobile object which is a special kind of product with specific features. We will use overloading and overriding to extend the behaviour from the core Product class. (Inheritance)**

DEVLABS ALLIANCE

## Lab 3

1. Write a program where you have an interface Bank with getInterest() and then, two classes which implement the Bank interface and override the getInterest() in each class.

2. Write a program where your class will overload calculatePerimeter() for different shapes, each with different set of parameters, and print the values from each implementation.

## Conditional Statements

## Conditional Statements

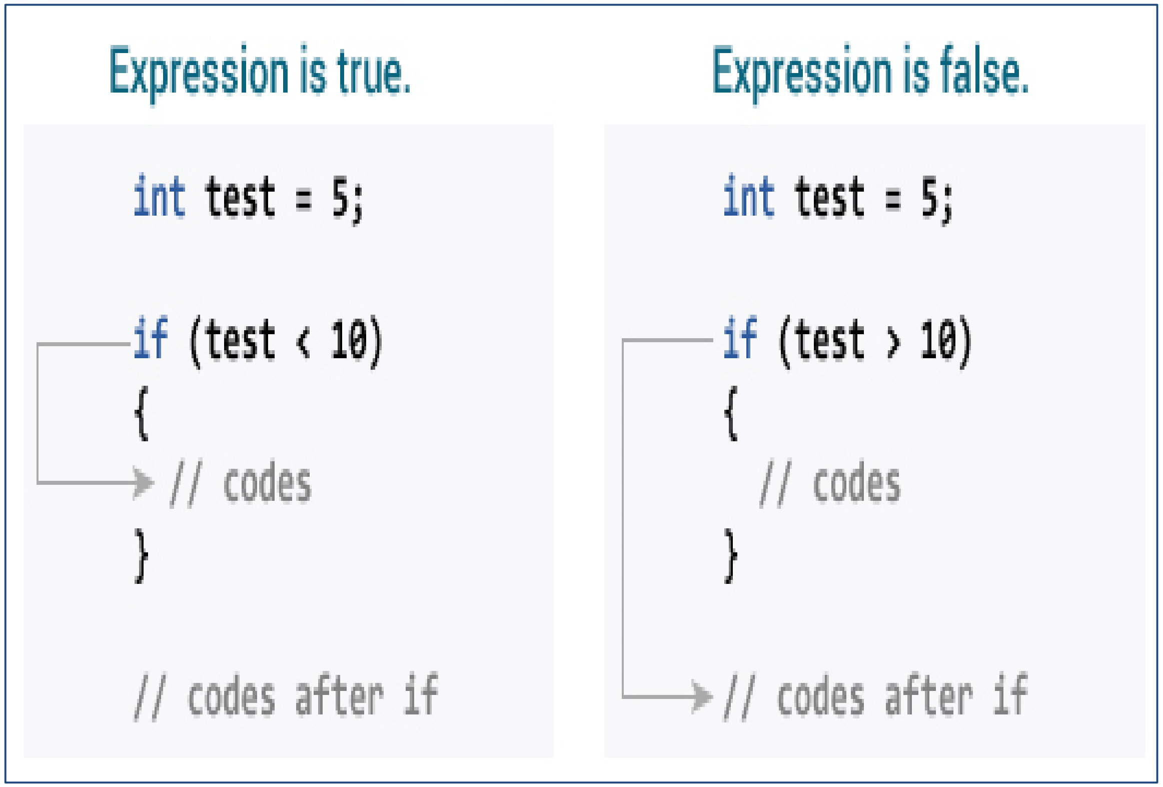
Conditional statements are used to control the flow of execution based on certain conditions.

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

- if
- if-else
- if-else-if
- nested-if
- switch-case

# If Statement

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



Expression is true.

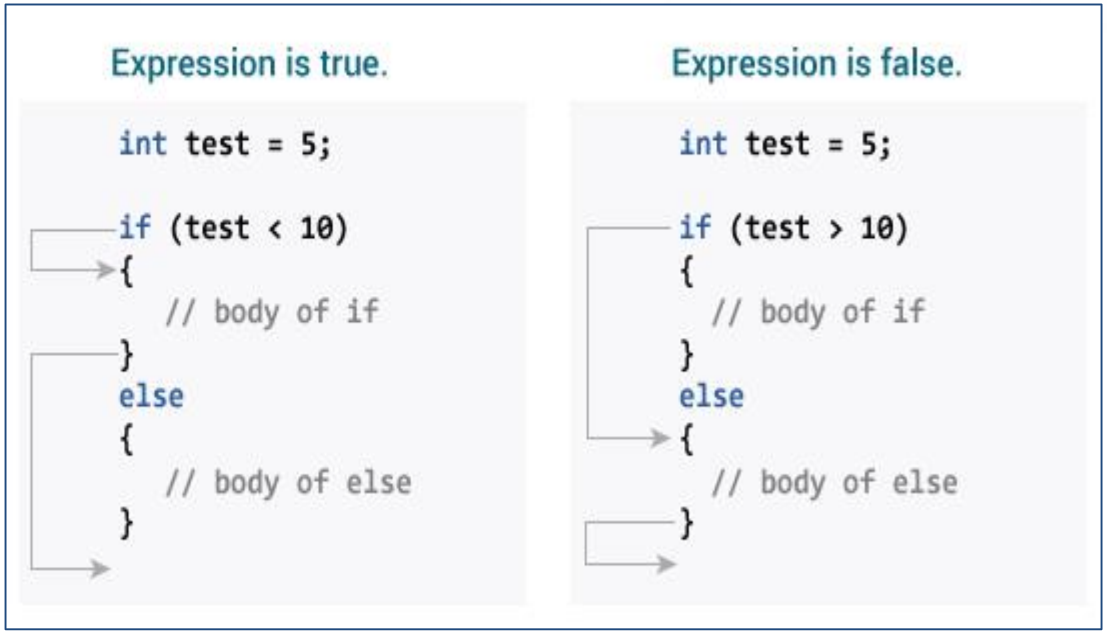
```
int test = 5;

if (test < 10)
{
  // codes
}

// codes after if
```

Expression is false.

```
int test = 5;

if (test > 10)
{
  // codes
}

// codes after if
```

# If-else Statement

```
if (condition)
{
   // Executes this block if condition is true
}
else
{
   // Executes this block if condition is false
}
```



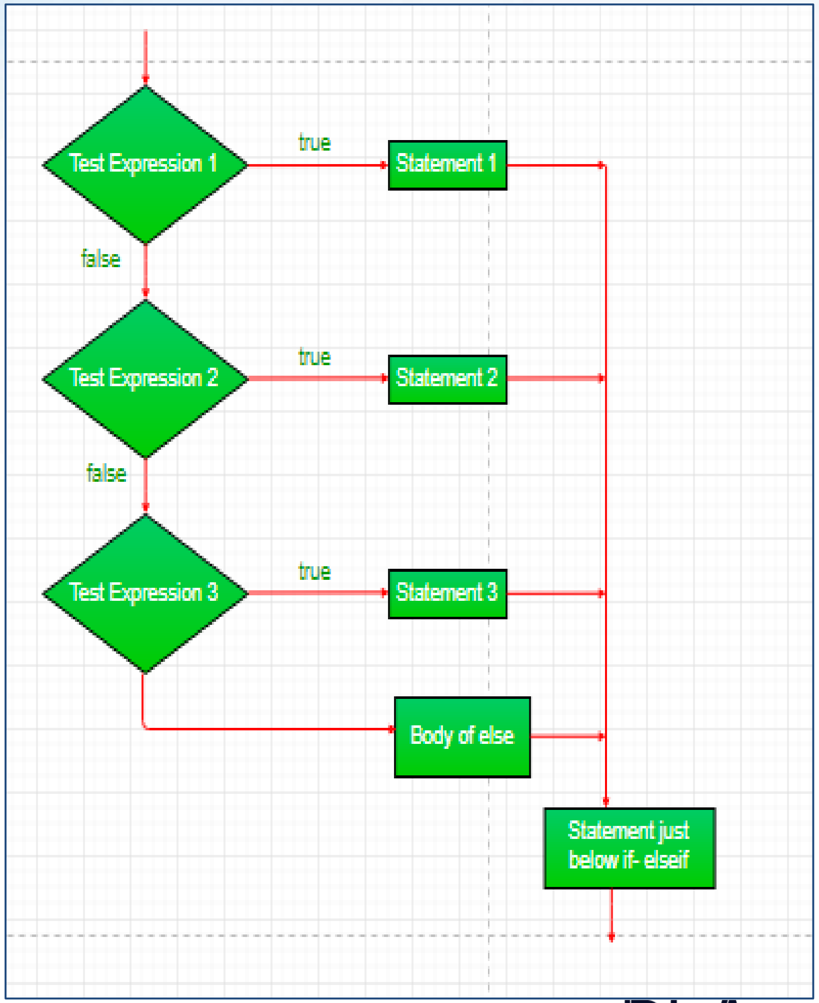
```
int test = 5;

if (test < 10)
{
   // body of if
}
else
{
   // body of else
}
```

```
int test = 5;

if (test > 10)
{
   // body of if
}
else
{
   // body of else
}
```

# If-else if Statement

If-else-if statement is used when we need to check multiple conditions.
We have only one "if" and one "else", however we can have multiple "else if". It is also known as **if else if ladder**.

```
if(condition_1) {
   /*if condition_1 is true execute this*/
   statement(s);
}
else if(condition_2) {
   /* execute this if condition_1 is not met and
    * condition_2 is met
    */
   statement(s);
}
else{
statement(s); //if none of the condition is true
    * then these statements gets executed
}
```

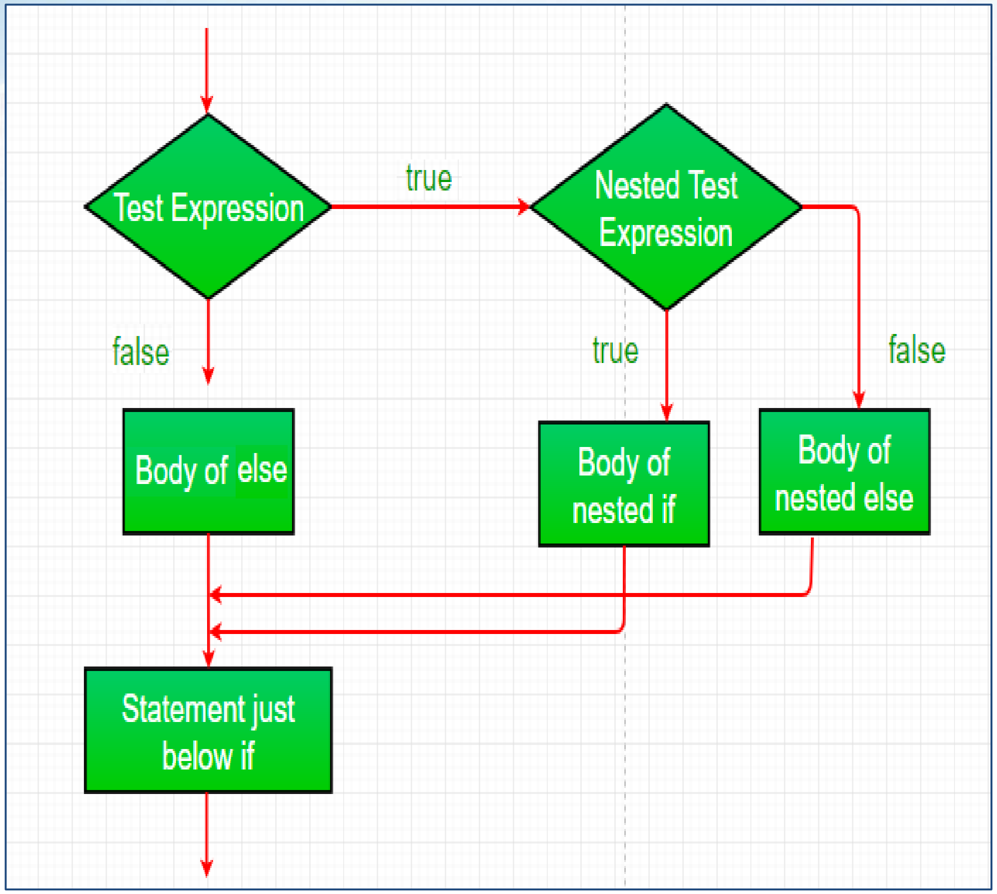


DEVLABS ALLIANCE

# Nested if Statement

A nested if is an if statement that is the target of another if or else.
Nested if statements means an if statement inside an if statement.

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

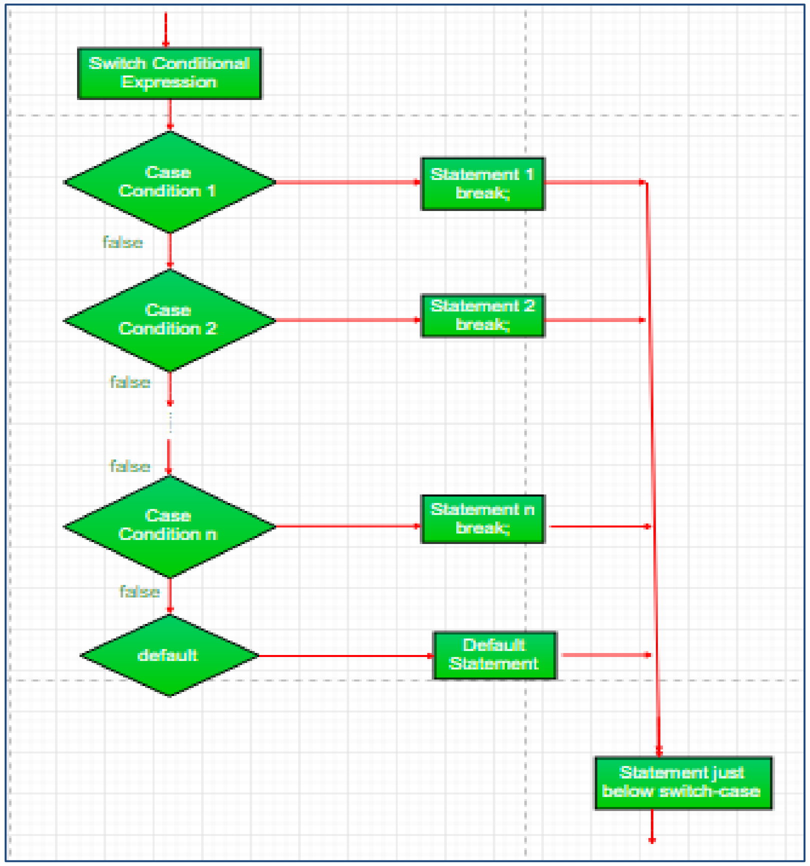


DEVLABS ALLIANCE

# Switch case Statement

- Evaluates the expression and compares with values of each case label.
- Executes all statements of the matching case label.
- The break statements are important because if they are not used, all statements after the matching case label are executed in sequence until the end of switch statement.

```
switch (variable/expression)
{
      case value1:
        // statements
        break;
      case value2:
        // statements
        break;
        .. .. ...
        .. .. ...
      default:
        // statements
  }
```



DEVLABS ALLIANCE

## Lab 4

1. Write a program to print the largest of three numbers using if-else if

1. Write a program to check whether a character is a vowel or consonant using switch case.

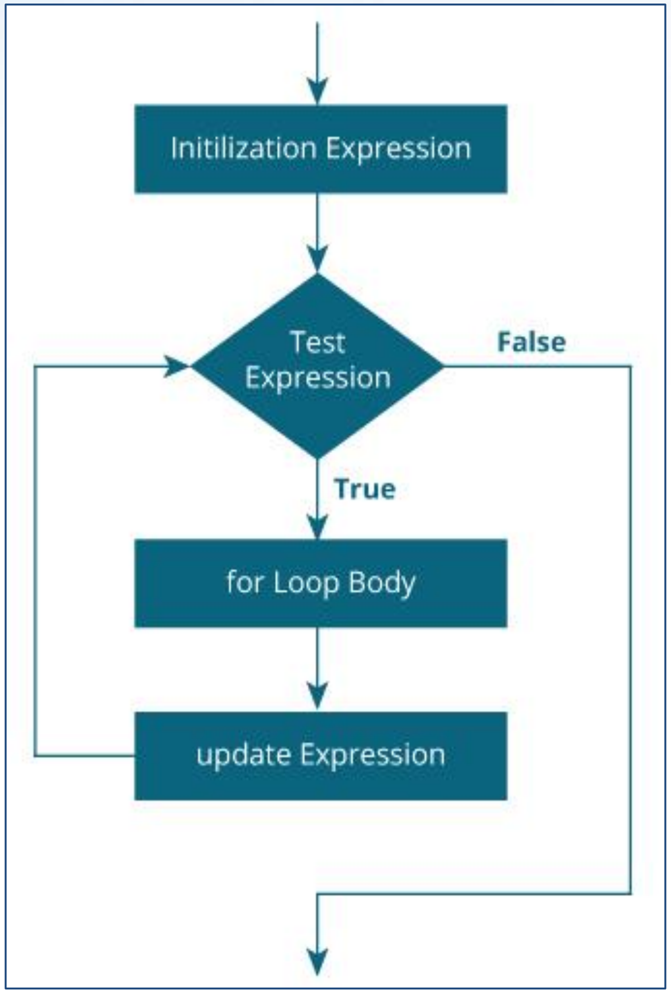
1. Write a program to check whether the no. entered from user is positive or negative.

## Loop Control

# for Loop

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

```
class ForLoopDemo
 {
   public static void main(String args[])
       {
        for(int i=10; i>1; i--)
         {
            System.out.println("The value of i is: "+i);
        }
      }
 }
```

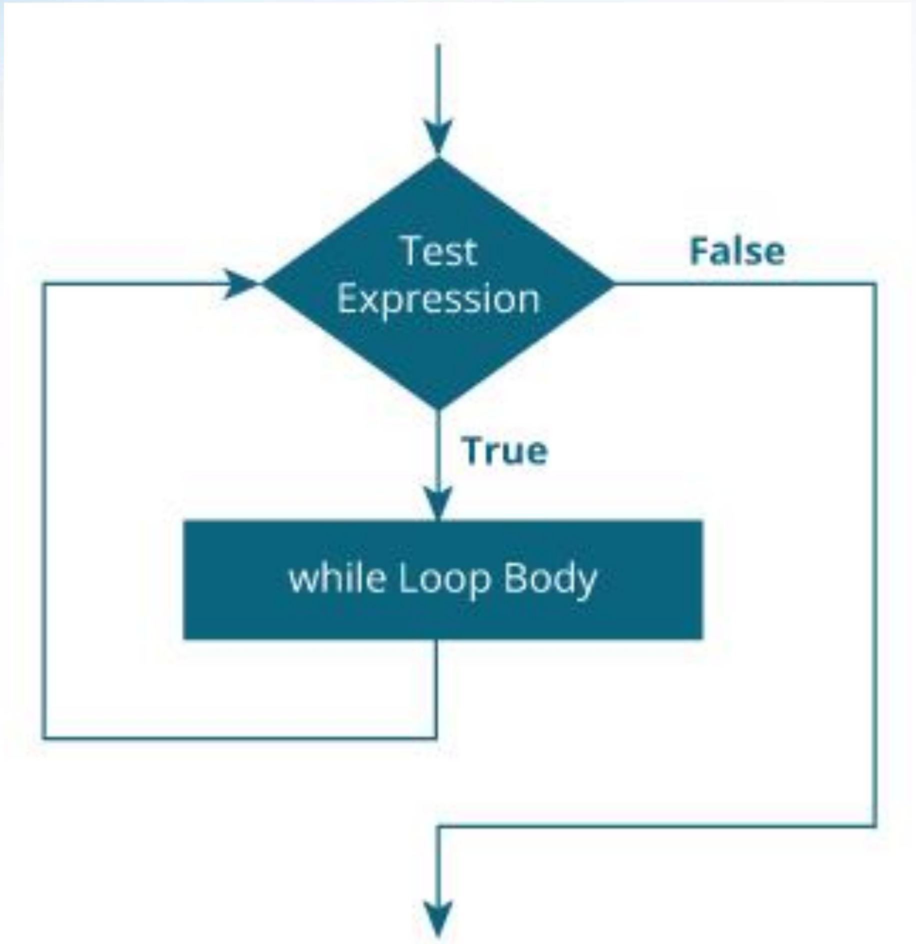


DEVLABS ALLIANCE

## while Loop

```
while (testExpression)
 {
   // codes inside body of while loop
  }
```

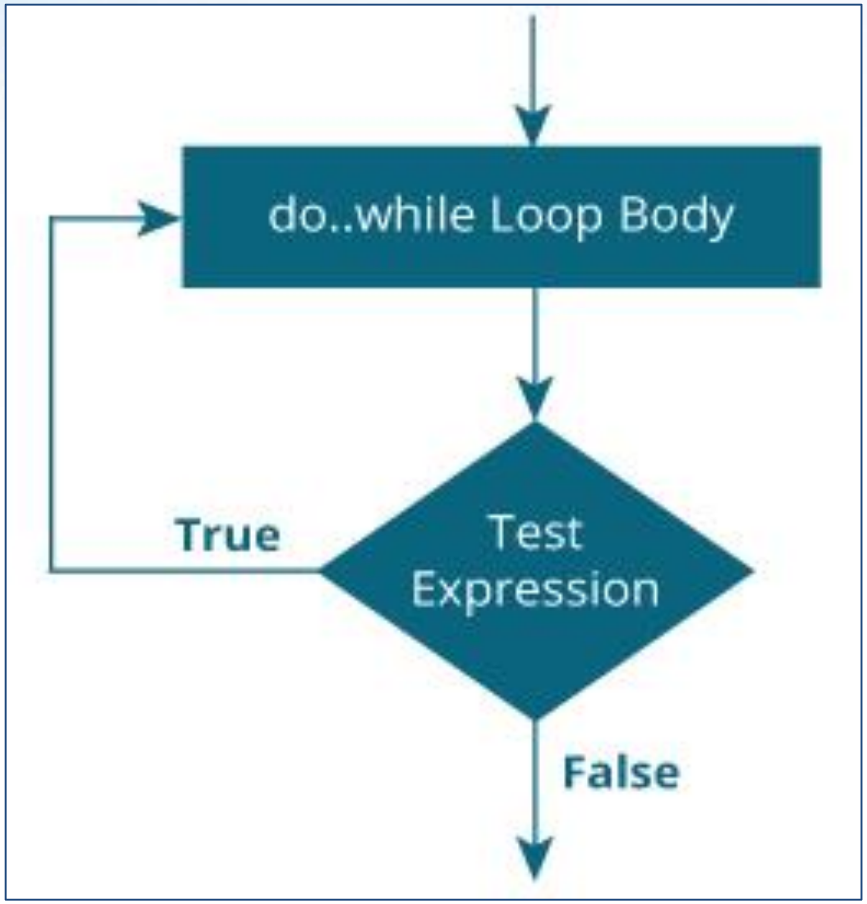
```
class WhileLoopDemo
 {
        public static void main(String args[])
        {
            int i=10;
            while(i>1)
           {
              System.out.println(i);
              i--;
          }
       }
}
```



# do-while Loop

```
do {
    // body of do while loop
} while (testExpression);
```

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



DEVLABS ALLIANCE

## for-each Loop

In Java, there is another form of for loop (in addition to standard for loop) to work with arrays and collection, the enhanced for loop.

```
for(data_type variable : array | collection)
{
//body of for-each loop
}
```

```
class ForEachDemo
{
public static void main(String[] args)
{
    char[] vowels = {'a', 'e', 'i', 'o', 'u'};
    for (char item: vowels)
    {
       System.out.println(item);
    }
  }
}
```

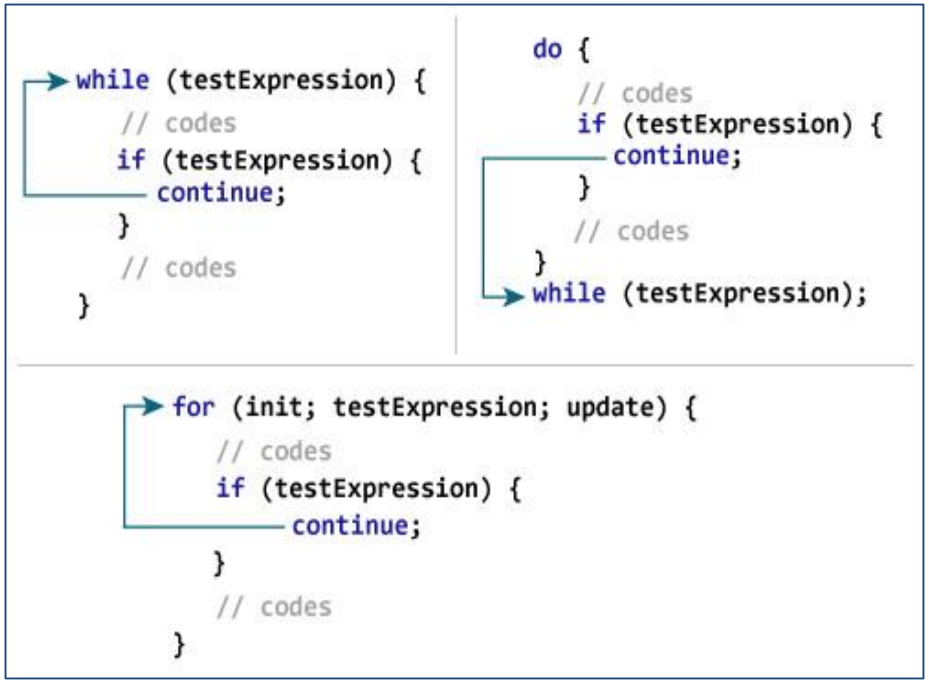
```
for (type var : array)
{
   statements using var;
}
```

is equivalent to:

```
for (int i=0; i<arr.length; i++)
{
   type var = arr[i];
   statements using var;
 }
```

## Loop Control Statements

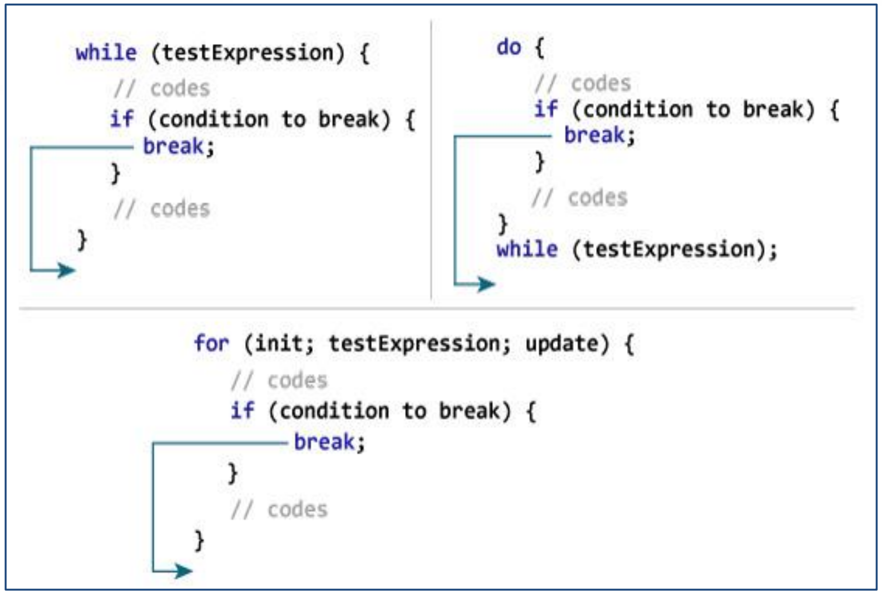
| Statement | Description |
|---|---|
| Continue | Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |
| Break | Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch. |



```
while (testExpression) {
   // codes
   if (testExpression) {
      continue;
   }
   // codes
}
```

```
do {
   // codes
   if (testExpression) {
      continue;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (testExpression) {
      continue;
   }
   // codes
}
```



```
while (testExpression) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

```
do {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

DEVLABS ALLIANCE

## Lab 5

1. Write a program to print the table of 5, first 10 values.

2. Write a program to display prime numbers between two numbers where you get the 2 inputs for the range, from the user.

# Lab 5

A college professor is trying to compute marks and grades for her class of 60 students. There are 5 subjects that she needs to mark them on and then, accordingly print their grades.

Lets see how we can help her using loops and conditional statements.

The five subjects are :

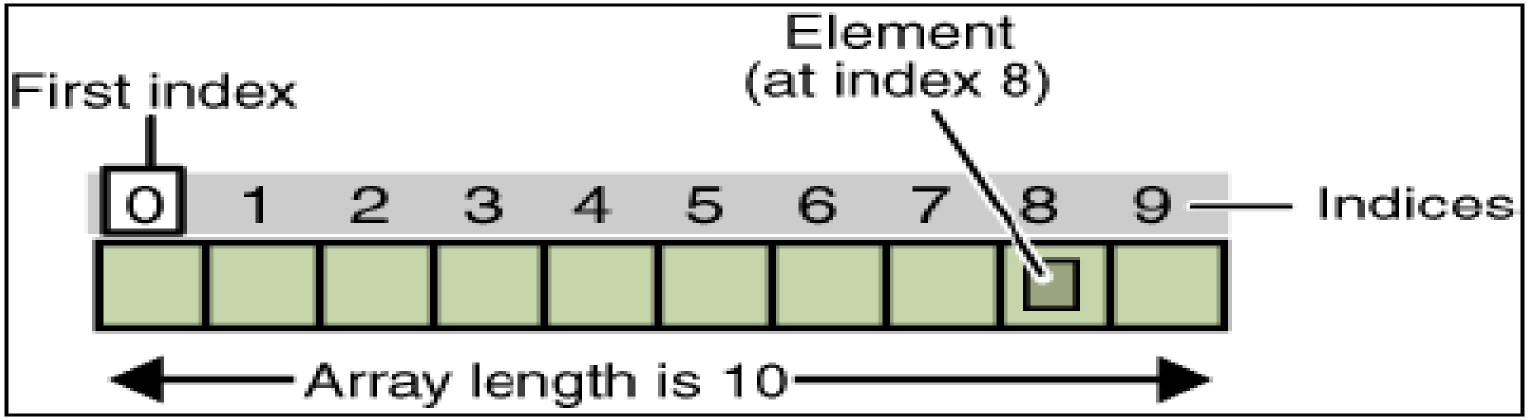
| | | | | |
|---|---|---|---|---|
| **Electronics** | **Engineering Drawing** | **Civil** | **Mechanical** | **Maths** |

| **Total Marks** | **Grades** |
|---|---|
| Below 40 | Poor |
| 40-60 | Average |
| 60 to 75 | Good |
| 75 and above | Very Good |
| 90 and above | Excellent |

## Arrays

# Arrays

- An array is a data structure which stores a fixed number of sequential elements of the same type.
- You can think of array as a collection of variables of the same type.
- 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.



## Types of Arrays

Single-dimensional Array

Multi-dimensional Array

Initialization

```
int d[] = new int[5];
```

```
int table[][]= new int[4][5];
```

| Value | 12 | 24 | 45 | 68 | 90 |
|---|---|---|---|---|---|
| Index | d[0] | d[1] | d[2] | d[3] | d[4] |

Index in Y dimension

Index in X dimension

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 2 | 23 | 12 | 54 | 42 |
| 1 | 6 | 21 | 14 | 53 | 65 |
| 2 | 28 | 1 | 17 | 7 | 34 |
| 3 | 25 | 5 | 8 | 62 | 26 |

## Array Examples

**Examples for single-dimensional arrays**

- int[] values = { 10, 100, 1000 };
- String[] names = {"Joe", "Jane", "Juan"};

**Examples for multi-dimensional arrays**
Implemented as arrays of arrays

- int[][] twoD = new int[64][32];
- String[][] cats = {{ "Caesar", "blue-point" }, { "Heather", "seal-point" }, {"Ted","red-point" }};

**Note:** Number of elements in each row need not be equal

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

## Case Example

Priya needs help again...this time she wishes to store invoices of 20 customers with the amount of each and the invoice number, but she wants to store them together in one place.

Can she use arrays? Lets see.

## Lab 6

1. Write a program to calculate the average of the elements in an array.

2. Write a program to print the elements of a two-dimensional array.

# Assignment 1

1. **Write a program to calculate the factorial of a number using while loop.**

   The factorial of a positive number n is given by:
   factorial of n (n!) = 1 * 2 * 3 * 4 * ... * n

2. **Write a program to print fibonacci series (10 values).**

   A series where the next term is the sum of previous two terms.
   The first two terms of the Fibonacci sequence is 0 followed by 1.
   The Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

3. **Write a program to sort the elements of an array in ascending order.**

4. **Write a program to check current year is leap year or not. User will enter year value.**

5. **Write a program to print first 10 prime numbers.**

6. **Write a program to calculate area of triangle. User will enter the values for base and height of the triangle.**

DEVLABS ALLIANCE

## Assignment 1

**7. Write a program to print the sum of first 20 natural numbers.**

**8. Write a program to reverse the elements of an array where the array size as well as the array values are entered by the user.**

**9. Write a program to print only even numbers till 50.**

**10. Write a program to print this output using two-dimensional array.**

```
Triangle Array
0
00
000
0000
00000
000000
0000000
00000000
000000000
0000000000
```

# Core Java - Table of Contents

## Strings in Java

## What is a Java String?

- String in Java represents a sequence of characters.
- For a single alphabet or letter, we have char data type but for storing data like words, sentences or paragraphs, we need Strings.
- String is a built-in class used to create, process and manipulate Strings in java, it is in the java.lang package.
- String in Java is an object of class java.lang.String.
- String is a reference type, not a primitive type.
- Java String works in the same way, as an array of characters.

## Creating a String object

- There are two ways to create a new String object:
  - ❖ **By String literal:**

    Java String literal is created by using double quotes.

    ```
    String s ="Welcome";
    ```

  - ❖ **By new keyword**

    ```
    String s = new String("Welcome");

    char[] ch = {'j','a','v','a','p','o','i','n','t'};
    String s = new String(ch);
    ```

**Java String Pool refers to a collection of String literals which are stored in heap memory.**
**String Pool helps in saving memory space for Java runtime.**

DEVLABS ALLIANCE

# String Pool

String str = “Hello World”; //String literal

String str1 = “Hello World”; //String literal

String str3 = “Core Java”; //String literal

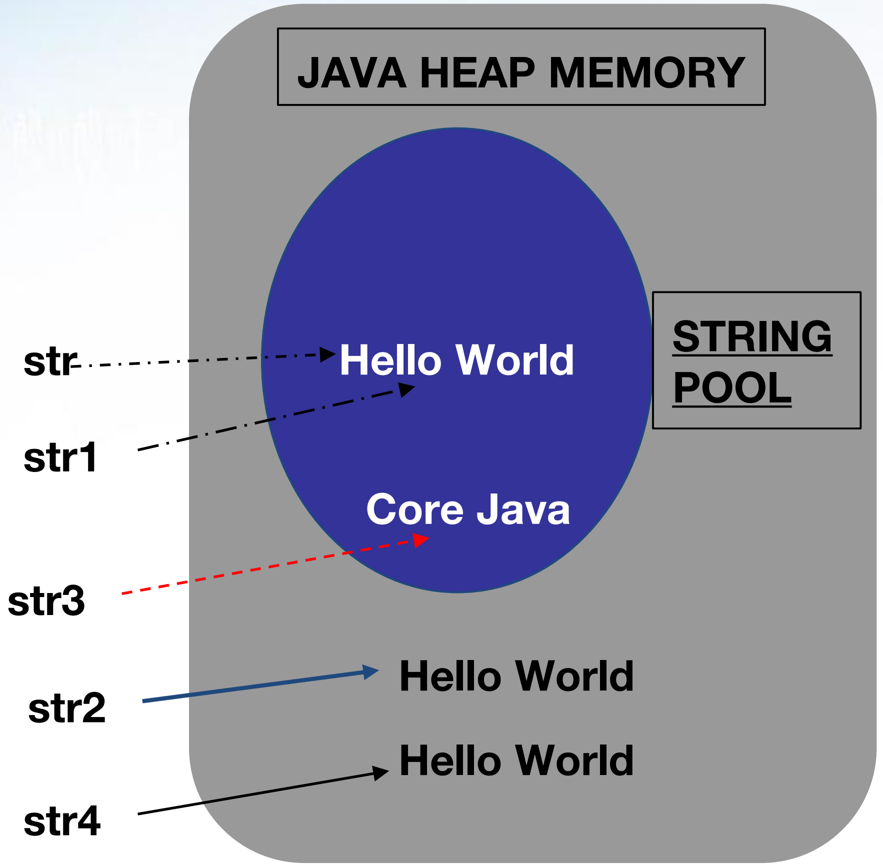
String str2 = new String(“Hello World”);

String str4 = new String(“Hello World”);

String Literals are created only once.

- ❖ They are shared, save memory space.
- ❖ They are constants.

If String is created using new keyword,
a new object is always created in memory,
even if another object already exists in memory, with the same value.



## String is immutable

- String is ***immutable in nature***, the data inside a String object cannot be changed.

- You can never modify a **String** object. If we modify, a new object is always created that contains the modified character sequence.

- You can perform various operations on String: like searching an alphabet or word in sentences, dividing strings.

- You can create new Strings from an existing String:

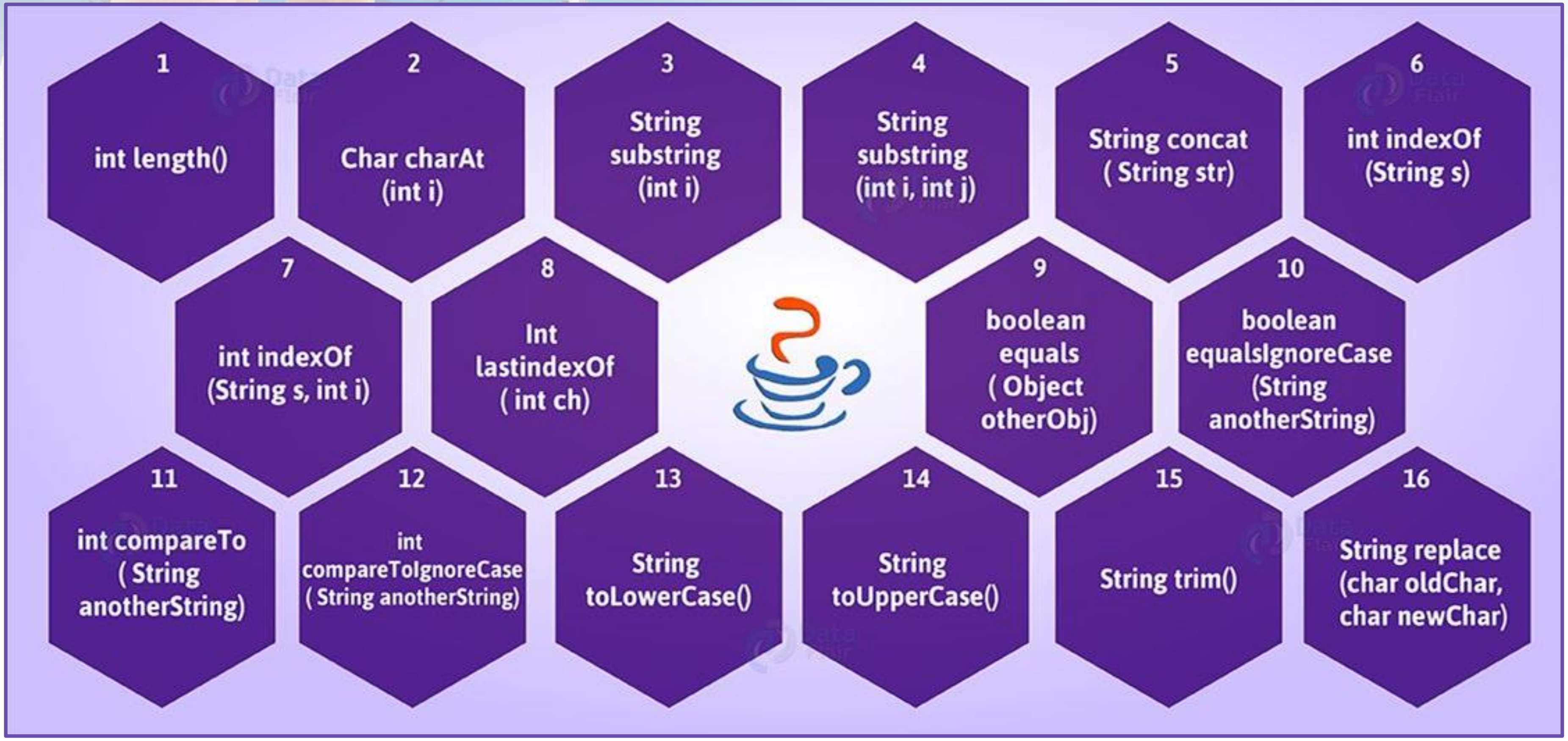
  She sells sea shells on a sea shore.

  ❖ Create Substring : sea shells on a sea shore.
  ❖ Create new Strings : She sells on a sea shore.

- You can't use the relational or equality operators (==, <=, >=) to compare the values stored in strings.

DEVLABS ALLIANCE

## String Methods



1. int length()
2. Char charAt (int i)
3. String substring (int i)
4. String substring (int i, int j)
5. String concat ( String str)
6. int indexOf (String s)
7. int indexOf (String s, int i)
8. Int lastindexOf ( int ch)
9. boolean equals ( Object otherObj)
10. boolean equalsIgnoreCase (String anotherString)
11. int compareTo ( String anotherString)
12. int compareToIgnoreCase ( String anotherString)
13. String toLowerCase()
14. String toUpperCase()
15. String trim()
16. String replace (char oldChar, char newChar)

## String Methods

| | |
|---|---|
| char charAt(int index) | returns char value for the particular index |
| length() | returns string length |
| toLowerCase() | returns formatted string |
| toUpperCase() | returns formatted string with given locale |
| substring(int beginIndex) | returns substring for given begin index |
| substring(int beginIndex, int endIndex) | returns substring for given begin index and end index |
| contains(CharSequence s) | returns true or false after matching the sequence of characters |
| equals(Object another) | checks the equality of String with object |
| isEmpty() | checks if string is empty |
| concat(String str) | concatenates two Strings |
| replace(char old, char new) | replaces all occurrences of specific char value |
| replace(CharSequence old, CharSequence new) | replaces all occurrences of specified CharSequence |
| toCharArray() | convert a String to a  new character array |
| endsWith() | checks if a String ends with the specified suffix |

DLA DEVLABS ALLIANCE

# Lab 7

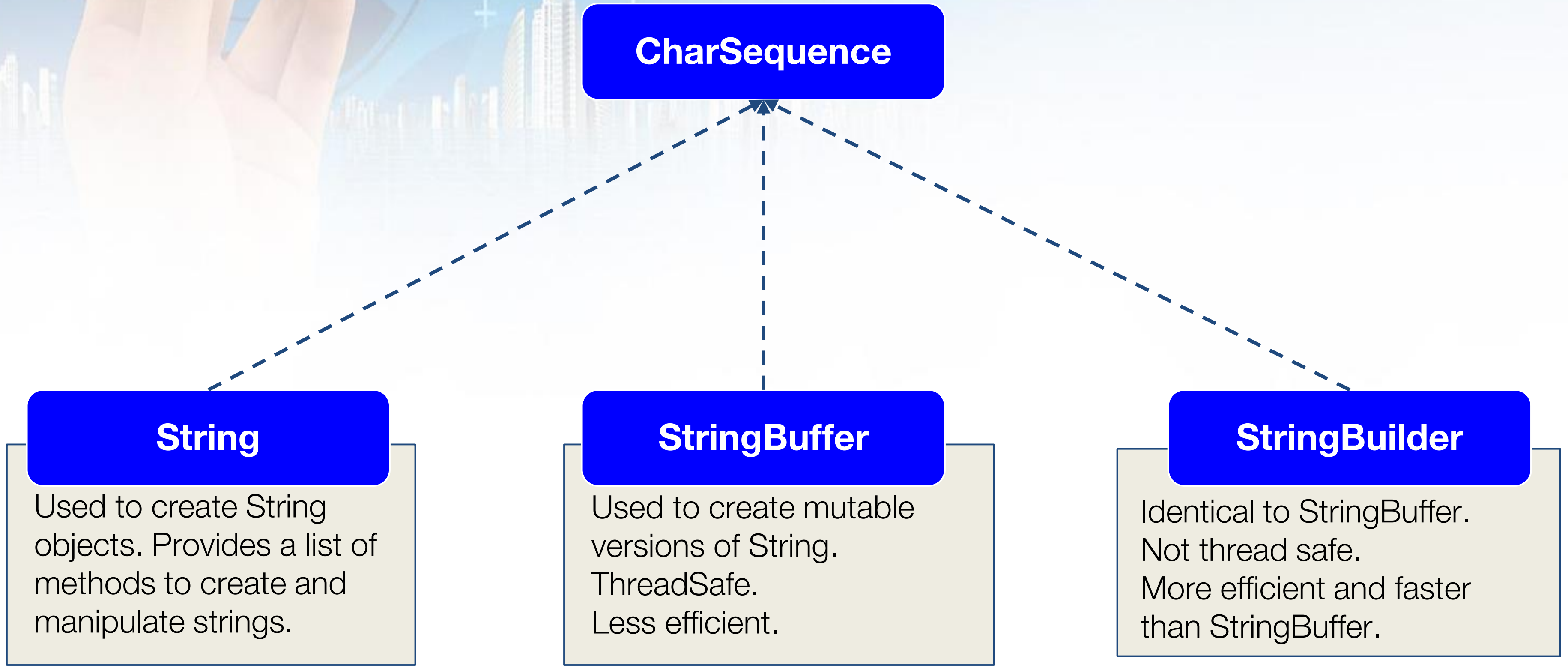
1. **Write a program to sort the Strings in the array: (Bubble Sort)**
   String str[] = “Ciaz, Alto, Swift, Dezire, Brezza”
   Try with user input also, using Scanner.nextLine()

2. **Write a program to find the uppercase and lowercase characters in the string below.**
   String characters = "AbCdefGHijklMNOpqRstUVwxyZ@%*234";
   Use char.isUpperCase(char c) and char.isLowerCase(char c)

3. **Write a program to trim the String : "** She sells sea shells on the sea shore **".**

4. **Write a program to replace character ‘P’ with ‘F’ in the String:**
   “Pan Pun Prank Pit Pat”

## StringBuffer & StringBuilder



**String**

Used to create String objects. Provides a list of methods to create and manipulate strings.

**StringBuffer**

Used to create mutable versions of String.
ThreadSafe.
Less efficient.

**StringBuilder**

Identical to StringBuffer.
Not thread safe.
More efficient and faster than StringBuffer.

## String Exercise 2

**Let us further explore Strings:**

- isEmpty()
- concat()
- StringBuilder : append()
- StringBuffer : append()
- CharSequence can point to String, StringBuilder, StringBuffer
- toString() : returns the String representation of any object.

## Lab 8

1. Write a program to compare two strings and check if they are equal.
2. Write a program to append Strings using StringBuilder and StringBuffer.
3. Perform validations for user input fields in a form, using String API
4. Write a program to concatenate 2 strings.

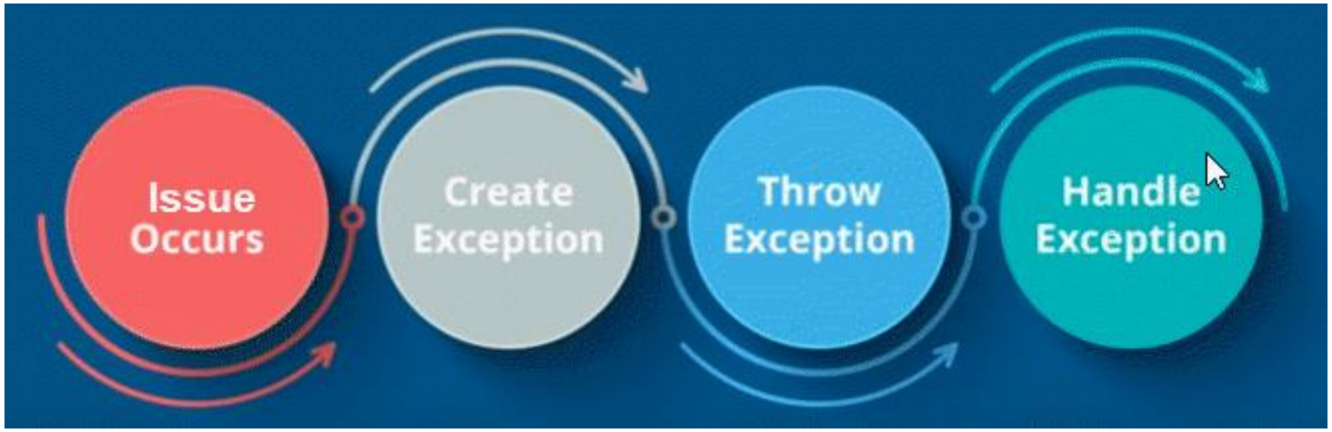
## Exception Handling

# Exceptions

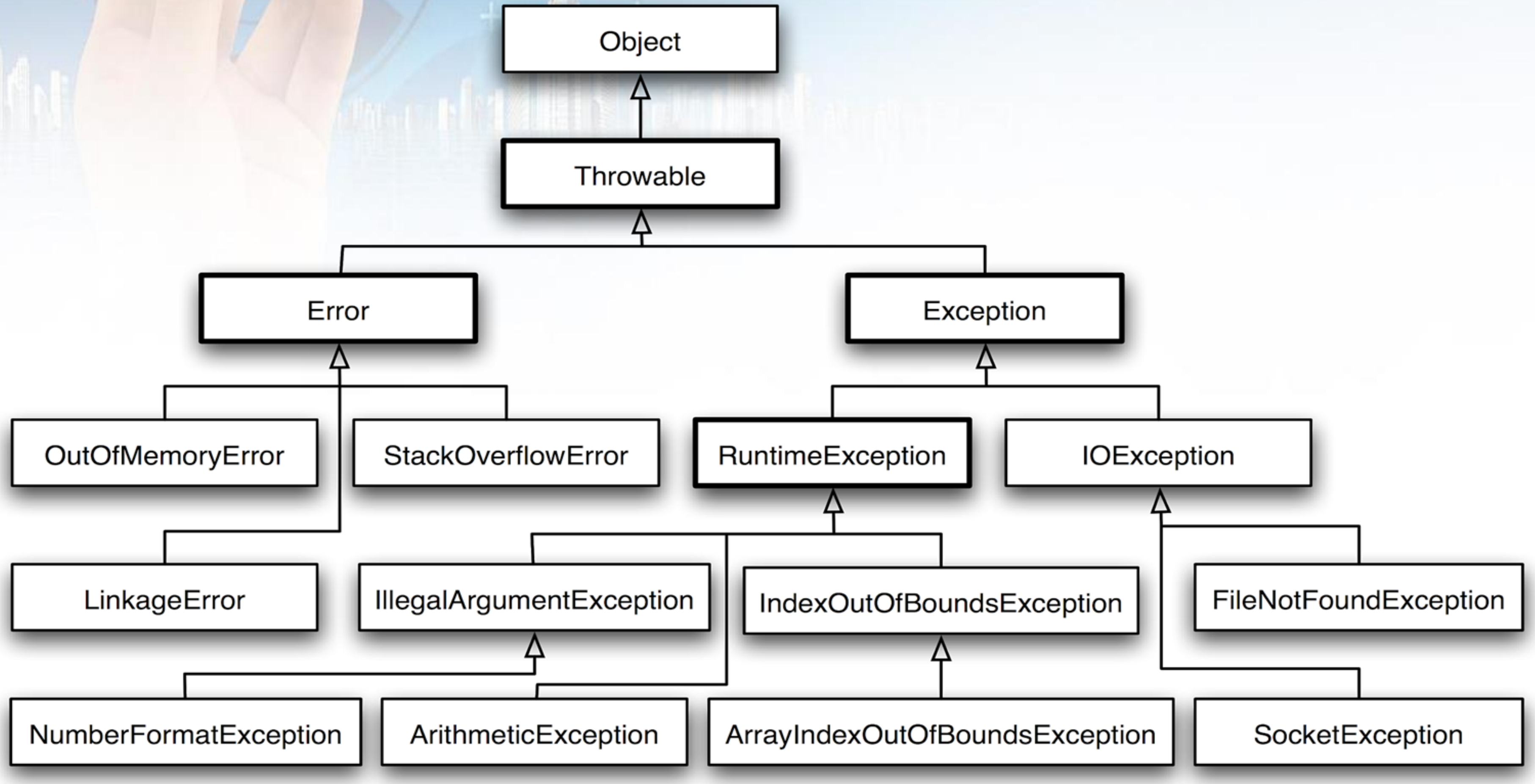
An exception is an unwanted event that disrupts the normal flow of the program.

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Exception handling ensures the normal flow of the application.



# Exception Hierarchy



DEVLABS ALLIANCE

# Errors vs Exceptions

| Error | Exception |
|---|---|
| java.lang.Error type objects | java.lang.Exception objects |
| Impossible to recover from errors | Recovery and handling possible |
| Application prefers to crash rather than handling errors | Exceptions can be handled programmatically, corrective actions can be taken. |
| Unchecked Type | Checked and Unchecked both |
| Happen at runtime | Runtime and Compile time both |
| Caused by the environment that the application is running in | Caused by the application |
| Examples: java.lang.StackOverflowError, java.lang.OutOfMemoryError | Checked: SQLException, IOException<br>Unchecked: ArrayOutOfBoundsException, NullPointerException, ClassCastException |

DEVLABS ALLIANCE

## Exception Types

**EXCEPTIONS**

**CHECKED EXCEPTIONS**

- ➔ All exceptions other than Runtime Exceptions.
- ➔ Checked exceptions are checked by the compiler at the compile time.
- ➔ If these exceptions are not handled/declared in the program, you will get compilation error.
- ➔ SQLException, IOException, ClassNotFoundException etc.

**UNCHECKED EXCEPTIONS**

- ➔ Unchecked exceptions occur at the time of program execution.
- ➔ Also called Runtime Exceptions.
- ➔ Not checked at compile time, hence, compiler will ignore an unchecked exception.
- ➔ For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

## Basic Exception Format

```
class Exception
{
 public static void main(String args[])
 {
            try
            {
          //code that may throw exception
        }
        catch(Exception e)
        {
           // exception-handling code
        }
        finally
        {
          //important code : always executed
        }
```

DEVLABS ALLIANCE

# Exception Keywords

| Keyword | Description |
|---|---|
| **try** | We place the code that throws exception inside the “try” block. A try block must be followed by either a catch block or finally.<br>Can’t use try block alone. |
| **catch** | “Catch” block is used to handle the exception.  It must be preceded by try block and can’t be used alone. Can be followed by finally block later. |
| **finally** | “Finally” block is used to execute the important code of the program. It is always executed whether an exception is thrown or not. |
| **throw** | “Throw” is used to throw an exception. |
| **throws** | “Throws” is used to declare exceptions. It doesn’t throw an exception. It is always used in the method signature. Indicates that the method can throw a |

DEVLABS ALLIANCE

# Throw vs throws

| Throw | Throws |
|---|---|
| Java throw keyword is used to explicitly throw an exception. | Java throws keyword is used to declare an exception. |
| Checked exception cannot be propagated using throw only. | Checked exception can be propagated with throws. |
| Throw is followed by an instance. | Throws is followed by class. |
| Throw is used within the method. | Throws is used with the method signature. |
| You cannot throw multiple exceptions. | You can declare multiple exceptions. Example : public void method() throws IOException,SQLException. |

## Lab 9

1. **Write a program with a try-catch-finally block.**

2. **Write a program that throws ArrayOutOfBoundsException, using a String array.** Use try, catch block.

3. **Write a program to throw parseException while parsing a date entered by user. Convert from dd/mm/yyyy format to yyyy/mm/dd if input is valid.** (exception will occur when an invalid input is entered)

4. **Write a program that throws NumberFormatException. Use try, catch block.** (exception thrown when user inputs string when int is expected)

## Final vs finally vs finalize

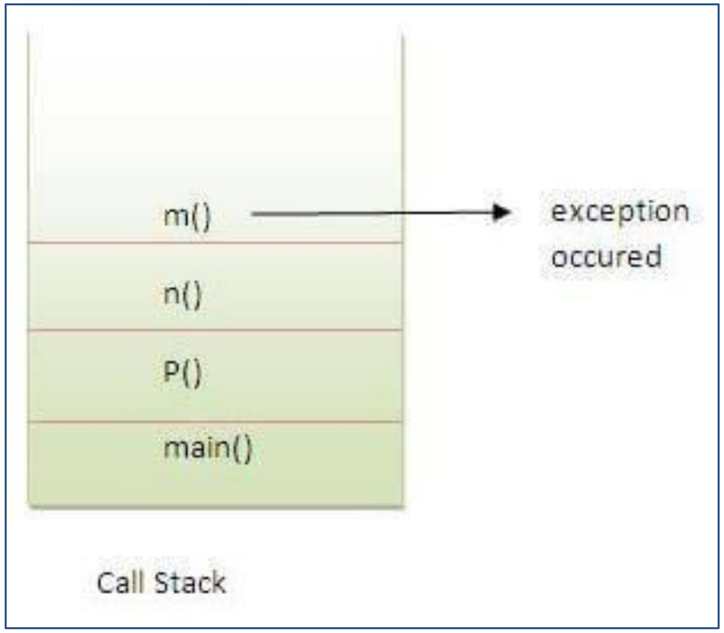
| Final | Finally | Finalize |
|---|---|---|
| Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed. | Finally is used to place important code, it will be executed whether exception is handled or not. | Finalize is used to perform clean up processing just before object is garbage collected. |
| Final is a keyword. | Finally is a block. | Finalize is a method. |

Garbage collection in Java happens automatically during the lifetime of the program, eliminating the need to deallocate memory and thereby avoiding memory leaks. All unreferenced objects' memory is reclaimed.
Explicit call : System.gc()

DEVLABS ALLIANCE

## Exception Propagation

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method,If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.



## Things to Remember

- You can have ***nested try blocks*** also, i.e. one try catch block inside an outer try block.

- Any number of catch methods may follow a try block. (***Multiple catch blocks***)
    - If one catch block is executed, all other catch methods are ignored.

    - The more specific ones should be placed first and then the most generic catch block should be placed at last.

- There are many rules if we talk about ***method overriding with exception handling***.
  **If the superclass method does not declare an exception**
    – If the superclass method does not declare an exception, subclass overridden method cannot declare any checked exception but it can declare unchecked exception.

  **If the superclass method declares an exception**
    – If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

DEVLABS ALLIANCE

## Lab 10

1. **Write a program to handle multiple exceptions. (multiple catch blocks)**

## Assignment 2

1. **Write a program to print the occurrence of each character in the String**
   DevLabs Alliance Training

2. **Write a program to check if a given string is a palindrome or not.**
   Palindrome example : trurt

3. **Write a program to check "brown" is present in the string : A brown fox ran away fast**

4. **Write a program to convert String to a character array and character array to String.**

5. **Write a program to throw NumberFormatException and handle it appropriately with proper message.**
   If you pass invalid input to parseInt(str), this exception will be thrown.

6. **Write a program where a method declares that it throws ArithmeticException.**

## Assignment 2

8. Write a program with nested try blocks.

9. Write a program to re-throw an exception. ( throw inside catch block)

# Core Java - Table of Contents

DEVLABS ALLIANCE

## Collections Framework

## Collections Framework

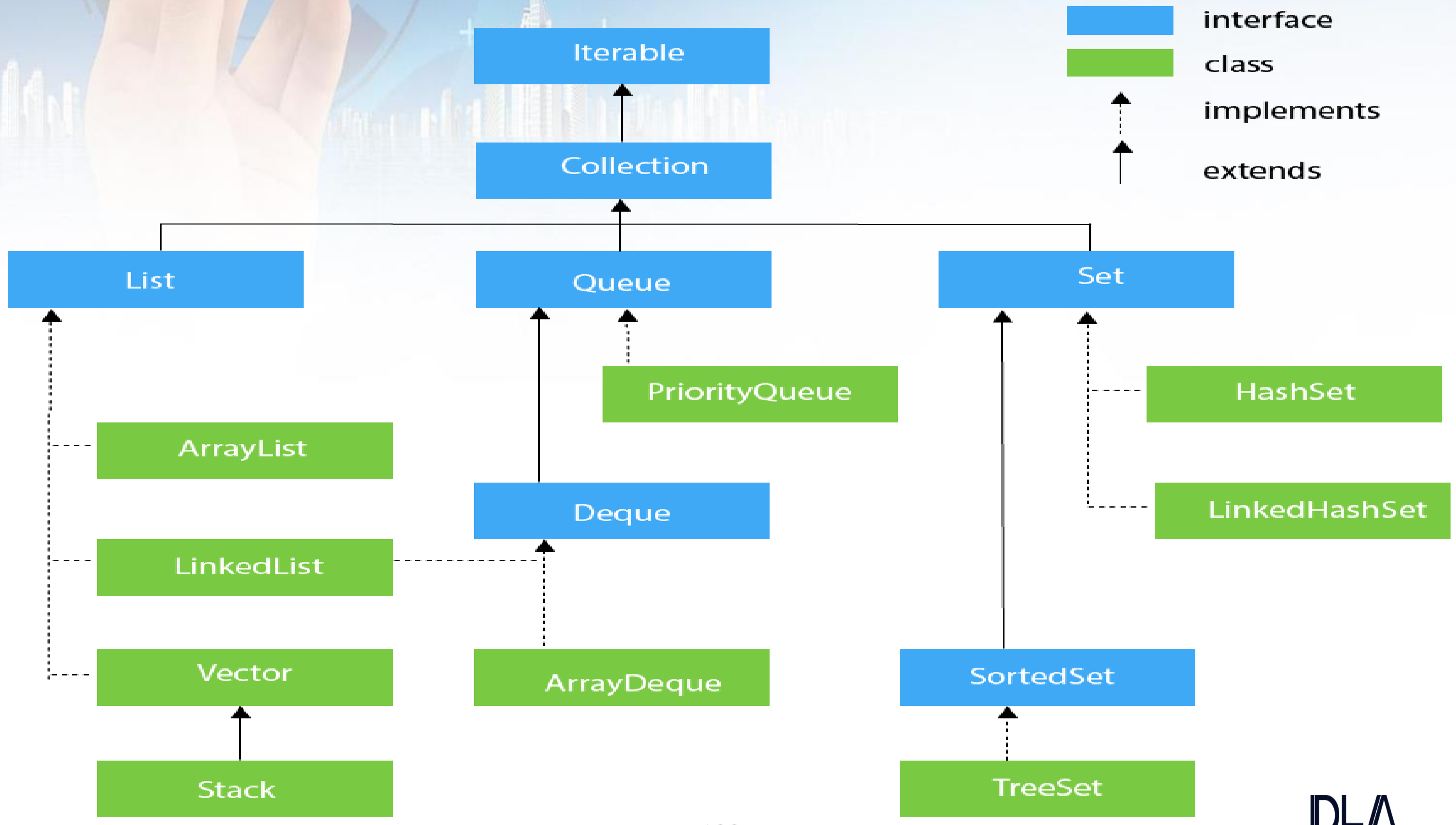
Collections are the containers that groups multiple items in a single unit.

Java Collections is a framework that provides a unified architecture to store and manipulate a group of objects.

Using Java Collections, various operations can be performed on the data like searching, sorting, insertion, manipulation, deletion etc.

Collections Framework provides many interfaces and classes in java.util package, with tons of useful functions, to process the data efficiently.

# Collection Hierarchy



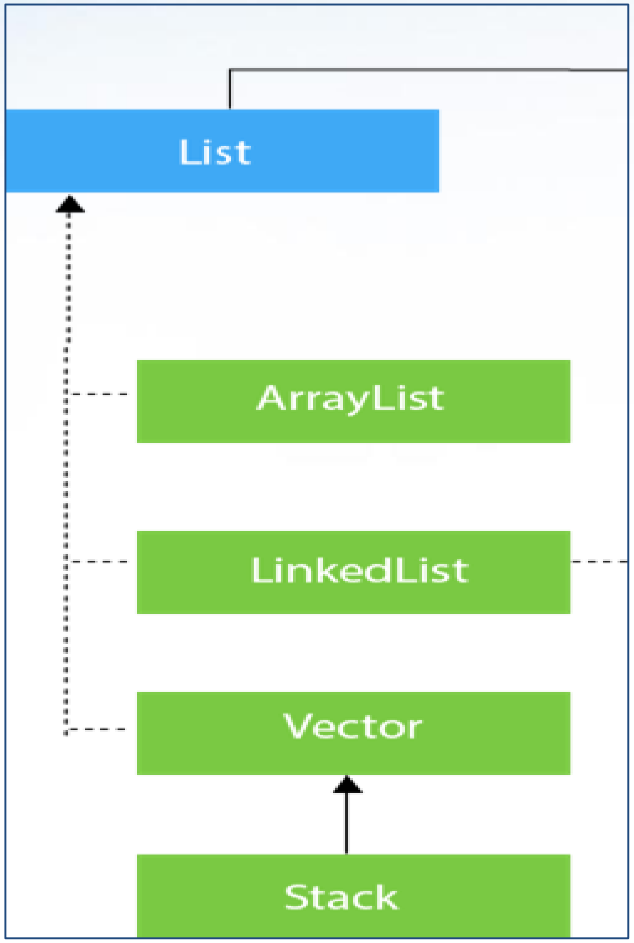
DEVLABS ALLIANCE

# List Interface

- List is a data structure in which we can store the **ordered collection** (a sequence) of objects.
- It ***can have duplicate values.***
- Lists ***can be empty – no elements***.
- List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

➔ List <data-type> list1= **new** ArrayList();
➔ List <data-type> list2 = **new** LinkedList();
➔ List <data-type> list3 = **new** Vector();
➔ List <data-type> list4 = **new** Stack();



# List Interface : Methods

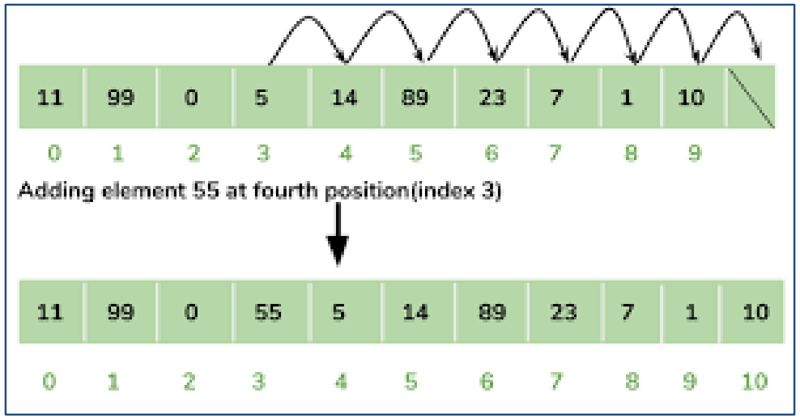
| Method | Description |
|---|---|
| add(int index, E element) | It is used to insert the specified element at the specified position in a list. |
| remove(int index) | It is used to remove the element present at the specified position in the list. |
| size() | It is used to return the number of elements present in the list. |
| subList(int frmIndex, int toIndex) | It is used to fetch all the elements lies within the given range. |
| indexOf(Object o) | It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element. |
| equals(Object o) | It is used to compare the specified object with the elements of a list. |
| get(int index) | It is used to fetch the element from the particular position of the list. |
| isEmpty() | It returns true if the list is empty, otherwise false. |
| contains(Object o) | It returns true if the list contains the specified element |
| toArray() | It is used to return an array containing all of the elements in this list in the correct order. |

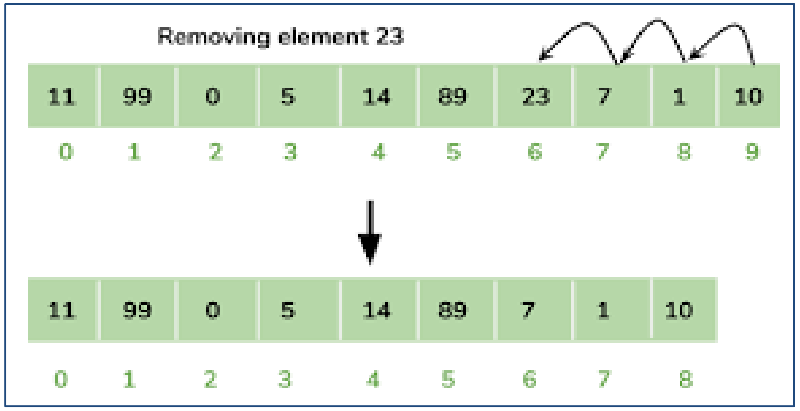
DEVLABS ALLIANCE

# ArrayList

**public class** ArrayList<E> **extends** AbstractList<E> **implements** List<E>, RandomAccess, Cloneable, Serializable

- ArrayList is ***a resizable-array implementation*** of the List interface.
- ArrayList ***can dynamically grow and shrink after addition and removal of elements***.
- It implements all list operations, and permits all elements, including null.
- It ***can contain duplicate elements***.
- It ***maintains the insertion order***.
- Allows ***random access*** through indices.
- Manipulation is slow as a lot of shifting needs to occur if any element is removed.





# Array vs ArrayList

| Array | ArrayList |
|---|---|
| Array is a fixed length data structure whereas. We cannot change length of array once created in Java. | ArrayList is a variable length Collection class. ArrayList size can be changed. |
| Array can contain both primitives and objects in Java. | We cannot store primitives in ArrayList, it can only store objects. |

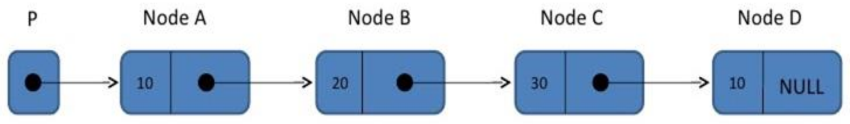
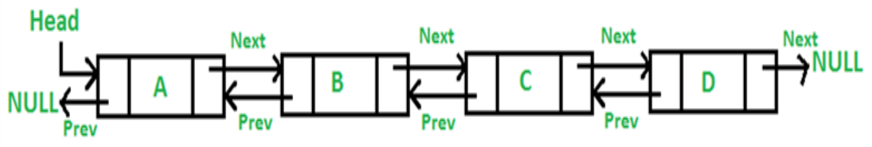
DEVLABS ALLIANCE

# LinkedList

**public class** LinkedList<E> **extends** AbstractSequentialList<E> **implements** List<E>, Deque<E>, Cloneable, Serializable

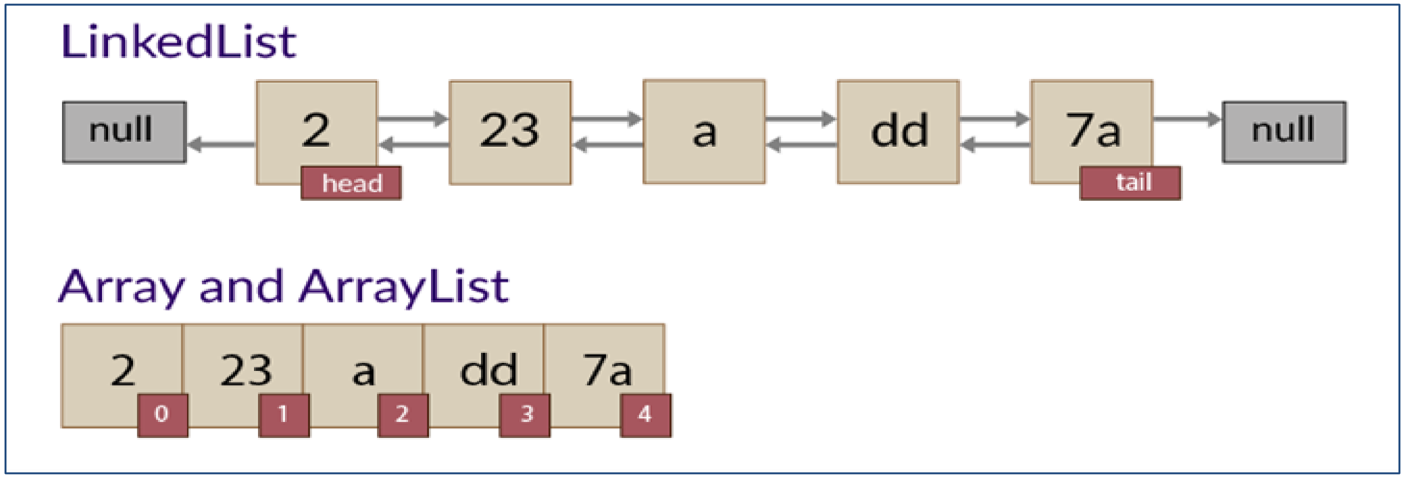
LinkedList is a list implementation where the elements are linked with each other using pointers.

- It can contain duplicate elements.
- It also maintains insertion order.
- It is non-synchronized.
- Manipulation is fast because no shifting needs to occur.

| **Singly Linked List** | **Doubly linked list** |
|---|---|
| ● Content<br>● Pointer to the Next Node | ● Pointer to the previous node<br>● Content of the element<br>● pointer to the next node |
| We can add or remove elements from one end. | We can add or remove elements from both sides. |
|  P<br>Node A<br>Node B<br>Node C<br>Node D<br>10<br>20<br>30<br>10<br>NULL |  Head<br>NULL<br>Prev<br>A<br>Next<br>Prev<br>B<br>Next<br>Prev<br>C<br>Next<br>Prev<br>D<br>Next<br>NULL |

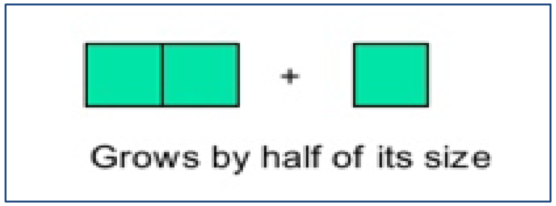
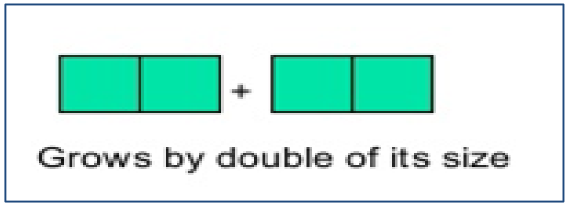
# ArrayList vs

| ArrayList | LinkedList |
|---|---|
| Implements list as an array. | Implements list as a doubly-linked list with a header node. |
| Provides random access to all elements. | No random access, need to traverse to access the ith element. |
| Insertion and removal required shifting of subsequent elements. | Insertion and removal done by rearranging the links - no shifting. |
| Needs to be resized when out of space. | Nodes are allocated and removed as necessary. |



# ArrayList vs Vector

ArrayList and Vector both implement the List interface and maintain insertion order.

| ArrayList | Vector |
|---|---|
| Not synchronized or thread-safe. | Synchronized and thread-safe. |
| Grows by 50% if the number of elements exceeds from tis capacity. | Vector grows by 100%, i.e. doubles itself if the number of elements exceeds from tis capacity. |
| High performance. | Slow performance. |
| Used in single-user application. | Used in multi-user application. |
|  |  |

## Lab 11

Write Java programs to:

1. Create an ArrayList and insert and retrieve value at a specified index.
2. Check whether a particular element exists in the list (Search). Print the index too.
3. Remove a particular element in ArrayList.
4. Set and get values from a Linked List.

## List Iterator Interface

ListIterator Interface is used to traverse the elements in backward and forward direction.

```
public interface ListIterator<E> extends Iterator<E>
```

| Method | Description |
|---|---|
| void add(E e) | This method inserts the specified element into the list. |
| boolean hasNext() | This method returns true if the list iterator has more elements while traversing the list in the forward direction. |
| E next() | This method returns the next element in the list and advances the cursor position. |
| boolean hasPrevious() | This method returns true if this list iterator has more elements while traversing the list in the reverse direction. |
| E previous() | This method returns the previous element in the list and moves the cursor position backward. |
| void remove() | This method removes the last element from the list that was returned by next() or previous() methods |
| void set(E e) | This method replaces the last element returned by next() or previous() methods with the specified element. |

## Collections Class

**public class** Collections **extends** Object

Java collection class is used exclusively with static methods that operate on or return collections.

| Method | Description |
|---|---|
| addAll() | It is used to adds all of the specified elements to the specified collection. |
| list() | It is used to get an arraylist containing the elements returned by the specified enumeration in the order in which they are returned by the enumeration. |
| max() | It is used to get the maximum value of the given collection, according to the natural ordering of its elements. |
| min() | It is used to get the minimum value of the given collection, according to the natural ordering of its elements. |
| copy() | It is used to copy all the elements from one list into another list. |
| sort() | It is used to sort the elements presents in the specified list of collection in ascending order. |

# Lab 12

Write Java programs to:

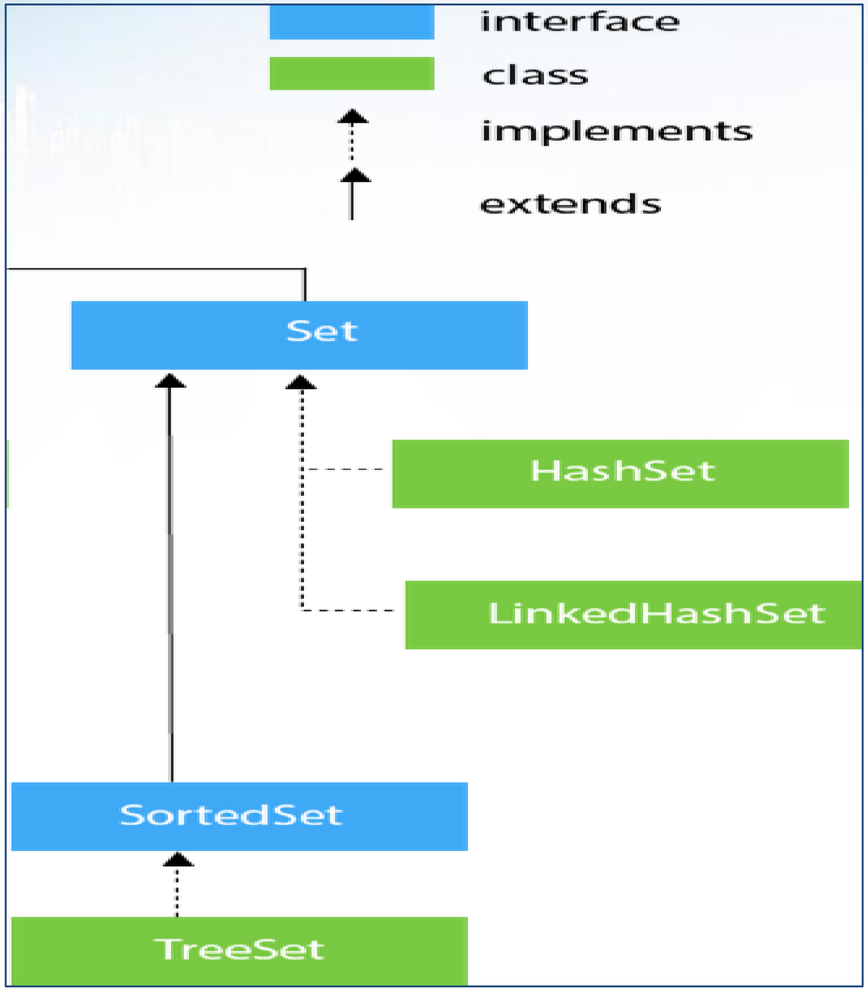
1. Join two lists using addAll() from List

2. Add books to a List and then, read the list of books using ListIterator, in both backward and forward directions.

3. Find the max and min of a String list using Collections class. Also, sort the list and add more Strings using addAll() from Collections class.

# Set Interface

- Set is a collection that contains **no duplicate elements.**
- Sets contain no pair of elements e1 and e2 such that e1.equals(e2), and has at most one null element.
- As implied by its name, this interface models the mathematical *set* abstraction.



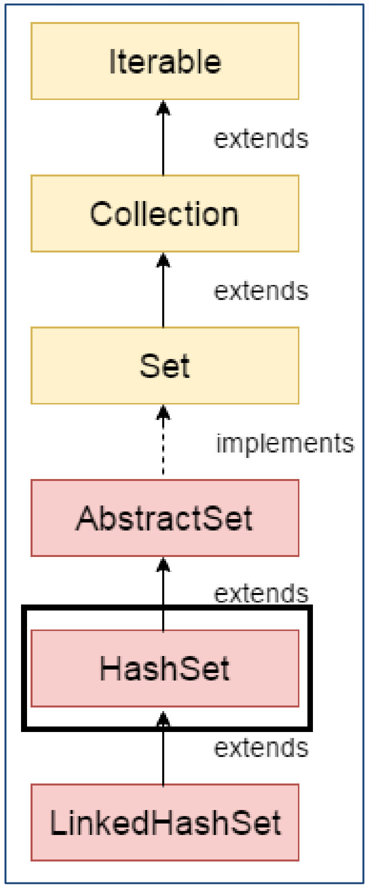
DEVLABS ALLIANCE

# HashSet

**public class** HashSet<E> **extends** AbstractSet<E> **implements** Set<E>, Cloneable, Serializable

It is used to create a collection that uses a hash table for storage.

- HashSet stores the elements by using a mechanism called **hashing.**
- Doesn’t allow duplicates. If you add a duplicate element, the old value would be overwritten.
- Allows null values but if you insert more than one null values, it would still return only one null value.
- Non-synchronized.
- Doesn't maintain any order. Elements are inserted on the basis of their hashcode.
- Preferred for search operations.

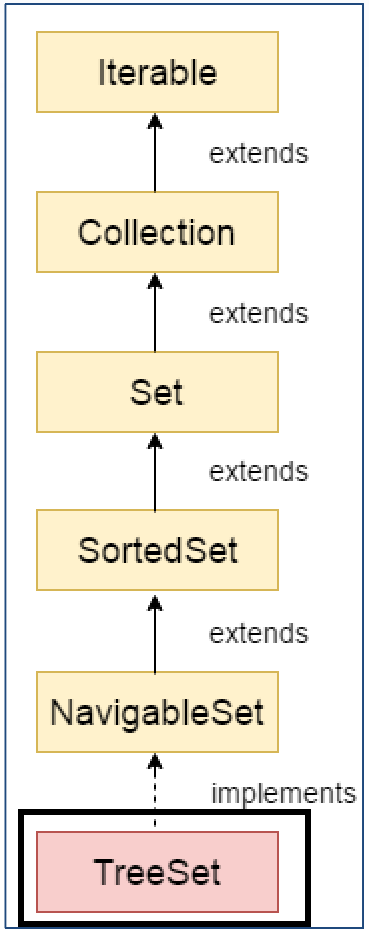


## TreeSet

**public class** TreeSet<E> **extends** AbstractSet<E> **implements** NavigableSet<E>, Cloneable, Serializable

Java TreeSet class implements the Set interface that uses a tree for storage.

- Maintains ascending order.
- Contains unique elements only like HashSet.
- Access and retrieval times are quiet fast.
- Doesn't allow null element.
- Non synchronized.

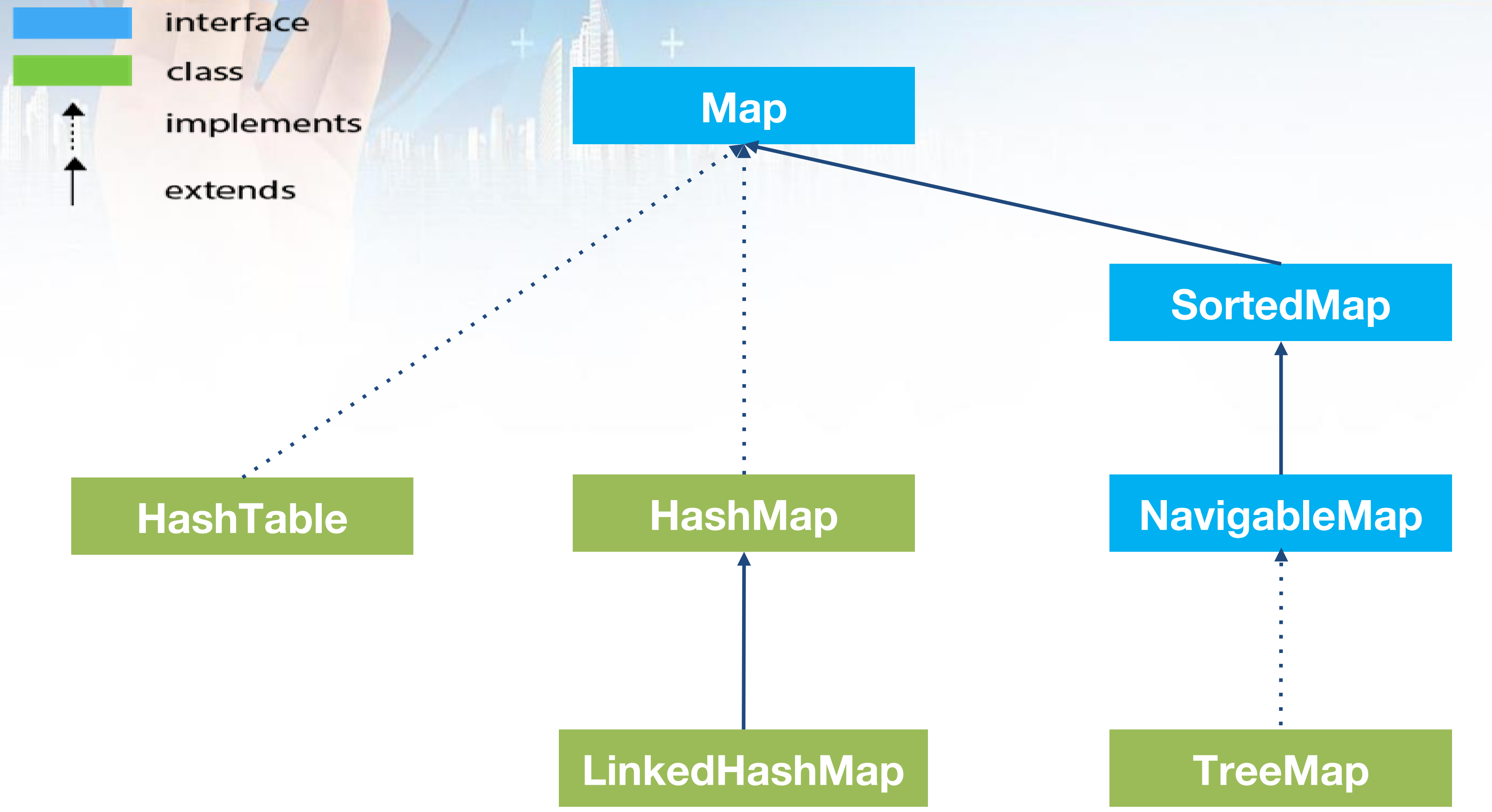


# Lab 13

Write Java programs to:

1. Create a HashSet and insert and retrieve values. Also, print the size of the HashSet.
2. Convert TreeSet to an integer array.
3. Remove specific element from TreeSet

# Map Interface



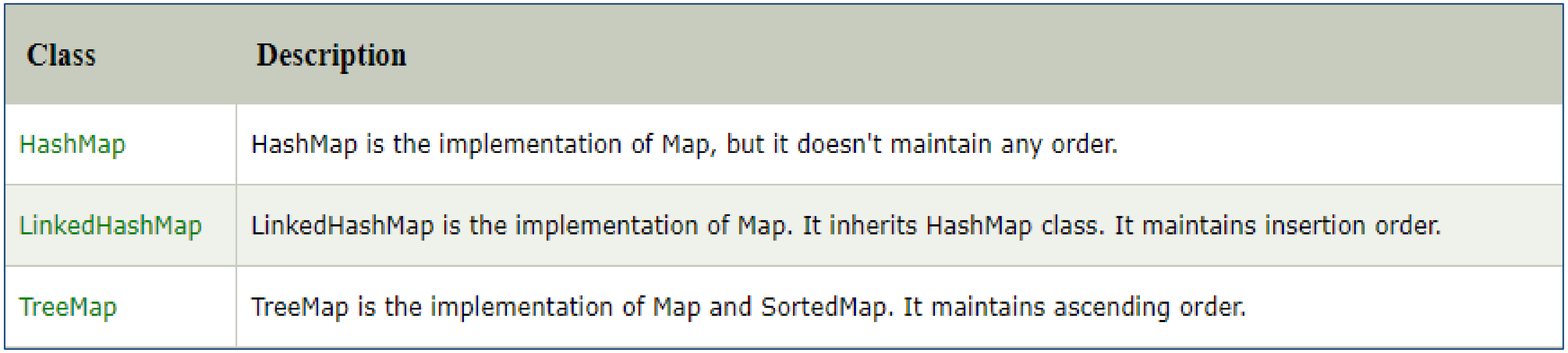
## Map Interface

Unlike List and Set, Map does not implement Collection interface.

It is a collection of ***key-value pairs***.
A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains ***unique keys***. Duplicate values are allowed.

A map is useful if you have to search, update or delete elements on the basis of a key.

There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap.



| Class | Description |
|---|---|
| HashMap | HashMap is the implementation of Map, but it doesn't maintain any order. |
| LinkedHashMap | LinkedHashMap is the implementation of Map. It inherits HashMap class. It maintains insertion order. |
| TreeMap | TreeMap is the implementation of Map and SortedMap. It maintains ascending order. |

DEVLABS ALLIANCE

## HashTable

**public class** Hashtable<K,V> **extends** Dictionary<K,V> **implements** Map<K,V>, Cloneable, Serializable

**K**: It is the type of keys maintained by this map.

**V**: It is the type of mapped values.

Java Hashtable implements a hash table, which maps keys to values. Uses hashing technique.

Hashing is the process of converting an object into an integer value. The integer value helps in indexing and faster searches.

A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the hashcode() method.

A Hashtable contains values based on the key.

- Contains unique elements.
- Doesn't allow null key or value.

## HashMap

**public class** HashMap<K,V> **extends** AbstractMap<K,V> **implements** Map<K,V>, Cloneable, Serializable

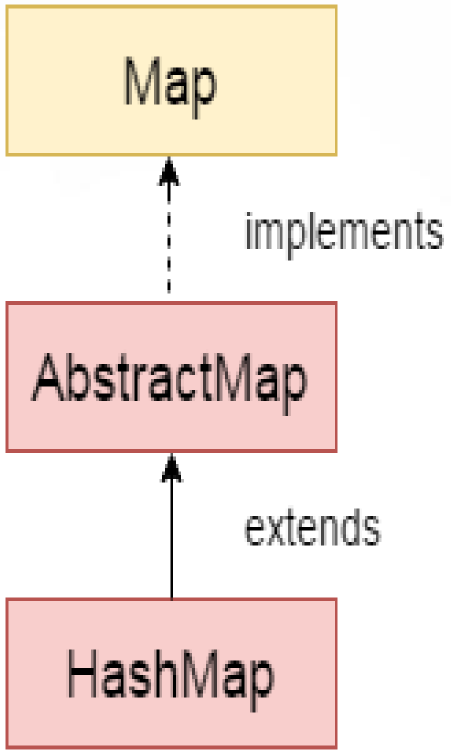
**K**: It is the type of keys maintained by this map.

**V**: It is the type of mapped values.

Java HashMap class implements the map interface by using a hash table.

This implementation provides constant-time performance for the basic operations (get and put), assuming the hash function disperses the elements properly among the buckets.

- Contains values based on the key.
- Contains only unique keys.
- May have one null key and multiple null values.
- Non synchronized.
- Maintains no order.



DEVLABS ALLIANCE

## TreeMap

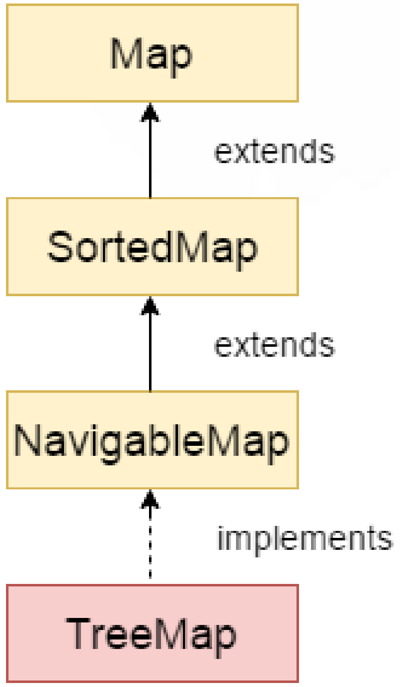
**public class** TreeMap<K,V> **extends** AbstractMap<K,V> **implements** NavigableMap<K,V>, Cloneable, Serializable

**K**: It is the type of keys maintained by this map.

**V**: It is the type of mapped values.

Java TreeMap class is a tree based implementation.
It provides an efficient means of storing key-value pairs in sorted order.

- Contains values based on the key.
- Contains only unique elements.
- Cannot have a null key but can have multiple null values.
- Non synchronized.
- Maintains ascending order.



# Lab 14

Write Java programs to:

1. Check a particular key exists in hashmap
2. Iterate through keys of a hashtable.
3. Remove key value pair from a hashtable.
4. Convert values of a map to a List.

## Assignment 3

Write java programs to :

1. Find duplicate characters with their occurrences count using HashMap.
2. Reverse an Arraylist.
3. Check a particular key exists in HashTable.
4. Convert keys of a map to a list.
5. Copy all elements of a HashSet to an Object array.
6. Get highest and lowest value stored in TreeSet
7. Sort ArrayList of Strings alphabetically.
8. Get Set view of keys from HashTable.

## Q&A

For any open questions, be in touch with us at training@devlabsalliance.com

DEVLABS ALLIANCE

Visit us at www.devlabsalliance.com

DLA
DEVLABS ALLIANCE