**Full Stack Development In Java (Full Stack Developer)**

Front End Tech (Front End Developer)

HTML, CSS, JS, React/Angular, Bootstrap

Backend Tech (API Developer)

Core Java

Servlet/JSP

Spring Boot

DataBase (Data Base developer/ (DBA))

MySql

Oracle

Junit Test

Maven

Git and GitHub

Postman

Swagger

Core Java

Basics Of Java

Installation, Writing First java program, Compile and run, Variables, Control flow statement, Arrays and String, method, class, Object

OOPs

Class, Object, Inheritance, Polymorphism, Abstraction, Encapsulation, Constructor, Super, this, static, final keyword, import and package, access modifier.

Advance Concepts in Core java

**Exception**, **Threading**, **Collection**, DB connection, File Handling.

**Java Language**

1. It is an Programming Language, use to write a code/Program.
2. Type Application
   1. Desktop Application
   2. Web Applications
   3. Mobile Application
   4. Embedded Application
   5. Console Application

**Java Editions**

1. These editions are introduce int java 2nd version.
2. There are 3 Editions of Java
   1. **JSE** – Java Standard Edition (Core Java)

Can develop a Console and Desktop applications.

Netbeans/Eclipse/IntelliJ

* 1. **JEE** – Java Enterprise Edition (Advance Java)

Can develop a Web Application.

IRCTC, SBI Banking, Kotak Banking

* 1. **JME** – Java Micro Edition

Can develop Mobile (Android) and embedded application.

**Setting system for Core Java Application Development**

1. JDK Download and Install

Download JDK 8 onwards

<https://www.oracle.com/java/technologies/downloads/>

1. Check after installation whether Jdk created inside programs file or not

C:\Program Files\Java\jdk-11.0.11

1. **Setting up the Environment variable**
   1. It is required to set the path of the java compiler and interpreter and other java components.
   2. To set the environment variable go to start and search for **“environment variable”**
   3. Or else, Right click on “**This PC**” then go to “**properties**”

Select “**Advance System and Setting”** option from the right side of the new window

Click on “**Environment Variables..”** button.

Then follow the next step in the “**System Variable”** section

* 1. Steps
     1. Java Home
        1. Click on “**New**” button from the System Variable



* + 1. JDK path
       1. Search for **Path** variable from the System Variables list.
       2. Click on “**Path**” variable and click on “**Edit**” button.
       3. Click on “New” button in the new window
       4. And set the value as

**%JAVA\_HOME%\bin**

Or

**C:\Program Files\Java\jdk-11.0.11\bin**

**Verify Your Setup**

1. Open a command Prompt
2. type “**javac”** command
3. type “**java -version”** command

**Java Components**

JDK

JDK is a Java Development Kit

This component is use to compile and run the java programs.

JDK is a collection of JRE, JVM and API

JRE

JRE is a Java Runtime Environment

This component is only use to run the java programs.

JVM

1. JVM is a Java Virtual Machine.
2. This component is use to execute java programs.
3. Assign the memory for the program execution, creating and managing threads and communicate with the OS wherever required

APIs

1. API is an Application Programming Interface.
2. These are the predefine class, interface, functions which is provided by java to reduce developer efforts.
3. rt.jar file has all java APIs

**Write first Java Program**

1. Open a Notepad file
2. Write a Java Program
   1. Create a java class
      1. Everything in java must be write inside class, excepts Import and package statement.
      2. Syntax:

**public class <NameOfClass>**

**{**

**}**

* 1. Write a main method
     1. Main method is the start point of java application
     2. JVM call this method whenever java application runs.
     3. Syntax:

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

**{**

**}**

* 1. Write Executable statement
     1. Which executes and produce an output

**System.out.println(“Hello, This is my First Java Program”);**

1. Save Notepad file
   1. File Name must be a same as public class name.
   2. File extension must be a .java
   3. Example : "WelcomeJava.java"
   4. Java File is also known as **source file** and the code written inside java file is called as **Source code**.
2. Open a Command Prompt
   1. Command prompt must points to a location where you save your source file (java file)
3. Compile the java File
   1. To compile java file use following command

“**javac filename.java**”

Example : javac WelcomeJava.java

* 1. After the compilation step you will get and .class file if there is not syntax error.
  2. In this stage higher level code will converted into lower level code which is also known as byte code.
  3. .class files contains the byte/binary code
  4. .class files are generated for the classes in a source code and not to a java file.

1. Run the Java Program
   1. To Run the java program use following command

“**java ClassName**”

Example: java WelcomeJava

* 1. When you run any java class JVM will execute the main method.



Rules For Java Program

1. Once Source file may have more than one class.
2. But only one class must be public and file name must be same as public class name.

**Keyword**

1. Keyword is the reserve words by Java language.
2. You cannot use these words for your purpose while writing a program.
3. All keyword are always in small case
4. Java Is case sensitive language.
5. Examples:

public, static, void, class, int, short, boolean, double, float, char, long, if, else, while, do, while, enum, extends, implements, new, false, true, null etc.

Literals

1. Are the values which also consider in the keyword list.
2. Examples: true, false, null

Identifier

1. Are the words which develop use for own purpose.
2. Identifiers are use for a class, method, variable, constant, Object name.
3. Rules and Convention
4. Rules for Identifier
   1. All identifiers are case sensitive.
   2. Identifiers can contain alphabets, number and symbols.
   3. Allowed symbols are $ and \_
   4. Identifier must be start with alphabets or Symbols, it must not be start with number.
   5. Identifier must not be as keywords.
   6. Spaces are not allowed in Identifier.
   7. There is no character limit to create an identifier.

|  |  |
| --- | --- |
| **Identifier** | **Valid/Invalid** |
| Employee1 | Valid |
| printDetails | Valid |
| Next record | Invalid |
| $salary | Valid |
| @email | Invalid |
| 1stNumber | Invalid |
| Number1 | Valid |
| \_$\_ | Valid |
| \_\_\_\_ | Valid |
| 12\_34 | invalid |
| \_123 | Valid |
| \_ | Valid |

**Conventions for creating Identifier.**

1. Convention For Class
   1. Class Name should be start with capital letter.
   2. If Class name is combination of more than one word, then every word should start with capital case.
   3. Example: String, Welcome, EmployeeDetails, StudentAddressDetails, Student
2. Convention For methods/variable/Object
   1. methods/variable/Object name should be start in small case.
   2. If Class name is combination of more than one word, then after 2nd word onwards every word should start with capital case.
   3. Example: println, main, printEmployeeDetails, employeeName.
3. Convention For Constant
   1. Constants are always in capital case
   2. If It is a combination of more than one word, then use \_ between all the words (Snaky Way).
   3. Example: PI, GRAVITY, COMPANY\_NAME

**Data Types and Variable.**

**Data Type**

1. Are the types of values which can be use in a java program.
2. There are 2 categories of Data Type
   1. Primitive Data Type
   2. Non-Primitive Data Type



Integer Data Type

1. Can store a values which do not have any data type.
2. These values can be a negative or positive
3. Example : 23, 32, -12, -22

Decimal Data Types

1. These are the type using which you can store the values with decimal point.
2. These values can be a negative or positive
3. Example : 3.14, 9.8, -23.12

Text Data Type

1. You can store a single char or even a numeric value.
2. Char values must be inside single quotes(‘value’)
3. Example: ‘M’, ‘F’, 65, 66, ‘@’, ‘A’

Logical Data Types

1. You can store a value only in form of **true** and **false**

**Formula to calculate range of values for each data type(Except char and boolean)**

**-2 n-1 to 2 n-1 -1**

Example: byte = -2 8-1 to 2 8-1 -1

-128 to 127

Here, n is the number of bits

**Variable**

1. Variable is a name of memory location.
2. Variables are use to store a values.
3. Variables can be use in a math expression.
4. Variables can use to show value the user.
5. Variables can be use to assign values to another variable.
6. Syntax:



1. There are two parts to create variable
   1. **Declaration of variable**

Syntax: datatype identifier(VariableName);

* 1. **Initialization of Variable**

Syntax: identifier(VariableName) = value;



Task-25-Nov

1. Create two int numbers and print the addition, subtraction, multiplication, division.

**Values Representation in Java**

1. Octal Value
   1. The values which is start with 0 are the octal values.
   2. Example: 012, 022
2. Binary Values (allowed in Java 1.7 and onwards)
   1. The values which are start with 0B or 0b are the binary values.
   2. Example: 0B1010 or 0b1010
3. Decimal Values
   1. The values without any prefix
4. Hex-Decimal
   1. The values which are start with 0X or 0x are the hexa values
   2. Example: 0X12, 0XCAFE, 0XDAD

**Primitive** **Variable Casting**

Casting a process in which you can convert value form one data type to another.

**Implicit Casting**

1. Is a casting which is done automatically by Java.
2. Example:

byte x = 10;

int y = x; //--- Implicit casting

**Explicit Casting**

1. The casting cannot be perform automatically by java and developer has to provide extra provision for this.
2. There is always a risk of getting logically incorrect output in case of explicit casting. And Developer has to make sure of it.

Example:

int y = 10;

short z = (short) y; // explicit casting.

**Rules to create Variables**

1. For float variable
   1. Float values must have suffix ‘f’ or ‘F’
2. For Long Variable
   1. Long values must have suffix ‘l’ or ‘L’
3. For char variable
   1. Char values can be numeric also but it never a negative values.

**Types of Variables.**

1. Variable types are decided on the bases of the location of the variable.
2. There are 3 types of variables
   1. Local Variable
      1. These variables are created inside the method.
      2. These variables are accessible inside only the methods.

* 1. Instance Variable
     1. These variables are created inside the class and outside of the any method.
     2. These variables can be access in side class directly and can be access in static method or outside class by using instance/Object of the class.
  2. Static/class variable (**Global**)
     1. These variables are use to create inside the class and outside any method by using static keyword.
     2. These variables can be access directly inside class and can be use by class name outside of the class.

**Default Values for a Variable.**

|  |  |
| --- | --- |
| **Data Type** | **Values** |
| byte short int long | 0 |
| double float | 0.0 |
| char | 0 or \u0000 |
| boolean | false |
| Non-primitive | null |

**Note: This default values are only applicable to instance and static variables. These Vales are not applicable to a local variable.**

**Comments in Java**

1. You can skip the lines for the execution.
2. There are 3 types of comments in Java
   1. Single Line Comment
      1. Is use to comment a single line from the code.
      2. Syntax:

**//** line to comment

* 1. Multi line Comment
     1. Is use to comment multiple lines from the code.
     2. Syntax:

/\*

Line1 to comment

Line2 to Comment

\*/

* 1. Documentation Comment
     1. Is use to provide a documentation for the class, methods and variables.
     2. These commands will be added in the java documentation and also inside compiled code.
     3. Syntax:

/\*\*

Documentation

\*/

**Control Flow Statement**

1. It is use to control the execution flow of the program.
2. There are 3 typed of control flow statement
   1. Sequential statement execution.
      1. This is the default implementation of any program.
   2. Conditional Statement.
      1. This is use to execute the statement(s) based on a specific scenario or condition.
      2. There are 2 ways to achieve conditional statement
         1. If-else conditional Statement and its variable.
         2. Switch cases.
   3. Looping Statement.
      1. This is use to execute the statement(s) multiple time until the specific condition is true.
      2. There are 3 typed of looping statement
         1. while
         2. do-while
         3. for and enhance for loop.

**If-else conditional Statement**

1. The statement(s) executes if the condition is true.
2. Syntax:

if (boolean condition)

{

Statement(s)

}

Note: The statement(s) from the if body executes only if the condition is true and if the condition is false then it will not executes the statement.

1. Syntax:

if (Boolean Condition)

{

Statement(s)

}

else

{

Statement(s)

}

Note: The statement(s) from the if body executes only if the condition is true and if the condition is false then it will executes the statement from else block. Else block must be followed with the if block

Task 30-Nov-2021:

Declare and initialize 2 int variables, compare the variable and print the largest value from those 2 values.

else-if ladder

1. You can also set the condition for the else block.
2. You can create any number of else if for a single if block
3. Syntax:

if (Condition) {

Statement(s)

}

else if (Condition) {

Statement(s)

}

else if (Condition) {

Statement(s)

}

else if (Condition) {

Statement(s)

}

else {

Statement(s)

}

Note:

1. else block is not mandatory, if you are writing else block then it must be a last block
2. if any of the condition becomes true then it will execute the statement from that block after that control will come outside the else-if structure

**Nested If-else**

1. one if condition inside another.
2. Syntax:

if(condition) {

if(condition) {

}

}

else {

if(condition) {

}

}

Task 1-Dec-21

Create a percentage variable with some value.

Print the grade based on Percentage value.

Percent is between 75-100 – Congrats!! You got **Distinction**

Percent is between 60-74 – Congrats!! You got **First Class**

Percent is between 50-59 – Congrats!! You got **Second Class**

Percent is between 40-49 – Congrats!! You got **Pass Class**

Percent is between 0-39 – Sorry!! You are Fail..

Percent not is between 0-100 – Invalid Percentage

**Switch Case**

1. To check the exact equality of the values you can use a switch case.
2. **Syntax**

switch(value)

{

case label/value:

Statement(s)

break;

case label/value:

Statement(s)

break;

case label/value:

Statement(s)

break;

case label/value:

Statement(s)

break;

default:

Statement(s)

}

**Rules For Switch Case**

1. Switch Case value must be **byte, short, int, char, enum, String** (Jdk1.7)
2. Case labels data type must be same as the data type of switch case value.
3. Case labels must be unique.
4. If multiple case has a same execution then you can combine a multiple cases.
5. Default is not always mandatory and can be write anywhere inside a switch case.

**While Loop**

1. It is a looping statement and use to execute a statement(s) multiple time.
2. In while loop first it will check the condition if condition is true then only it allows to execute the loop, if condition is false then loop we not execute. It is also known as entry level or Pre check.
3. Syntax:

Variable initialization (Start point)

while(condition)

{

Statement(s)

Increment/Decrement

}

**Note: The statements from the while loop will executes until the condition is true. Once condition becomes false while loop will stop execution.**

**Task: 2-Dec-2021**

Declare and initialize an int variable ‘num’ and print the table of num. Using While.

Example: int num = 5;

Output: 5

10

15

.

.

.

50

**do-while**

1. It is a looping statement and use to execute a statement(s) multiple time.
2. In this loop first it executes the statements and then check the condition. If condition is true then loop execution will be continue else the loop will be stop executing. This is also known as Exit level check or Post check
3. Syntax:

Variable initialization (Start point)

do

{

Statements(s)

Increment/Decrement

}

while(condition)**;**

Note: do-while loop executes at least once even though condition is false.

**Task-2-Dec-2021**

Write a Program to print the all number between 2 numbers.

In this Program declare a 2 int variables and print the all the number between them. Using Do-While

Example: int num1= 2

Int num2 = 7

Output: 2

3

4

5

6

7

**For Loop**

1. It is a looping statement and use to execute a statement(s) multiple time.
2. Syntax:

for (declaration/initialization **;** Condition **;** increment/decrement/Statement)

{

Statement(s)

}

1. In the for-loop declaration/initialization, condition and increment/decrement is optional.
2. If you don’t provide a condition inside for loop then it will be consider as true.

**Nested Looping**

1. One loop inside another loop.
2. Mostly nested looping is use to work with row and column structure.
3. Syntax:

for ( ; ; ) 🡪 Row

{

for ( ; ; ) 🡪 Column

{

}

}

**Enhance For Loop**

1. It is use to iterate all the values from the collection.
2. In this loop, Index Will be manage internally by java
3. ArrayIndexOutOfBound exception can be avoid using this loop.
4. Syntax:

for(datatype var **:** collection)

{

Statement(s)

}

Task:3-Dec-21

1. WAP to print the all even numbers between 1 to 50

Hint: 4%2 = 0

5%2 = 1

(% : mod is use to get the reminder of the division)

1. WAP to print following Pattern

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

**Array**

1. Array is a group/collection of similar type values.
2. Array must be of fixed in size.
3. Array can store only a same type of data.
4. Array is a non-primitive data type.
5. Array can store primitive and non-primitive.
6. Array is indexed based, index is always managed by java internally, and index always start with 0.
7. If you used index improperly then you may get and

java.lang.ArrayIndexOutOfBoundsException

1. Array can be of different types
   1. 1-D Arrays
   2. 2-D Array
   3. Multi-Dimensional Array (Jagged Array)

**1-D Array**

1. To use 1-D array you have to follow 3 steps
   1. **Declaration of Array**
      1. We can declare a type of array going to use.
      2. Syntax:

DataType NameOfArray **[]**;

int marks**[]**;

* 1. **Instantiation/Object of Array**
     1. You have to create an Object of array using **new** Operator.
     2. During instance creation of array you have to provide size of an array.
     3. After the instance creation of array internally memory gets allocated for every block of array and it will be initialize by default values.
     4. Syntax:

NameOfArray = **new** DataType[size];

marks = **new** int[5];

****

* 1. **Initialization of Array**
     1. Assigning values to an array.
     2. To set or get values inside array you have to use the index.
     3. Syntax:

NameOfArray[index] = value;

marks[0] = 55;

marks[1] = 88;

marks[2] = 76;



**Length Function in array.**

1. Length is use to get the total number of values inside array.
2. LastIndex = length – 1;
3. Syntax:

nameofArray.length;

Task 4-5 Dec-2021

1. Create an array of 10 int random numbers.
2. Print the count of even numbers and odd number (By Normal For Loop)
3. Print the average of all the number (By Enhance For Loop)
4. Find the Max and Min number from array (any for loop)

**2-D Array**

1. This array is use to store values in row and column format.
2. **Declaration of 2-D Array**

DataType NameOfArray [] [];

int marks[][];

1. **Instance Creation of array**

NameOfArray = **new** DataType[RowSize][ColumnSize];

marks = new int[3][5];

1. **Initialization of array**

NameOfArray[Row-Index][Column-Index] = value;

marks[1][1] = 60;

marks[1][3] = 80;



**Different Ways to create 2-D Array**



**Length function in 2-D**

1. **arrayName.length** -> return the total number of rows
2. **arrayName[row\_index].length** -> return the total number of column for given row.

**Task-1: 6-Dec-2021**

WAP to store 4 student 5 subjects marks.

1. Print the percentage of each student.
2. Print the highest percentage.
3. Print the Grades of Every Student (1st Class, Distinction, 2nd Class)

**Multi-dimensional/Jagged Array**

1. In this type of array the number of rows are fixed but each row can have different number of column.

**Class, Method, Object In java**

**Class:**

1. Class is a collection data member (variables/state) and member function (Behavior/method). (Class is a collection of state and Behavior)
2. Along with variables and methods you can also create another class inside one class (Nested Class), constructor, blocks.
3. Syntax:



**Method**

1. Method is a collection of variables and executable statement.
2. Methods are also use to distribute your logic into smaller chunks (small block).
3. You can also reuse the code written inside a method.
4. Along with variables and statement you can also create a class inside method.
5. At the time of calling methos if method has some input parameters then you have to pass a values to every parameter which is called as arguments.
6. You never create one method inside another method.



1. Methods are of 2 types
   1. Instance Methods
      1. Is a method which can be call only by using class Object.
      2. One instance method can call another method from class directly.
   2. Static Methods
      1. Are the methods which is created using static non-access modifier.
      2. To call static method we don’t require the class Object and it can be call by using directly class name also.
      3. Static methods can access only the static members (variables and methods) of class.

**Object**

1. Object are the instance of class.
2. Objects represents the properties of the class.
3. Each Object has independent existence.
4. Using Object you can access the properties of one class into another class.
5. Variables or methods can be access using dot(**.**) operator.
6. Syntax:





Task-1

Create a class Student create id, contact, gender of the student.

Create 3 objects of Student from the main method and set the value of student and also print value of each student on console

**String In Java**

1. String is a non-primitive data type
2. String is an array of character.
3. In Java String values store in double quotes (“value”)
4. String can be created in java by multiple ways.
   1. String class
   2. StringBuffer class
   3. StringBuilder class
   4. StringTokenizer class
5. **String class In Java**
   1. String class is **Build-in class** in java.
   2. String class present inside **java.lang** package.
   3. String class is a **final class.**
   4. By Using String you can store a string values (array character)
   5. String values are indexed based.
   6. Inside String class you will get multiple methods/function using which can perform operations on existing string.
   7. To store strig value you have to create Object of String class.
   8. String **objects are immutable**. Once we assign a values to these Object it value never change by any of the string class method.
   9. There are 2 ways to get the object of String class.
      1. With new Operator

**String str1 = new String("Hello");**

* + 1. Without new operator

**String str2 = "Hello";**

**SCP (String Constant Pool)**

1. It is a memory location from the heap
2. In side SCP String objects will be created.
3. The String value which is created without new operator are created inside SCP.
4. When creates any String Object inside SCP, it will first check whether same value is present or not, if object with same value is present then no new object will creates else it will create new object.



**StringBuilder**

1. StringBuilder is a java build-in class.
2. Using this class you can store a string values.
3. There are multiple methods available inside this class to perform operation on exiting string.
4. The Object of StringBuilder is mutable (the values of the StringBuilder will keep on changing once perform any operation using methods.)
5. StringBuilder is present inside java.lang package.
6. StringBuilder is a final class.
7. The Object of this can be create by using new operator only.

**StringBuffer**

1. StringBuffer is a java build-in class.
2. Using this class you can store a string values.
3. There are multiple methods available inside this class to perform operation on exiting string.
4. The Object of StringBuffer is mutable (the values of the StringBuffer will keep on changing once perform any operation using methods.)
5. StringBuffer is present inside java.lang package.
6. StringBuffer is a final class.
7. The Object of this can be create by using new operator only.
8. StringBuffer methods are synchronized.
9. Hence, Object of StringBuffer is thread safe.
10. StringBuffer is slower than StringBuilder

Interview Question

1. Difference between String, StringBuffer and StringBuilder
2. Difference between StringBuffer and StringBuilder

Eclipse IDE:

<https://www.eclipse.org/downloads/packages/release/2021-09/r>



**Select Java Perspective of eclipse**



**Select Java and click on Open button**



**To Get Console at the bottom of screen.**

Go To “Windows” menu -> select “Show View” option -> select “Console” option

**Create Java Project in Eclipse**

Go to “File” menu -> Select “New” option -> select “Java Project” option -> New Window will be popup.



Core Java Project Structure



**Create Java Class Inside project**

Right click on “src” folder -> go to “New” option -> select “Class” option -> new Window will popup



**Object Oriented Programming concepts (OOPs)**

1. Java use all the OOPs concepts.
2. Java is not fully object oriented programming language. In Java primitive variables are not in the form of object and this is one of the reason for Java is not fully object oriented.
3. Class and Objects are the base of object oriented programming concepts.
4. There 4 OOPs concepts
   1. **Encapsulation**
      1. Wrapping of data member(variable) and member function(method) into single unit. Use to achieve data hiding.
   2. **Inheritance**
      1. It is a way to access properties (variables & methods) of parent class into child class. Use for code reusability.
   3. **Polymorphism**
      1. Is away to perform single action into multiple ways (One thing has multiple form).
   4. **Abstraction** 
      1. Hiding the internally/complex details and showing the only functionalities/required detail to the user.

**Encapsulation**

1. Wrapping of data member(variable) and member function(method) into single unit.
2. Java class is also consider as a encapsulation.
3. Perfect encapsulated class is a class which has all the instance variables as private and provide access to them using getters and setters.
4. Getter method is use for getting value of the variable.
5. Setter method use for set the value of the variable.
6. Can achieve Data hiding using Encapsulation.
7. Encapsulation is one of the way to achieve loose coupling.
8. You will have a full control on the values which is set or get from the application.
9. Also can be decide who can access what.

Task-1

Create a Student Class which has following Variable

Id, name, gender.

Set and get the student data from another class. Apply following validation.

Id must be between 1-1000

Gender must be Male, Female, Other

**Inheritance**

1. It is a way to access properties (variables & methods) of parent class into child class.
2. In this OOPs concept the parent and child relation will be created between a classes which is also known as **IS-A relation in java.**
3. Inheritance can be achieved in java using **extends** keyword.
4. There are 5 types of inheritance in OOPs. In this java supports only 3 types directly and 2 types can be achieved in java using interface.



1. In Java One class can inherits(extends) the properties from one class only at a time.
2. In Java **Object class is a parent of all java classes**. Either directly or indirectly.
3. You can only inherit the non-private properties of parent class into child class.
4. Every Sub class constructor calls super class default or no parametrized constructor internally. This is happened due to a super keyword.

**Object Class**

1. Object class is the java build-in class.
2. Object class is the parent of all the java classes.
3. In Object class some of the common methods are present which is accessible in all the java classes.
4. Some of the common methods are
   1. **toString()**: this methos calls whenever you print the object of class directly. I will return the String value as an output
   2. **hashCode():** this method return the hashcode of the object.
   3. **equals(Object):** this methos is use to compare two object.
   4. **wait(), wait(int),wait(int,long):** all these method are use tom multi-threading
   5. **notify(), notifyAll():** these methods are also use in multi-threading.

**Constructor**

1. Constructor is a special kind of method, which is use to initialize the instance variable.
2. Constructor is use to construct the values/memory at the time of object creation.
3. There are some rules to create Constructor in class.
   1. Constructor name must be same as class Name.
   2. Constructor must not have a return type.
   3. Constructor can be crate by using any of the access modifier like public, private, protected, default.
   4. There can be more than one Constructor present inside a class.
   5. Constructors gets called at the time of object creation, and cannot be call using an object and dot operator.
   6. Every class has constructor, if it is not provided explicitly then, the default constructor will be provided by java. But if you provide constructor inside class explicitly then java will not create a default Constructor.

**super keyword**

1. Super keyword is use to access the variable, methods, constructor of the super class.
2. Super is not an object of super.
3. Calling constructor of Super class using super keyword
   1. By default every sub class constructor calls super class default/No param constructor by suing super keyword.
   2. Internally java adds a super() as a first statement of every constructor.
   3. By using this you can call a constructor manually also.
   4. Constructor calling line must be a first line in a constructor.
4. Using super keyword to call super class variable and methods
   1. If sub class has a property same as super class, then sub class always gives the priority to a sub class properties.
   2. To access the super class properties in this case you have to take a help of super keyword.
   3. You can access a super class variable and methods from anywhere inside the sub class.

**this keyword**

1. This keyword is use to access the variable, methods, constructor of the same class.
2. This keyword is points to same class current object
3. Calling constructor of same class using this keyword
   1. You can call a constructor of same class from another constructor using this keyword.
   2. Constructor calling statement using this keyword must be a first statement in a constructor.
   3. You cannot use super and this at a time in a constructor.
4. Accessing variables and methods of same class using this keyword.
   1. You can access the variables and methods of same class using this keyword.
   2. Accessing variables and methods can be done at any location of the class.

**final keyword**

1. Final Keyword is use to create a constant.
2. Final keyword can use for class, variable, method.
3. **Final variables** 
   1. By Creating final variable, you can make the variable constant (fix value).
   2. Final variables can be an instance, local variable.
   3. Final instance variable has to initialize at any one to the location out of 3
      1. At the time of declaration.
      2. Inside every constructor of class.
      3. Inside initializer block.
4. **Final Class**
   1. Final classes cannot be use (in inheritance) as a parent class.
   2. By making class as final, you can make the implementation of class fixed for every user.
5. **Final Method**
   1. Final methods cannot be override. And the implantation of the method is fixed.

**static keyword**

1. Static keyword is use to create an object independent component.
2. Static keyword can be used for Class, variable, method
3. **Static Variable**
   1. You can only create static variables inside class.
   2. Static variables cannot be created inside method.
   3. Static variables can access without using class Object, and can be access by using a class name.
4. **Static methods**
   1. Static methods can be access without object just by using a class name.
   2. You cannot access the non-static properties of the class inside static methods directly.
   3. You cannot use super and this keyword inside the static method.
   4. Static method cannot be override.
5. **Static Class**
   1. Static class cannot be an outer class.
   2. Static class must be an inner class.

**Package and Import.**

**Package:**

1. Package is a group (collection) of java/class files.
2. These java/class files are of similar functionality.
3. You can consider it as a folder. When you create package inside java program it creates a folder inside a file system.
4. To create package use following steps.
   1. To crate package, you have to use package keyword.
   2. Package statement must be a first line in your source file.
   3. Package statement is appliable of the all classes form a source.
   4. Package statement can be write only once in a file.
5. To access the class from one package to another you have to use an import statement.

**Import:**

1. Import statements are use to access the classes from one package to another.
2. Import statement must be after package statement.
3. You can write more than one import statement in a file.
4. Java adds import all the classes from java.lang package internally.
5. Syntax:

import packagename.className; // to access a single class from a package

import packagename.\*; // to access all the classes from a package.

**Access Modifier**

1. Access modifiers are use to manage the access of class, variable, methods, constructor etc.
2. There are 4 access modifiers in Java
   1. public
   2. protected
   3. default/package (default access modifier if not provided any access modifier)
   4. private

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Access Modifier** | **Same Class** | **Different class in same package by Object** | **Different class in same package by Inheritance** | **Different class in Different package by Inheritance** | **Different class in Different package by Object** |
| public | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** |
| protected | **Yes** | **Yes** | **Yes** | **Yes** | **No** |
| default/package | **Yes** | **Yes** | **Yes** | **No** | **No** |
| private | **Yes** | **No** | **No** | **No** | **No** |

1. The highest restrictive access modifier is **private**
2. The lowest restrictive access modifier is **public**
3. Level from lowest to highest

**public --> protected --> default --> private**

**Polymorphism**

1. Single thing can be use in a different way is known as polymorphism.
2. **Poly** means manyand **morphism** means form.
3. In Java Polymorphism is of 2 types.
   1. **Compile time Polymorphism**
      1. A method calls resolves at time of compilation and same will be followed at execution time.
      2. **Overloading** is an example of Compile time polymorphism.
   2. **Runtime Polymorphism**
      1. A method calls linked at time of compilation, but different method is called at the time of execution.
      2. Also one of the way to achieve loose coupling.
      3. **Overriding** is an example of Runtime polymorphism.

**Overloading**

1. Overloading can be achieved in same class or in sub class.
2. In Overloading method has to create with same name and different parameter list.
3. Different Parameter List can achieve by 3 ways
   1. Change the data type of parameter
   2. Change the number/count parameter
   3. Change the sequence of parameter
4. You can change the return data type.
5. You can change the access modifier.
6. You can overload a final method.
7. Can overload a static method.
8. Can throw new and broader checked Exception.
9. By overloading you can improve a readability of the program.
10. Example of overloading in java build-in class.

StringBuffer, StringBuilder

**Overriding**

1. Overriding can be achieved in Sub class only.
2. In Overriding method has to create with same name and same parameter list.
3. Return datatype must be same if it is primitive or void, or it can be sub type (applicable of non-primitive data type).
4. Access modifier must be same or less restrictive.
5. You cannot override final method.
6. Cannot override a static method.
7. Private methods cannot be override.
8. Cannot throw new and broader checked exception.
9. Overriding is use to improve a readability of the program and also, we can change the implementation provided by super class method inside sub class.

Task

1. List the difference between overloading and overriding.
2. Write a single program to demonstrate overloading and overriding.

**Runtime Polymorphism**

1. Method call is resolve at the time of execution/runtime.
2. Run time polymorphism is also known as dynamic polymorphism.
3. To achieve runtime polymorphism, you should follow the steps
   1. There must be an inheritance between a Class.
   2. There must be an Overriding in the sub-class.
   3. The method call has to do using polymorphic object.
4. What is Polymorphic Object
   1. The Object which has a parent class reference and child class object.
   2. Every parent class reference holds the object of child class.

Task :



**Abstraction**

1. Hiding the internally/complex details and showing the only functionalities/required detail to the user.
2. There are 2 ways to achieve abstraction in java.
   1. Abstract Classes
      1. Abstract is use to achieve 0-100% abstraction (Partial Abstraction).
   2. Interface
      1. Interfaces are used to achieve 100% abstraction. (till jdk 1.7)
3. **Abstract Class**
   1. Abstract classes are use to achieve partial abstraction.
   2. Abstract class is the class which is define using **abstract keyword**.
   3. Abstract class can have abstract and non-abstract method.
   4. Abstract method is a method, which do not have any implementation and we just declare the method.
   5. You cannot create an object of abstract class. But it can be used as a reference to create polymorphic objects.
   6. Abstract classes are mostly used as a top level called in a hierarchy.
   7. It is not mandatory to create abstract method inside abstract class, you can create an abstract class without any abstract methods also.
   8. Abstract classes can be extends on another class.
   9. If you extends abstract class on non-abstract class (concrete class) then, you have to provide implementation (overriding) for all the abstract methods of the abstract class inside concrete sub class.
   10. If you extends abstract class on another abstract class then, you do not have to provide implementation for the abstract method.
   11. Constructor can be create inside abstract class. These constructors will be invoked whenever you create an object of sub class.
   12. Can create static, final methods inside abstract class. But those method must be a non abstract.
   13. Cannot mark abstract method as private.
   14. You cannot use a combination of abstract, static and final.
   15. Can create method with any access modifier inside abstract class.
   16. Can create variable with any access modifier and non-access modifier inside abstract class.

**Interface**

1. Interfaces are used to achieve 100% abstraction. (till jdk 1.7)
2. Interfaces are not a class. But every interface will get a .class file after compilation.
3. Interface can be created by using **interface keyword**.
4. All the methods from the interface are by default public and abstract.
5. In the interface you cannot create non-abstract method, default methods are allowed from java 1.8 onwards.
6. All the variables create inside interface are by default consider as public static final.
7. You cannot create Object of interface, but it can be used as a reference to create polymorphic object.
8. Interfaces are use as a top level in a hierarchy.
9. Interfaces are always implements by class. Here, also IS-A relation will be form between class and interface.
10. If you implements interface on any concrete (non-abstract) class then, it is mandatory to provide implementation for all the abstract method of interface.
11. If you implements interface on any abstract class then, it is not mandatory to provide implementation for all the abstract method of interface.
12. Cannot create constructor inside interface.
13. All the members (variables and method) of interface are public, you cannot use any other access modifier than public inside interface.
14. One interface can extends one or more other interface.



1. Multiple inheritance can be achieved by following ways using interface
   1. One interface extends more than one interface.
   2. One class can implements more than one interface.
   3. One class extends another class and implements one or more interfaces.



**Types of Interface**

1. **Simple/regular interface**
   1. The interface with abstract or default methods
2. **Marker Interface**
   1. The interface without any member.
   2. It is also known as blank interface.
   3. These are used to notify JVM to provide special functionality while execution of the program.
   4. Examples, Serializable interface.
3. **Function Interface** 
   1. The interface with only one abstract method. It can have any number of default methods.
   2. These interfaces are introduced in Java 1.8 onward.
   3. These interfaces are use for functional programming using lambda expression.
   4. Build-in functional interface of java present in **java.util.function** package.
   5. Java provided @FunctionalInterface annotation to ensure create interface is a Functional Interface

**Interface in Java 1.8 onwards**

1. Default Methods are allowed in interface.
2. Static implemented methods are allowed in Interface. Hence, you can create a main method and execute the interface in java-8.
3. Functional Interface are introduced.

Task:

1. List the difference between abstract class and interface.
2. WAP for following structure



**Exception handling**

1. **Exception**: Exception is an unwanted scenario which occurs at the time of program execution due to which program execution terminate abnormally.
2. **Exception Handling**: Exception handling is a process by which we can avoid the abnormal termination of the program if any exception occurs during the runtime. Also it can be use to provide any alternative way if any exception occurs.

****

**Exception Hierarchy**



**Error:** Error can be occurs during the program execution or before starting of the execution. It is not recommended to handle the errors. Program cannot be recovered after getting a error inside program. Errors mostly occurs due to system issues.

|  |  |
| --- | --- |
| **Exception** | **Error** |
| Exception are Recoverable | Errors are non-recoverable |
| Exception can be handle and provide an alternative way to continue program execution | Error can be handle but cannot continue the program execution. |
| Exceptions mostly occurs due to logical issues or user inputs issues. | Error are mostly occurs due to a user system issue. |

**Keywords use in Exception Handling**

There is total 5 keyword use in exception handling

1. **try**:
   1. It is a block which use to write a statement which may cause an exception.
2. **catch**:
   1. It is a block which is use to handle/catch the exception which is thrown from the try block. In side this block you can also provide the alternative statement which executes if any exception occurs.
3. **finally**:
   1. This block is use to write a statement which has to be execute irrespective of try and catch execution (Statement which executes always).
4. **throw**:
   1. This keyword is use to throw/raise any exception manually from any of the statement.
5. **throws**:
   1. This keyword is use to populate exception at the caller level or outside method.

**Try block**

1. You can write a statement inside try block which may throw an exception.
2. Try block must be write with catch or finally or both.
3. Syntax:

try {

statement(s)

}

**Catch block**

1. Catch block is use to handle the exception. While writing a catch block you must provide the exception type.
2. Syntax:

catch(ExceptionType/class ref\_variable) {

statement(s)

}

1. Can create more than one catch block of a single try.
   1. In try with multiple catch, you cannot handle/write a parent catch block before child.
2. Nested try catch, One try can have a try-catch structure or any catch block can have try-catch structure.

**Internals of throwing Exception**

1. Exception throws from the program during the execution/runtime only.
2. First step is Exception Scenario/condition/cause will be identify.
3. Second step is to identify the exception type and create object of Exception.
4. Third step is to throw object of exception from the statement level.

**finally block:**

1. Finally block will give a guarantee of execution.
2. This block executes always irrespective of try or catch execution.
3. This block is mostly used for resource closing/cleanup activities.
4. Finally block must be write after try or try catch.
5. For one try block you can write maximum one finally block.
6. Finally, will not execute in the following cases.
   1. Calling a System.exit() manually.
   2. JVM crash before execution of finally block.

**throw**

1. Throw keyword is use to throw any exception manually.
2. Using throw you can raise object of execution manually.
3. It is use to throw checked, unchecked exception and custom exception.
4. Mostly it is use to raise a custom exception.

**Custom Exception**

1. Custom exception is the use define exception for customized requirement.
2. To Create a Custom exception, you have to create a java class.
3. Extends you java class with any of the exception class.
4. Then, you can provide you own functionality for the exception.

**Types of Exception**

1. There are 2 types of exception
   1. Checked Exception (compile time exception)
      1. This exception has to handle at the time of coding, else compiler will not allow to compile the program.
      2. Java Compiler will force us to handle these types of exceptions.
      3. As a precaution compiler is forcing developer to handle the exception.
      4. If any checked exception occurs at the execution time, program never terminate abnormally because handling is already provided for these exceptions.
      5. Example: IOException, SqlException, InterruptuedException etc.
   2. Unchecked Exception
      1. These exceptions are not force by java compiler to handle at the compilation time.
      2. If this exception occurs during the execution of the program and if it is not handle then program can terminate abnormally
      3. Example: ArithmaticException, NullPointerException, IndexOutOfBoundException etc.

**throws**

* 1. Is use to propagate the exception outside method at the caller level.
  2. Can skip the exception handling for the targeted method and propagate this exception to a caller level.
  3. Throws is use at method declaration level.
  4. Throws must be followed with Exception class name.
  5. You can throw any (checked, Unchecked, custom) exception from the method declaration.
  6. You can also add more than once exception class into declaration level by throws.

**Java 7 enhancements in Exception**

1. Catch with multiple exception using OR clause
   1. Using this feature you can club multiple exception into a single catch block.
   2. You cannot use exception from the same hierarchy.
   3. Syntax:

**catch** (ArithmeticException | NullPointerException | IndexOutOfBoundsException e) {

System.***out***.println("Got Exception");

}

1. Try with resource. 
   1. In this structure the resource closing will be taken care by java internally.
   2. **Syntax:**

try(Resources/Object of the classes which needs to be close)

{

}

Task:

1. Difference between checked and unchecked exception.
2. Difference between throw and throws.
3. Difference between final and finally.
4. Difference between Exception and Error.

**User Input in Java using Scanner Class**

1. Scanner class is a java build-in class.
2. Scanner class is use to accept an input from the user using console, file, string.
3. Scanner class is present inside java.util package.
4. Scanner class is use to accept use inputs at runtime.
5. You have to create an object of Scanner class before accepting values from the user.
6. Object Syntax: Scanner scan = **new** Scanner(System.***in***);
7. There are multiple methods available inside scanner class to accept a user values.
   1. next(): to accept the string word from the user.
   2. nextInt(): to accept int value
   3. nextFloat(): to accept float value
   4. nextDouble(): to accept double value
   5. nextLine(): to accept the complete line with multiple words.

**Multi-Threading**

**What is Thread?**

1. Thread is an individual block of execution which executes independently.
2. Every thread has its own call by stack memory to execute independently.
3. Thread is light weight process. It is also known as a small part of a process.

**Why Thread?**

1. Multi-threaded applications are always faster than single threaded application (Performance can be improve).
2. Every thread executes independently, and hence single thread failure will not cause to a process failure.
3. You can also execute a task in parallel.
4. Can utilize 100% of CUP resource.

**How to create Thread in Java**

1. Every Java Program executes using a thread.
2. Java creates a thread internally for every program. This thread is a **main thread**.
3. Custom thread can be created by 2 ways.
   1. Extending **Thread class**
   2. Implementing **Runnable interface**
4. Every thread has some task and that has to assign inside a **run()** method. You have to override run method from the Thread class or Runnable interface.
5. Thread will always execute by JVM. You cannot have full control for the execution of the thread.
6. We can only create a thread, assign task to thread and will make thread ready to run.
7. To make thread ready to run you have to call a **start()** method from the thread class or its sub class.

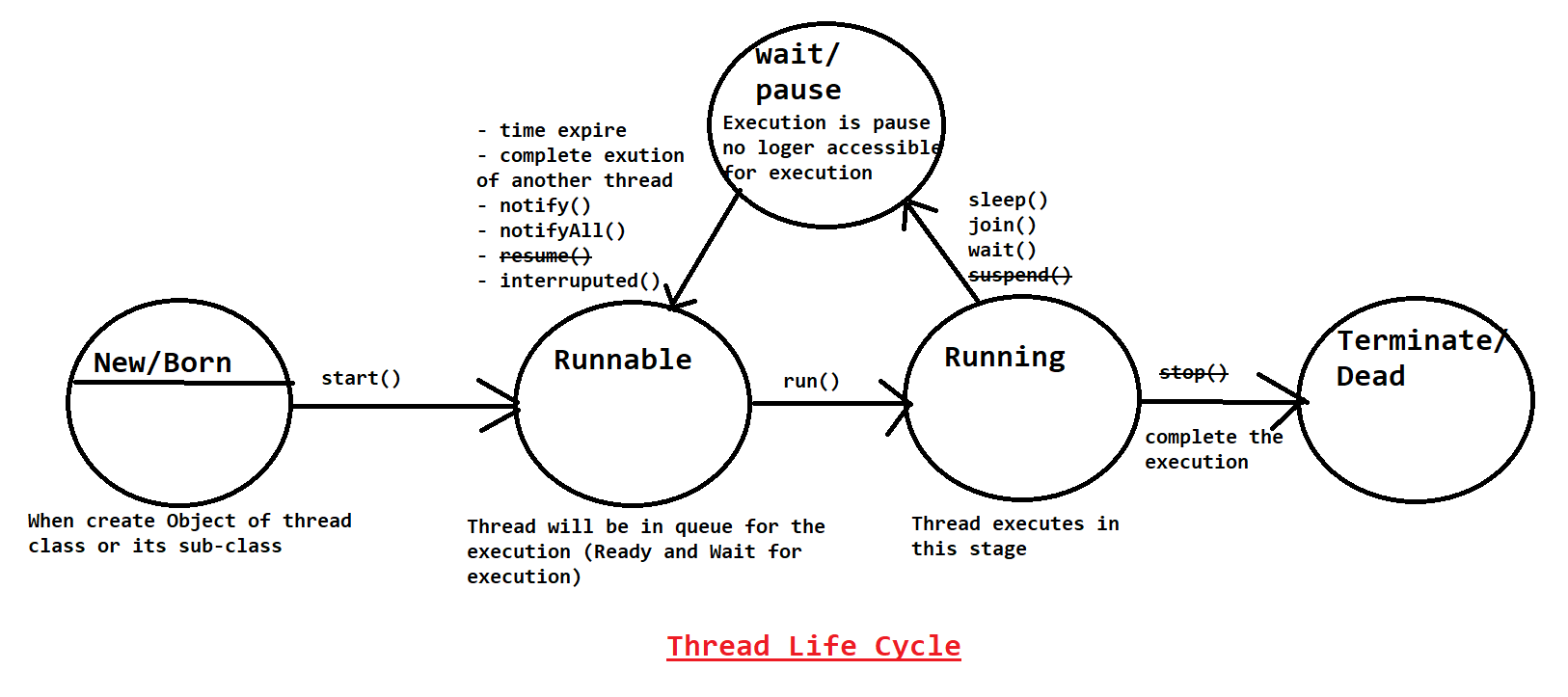
Thread Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| run() | Using this method, you can assign a task to a thread |
| start() | Using this method, you can make thread ready to run. |
| currentThread() | This is a static method in thread class, Using this method you can get the currently executed thread object.  Thread[main(name),5(priority),main(group)] |
| setName(String)  getName() : String | These methods are use to set and get the name of the thread. |
| setPriority(int)  getPriority() | These methods are use to set and get the priority of the thread. The priority must be between 1-10. Default priority is the 5 or the thread group priority. 1 – LOW\_PRIORITY  5 – MID\_PRIORITY  10 – HIGH\_PRIORITY  If priority not between 1-10 then you will get an java.lang.IllegalArgumentException |
| sleep(long) sleep(long, int) | This method is use to pause the execution of currently executed thread for a given time. Time is in milliseconds or milliseconds and nanoseconds. This is the static method of the thread class. This method throws checked exception (InterruptedException) |
| join() join(long) join(long, int) | This method is use to pause the execution of current thread and it will wait until the joined thread complete the execution or till the given time is expire. This is the non-static method from thread class. This method throws checked exception (InterruptedException) |
| ~~stop()~~ | This method is use to terminate/stop the execution of thread manually. |
| ~~suspend()~~ | By using this method you can pause the execution of thread. These thread will remain in Pause stage until some another thread calls resume method. |
| ~~resume()~~ | This method is use to resume the execution of suspended thread. |
| wait() wait(long)  wait(long, int) | By using this method current thread pause the execution and also thread will release the lock before entering into a wait stage. Wait thread has to notify by some another thread or it will come into runnable stage after time expire. This method has to execute inside a synchronization block or method only else it will throws an exception. |
| notify() | Using this method the one thread which is in the waiting stage will be notify and enter inside the runnable block. This method has to execute inside a synchronization block or method only else it will throws an exception. |
| notifyAll() | Using this method all the thread in waiting stage will ne notify and enter into runnable stage. This method has to execute inside a synchronization block or method only else it will throws an exception. |

Difference in Sleep and Join method?

**Thread Life Cycle**

1. Execution of the thread is divided into multiple stags. These stages are known as thread life cycle.
2. Thread life cycle will be manage by JVM.
3. There are 5 stages of life cycle
   1. New/Born stage
   2. Runnable stage
   3. Running Stage
   4. Wait/Pause stage
   5. Terminate/Dead stage



**Synchronization**

1. Synchronization is a process in which share resource can be invoked by thread sequentially (One by One).
2. In the multi thread with share resources, data inconsistency issue can occurs.
3. The data inconsistency issue can be resolve by synchronization.
4. In the synchronization process the object lock and unlock process happens internally.
5. There are two types of lock. Object and class level lock.
6. Synchronization can be use using the **synchronized** keyword.
7. You can mark method as synchronized or can create a synchronized block.
8. Synchronization can affect a program performance.
9. Synchronization can cause a dead lock scenarios also.

Difference in sleep, wait, join?

Difference in notify and notifyAll?

**Collection Framework**

1. Is a group of Objects which is also known as elements.
2. It is dynamic in size.
3. It is a collection of different type of objects.
4. You can store the elements using different criteria’s.
   1. Collection can be sorted
   2. Collection can order
   3. Collection can be of unique object
   4. Collection can be a duplicate object
5. Collection is divided into 3 types
   1. List: It is index based collection, Allows duplicate elements.
   2. Set: In is not index based, Store only unique elements.
   3. Queue: Is use for a FIFO priority.
6. All the classes, interface are present inside **java.util**

**Collection Interface**

1. Collection is the parent interface of all the interfaces and class present inside this framework.
2. Collection interface provided common methods which are present in all other classes of collection.
3. Some of the common methods are as below
   1. add(Object) : is use to add single object inside collection
   2. remove(Object): is use to remove a single object from the collection.
   3. contains(Object): is use to check a specific object is present inside collection or not.
   4. addAll(Collection): is use to add multiple object at a time inside collection.
   5. removeAll(collection): is use to remove a specific group of elements.
   6. containsAll(Collection): is use to check multiple objects present inside collection or not.
   7. clear(): is to remove all the object from the collection. To make collection Empty.
   8. size(): is to known the total objects present inside collection.
   9. isEmpty(): is use to check whether collection is empty or not.
   10. iterator(): is use to get one by one object from the collection.

**Collection Hierarchy**



**List Interface**

1. List is an index based.
2. List allows duplicate elements.
3. List is use to store object of different data type with dynamic in size.
4. List interface has following implemented classes
   1. ArrayList
   2. Vector
   3. LinkedList

**ArrayList**

1. ArrayList is the implemented class of List interface.
2. ArrayList is an indexed based.
3. ArrayList can store duplicate objects.
4. It store object of different data type and its dynamic in size.
5. ArrayList is backed by array. The elements store in form of array in ArrayList.
6. ArrayList has a default capacity 10
7. ArrayList is order, maintains the insertion order of the elements.
8. ArrayList also provide a random access to a elements.
9. Can perform operation on a specific index, due to randomly adding and deleting values on specific index shifting of the elements will take place.
10. ArrayList provide slower performance of the update operations
11. ArrayList is faster in the iteration/retrieval process
12. This class is introduced in jdk 2 version

**Vector**

1. Vector is the implemented class of List interface.
2. Vector is an indexed based.
3. Vector can store duplicate objects.
4. It store object of different data type and its dynamic in size.
5. Vector is backed by array. The elements store in form of array in Vector.
6. Vector has a default capacity 10
7. Vector is order, maintains the insertion order of the elements.
8. Vector also provide a random access to a elements.
9. Can perform operation on a specific index, due to randomly adding and deleting values on specific index shifting of the elements will take place.
10. Vector provide slower performance of the update operations
11. Vector is faster in the iteration/retrieval process.
12. Vector methods are synchronized.
13. Vector object is thread safe object can be access one thread at time in multi threaded application.
14. Vector is slower than ArrayList.
15. This class is introduced in jdk 1 version

**LinkedList**

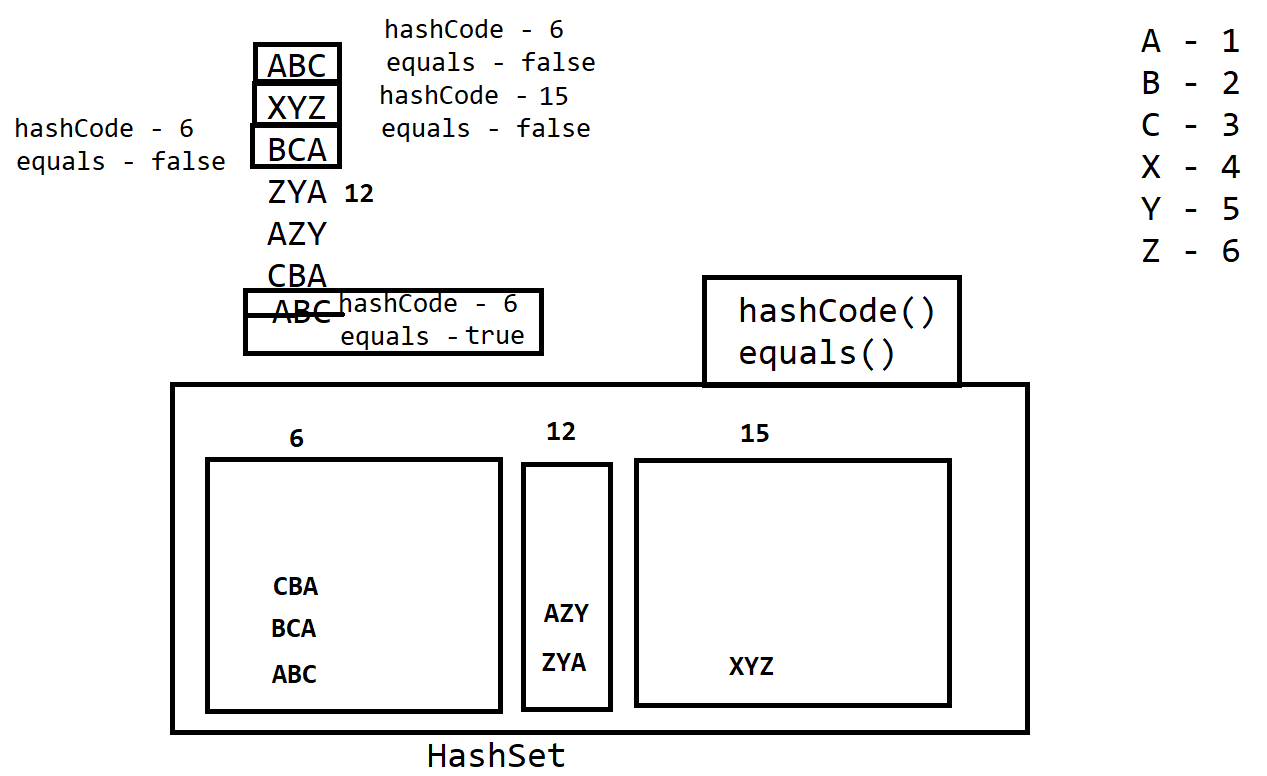
1. LinkedList is the implemented class of List, Queue ,Deque interface.
2. LinkedList is not indexed based.
3. LinkedList can store duplicate objects.
4. It store object of different data type and its dynamic in size.
5. LinkedList is based on Doubly linked list DS.
6. Linked List store the elements in Node form. Every node has a 3 part predecessor(previous), element, successor(next).
7. Linked List are faster for updation.
8. It is slower for iteration.
9. Radom access are not allowed here inside LinkedList.
10. Linked List maintains the insertion order.

**Set Interface**

1. Set is not indexed base.
2. Set is used to store unique values. And not allowed duplicate.
3. Set is dynamic in size and can store different data type of values except TreeSet.
4. Implemented class of Set
   1. HashSet
   2. LinkedHashSet
   3. TreeSet

**HashSet**

1. HashSet implements the Set Interface.
2. HashSet is not indexed base.
3. HashSet is used to store unique values. And not allowed duplicate.
4. HashSet is dynamic in size and can store different data type of values.
5. Is it based on Hashing algorithm.
6. Hashing algos are used for faster searching.
7. HashSet is a unorder and unsorted.
8. To maintain the uniqueness of the values internally hashCode() and equals() methods are called at the time of adding the object inside HashSet.
9. Example:



**LinkedHashSet**

1. LinkedHashSet extends the HashSet class.
2. LinkedHashSet is not indexed base.
3. LinkedHashSet is used to store unique values. And not allowed duplicate.
4. LinkedHashSet is dynamic in size and can store different data type of values.
5. Is it based on Hashing algorithm and doubly linked list algo.
6. Hashing algos are used for faster searching.
7. Linked algo use for faster updation.
8. LinkedHashSet is order, it maintains the insertion order.
9. To maintain the uniqueness of the values internally hashCode() and equals() methods are called at the time of adding the object inside LinkedHashSet.
10. Default capacity is 16 and load factor is 0.75.
11. It increases the capacity of LinkedHashSet after reaching the load factor.

**TreeSet**

1. TreeSet implements Set, NavigableSet, SortedSet interface.
2. TreeSet is not indexed base.
3. TreeSet is used to store unique values. And not allowed duplicate.
4. TreeSet is dynamic in size.
5. Cannot store different data type of values. All values must be of same data type.
6. TreeSet sorts the element by natural sorting order.
7. TreeSet id based on Balance Tree Algorithm.
8. To sort and maintain the uniqueness of the object Comparable or Comparator implementations are used.
9. To add any objects inside TreeSet, the class of an object has to implements Comparable or provide Comparator externally. Else you will get an exception during the run time.

**Comparable**

1. It is an interface which has to implements on targeted (the class whose object trying to add inside TreeSet) class.
2. After implementing the interface you have to override a abstract method

**public** **int** compareTo(Product o)

1. This method returns int has following means
   1. Both Objects are equals

Object1 == Object2 return 0

* 1. First Object value is grater than second Object value.

Object1 > Object2 return + values

* 1. First Object value is smaller than second Object value.

Object1 < Object2 return - values

1. Can provide single comparation technique at a time.

**Comparator**

1. It is an interface which has to implements on other than targeted (the class whose object trying to add inside TreeSet) class.
2. After implementing the interface you have to override a abstract method

**public** **int** compare (Product o1, Product o2)

1. This method returns int has following means
   1. Both Objects are equals

Object1 == Object2 return 0

* 1. First Object value is grater than second Object value.

Object1 > Object2 return + values

* 1. First Object value is smaller than second Object value.

Object1 < Object2 return - values

1. Can provide multiple comparation technique at a time.

**Task**

Find out the following difference

1. Difference in ArrayList and Vector
2. Difference in ArrayList and LinkedList
3. Difference in HashSet and LinkedHashSet
4. Difference in HashSet and TreeSet
5. Difference in Comparable and Comparator
6. Use and implementation of hashCode() and equals() methods

**Map**

1. It is not consider as a collection, because it not implements the property of collection interface.
2. Map is use to store key and value.
3. Each key and value pair are called as entry.
4. Key is always unique in Map and values can be duplicate.
5. Map has multiple implemented classes
   1. HashMap
   2. Hashtable
   3. LinkedHashMap
   4. TreeMap

**HashMap**

1. HashMap implements the property from Map interface.
2. HashMap is also a collection of entries (key and values).
3. HashMap keys must be unique and can be of difference data type.
4. Values can be duplicate and can be of different data type.
5. Map entries are dynamic in size.
6. Based on hashing algo.
7. HashMap entries are un-order and un-sorted.
8. HashMap can contains one null key and multiple null values.

**Hashtable**

1. Hashtable implements the property from Map interface.
2. Hashtable is also a collection of entries (key and values).
3. Hashtable keys must be unique and can be of difference data type.
4. Values can be duplicate and can be of different data type.
5. Map entries are dynamic in size.
6. Based on hashing algo.
7. Hashtable entries are un-order and un-sorted.
8. Hashtable cannot contains null key or null values.
9. Hashtable methods are synchronized.
10. Object of Hashtable is thread safe. Only one thread can access the object of Hashtable.
11. Hashtable is slower in the performance than HashMap.

**LinkedHashMap**

1. LinkedHashMap extends the HashMap and implements the property from Map interface.
2. LinkedHashMap is also a collection of entries (key and values).
3. LinkedHashMap keys must be unique and can be of difference data type.
4. Values can be duplicate and can be of different data type.
5. Map entries are dynamic in size.
6. Based on hashing algo and doubly linked list algo.
7. LinkedHashMap entries are order (maintain an insertion order) and un-sorted.
8. LinkedHashMap can contains one null key and multiple null values.

**TreeMap**

1. TreeMap implements the property from Map, NavigableMap, SortedMap interface.
2. TreeMap is a collection of entries (key and values).
3. TreeMap keys must be unique and must be of same data type.
4. Values can be duplicate and can be of different data type.
5. Map entries are dynamic in size.
6. TreeMap is based on balanced tree algo.
7. TreeMap stores entries in sorted by key.
8. Null keys are not allowed in TreeMap but can add null values.

**Wrapper Classes**

1. These are the Java classes which is provided for all the primitive data type.

|  |  |
| --- | --- |
| **Primitive type** | **Wrapper Class** |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |