**Core Java**

Basic Core Java

Core Java intro, Setup for Core Java, Writing, compilating and running first java program, Variables and Data types, Control flow statement (if, switch, while, do-while and for), Class, Object, methods, Array, Build in java classes (String, Scanner)

OOPs In Java

Inheritance, Polymorphism, Abstraction and Encapsulation. Constructor, Keyword (Super, this , final static), packages, imports.

Advance Topics in Core Java

Exception Handling, Threading, Collection, IO handling, JDBC.

JDK Download

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

Open JDK (32bit OS)

<https://developers.redhat.com/products/openjdk/download>

jdk-8u312-x86 MSI OpenJDK 8 Windows 32-bit

JavaDocs

<https://docs.oracle.com/javase/8/docs/api/>

**Java Setup Setps**

1. Download JDK
2. Install JDK
3. Verify Installation
   1. Go To C driver -> Program File folder -> Java -> JDK
4. Setup a Path of Java into Environment Variable
   1. ***Set JAVA\_HOME***
      1. Right Click on “This PC” / ”My Computer”
      2. Select **Properties** option
      3. Select “Advance System Settings” from the right side of the new window.
      4. Click on “Environment Variables…” button on new Window.



* + 1. Click on “**New**” button from the **System variable** section of the new window.



* + 1. Set the Variable Name and Variable Value on new Window.

**Variable Name : JAVA\_HOME**

**Variable Value : <JDK-Path> “C:\Program Files\Java\jdk-11.0.11”**



* 1. ***Set Path*** 
     1. Follow above step from 1 to 4
     2. Select “Path” variable from the System variable section.
     3. Click On “Edit” button
     4. Click on “New” button from the new Window.
     5. Add following Variable values:

***%JAVA\_HOME%\bin -> Windows 8 and above***

***;%JAVA\_HOME%\bin; -> Windows 7 and less***



1. **Verify Path set or not**
   1. Open a CMD
   2. Type Command

“javac”

“java -version”



**Java Language**

1. Java is use to write a programs/ to develop an application (software).
2. Can develop web application, Console based application, Mobile Application, desktop application, embedded application.
3. Java introduce in 90s.
4. Java Editions
   1. Java SE (J2SE)
      1. Java Standard Edition
      2. Also Known as Core Java
      3. Can develop console based and desktop applications.
   2. Java EE (J2EE)
      1. Java Enterprise Edition
      2. Also known as Advance Java
      3. Can develop Web Applications.
   3. Java ME (J2ME)
      1. Java Micro Edition
      2. Can develop Embedded and mobile application.

**Writing First Java Program**

1. **Write Java Code**
   1. Open Editor (Notepad)
   2. **Write a java program**
      1. Create a Java class.
      2. Everything in java must be write inside a class excepts import and package statement.
      3. Syntax:

**public class <Class\_Name>**

**{**

**}**

1. Create a Main method inside java class.

Main method is the start point of java program.

Inside method you can write an executable statement.

Syntax:

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

**{**

**}**

1. Write an executable statement.

System.out.println(“Hello, Welcome to Core Java First Program”);

1. **Save the file**
   * 1. File must have **.java** extension.
     2. File name must be a public class name.
2. **Compile java code**
   1. Here, java code will be converted from higher level language to lower-level language (Byte/binary code)
   2. Also, it will verify the syntax of the program while compilation.
   3. Open a CMD

The path of the CMD must be pointing to a location where you save your java file. (Go to Folder where you saved your java file and type “cmd” in the address bar and hit Enter.)

* + 1. Execute the following command

**javac FileName.java**

* 1. As an outcome of compilation step .class file will be generated which is a binary file.
  2. This file will be use further to execute java code.
  3. .class files create for the java classes from the java file.

1. **Run/Execute java code**
   1. Execute code using CMD
   2. To execute use following command

**java CassName**



1. One source file can have a more than one classes.
2. One source file must have only one public class all other classes must be a non public.
3. Source file name must be a same as public class name.
4. For every class, from the source file will get a separate .class file after compilation.

**Useful Components for Java Program**

* + - 1. JDK:
         1. Java Development Kit
         2. This is useful for the developer, It is consist of many components which helps java developer to create and execute java program.
         3. JDK consist of JRE (till jdk1.8), JVM, API, dev tools like javac, javap, javadocs , javaw
         4. JDK must be present on developer’s system.
      2. JRE:
         1. Java Runtime Environment.
         2. JRE use to execute the Java programs and you cannot compile the java code just by using a JRE.
         3. JRE must be present at client system.
         4. JRE consist of JVM and API.
      3. JVM
         1. Java Virtual Machine.
         2. JVM is help us to execute the java program.
         3. During the execution of java program, JVM will create memory virtually on the RAM, create and manage a thread to execute a java program.
         4. JVM works in 3 steps

**Class Loader**: in this stage JVM load a .class file into a memory.

**Byte Code Verifier**: Verify the byte code file whether it is compiled from the valid compiler, valid version, it no modified after compilation or not.

**Execution**

* + - 1. API
         1. Application Programming Interface
         2. Predefine functionalities, these functionalities will be provided through a java classes, interface and jar (collection of .class file) files.

**Keyword, Literals and Identifier**

**Keyword**

1. Keywords are the predefine words or reserve words by Java language.
2. These words cannot be use for custom requirement.
3. There are 52 keywords present inside java.
4. Java is case sensitive languages. All the keywords are in small cases.
5. Examples:

public, class, void, static, if, while, continue, do, for, private, protected, default, case, break, int, short, long, byte etc.

**Literals**

1. Literals are the values.
2. There some values which is predefine by java. These values are also included in keyword.
3. There are 3 literals

**true, false, null**

**Identifier**

1. Identifier are the words which can be define by the user/developer for own purpose.
2. Identifiers are use for class name, method name, variable name, object name, constants.
3. There are rules and conventions for creating and using Identifier.
4. Rules for Identifier
   1. Identifier are case sensitive.
   2. Identifier must not be a keyword.
   3. Identifier can contain character, numbers and symbols ($ \_).
   4. Allowed symbols are **$** and **\_**
   5. Identifier must be start with character(alphabets) or symbol, Identifier never starts with number.
   6. Spaces are not allowed in identifier.
   7. There is not length limit to create an identifier.
   8. Examples

|  |  |
| --- | --- |
| **Identifier** | **Valid/Invalid** |
| Int | Valid |
| int | Invalid |
| FirstClass | Valid |
| $salary | Valid |
| 1stClass | Invalid |
| Demo\_Program | Valid |
| New-class | Invalid |
| \_$\_ | Valid |
| \_100 | Valid |
| email@id | Invalid |
| \_ | Valid |

1. Conventions to create class
   1. First character of class name should be in capital case.
   2. If class name is combination of more than one words. Then every first character of the word should be in capital case.
   3. Example:

First, String, System, EmployeeDetails, EmployeeAdderssInfo

1. Convention to create methods, variables, object
   1. First word should be in small case.
   2. If identifier is combination of more than one word then, from the 2nd word onwards every words starts with capital case.
   3. Example:

out, args, main, println, printDetails, showEmployeeDetails

1. Convention to create constants
   1. All constants should be in capital case.
   2. If it is a combination of multiple words then use ‘\_’ between the words (Sanky way).
   3. Example

PI, COMPNAY\_NAME, GRAVITY

public class First

{

public static void main(String args[])

{

System.out.println("First Program");

}

}

**Keyword**: public, class, static, void

**Identifier**: First, main, String, System, args, out, println

**Data Type and Variables**

**Data Type**

1. Is use to assign a type of data.
2. There are 2 categories of Data types.



**Integer Type**

1. Numeric type of value without any decimal point.
2. These values can be a negative or positive.
3. Example: 112, -234, 346, 0

**Decimal Type**

1. Numeric type of values with decimal point.
2. These values can be negative and positive
3. Example: 234234.23, 3.14, 9.8

**Textual Type**

1. Can store a single character/symbol or even an integer type value
2. In this data type ASCII values can be store.
3. Char values cannot be negative hence, it is also known as unsinged integer.
4. Example: N, 78, @, 64, 0

**Logical Type**

1. Can store **true** or **false** inside this type
2. In java 0 consider as false and 1 consider as true internally, but you cannot store 0,1 values as a Boolean value.
3. Example: true, false

**Variable**

1. Variables are use to store a value/data.
2. Use to print the values as an output.
3. Use to assign values to any another variable.
4. Use in a match or logical expression (calculation).
5. Variables are use with the data types only.
6. Syntax to primitive create variable

**Datatype Identifier(Name) = value;**

**byte id = 12;**

****

**int a; // declaration of variable.**

**a = 20; // initialization of variable**

**Formula To calculate data type Range**

**- 2bits-1 to 2bit-1-1**

Can find the range of values for any data type except char and Boolean

byte: -28-1 to 28-1-1 : -128 to 127

**Rules For creating variable**

Rule for Float values

1. Float variable values must be use with a suffix ‘F’ or ‘f’

Rule for Long values

1. Long variable values must be use with a suffix ‘L’ or ‘l’

Rule for char values

1. Char values must be created inside single quotes.
2. Values must be a single char, symbol or any non-negative (ASCII) int value

**Important points**

1. The numeric value without any decimal point and data type is always get a memory of int.
2. The numeric value with decimal point and without data type is always get a memory of double.

**Casting in Primitive Data types**

1. Casting is a process in which one data type value converts into another data type.
2. There are 2 types of casting in java
   1. **implicit Casting**
      1. The casting perform internally automatically by Java is called implicit casting.
      2. There are not loss of any bits int this type of casting.
      3. Example:

int x = 10;

long y = x; // implicit casting

System.out.println(y);

* 1. **Explicit Casting**
     1. The casting has to perform the developer manually is called as explicit casting.
     2. There is always a loss of bits in this type of casting. And due to this it may give you logically incorrect output.
     3. Example:

byte b = 10;

b = **(byte)** (b + 1);

int x = 10;

short z = (short) x; // explicit casting

System.out.println(z);

**Values Representation**

1. Binary
   1. The Values which are start with 0b or 0B are the binary values
   2. These values are allowed in java from JDK 1.7 and onwards version.
2. Octal
   1. The Values which are start with 0 are the octal values
3. Decimal
   1. These are the regular values.
4. Hexa-Decimal
   1. The Values which are start with 0X or 0x are the hexa-decimal values

**Types Of Variables**

1. **Local variable**
   1. The variables which are created inside a method or at a methods parameter are called local variable.
   2. This variable scope (accessible) is only inside a method.
   3. Local variables have initialize before use.
2. **Instance Variable**
   1. The variables which are created inside a class and outside any method are called instance variable.
   2. Instance variable can be access inside class by any of the non-static method directly and can be access by object (instance) outside the class.
   3. Instance variable will be initialized by java automatically if no values provided.
3. **Class/Static Variable (global variable)**
   1. The variables which are created inside class and outside any method with static keyword are classed static/class variable.
   2. Static variables can be access inside class directly in any of the method. And also can be access outside class using only class name and without object of class.
   3. Static/class variable will be initialized by java automatically if no values provided.

**Example**

public class VariableDemo6

{

int a = 10; // Instance Variable

static int c = 30; // static/class Variable

public static void main(String ar[])

{

int b; // Local Variable

}

}

**Default Values**

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

**Note:** These default values are only applicable for an Instance and Static/class variable.

**Operators**

**Arithmetic Operator:** (always return some values except boolean)

+, -, \*, /, %

**Assignment Operator:** (always return some values except boolean)

=, +=, -=, \*=, /=

**Increment/Decrement Operator** (always return some values except boolean)

++, --

To increment and Decrement value by one.

These operators are use to increment/decrement the value and also assign the values.

There are two types of increment/decrement operations can be perform by this which Pre (++a, --a) and post (a++, a--)

Post operation: first assign the value then, increment/decrement the value of the variable.

Pre Operation: first increment/decrement the value of the variable then, assigns the value.

**Relational Operator:** (always returns Boolean (true/false) value)

>, <, >=, <=, ==, !=

These operators are used to compare the values.

These are use in the logical expression.

**Logical Operator**: (return numeric values and also Boolean values)

&, |

**Short Cruciate Operator:** (use for only logical expressions and always return Boolean value)

&&, ||

**Shifting Operator:** (Use for a numeric values)

>> (Right Shift) = Number / 2Bit Shift

<< (Left shift) = Number \* 2Bit Shift

**Comments**

1. To provide an information, write a documentation or to skip any line from the execution the comments are used.
2. There are 3 types of comments
   1. Single line comment
      1. To Comment out a single line

Syntax

// comments line

* 1. Multi line comment
     1. To comment out group line(multiple line)
     2. Syntax:

/\*

Lines

\*/

* 1. Documentation comments 
     1. Use to provide the documentation for the code.
     2. The information which is provided by this comment will be store inside java docs.
     3. Syntax:

/\*\*

lines

\*/

**Task**

1. Write a program display their sum of two number.

2. Write a program to calculate area and perimeter of Square

3. Write a program to calculate area and circumference of Circle.

4. Write a program calculate simple interest.

**Control Flow Statement**

1. To control the flow of java program execution.
2. There are 3 types
   1. Sequential statement execution
      1. This is the default execution of the java code.
   2. Conditional Statement
      1. You can manage the execution of line based on specific condition.
      2. To achieve this can use if statement and its variations, switch cases.
   3. Looping Statement
      1. You can execute a block of lines multiple time.
      2. To Achieve this can use while, do-while, for, enhance for loops.

**Conditional Statement**

1. Conditional Statement is use to execute a specific line or block of line based on condition/scenario.
2. There are multiple ways to achieve this
   1. If conditional statement and its variations like if, if-else, else-if, nested if.
   2. Switch cases.

**If Conditional Statement**

1. **if Statement** 
   1. if is a keyword, it must be followed with a Boolean expression.
   2. The expression which return either true or false is a Boolean expression
   3. The if curly brackets are optional if there is single line of execution is there of if statement. But it is always a good programming practice to use this brackets.
   4. The statement(s) from the if brackets only executes if condition is true.

Syntax:

if(Boolean Expression)

{

Statement(s)

}

1. **if-else statement**
   1. else is also a keyword.
   2. Else cannot be use alone and it must be followed with if condition.
   3. In this structure, if bracket is executes on the condition true, otherwise else statement(s) will executes.
   4. Syntax

if(condition)

{

Statement(s)

}

else

{

Statement(s)

}

**else-if statement**

1. If you wants to check the condition for the else part then can use else if structure.
2. This structure is also known as else-if ladder.
3. In this structure, if any of the condition becomes true then, it will never execute the other conditions.
4. Syntax

if(condition)

{

Statement(s)

}

else if(condition)

{

Statement(s)

}

else if(condition)

{

Statement(s)

}

else

{

Statement(s)

}

**Nested if statement**

1. One if has another if is called nested if.
2. You can also write if structure inside else.
3. Syntax:

if (Condition)

{

if(Condition) {

}

}

**Switch Cases**

1. Switch statement has a cases.
2. switch is a keyword, in switch structure case, break, default keywords are used.
3. Syntax:

switch(Value)

{

case label:

Statement(s)

break;

case label:

Statement(s)

break;

default:

Statement(s)

}

1. Rules to use Switch cases
   1. Switch value must be a **byte, short, int, char, enum(JDK 1.5), String(JDK 1.7).**
   2. Case labels must be unique.
   3. Case label data type must be same as value data type.
   4. Case labels are must be constant (fixed value).
   5. Break is not a part of syntax (not mandatory), you may get an logically incorrect output if you don’t write break statement.
   6. default can be write at any of the location in switch case. It need not to be at the end of switch.
   7. If multiple cases has similar execution then you can combine the multiple cases.

Task:

Print the following output using Switch case

char ‘M’ or ‘m’ print **Monday**

char ‘T’ or ‘t’ print **Tuesday or Thursday**

char ‘W’ or ‘w’ print **Wednesday**

char ‘F’ or ‘f’ print **Friday**

char ‘S’ or ‘s’ print **Saturday or Sunday**

**Looping statement**

1. These types of statements are used to execute a line of block of line multiple times.
2. There are 3 types of Looping statement
   1. while
   2. do-while
   3. for
   4. Enhance for loop (Used for only a collection)

**While Loop**

1. While loop is use to execute statements multiple time.
2. While loop is pre condition check i.e. it will check the condition before entering inside the while execution body.
3. Syntax:

Variable declaration and initialization (start point of the loop)

while(Boolean-Expression)

{

Statement(s)

Increment/decrement variable

}

**do-while Loop**

1. do-while loop is use to execute a statement multiple time.
2. Do-while is post condition check i.e. it will first execute the execution body once and then check for the condition.
3. In this case loop execution body executes at least once.
4. Syntax

Variable declaration and initialization (start point of the loop)

do

{

Statement(s)

Increment/decrement variable

}

while(Boolean-Expression);

**for loop**

1. For loop I use to execute statement(s) multiple time.
2. If you know start and end point of the loop then for loop will be prefer.
3. Syntax:

for (declaration &initialization **;** Boolean-expression **;** increment/decrement/statement(s))

{

Statement(s)

}

1. The while is divide into 3 parts by 2 semi-colons.
   1. 1st part is optional (can be blank) or can initialize the variable.
   2. 2nd part is optional (can be blank) or can write a Boolean expression. If it kept blank then by default it considers a true.
   3. 3rd part is also optional or can write increment/decrement or even a statements, which must be separated by comma (,)

Task

1. Write a program to print the table of any number using while and for.

Example: if num = 5

5

10

15

20

.

.

.

50

**Nested Looping**

**Nested For Loop**

1. One for loop inside another for loop.
2. These types of for loop are mostly use to work with row and column (table) structure.
3. Syntax:

for ( ; ; ) { ------------🡪 Row

for ( ; ; ) { ------🡪 Column

}

}

Task:

Print the pattern (Hint: pyramid patter user 2 inner foe loop int to manage spaces and another to manage \*)

\* \* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \*

**Enhance For Loop**

1. In loop is use to iterate values from the collection.
2. In this loop index will maintain internally.
3. This loop is use to iterate all the values from the collection, and you cannot perform indexed based iteration using this loop.
4. Using this loop you can avoid a changes of ArrayIndexOutOfBoundException at runtime.
5. Syntax

for( DataType variable **:** collection )

{

}

Example:

int[] percent = {78,89,56,79,78,54,67,87,78,45,51,78,87};

for(int var : percent)

{

System.out.println(var);

}

**Array**

1. Array is a group of values/object.
2. It’s a group of same type of values/object.
3. Array is fixed in size.
4. To use array you must know the size and the type of values.
5. Array are a non-primitive data type.
6. Array is an indexed based.
7. Index will be created and maintain internally.
8. Index are always start from 0.
9. If you wants set or get (access) the values from array you have to use index.
10. To create Array, have to follow 3 steps
    1. Array Declaration.
    2. Array Instantiation (Object creation).
    3. Array Initialization.
11. Arrays are of different types, like 1-D array, 2-D array, Multi-dimensional (Jagged Array).
12. If you handle index improperly then you will get an ArrayIndexOutOfBoundsException

**1-D Array**

1. In 1-D array can store multiple values in row format.
2. Syntax:

**Array Declaration**

**Data-type identifier[];**

Example: double percent[];

**Array Instantiation (Object Creation)**

**identifier = new Data-type[Size];**

Example: percent = new double[7];

**Array Initialization**

**identifier[index] = value;**

Example: percent[0] = 67.87;

1. While creating object of an array, we must have to assign size of array.
2. While declaration of array you can create array bracket anywhere except before data type.
3. To Access the values from array you have to use index.
4. After instance creation of an array every block of array will be initialize by default value.
5. Total number of values can be get using length function in array.
6. **array.length** this will return the total number of values present inside array.
7. **array.length - 1**  : To get the last index of array.



1. Different Ways to create 1-D Array

int arr[]; // declare

arr = new int[5]; // instance creation

arr[0] = 10; // initialization

arr[1] = 20;

arr[2] = 30;

arr[3] = 40;

arr[4] = 50;

int arr[] = new int[5]; // declare and instance creation

arr[0] = 10; // initialization

arr[1] = 20;

arr[2] = 30;

arr[3] = 40;

arr[4] = 50;

int arr[] = new int[] {10, 20, 30, 40, 50}; // declare ,instance creation and initialization

int arr[] = {10, 20, 30, 40, 50}; // declare and initialization (instance creation happens internally)

**2-D Array**

1. Can store values in row and column structure.
2. Syntax:

**Array Declaration**

**Data-type identifier[][];**

Example: double percent[][];

**Array Instantiation (Object Creation)**

**identifier = new Data-type[Row\_Size][Column\_Size];**

Example: percent = new double[3][4];

**Array Initialization**

**identifier[row\_index] [column\_index] = value;**

Example: percent[0][1] = 67.87;



1. Length in 2-D

percent.length // total number of rows

percent[0].length // total number of values in 0th row

3. Different Option to create 2-D Array

double percent[][];

percent = new double[3][4];

percent[0][0] = 56.78;

percent[0][1] = 77.78;

double percent[][] = new double[3][4];

percent[0][0] = 56.78;

percent[0][1] = 77.78;

double percent[][] = new double[][] { {78, 65, 78, 89}, {78, 87 , 67, 89}, {87, 67, 65, 90} } ;

double percent[][] = { {78, 65, 78, 89}, {78, 87 , 67, 89}, {87, 67, 65, 90} } ;

**Multi-Dimensional Array (Jagged Array)**

1. The Row size is fixed and column size is differ for every row.
2. Example:

double percent[][] = {

{67, 79, 80, 67},

{76, 56},

{56, 78, 99}

};

**Class, Methods and Object**

**Class**

1. Class is a collection of state/Data/Variable and behavior/function/method.
2. Along with Variables and method you can also create constructure, create another class which is known as inner class.
3. Classes are of 2 types
   1. Build-in class: Is a pre define class by Language.
   2. Custom Class: is created by developer or manually.
4. Syntax:



**Method**

1. Methods are the collection of variables and the executable statement.
2. Along with variables and statements you can also create class inside method (Inner class).
3. Method can accept input parameters/Arguments which is use during the execution of the method and also can return a value after execution.
4. Method can have multiple input parameters but can return maximum 1 value at a time.
5. Methods have to call/access manually to execute.
6. Syntax:



1. You cannot create one method inside another method.
2. There are 2 types of method
   1. Instance method
      1. Instance method can be access/call by using class object only.
      2. You can access/call another instance or static method of same class directly without object.
   2. Static/class method
      1. To access/call the static method you do not required object, and this method can be called using Class Name.
      2. Static method can access/call only static variables and method of same class directly.

**Object**

1. Object is an instance of class.
2. Objects are use to present the Data member and member function of the class.
3. Each Object has separate Heap memory and independent existence.
4. Using object you can access/execute/call the members (data and function) of class.
5. To access the members of class you have to use dot (**.**) operator.
6. Syntax:



**Memory Footprints for Objects**



Stack Memory:

1. Inside Stack Memory method executions are happed.
2. All the local variables are created inside this memory.
3. Stack is based on LIFO (Last In First Out) structure.
4. Stack keep on clearing a memory after completing the execution of the method.
5. The method which enters last in the stack memory will clear first.

Heap Memory:

1. All the objects will be created inside Heap Memory.
2. This is the sharable memory between all the methods and all the threads.
3. Heap memory will be clean up using a special process called as Garbage Collection (GC) which is done by java internally after every interval of time.

**String In Java**

1. String is a non-primitive data type.
2. String is an array of character.
3. In Java String values are created inside double quotes (“Value”)
4. To create and use string Java provided various build-in classes.
   1. String Class
   2. StringBuffer Class
   3. StringBuilder Class
   4. StringTokenizer Class
5. By Using these classes can store string value also can perform operations on String.

**String class**

1. String class is a java **build-in class**.
2. This class is present inside **java.lang package**.
3. String class is a **final class**.
4. Using String class can Store values of array of character type. String value always store in array of char format and hence every char is store into a separate index.



1. String class provided multiple function/methods using which can perform operation on existing string value.
2. String class **object is immutable**. Once assign a value to a string its values remain same even after applying any of the function of String class.
3. To use a String in java program, you have to create Object of String class.
4. There are 2 ways to get the Object of String.
   1. With new Operator.

Example: **String str = new String(“<Value>”)**

* 1. Without new Operator.

Example: **String str = “Value”;**

1. The memory allocation for the String objects created without new operator will be allocated inside String Constant Pool (SCP).
2. SCP will store the String Object, but while storing an Object it will check whether already same value object is present inside memory or not. If object with same value is present then, it will not create new object and return same memory allocation else, the new object will be created.

****

Comparing Values Using == (Relational Operator)

1. Comparing Primitive variable using this operator will always give you logical correct answer.
2. Comparing object (non-primitive) variable using this operator will give you logically incorrect answer.

To Compare two object you can use .equals() function instead of == operator. If you use == operator for object it will compare their references instead of their values.

**StringBuilder class**

1. Using this class can **store String values**.
2. This is a java **build-in class.**
3. StringBuilder class is present **inside java.lang package**.
4. To Store the value inside StringBuilder you have to create object of this class.
5. There are multiple function/methods present inside this class to perform operation on string.
6. **SCP is not applicable** for StringBuilder.
7. StringBuilder **objects are mutable**. By Applying any of the function of StringBuilder the original value keep on changing.
8. StringBuilder is a **final class**.

**StringBuffer class**

1. Using this class can **store String values**.
2. This is a java **build-in class.**
3. StringBuffer class is present **inside java.lang package**.
4. To Store the value inside StringBuffer you have to create object of this class.
5. There are multiple function/methods present inside this class to perform operation on string.
6. **SCP is not applicable** for StringBuffer.
7. StringBuffer **objects are mutable**. By Applying any of the function of StringBuilder the original value keep on changing.
8. StringBuffer is a **final class**.
9. All the method of **StringBuffer is synchronized**.
10. Object of **StringBuffer is thread safe**. At a time only one thread can perform an operation on StringBuffer object. Other thread has to wait until completing execution of first thread.
11. **StringBuffer is slower in performance** than StringBuilder

**StringTokenizer class**

1. StringTokenizer class is present inside java.util package.
2. This class is use to convert string into tokens using delimiter.
3. To iterate on tokens it have provided multiple function.

FAQs

1. What is String class?
2. What is SCP in String?
3. Difference in StringBuilder and StringBuffer?
4. Difference in String, StringBuilder and StringBuffer?
5. What is Immutable and why String is Immutable?
6. How to compare two String values?

**Input From User**

1. Command Line Argument.
   1. In This technique you can accept values from the user at the time of giving execution command.
   2. These arguments (values) are received inside String array of the main method.
   3. And all the arguments are store into a separate index of the String array.
   4. All arguments are always received into a string format.
   5. Values has to provide before start execution of the program.
   6. If user do not pass expected values in in a expected format then there are chances of getting exception.
2. **Scanner Class.**
   1. Scanner is a build-in class.
   2. This class is use to accept values from user using cmd/console, file, String.
   3. Scanner class is present inside **java.util package**.
   4. This class is a final class.
   5. Using Scanner can accept values from the user at run time.
   6. Using Scanner you can accept values of any data type.
   7. To use scanner you have to create Object of this class and can use the function to accept the values.
   8. To accept the values from the user use nextXXX() methods which is provided for different data type.
   9. Close the scanner object once you are done by using it, this will avoid the memory leak issues.

**Eclipse IDE**

1. Eclipse is an IDE for writing a code.
2. In the IDE you can get the project structure, Code Editor and Console inside a same window.
3. IDE also provide the intelligence.
4. Provide a real/live time compilation.
5. IDE also generated Boiler (repetitive) plate code.
6. Download Eclipse IDE.

<https://www.eclipse.org/downloads/packages/>



1. Extract a zip file which is downloaded.
2. Click on the eclipse application file.
3. Create/select workspace (a location where all java project will be created). And then click on Launch.
4. Set the perspective of the eclipse to work on core java application.



**Create First Java Project**

1. Go To “File” Menu -> Go To “New” Menu item.
2. Select “Java Project” option
3. Provide Project Name and Unchecked the Module check box at the bottom of the new window.
4. Click on “Next” and then “Finish” button
5. Java Project structure will be as follows



**Steps To create First Java Class in Project**

1. Right click on “src”
2. Got to “New” option and select “class” option.
3. Provide a class Name and select check box to create main method.

**OOPs (Object Oriented Programming)**

1. Class and object are a basic of the OOPs.
2. There are 4 main OOPs concepts are there which is implemented by Java.
   1. Encapsulation
      1. Wrapping of data member (variables) and member function (methods) into single unit.
   2. Inheritance
      1. One class inherits the properties of another class using extends keyword.
   3. Polymorphism
      1. One thing can be use in a different way.
   4. Abstraction
      1. Hiding the complexity and showing only the required functionality to the user.

**Encapsulation**

1. Wrapping of data member (variables) and member function (methods) into single unit.
2. Class is also an example of Encapsulation.
3. Java recommends to access data member of class using method and not to provide access directly outside class.
4. This can be achieved by hiding data member and providing access to them using getter and setter method.
5. Getter Methods are use to get the value of the variable. Getter methods always return values and you can use “get” as a prefix for method.
6. Setter Method is use to set the value of the variable. Setter method will return void and always accept a value which assigns to an instance variable, you can use “set” as a prefix for a method.
7. To hide the data member, you can use a private access modifier.

****

1. Advantage of encapsulation
   1. It is a one of the way, to achieve loose coupling.
   2. New code can be added easily and also modify existing code without affecting another code.
   3. Can manage Who can access What.

**Inheritance**

1. One class inherits the properties (Variables, Methods) of another class using extends keyword.
2. There will be a parent and child relation establish between the classes after using inheritance. This relation is also known as **IS-A Relation** in Java.
3. In OOPs concepts there are 5 types inheritance. From these only 3 directly supported in java and 2 are not directly supported but it can be achieve using interface in java.



1. In Java One class can inherits (extends) the property from only one class a time.
2. In Java Object class is a parent class of all the java classes either directly or indirectly.
3. In the inheritance private properties and constructor cannot be inherit in to child class.
4. Advantage of Inheritance
   1. Code Reusability.
   2. Extensibility.
   3. To achieve runtime polymorphism Inheritance is required.

**Object Class**

1. Object class is a build-in class in java.
2. Object class is a parent of all the java classes.
3. In this class common function/methods are provided which can be use in all the classes.
4. Some of the methods of Object class.
   1. **toString():** This method called whenever you print the object of the Java class directly. It will by default return the combination of ClassName@HashCode
   2. **hashCode():** this method return the hash code of the object in int format.
   3. **equals(Object):** is use to compare two object with their references.
   4. **wait(), wait(int), wait(int, long)**: this method is use to pause the execution of current thread. This method is use in case of multi-threading.
   5. **notify(), notifyAll():** this method is use to notify the thread which is in wait state. This method is use in case of multi-threading.
   6. **getClass():** to get the Class object.
   7. **finalized():** this method gets invoked at the time of garbage collection

**Constructor**

1. Is a special method of the class. Which do not have return type.
2. Constructor is use to initialize the instance variable at the time of object creation.
3. To Create constructor, you have to follow the following rules.
   1. Constructor name must be a same name as class Name.
   2. Constructor must not have a return data type.
   3. Constructor can be create using any of the access modifier (public, private, protected, default).
   4. Can create more than one Constructor in a class. Which is also known as constructor overloading.
   5. Every class has a Constructor, if it is not created manually then, java provides a default constructor at the time compilation and if any Constructor provided manually then java never provides default constructor.
   6. Constructors cannot be called directly using object and dot operator, it will automatically called whenever you create object of class.
   7. Every constructor of the sub class by default calls super class default/no-param constructor.

**this, super, static, final Keyword**

**this keyword**

1. This is a predefine java keyword.
2. It is a current class object.
3. Using this keyword you can **access the method, instance variable and constructor of same class**.
4. One constructor can call another constructor of the same class using this keyword.
5. Constructor call must be the first statement in a constructor.



**Super Keyword**

1. Super is a predefine word in java.
2. It is use to **access the method, variable and constructor of the super class**.
3. Super Keyword is mainly used for a super class property.
4. Super is not an object of Super class.
5. Every constructor of the sub class by default calls super class default/no-param constructor.
6. Internally java adds a super keyword in every constructor as a first line.
7. Constructor call must be the first statement in a constructor.

**Final Keyword**

1. Final is a predefine word in java.
2. Using final keyword can create a constant (fixed values, implementations).
3. Final keyword can be used for a variable, method and class.
4. Final variables can be an instance, local or static variables.
5. By Making variable as final, you can fix the value of that variable and these values cannot be change.
6. The implementation of the final method will be fixed. These methods cannot be override.
7. Final classes cannot be inherited. String, StingBuffer, StringBuilder, System class are the build-in final classes.

**Static Keyword**

1. Static is a predefine word in java.
2. Can create static variable, method, classes (inner class), static block.
3. Static properties will load into memory at the time of class loading.
4. Static properties can be access without creating object of class and just by using class name.
5. Static Variables can be created outside method which is also known as class/static variable.
6. You cannot create static variable inside method.
7. Static methods can access only a static property of the class directly, to access non static properties of the class you have to create an Object of the class.
8. In side static method you cannot use this and super keyword.
9. Static variables are the sharable variables between all the objects. That is, objects of the class will not have a separate copy of the static variables.
10. Static class must be an inner class, outer class never can be a static.
11. Static block is use to initialize the static variable. Static block gets invoked before constructor.

**Task: Inheritance, Super and this**

****

**Package, Import and access modifier.**

1. **Package:**
   1. Package is a group of java class having similar functionalities.
   2. Package is a java keyword.
   3. Package is nothing but a folder in the file system.
   4. To create any class inside package, you have to write a package statement as a first statement in a java file.
   5. Package statement must be a first statement in a java file and it must be outside class.
   6. There can be maximum one package statement in a source file.
   7. Syntax:

**package packageName1. packageName2. packageName3;**

1. **Import:**
   1. Import statements are use to include java class from one package into another class of different package.
   2. import is a keyword in java
   3. Import statement must be present inside a source file and outside any class.
   4. Import statement must write after package statement.
   5. There can be multiple import statement in a class.
   6. Every java class by default import the classes from the **java.lang package.**
   7. Syntax:

**import packageName1.packageName2.ClassName; (Syntax to import single class from package)**

**import packageName1.packageName2.\*; (Syntax to import all the classes from package)**

1. **Access Modifier**
   1. You can manage the access of the properties (methods, variables) from the class.
   2. There are 4 types of access modifiers
      1. public
      2. protected
      3. default/package
      4. private

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Access Modifier** | **properties into same class** | **properties of one class into another class of same package using object** | **properties of one class into another class of same package using Inheritance** | **properties of one class into another class of different package using Inheritance** | **properties of one class into another class of different package using 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. Highest to lowest restriction

private (Highest Restrictive)

default/package

protected

public (lowest restrictive)

****

**Polymorphism**

1. One thing can be use in a different way.
2. Single things having a multiple form is a polymorphism.
3. Practically it can be achieved by create a method with same name and different implementation.
4. There are 2 types of polymorphism
   1. **Compile time polymorphism** 
      1. The method call resolve at time of compilation and same will be followed at execution time is called compile time polymorphism.
      2. The example/can be achieve using **Overloading**.
   2. **Run time polymorphism**
      1. The method call resolve at the time of execution is called run time polymorphism.
      2. The example/can be achieve using **Overriding**.
5. **Overloading**
   1. In the overloading method name must be same and the input parameter must be different.
   2. Overloading can be done in the same class or in the sub-class.
   3. In overloading Input parameter list must be different by following options
      1. Can change the datatype of parameter.
      2. Can change the number of parameters.
      3. Can change the sequence of parameter.
   4. Can change the return data type.
   5. Can change the access modifier.
   6. Can throw new broader checked exception.
   7. Can overload final methods.
   8. Can overload static method.
   9. The overloading method will be invoked based on the arguments (value) pass at the time of method call.
6. **Overriding**
   1. Overriding is use to change the default implementation provided by the super class method inside sub class.
   2. In the overriding method name must be same and the input parameter must be same.
   3. Overriding has to do inside sub-class only.
   4. Return data type must be same (if it is primitive or void) or it can be sub-type.
   5. Access modifier can be change but it must be less restrictive.
   6. Private methods cannot be override. It won’t give a compile time error but it will not consider as a valid overriding.
   7. Final methods cannot be override.
   8. Static methods cannot be override.
   9. Cannot throw new and broader checked exception, but can throw the child checked exception.
   10. @Override annotation is provided by java to verify the method overriding at the time of compilation. If you not followed the overriding rules the you program won’t compile.
7. Every Super class reference can hold the object of its sub class. This type of object is known as polymorphic object.

Syntax: **ParentClass ObjName = new ChildClass();**

1. To achieve Runtime Polymorphism, you have to follow the following steps.
   1. There must be inheritance between a class.
   2. There must be a method overriding.
   3. Method call must be happened using polymorphic object.



Task:

1. List the difference between compile time and run time polymorphism.
2. List the difference between Overloading and overriding.
3. What is run time polymorphism explain with the help of program.



**Abstraction**

1. Hiding the complexity and showing only the required functionality to the user.
2. There are two options to achieve the abstraction in Java
   1. Abstract Class
      1. Using abstract class you can achieve 0-100% abstraction (partial or full abstraction)
   2. Interface
      1. Using interface you can achieve 100% abstraction.

**Abstract Class**

1. Abstract class is one of the ways to achieve abstraction in java.
2. Can achieve 0-100% abstraction.
3. Abstract classes are the classes which is declare by using **abstract keyword**.
4. You can create an abstract methods inside abstract class.
5. Abstract method is a method which is created without any implementation and just declare that method.
6. Abstract classes are mostly used as a top-level class in a hierarchy.
7. Abstract classes can be extends by a non-abstract (Concrete) class, here the non-abstract class has to provide implementation (override) for all the abstract method.
8. Abstract classes also can be extend by another abstract class, here there is no need to provide implementation of abstract methods. The abstract methods will be inherited inside sub class.
9. You cannot create the Object of abstract class, but it can be used as a reference.
10. Abstract methods cannot be private.
11. Abstract method cannot static or final.
12. Abstract classes cannot be final or static.
13. You can create static and final method inside abstract class but those methods must a non-abstract.
14. Abstract class can have constructor. These constructors get invoked whenever you crate the object of sub class of abstract class using super keyword.

**Interface**

1. Interface is another way to achieve abstraction.
2. We can achieve 100% abstraction.
3. Interfaces are not a class and it will be created using an interface keyword.
4. Every interface will get a separate .class file after compilation.
5. Every variable in the interface is by default public static and final.
6. Every method of the interface is by default public abstract.



1. Every thing in the interface is public, and you cannot use any other access modifier inside interface.
2. Interfaces are mostly used as a top-level in a hierarchy.
3. Interfaces are always implements on the concrete class or abstract class.
4. If interface implements on the concrete class, then you have to implements (override) all the abstract methods in class.
5. If interface implements by abstract class, then it is not mandatory to implement the abstract, the abstract method from the interface can be carry/inherits inside abstract class as it is.
6. You cannot create Object of interface but it can be used as a reference.
7. You cannot create constructor inside interface.
8. One interface can be extends another interface.



1. One interface can extends more than one interface. And this is the way to achieve multiple inheritance.



1. Cannot create final method inside interface.

**Interface in JDK 8**

1. In JDK 8 onwards you can create implemented method inside interface. That method is called as default method.
2. In interface you can create static method with implementation.
3. You can create main method inside interface and interface can be execute independently.
4. In JDK-8 functional interfaces are introduced which is use to achieve functional programming and for lambda expression.

**Type of Interface**

There are 3 types of interfaces in java

1. Normal/Regular interface.
2. Marker Interface
   1. The interface without any member.
   2. This is also known as blank interface.
   3. Serializable in an example of marker interface.
   4. By looking into a marker interface java provides a special implementation at the time of code execution.



1. Functional Interface (Jdk-1.8)
   1. The interface which must contains only one abstract method.
   2. There can be any number of default and static method inside functional interface, but there must be a one abstract method present inside interface.
   3. This type of interface introduced in jdk 1.8 and onwards.
   4. Java Provided following annotation to make sure that the interface is functional interface or not.

**@FunctionalInterface**

* 1. This type of interfaces is used in function programming and lambda expression.
  2. Java Provided multiple functional interfaces and they are all inside **java.util.function** package

Lambda Expression:

1. Lambda implementation is a way to implement a method of the functional interface.
2. It can be achieved by using lambda symbol

(param) **-> {Implementation}**

**Exception Handling**

**Exception:** Unwanted scenario/situation occurs during the execution of the program and due to this program execution terminated abnormally this is an exception scenario.

**Exception Handling:** It is a scenario where the exception will be handled and provide an alternative way so that we can avoid the program termination and continue program execution by handling the exception.

Keywords used in Exception Handling

1. **try**: try is a block in which you can write a statement which may throw exception.
2. **catch**: catch is block where the exception will be handled which is thrown from the try block.
3. **finally**: finally is a block which executes always irrespective of the execution of try and catch.
4. **throw**: throw is use to throw object of exception manually (raise exception manually). Mostly it used for custom exception.
5. **throws**: throws is use to declare exception to be thrown from the method instead of handling it (propagating the exception)



**Exception Hierarchy**



**Error**

1. Errors are occurs mostly due to a system issues.
2. Errors are non-recoverable.
3. It is recommended not handle error, because you cannot continue program execution on error occurrence.

|  |  |
| --- | --- |
| **Checked Exception** | **Un-Checked Exception** |
| The exception which is checked by compiler | Un-checked exceptions are not checked by compiler. |
| If any checked exception scenario is there inside code than compiler will force developer to handle the exception. | There is no checked on this type of exception and hence program can be compiled if no handling provided. |
| In Case of checked exception program never terminate abnormally. Because exception handling will always be provided for these exception. | In this case program may terminate abnormally if unchecked exception occurs and it is not handled. |
| Exception: SqlException, IOException etc. | Example: NullPointerException, ArrayIndexOutofBoundException etc. |

**try**

1. Try is a keyword which used in exception handling.
2. Try is a block inside this block you can write a lines which my throws exception.
3. Try block must be write with catch or finally or both.
4. From the try block object of exceptions will be thrown which is further cached by catch block.
5. From the single try block there can be multiple exception thrown.
6. Syntax:

**try {**

**statement(s)**

**}**

**catch**

1. Catch block is used to catch the exception thrown from try block.
2. Catch Block catches the object of exception.
3. Inside catch block you can provide an alternative way as an exception handling.
4. Every catch block is used to catch a specific type of exception (While creating catch block you have to provide type of exception).
5. If single try block throwing more than one exception then you can write more than one catch block for a single try.
6. In a try with multiple catch case, you must not handle parent exception before child exception, but parent exception can be write after child exception.
7. Syntax:

**catch(ExceptionType ref) {**

**statement(s)**

**}**

**finally**

1. Finally block is use to write a statement which has be execute always.
2. Finally block will give you guarantee of execution.
3. For single try block there can be one finally block.
4. Mostly it is used for a resource closing purpose.
5. Syntax:

**finally {**

**statement(s)**

**}**

**throw**

1. Throw is use to raise exception manually.
2. Using this keyword you can throw object of exception from the statement level.
3. Throw keyword is used at a statement level inside method.
4. Mostly it is used for a custom exception.

**Custom Exception**

1. Create a java class
2. Extends the java class from Exception class or any of its sub class.
3. Can provide your own implementation for the exception.

**throws**

1. Throws keyword is use to propagate exception outside method.
2. Exception can be thrown outside method at calling level with providing exception handling to it.
3. Exception will always throw at the method declaration level.
4. Throws must be followed with class name.
5. You can throws more than one exception at a time using this keyword.

**JDK 1.7 Enhancements in Exception Handling**

1. **Try with resource**
   1. This feature is provided to avoided writing common code of closing a resource.
   2. Syntax:

**try (Create Resources)**

**{**

**Statement(s)**

**}**

* 1. This feature will help developer to close the resources, which is created inside try block automatically once execution come outside try block.
  2. Internally it follows a try-finally structure.
  3. So that this try with resource can be write without finally or catch block also.

1. **Multiple exception type in Single catch**
   1. If multiple exception having similar execution then you can combine the multiple type of exception into a single catch block.
   2. Syntax:

**catch** (Exception1 | Exception2 | Exception3 e) {

System.***out***.println("Somtime went wrong, try after some time..");

}

* 1. The exception listed out in side catch block must not have a parent child relation.

FAQ:

1. What is exception and What is exception handling
2. What is the different keyword used in Exception handling?
3. What is the use to try and catch?
4. When and why use finally.
5. Difference between Checked and Unchecked Exception
6. Difference between Exception and Error
7. Difference between throw and Throws
8. Difference between finally and final
9. What is custom exception and how to create custom Exception.

**Multi-threading**

What is Thread?

1. Thread is a light weight process or a part of process.
2. Every thread executes independently.
3. Every thread has it own call by stack memory.
4. Thread is a small part/execution of the program.
5. Every java program executed by a thread.
6. The default thread which is created by java is knowns as main thread.
7. You cannot have a full control on the thread execution, thread will be executed by JVM.

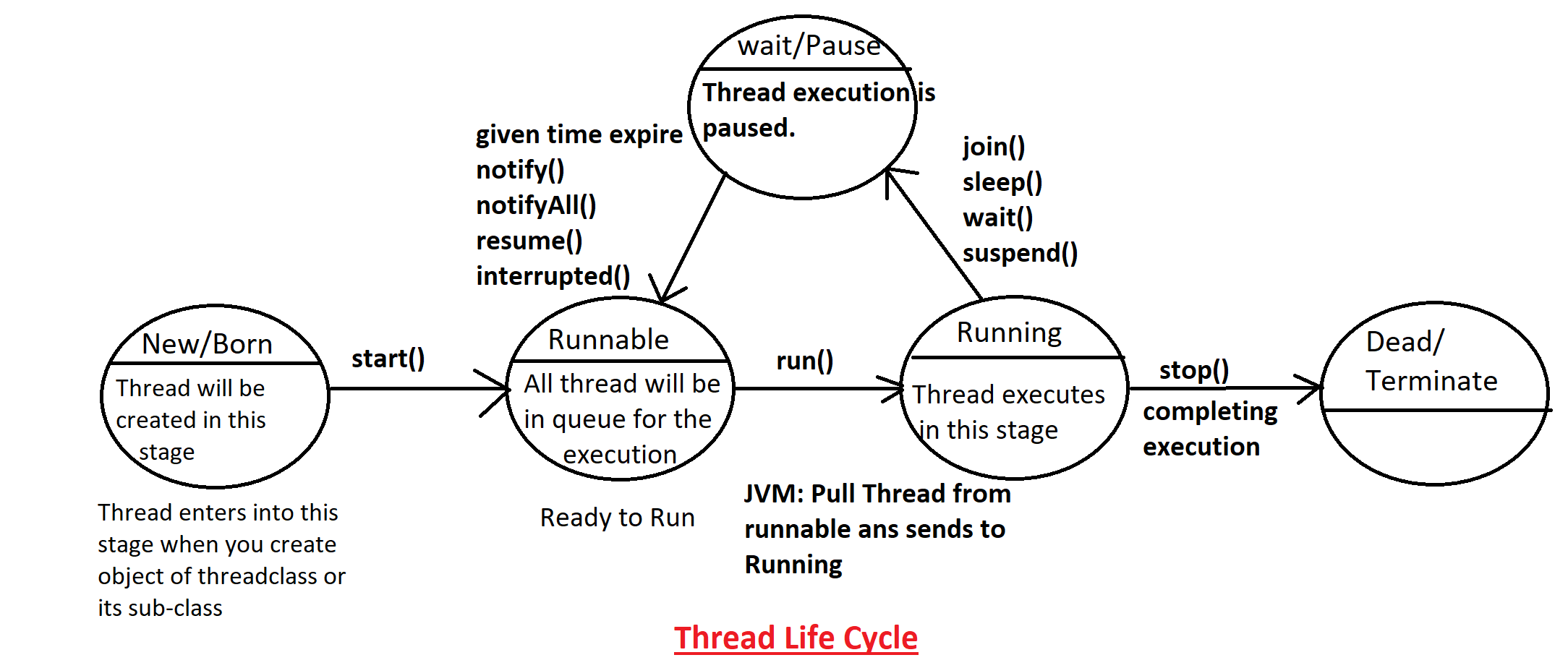
Why Thread?

1. Multi-threaded applications are always faster that single threaded application.
2. In multi-threaded application if any thread failed to complete the execution so it will not directly affect another thread and another can execute independently.
3. Each thread has its own memory so that it can be execute independently.

How To create thread in Java?

1. Thread can be created in java by 2 ways
   1. Runnable Interface
      1. This interface has only one method which “run()”
      2. This interface is also known as function interface.
   2. Thread class
      1. Thread class internally implements Runnable interface.
      2. Thread class has multiple method

**Thread Life Cycle**

****



**Common Methods of Thread class**

|  |  |
| --- | --- |
| **Method** | **Description** |
| currentThread() | This is a static method of the Thread class. This thread return the object of currently executed thread. From this object you can get the thread details like thread name, priority and thread group |
| run() | Using this method you can assign task to a thread. This method will be called by JVM whenever thread start executing. |
| start() | This method is used to make thread ready to run. Thread executing cannot be start by this method. You can just make thread ready for execution. |
| setName(String) getName() | These methods are use to set and get the name of thread. |
| setPriority(int) getPriority() | These methods are use to set and get the priority of the thread. Priority Must be between 1-10. It you use any other value as a priority then It will throw IllegalArgumentException 1 – MIN\_PRIORITY 5 – NORMAL\_PRIORITY 10 – MAX\_PRIORITY |
| ~~stop~~() | This is a deprecated method. Using this method, you can stop the execution of the thread permanently. The stopped thread cannot resume again. |
| ~~suspend()~~ | This is a deprecated method. Using this method, you can pause the execution of the thread until some another thread calls resume method. |
| ~~resume()~~ | This is a deprecated method. Using this method, you can resume the execution of the suspended thread. Here thread will come into Runnable stage from pause stage. |
| sleep(long) sleep(long, int) | This is a static method, you can pause the execution of the thread for a specific milli seconds, this method throws InterruptedException which is checked exception. |
| join()  join(long)  join(long, int) | Using this method, you pause the execution of current thread and it will wait until completing execution of another thread or until the given time expires. |
| wait()  wait(long)  wait(long, int) | This method is present inside Object class, using this method you can pause the execution of the thread and also thread will release the lock. This method has to use in case of synchronization only. This method throws InterruptedException which is checked exception. |
| notify() | This method is present inside Object class, using this method you can notify a single thread which is waiting for object. |
| notifyAll() | This method is present inside Object class, using this method you can notify multiple threads which are waiting for object. |

**Synchronization**

1. This is a process where single thread accesses the shared object at a time.
2. This concept is only required in multithreading and in case of multiple thread using share resource.
3. This is a process where Shared object will be locked and unlocked automatically while thread execution.
4. This concept is use to resolve the data inconsistency issue.
5. Synchronization can be used by 2 ways
   1. Declare method as synchronized.
   2. Can create synchronized block.

Syntax:

synchronized(object)

{

Statement(s)

}

1. Using synchronization can affect the program performance.
2. Synchronization can lead to a dead lock scenario.

**Collection Framework**

1. Collection is a group of objects in java. These objects are also known as elements.
2. There can be a different criteria to store the objects which can be achieved by using collection.
3. Collection has provided different type of classes to achieve different ways to collect objects.
4. Collection is an interface in java which is also consider as a parent interface of all the collection.
5. All the Classes and interfaces present inside collection is available inside **java.util package**
6. The collection is divided into 3 types
   1. **List:** List is indexed Bases and it allows duplicate elements.
   2. **Queue:** Is used for FIFO priority.
   3. **Set:** Set is used to store unique elements.
7. Mostly collection is based on Data Structure.
8. **Collection Hierarchy**



**Collection Interface**

1. Collection is a parent of all the types of collection.
2. Collection is used to store group of objects into a single entity.
3. The Objects can be of different type.
4. Collection interface has some common methods which is present inside all other collection types.
   1. **add(object):** this method is use to insert new Object inside collection.
   2. **remove(object):** using this method you can remove objects from the collection.
   3. **contains(object):** using this method you can check whether Objects are present inside collection or not.
   4. **size():** you can get the total number of objects present inside collection.
   5. **isEmpty():** is use to check whether collection is empty or not.
   6. **clear():** is use to remove all the Object from the collection, and it will make your collection empty.
   7. **addAll(Collection):** this method is use to add multiple objects at a time in side collection.
   8. **removeAll(collection):** this method will remove a group of objects from collection.
   9. **containsAll(collection):** Use to search multiple object at a time inside collection.
   10. **iterator():** This method is use to get the value from collection one by one.

**Wrapper Classes**

1. In the collection you cannot store primitive values
2. All the values must be in the form of object only.
3. If you want to store any values which is of type primitive these types of values can be store as a object of wrapper classes.
4. Wrapper classes are the classes which provided for every primitive type.

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

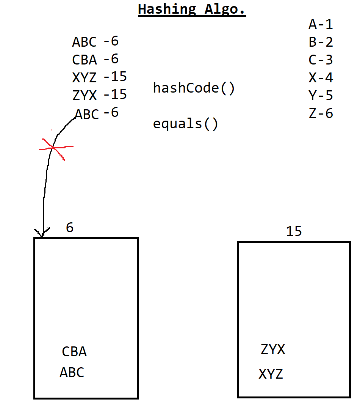
1. Between wrapper class and primitive data type java internally converts the values and this is known as auto boxing.

**Linked**

1. It is based on Doubly Linked List Algo.
2. Elements are store in the order way (insertion order maintained)
3. Updation is faster and iteration is slower.
4. Every element in this store the address of previous element (predecessor) and next element (successor)

**Hash**

1. It is based on the hashing algo.
2. This algo is use to search element in faster way.
3. In the Hashing algo in java elements are store in unorder and unsorted structure.



**Tree**

1. This is based on a balance tree algo.
2. This will store the values in sorted way.

**List Interface**

1. It is Index based.
2. It allows duplicate elements.
3. You can store different type of objects inside collection.
4. It is dynamic in size (Can add remove elements dynamically no fixed size has to provide)
5. List interface inherits the property from Collection Interface.
6. List interface has following implemented classes
   1. ArrayList
   2. LinkedList
   3. Vector

**ArrayList**

1. All the elements inside array list are index based.
2. Can store duplicate elements.
3. Can store values dynamically without any fixed size.
4. Can store different data type of values.
5. All elements inside the ArrayList are order. It will maintain the insertion order.
6. ArrayList is always faster for Iteration but it is slower in the updation.
7. ArrayList implements a RandomAccess interface which allows you to randomly access the values on the specific index.
8. ArrayList is backed by array. Internally all the elements store in form of array.
9. The default capacity of the array list is 10. This can be change using array list constructor.

**Vector**

1. All the elements inside Vector are index based.
2. Can store duplicate elements.
3. Can store values dynamically without any fixed size.
4. Can store different data type of values.
5. All elements inside the Vector are order. It will maintain the insertion order.
6. Vector is always faster for Iteration but it is slower in the updation.
7. Vector implements a RandomAccess interface which allows you to randomly access the values on the specific index.
8. Vector is backed by array. Internally all the elements store in form of array.
9. The default capacity of the array list is 10. This can be change using array list constructor.
10. Vector methods are synchronized.
11. Object of Vector are thread safe, at a time only one thread can perform the operation on the vector object.
12. It is slower in the performance than ArrayList.
13. Vector is introduce in java in the JDK 1 version, so this is also known as legacy class.

**LinkedList**

1. LinkedList implements the properties of List, Deque and Queue interface.
2. It is based on the doubly linked list algo.
3. It is faster for updation.
4. It is slower in iteration.
5. LinkedList is not indexed based. No Array is used to store elements internally.
6. In LinkedList internally Node Object will be store and every Node has a 3 part.(Pre, value, Next)
7. LinkedList maintains the insertion order.
8. Can store different type of values and also duplicate values are allowed.
9. Three is not fixed size for this collection, It is dynamic in size.
10. RandomAccess not implemented by LinkedList

**Set**

1. Set is non indexed based.
2. Set store only unique values.
3. Dynamically growable (Size id not fixed).
4. Can add values for different Data Type. Except TreeSet.
5. There are 3 implemented classes
   1. HashSet
   2. LinkedHashSet
   3. TreeSet

**HashSet**

1. It implements the property from Set Interface.
2. It is non indexed bases.
3. It store only unique values.
4. There no fixed size and store different type of values.
5. It is based on Hash Algorithm.
6. Elements are in the unorder and unsorted form.
7. It provides faster searching.
8. HashSet internally use HashMap to store a values. In the HashMap key will be the element of set and value will be a dummy object.
9. The default capacity of the HashSet is 16.
10. The default load factor is 0.75. Using this you can set the percentage after which the capacity of the set will be increase.
11. hashCode() and equals() method are internally called to maintain the uniqueness of the HashSet.
12. If you are adding any custom class Object inside HashSet then you must implement the hashCode() and equals() metho to get the logically correct output(maintain the uniqueness of object).

**Task:**

Create a class Product with id, name, price fields.

Store 5 products in a collection, where each product must be unique by its ID.

And display a product detail on the console.

**LinkedHashSet**

1. It implements the property from Set Interface and also extends HashSet Class.
2. It is non indexed bases.
3. It stores only unique values.
4. There no fixed size and store different type of values.
5. It is based on Hashing and Doubly Linked List Algorithm.
6. Elements are in the order form but they are unsorted.
7. It provides faster searching and also faster updation.
8. LinkedHashSet internally use HashMap to store a values. In the HashMap key will be the element of set and value will be a dummy object.
9. The default capacity of the LinkedHashSet is 16.
10. The default load factor is 0.75. Using this you can set the percentage after which the capacity of the set will be increase.
11. hashCode() and equals() method are internally called to maintain the uniqueness of the LinkedHashSet.
12. If you are adding any custom class Object inside LinkedHashSet then you must implement the hashCode() and equals() metho to get the logically correct output(maintain the uniqueness of object).

**TreeSet**

1. It implements the property from Set, NavigableSet, SortedSet Interface.
2. It is non indexed bases.
3. It stores only unique values.
4. There no fixed size for the elements.
5. You cannot store different type of values.
6. TreeSet based on the balance tree algo.
7. It stores the values in an ascending order by default.
8. To maintain the uniqueness and sorting of the elements Comparable or Comparator has to use.
9. All the Java build in classes like String, Wrapper classes etc. implements Comparable internally.
10. Any try to use Custom class Objects inside TreeSet will throw exception, if you not implement Comparable or Comparator Interface.

**Comparable and Comparator Interface**

1. These interfaces are used to maintain the uniqueness and the sorting techniques for the TreeSet.
2. Both are the interface which has single abstract method, inside which you have to provide the sorting and uniqueness logic. These methods return an int value as a return data type.
3. Comparable has to use on a target (actual) class and Comparator can be create separately that the targeted class.
4. Logic to maintain the uniqueness and sorting of the elements
   1. Return any +ve int number if 1st Object value grater than (>) 2nd Object value.
   2. Return any -ve int number if 1st Object value smaller than (<) 2nd Object value.
   3. Return any 0 if 1st Object value equal to (==) 2nd Object value.

**Comparable Interface**

1. Implement this interface on a targeted class.
2. This interface is present inside java.lang package
3. This interface has one abstract method

**public** **int** compareTo(Object o)

1. You can provide single sorting technique using this interface.

**Comparator Interface**

1. This interface has to implement on a separate class than the targeted class.
2. The sorting logic will be present inside another class than the targeted class.
3. This interface is present inside java.util package
4. This interface has one abstract method

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

1. You can provide multiple sorting techniques using this interface.

Task:

Create class Product with Id, name, price, quantity fields

Create a “Cart” collection which may contains multiple product object. Each product from the Cart must be unique and sorted by id.

Print the Billing details at the end of the program.

**O/P**

Id Name Quantity Price

121 Smart Watch 1 10000

222 Smart Phone 1 15000

245 Bluetooth speaker 1 5000

Total Payable Amount: 30000

**Map**

1. Map is not a part of collection.
2. Map is use to store a key and value (Entry).
3. Map not implements the property from collection interface.
4. Map has multiple implemented classes
   1. HashMap
   2. LinkedHashMap
   3. Hashtable
   4. TreeMap

**HashMap**

1. HashSet can store entries (Key & value)
2. Key and value can be of different data type.
3. Key must be unique and values can be duplicate.
4. Any try to add duplicate key, it will replace the old value with the new values.
5. HashMap is unorder and un-sorted.
6. It is based on Hashing Algo.
7. The default capacity is 16 and load factor is 0.75
8. In the HashMap one null key and multiple null values are allowed.

**LinkedHashMap**

1. LinkedHashMap extends the properties of HashMap
2. LinkedHashMap can store entries (Key & value)
3. Key and value can be of different data type.
4. Key must be unique and values can be duplicate.
5. Any try to add duplicate key, it will replace the old value with the new values.
6. LinkedHashMap is order and un-sorted.
7. It is based on Hashing and doubly linked list algo.
8. The default capacity is 16 and load factor is 0.75
9. In the LinkedHashMap one null key and multiple null values are allowed.

**Hashtable**

1. Hashtable can store entries (Key & value)
2. Key and value can be of different data type.
3. Key must be unique and values can be duplicate.
4. Any try to add duplicate key, it will replace the old value with the new values.
5. Hashtable is unorder and un-sorted.
6. It is based on Hashing Algo.
7. The default capacity is 11 and load factor is 0.75
8. In the Hashtable no null key or null value allowed.
9. Hashtable method are synchronized and at a time only one thread can perform the operation on Hashtable object.
10. Hashtable is slower in the performance than HashMap.

**TreeMap**

1. TreeMap implements the properties from Map, SortedMap and NavigableMap interface.
2. TreeMap can store entries (Key & value)
3. Key must be of same data type and values can be of different data type.
4. Key must be unique and values can be duplicate.
5. Any try to add duplicate key, it will replace the old value with the new values.
6. TreeMap is sorted by key.
7. It is based on Balanced Tree algo.
8. TreeMap no null key is allowed but can have a null values.

Thread Implementation with Runnable

JDBC (Java Data Base Connectivity)

<https://dev.mysql.com/downloads/installer/>

<https://www.youtube.com/watch?v=7_dv1R-3p1c>