**Full Stack Development in Java**

1. Front End Technologies

HTML, CSS, JS, Jquery, ReactJS/AngularJS

1. Back End Technologies

Core Java, Advance Java, Spring, Spring Boot framework.

1. Data Base

SQL (Oracle or MySql)

1. DevOps Tools

Git, GitHub tool, Maven, Swagger/Open API, Postman.

**Core Java**

1. Basics of Core Java

Writing First Java Program, Compilation, Execution, Variable, Data type, Control flow statement, Array, Class, Object, Methods, Java Build in classes like String classes, scanner class.

1. OOPs (Object Oriented Programming) in Java

Class, Object, Encapsulation, Inheritance, Polymorphism, Abstraction, Constructor, Java Keywords like static, final, this, Super, Package and imports, access modifier.

1. Advance Concepts in Core Java

Exception Handling, Collection Framework, Threading, JDBC, Inner classes.

JDK Download

<https://www.oracle.com/java/technologies/downloads/#java11>

**Java Doc:**

[**https://docs.oracle.com/javase/8/docs/api/**](https://docs.oracle.com/javase/8/docs/api/)

**Setup Java on Local System**

1. Download JDK

<https://www.oracle.com/java/technologies/downloads/#java11>

1. Install JDK
2. Verify Installation
   1. Go to **C Drive / Program Files / Java / JDK 11**
   2. Check for the file and folder like (bin, Config, lib etc..)
3. Setup Java Environment variable
   1. Go to start -> type “Environment”



* 1. Select “Environment Variables..” button
  2. Setup **JAVA\_HOME**
     1. Click “New” button from the System variable section
     2. New Popup will be shown, set the Variable Name and value.

Variable Name: **JAVA\_HOME**

Variable Value: <Path of the JDK folder> **C:\Program Files\Java\jdk-11.0.11**



* 1. Setup **Path**
     1. Select **“Path”** variable from the list of System Variable section. And Click on “Edit” button
     2. Set New variable as“**%JAVA\_HOME%\bin**”



* 1. Verify the setup
     1. Open Command Prompt
     2. And try following command

**java -version**

**javac**



**Write, Compile and Execute First Java Program**

1. Open Notepad.
2. Write a java code.
   1. Create a class
      1. Every thing in java must be write inside a class. Except import and package statement.
      2. Class is a collection of state/data/variable and behavior/function/method.
      3. Syntax:

**public class <ClassName> {**

**// Variable and Methods**

**}**

* 1. Create a Main method
     1. Main method is the start point of the program.
     2. All the programs always start executing from the main method.
     3. Main method internally called and execute by java (JVM) when you run the java program.
     4. Syntax:

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

**}**

* 1. Write an executable statement inside main method.
     1. The statement which executes and produce an output.
     2. Like Print statement. Print statement print the output on the console.
     3. Syntax for print statement:

**System.out.println(“Message”);**

1. Save the File
   1. Save a file into a specific folder using .java extension.
   2. File extension must be **.java**
   3. File name must be a same as public class name.
2. Open a command Prompt (CMD)
   1. CMD is use to compile and execute the java program.
   2. CMD must points to a location where you store you .java file.



* 1. Another option to open a CMD is, open a folder where you create you .java file -> type “CMD” in the address and hit “Enter” Button.

1. Compile the Java Code.
   1. In this step .java (Higher level language) gets converted into .class (lower-level language)
   2. In the compilation stage code syntax will be verified.
   3. As a successful execution of the compilation step, you will get a .class file.
   4. Command to compile java file.

**javac FileName.java**

1. Execute the Java Code
   1. In this step the compiled code will be executed using a JVM.
   2. The main method will be internally called and execute.
   3. Command To Execute Java Code.

**java ClassName**



****

1. While execution of the java program .class files will be used. And for the modification in the java code .java files are used.
2. One file (Source file) can have more than one classes, but there must be a one public class in a source file.
3. The source file name must be same as a public class name.
4. The .class file will be generated for a classes, created inside a source file. For example if one source file contains 3 classes then after compilation of the source file you will get 3 .class files.
5. Every java class from the source file can be execute independently.
6. Class can be compiled successfully without main method, but you cannot execute the class independently using command.

**Keyword, Identifier and literals**

**Keyword in Java**

1. Keywords are the words which is reserve by java language.
2. You cannot use those words for your own purpose like while creating identifier.
3. There is total 51 keywords available in java.
4. In this 3 are a literal which is also consider as keyword.
5. Every keyword in java is in small/lower case.
6. Example:

public, static, void, class, private, protected, default, if, switch, else, while, break, int, short, long, double, char, boolean, true, false, null etc.

**Literal**

1. The values which is reserve by java.
2. There are 3 literals, and all are also considered as a keyword.
3. Literals are: **true, false, null**

**Identifier in Java**

1. Identifier are the words which is used by programmer/developer for own purpose.
2. Identifier are used to identify the entities inside program.
3. Identifier are used for class name, method name, variable name, constant variables, objects etc.
4. There are rules and conventions for the identifier.
5. Rules to create Identifier
   1. Identifier must contain characters, number and symbols ($ \_)
   2. Allowed symbols are $ and \_
   3. Identifier must be start with character or symbol.
   4. Identifier must not be start with a number.
   5. Spaces are not allowed in the identifier.
   6. Identifier must not be a java keyword.
   7. Identifier are case sensitive.
   8. There are not character limits in the identifier. But the recommended is to create short and meaningful identifier.
   9. Example

|  |  |
| --- | --- |
| **Identifier** | **Valid/Invalid** |
| Employee | Valid |
| Email Id | Invalid |
| 1StNumber | Invalid |
| Number2 | Valid |
| display-details | Invalid |
| email\_id | Valid |
| $salary | Valid |
| \_$\_ | Valid |
| \_123 | Valid |
| 123Demo | Invalid |
| int | Invalid |
| Class | Valid |



Conventions for Class Name

1. Class name should be start in capital/upper case.
2. If class name is combination of more than one word, then every word should be start in capital case.
3. Example: Welcome, String, System, EmployeePersonalDetails, ArrayList

Conventions for method, variable, Object Name

1. Method, variable, object name should be start with lower/small case.
2. If method name is combination of more one word, then 2nd word onwards it should be start in capital.
3. Example: main, println, displayEmployeeDetails, empName

Conventions for Constant

1. Constant should be capital case.
2. If constants name has more than one word, then between every word you should use \_
3. Example: GRAVITY, PI, COMPNAY\_NAME

**Data Type and Variable**

1. Data is the information or the values which may required to perform the program execution.
2. These data can be a different type and here we need a Data type to store the different type of values.



**Integer Type:**

1. These are the numeric type of variable.
2. The numeric values without any decimal point.
3. These numeric values can a negative or positive.
4. Example: 121, -23, 0

**Decimal Type:**

1. These are the numeric type of variable.
2. The numeric values can be with decimal point.
3. These numeric values can a negative or positive.
4. Example: 343452.23, -234.45, 0

**Textual Type:**

1. Here you can store a single character/letter or a symbol.
2. In this data type you can also store a numeric value. Which is consider as an ASCII value.
3. You cannot store a negative numeric value in this data type. That is why it is also known as unsigned int.
4. To store a single char letter or symbol you have to store value in single quotes (‘A’)
5. Example: A, 65, @, 64

**Logical Type:**

1. Can Store a Boolean values.
2. These values must be a **true** or **false.** No Other values are allowed in the logical type.
3. Example: true, false

**Variables in Java**

1. Variables are used to store values.
2. Variables are used in an expression (math expression, logical expression).
3. Variables are used to display values (as an output) to the user.
4. Variable values can be assigned to another variable.
5. Syntax to create variable

**DataType** **variable-name; // declaration of variable**

**Variable-name = value; // initialization of the variable**

**OR**

**DataType variable-name = value;**



**Formula to calculate the range of values which can be store inside variable**

**-2 no.Bits-1  to 2 no.Bits-1  -1**

Byte: -128 to 127

**Rules for creating variable**

1. long
   1. the value of long variable has to ends with L or l.
2. float
   1. the value of must be ends with F or f.
3. char
   1. In char can store a numeric value but must not be negative.
   2. And the letters or symbols must always in single quotes(‘’)

**Primitive variable Casting**

1. Casting is a process where you can convert value of one data type into another data type.
2. There are 2 type of casting in java
   1. **Implicit Casting**
      1. This type of casting done by Java internally/automatically and develop do not have to write any extra code for this.
      2. Example:

**byte b = 10;**

**int c = b; // Implicit Casting**

In the above example byte value converted into int automatically by java

* 1. **Explicit Casting** 
     1. The Casting has to do manually by writing the code by developer is called as explicit casting.
     2. If the explicit casting not done properly then you may get a logically in correct output.
     3. Example:

**int x = 12;**

**byte y = (byte) x; // Explicit Casting**

In the above code the int value is converting into a byte data type for this you have to explicitly write some code.



**Values representation in Variables**

1. Decimal
   1. These are the regular values which normally use in a program.
   2. The default format to display output or accept input will be decimal.
2. Octal
   1. The vales start with 0 are consider as a octal.
3. Hexa Decimal
   1. The values starts with 0x or 0X are consider as a hex values
4. Binary
   1. The values start with 0b or 0B are consider as binary values.
   2. Binary representation is allowed in java from JDK 1.7 version.

Types of variables

1. **Local Variable**
   1. The variables created inside a method or as an input parameter of the method are called local variables.
   2. Has to initialize at the time of declaration or before use.
2. **Instance Variable**
   1. The variables which are created inside class and outside any method are called instance variable.
   2. Instance variable will get the default value if it is not explicitly provided.
3. **Static/class Variable** 
   1. The variables which are created inside class and outside any method using static keyword are called static/class variable.
   2. Instance variable will get the default value if it is not explicitly provided.



**Default Values**

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

**Comments In Java**

1. Comments re use to skip the execution of the statement from the source code.
2. There are 3 types of comments in java.
   1. Single line comment
      1. Is use to comment the single line form the source.
      2. Syntax:

// statement

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

/\*

Statement(s)

\*/

* 1. Documentation Comment
     1. Is use to write a documentation inside a code.
     2. This comments will also be added inside the compiled code (in .class files).
     3. Syntax:

/\*\*

Statement(s)

\*/

**Operator**

**Arithmetic Operator**

**+, -, \*, /, %**

**Assignment Operator**

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

**Increment/Decrement Operator**

**++, --**

1. **These operators are used to increment or decrement value by 1.**
2. **There are two ways to use this operator**
   1. **Pre**
      1. Pre increment: ++a
      2. Pre decrement: --a
   2. **Post**
      1. Post increment: a++
      2. Post Decrement: a—

**Post:** Assign value then Update the value

**Pre:** Update the value then Assign value

**Relational Operator (Always return output in boolean, mostly these operators used in logical/conditional expression)**

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

**Logical Operator (it can return numeric values or Boolean value, mostly these operators use to combine 2 or more conditional expression)**

**&, |**

**Short Cruciate Operator**

**&&, ||**

****

**Control Flow Statement**

1. Program execution is always happened sequentially line by line.
2. To control this sequential execution of the statement you can use Control flow statement.
3. There are 3 types here
   1. **Sequential statement execution**
      1. The default program execution is always sequential.
   2. **Conditional statement**
      1. To execute specific statement(s) based on scenario or condition.
      2. To achieve this, you can use **if statement** and its variations or **switch cases**.
   3. **Looping Statement**
      1. To Execute specific statement(s) multiple time.
      2. To achieve this, you can use **while**, **do-while**, **for**, **enhance for**.

**Conditional Statement**

1. Use this whenever you wanted to execute specific statement or block of statement based on scenario.
2. There are multiple options to use conditional statement
   1. If conditional statement and its variable like if, if-else, else-if, nested-if.
   2. Switch Case

**If Conditional Statement**

1. If the Boolean Expression is true then only it will execute the statements written inside if body.

**If Syntax:**

**if(Boolean Expression)**

**{**

**Statement(s)**

**}**

1. In the following syntax the statement written inside if gets executed only if condition is true and if condition is false then it will execute the statements written inside else block.

**if-else Syntax**

**if(Boolean expression)**

**{**

**Statement(s)**

**}**

**else**

**{**

**Statement(s)**

**}**

1. In this following syntax you can apply a condition for else block. This is also known as else-if ladder. In this case if any of the Boolean expression is true then it will not evaluate other Boolean expression. If none of the expression is true then it will execute lese block. Here else block is always optional.

**else-if Syntax:**

**if(Boolean expression)**

**{**

**Statement(s)**

**}**

**else if(Boolean expression)**

**{**

**Statement(s)**

**}**

**else if(Boolean expression)**

**{**

**Statement(s)**

**}**

**else**

**{**

**Statement(s)**

**}**

**Nested-if syntax:**

**if(Boolean expression)**

**{**

**if(Boolean expression)**

**{**

**Statement(s)**

**}**

**}**

**Task:**

Declare a double variable percentage, and find the grade for the given percent value.

Percentage = 0 to 39.99 -> Fail

Percentage = 40 to 49.99 -> Pass Class

Percentage = 50 to 59.99 -> Second Class

Percentage = 60 to 74.99 -> First Class

Percentage = 75 to 100 -> Distinction

Percent not in = 0 to 100 -> Invalid Percentage

**Switch Case**

1. Switch has a cases which mapped foe a provided values.
2. Syntax:

switch(value)

{

case label:

Statement(s);

break;

case label:

Statement(s);

break;

case label:

Statement(s);

break;

default:

statement(s)

}

1. Rule for Switch
   1. As a switch value you can use **byte, int, short, char, String(Jdk1.7), enum(jdk1.5)** data types only. No other data type is allowed as a switch values.
   2. Case label must be of same data type as switch value.
   3. Case label must be unique.
   4. Break is not a part of syntax, but it has to use to get a logically correct output.
   5. You can combine multiple cases if all the cases has similar execution.
   6. Default is not mandatory and it can be skipped.

**Task: (Using Switch Case)**

Declare an int Variable day

Day is 1,2,3,4,5 -> It’s a Working Day

Day is 6,7 -> It’s a Weekend

**Looping Statement**

1. Looping statements are use to execute statement(s) multiple time.
2. This can be achieved by while, do-while, for, for-each (Enhance for)

* **While looping statement (Pre condition check)**

1. The loop statements execute will not execute if condition is false.
2. Syntax:

**Initialization (Start point)**

**while(condition)**

**{**

**Statement(s)**

**Increment/decrement**

**}**

Task:

Print 1-10 number using while loop

* **Do-While looping statement (post condition check)**

1. The loop statements executes at least once even if condition is false
2. Syntax:

**Initialization (Start point)**

**do**

**{**

**Statement(s)**

**Increment/decrement**

**}**

**while(condition);**

* **for looping statement**

1. in this loop you can combine 3 statements (start point, condition and increment/decrement) on a same line
2. Syntax:

**for ( initialization(start point) ; condition ; increment/decrement/statement )**

**{**

**Statement(s)**

**}**

****

**Task :**

**Print the table of given number using while, do-while and for loop.**

**Example: number = 5**

**o/p: 5**

**10**

**15**

**.**

**.**

**.**

**50**

**Nested Looping**

1. One loop is inside another loop
2. Nested looping is mostly used to work with row and column structure.
3. Can be achieved using any looping statement.
4. In the nested loop, outer loop is for row and Inner loop is for column

Example:

for( ; ; ) // row

{

for( ; ; ) // column

{

}

}



**Enhance For (for-each)**

1. It is use to iterate/loop through all the elements from the collection.
2. This for loop return every value one by one from the collection.
3. To iterate the values, you don’t have to provide indexes.
4. Syntax:

for(DataType variable **:** collection)

{

}

**Array**

1. Array is a collection of values or Objects.
2. It is a non-primitive data type.
3. Array is a collection of same type of values and it is of fixed in size.
4. Array internally store the value in index base.
5. Index are always maintain internally and always start from 0.
6. While handling the index if you use a wrong index then you will get an ArrayIndexOutOfBoundsException
7. Array are or different type
   1. 1-D Array
   2. 2-D array
   3. Multi-dimensional array (Jagged Array)

**1-D Array**

1. To use 1-D array you have to follow a steps.
   1. **Declaration of Array**
      1. We are going to declare an array variable and its type.
      2. Syntax:

DataType NameofVariable**[]**;

OR

DataType **[]**NameofVariable;

OR

DataType**[]** NameofVariable;

* + 1. Example:

int marks[];

* 1. **Instantiation (Object creation) of array**
     1. To create an instance/Object in java you have to use **new** keyword (operator).
     2. At the time of Object creation, you have to provide the size of an array.
     3. After Object creation the memory will be allocated inside a JVM.
     4. Syntax:

NameOfVariable = **new** DataType[**Size**];

* + 1. Example

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

* 1. **Initialization of Array** 
     1. In this step you can assign a value to an array.
     2. To set or get the value you have to use indexes.
     3. Syntax:

NameOfVariable[index] = value;

marks[2] = 30



**Different Ways to Create Array**

1. Create array using declaration, instance creation and Initialization on separate line.

int marks[]; // **Declaration**

marks = new int[5]; // **instance creation**

marks[0] = 67; // **initialization**

marks[1] = 77;

marks[2] = 57;

marks[3] = 65;

marks[4] = 61;

1. Create array by combining Declaration and Instance Create step on same line and Initialization on Separate line.

int marks[] = new int[5]; // **Declaration and instance creation**

marks[0] = 67; // **initialization**

marks[1] = 77;

marks[2] = 57;

marks[3] = 65;

marks[4] = 61;

1. Create array by combining Declaration, Instance Create and Initialization on same line

int marks[] = new int[] {67, 77, 57, 65, 61};

Note: You must not provide size of array at instance creation step

1. Create array by combining Declaration, Instance Create and Initialization on same line without new operator

int marks[] = {67, 77, 57, 65, 61};

**Array Length Function**

1. To Get the total number of values inside array
2. Example:

**marks.length**

1. To get the last index of array you can -1 the length.

**Last Index = marks.length – 1**

**Task:**

Create array with 10 int value. Print the count of even and odd values.

**2-D Array**

1. 2-D array is use to store values in row and column structure.
2. To use2-D array you have to follow a steps.
   1. **Declaration of Array**
      1. We are going to declare an array variable and its type.
      2. Syntax:

DataType NameofVariable**[ROW][COLUMN]**;

OR

DataType **[]**NameofVariable**[]**;

OR

DataType**[][]** NameofVariable;

* + 1. Example:

int marks[][];

* 1. **Instantiation (Object creation) of array**
     1. To create an instance/Object in java you have to use **new** keyword (operator).
     2. At the time of Object creation, you have to provide the size of an array.
     3. After Object creation the memory will be allocated inside a JVM.
     4. Syntax:

NameOfVariable = **new** DataType[**Row** **Size**][**Column Size**];

* + 1. Example

marks = **new** int[**5**][**4**];

* 1. **Initialization of Array** 
     1. In this step you can assign a value to an array.
     2. To set or get the value you have to use indexes.
     3. Syntax:

NameOfVariable[Row\_Index] [Column\_Index] = value;

marks[2][2] = 30

****

**Different Ways to Create Array**

1. Create array using declaration, instance creation and Initialization on separate line.

int marks[][]; // **Declaration**

marks = new int[3][5]; // **instance creation**

marks[0][0] = 70; // **initialization**

marks[0][4] = 50;

marks[1][2] = 60;

marks[2][1] = 80;

1. Create array by combining Declaration and Instance Create step on same line and Initialization on Separate line.

int marks[][] = new int[3][5]; // **Declaration and instance creation**

marks[0][0] = 70; // **initialization**

marks[0][4] = 50;

marks[1][2] = 60;

marks[2][1] = 80;

1. Create array by combining Declaration, Instance Create and Initialization on same line

int marks[][] = new int[][] { **{55 , 66, 77, 88, 99}**, **{67, 65, 78, 89, 76}**, **{88, 66, 76, 87, 85}** };

Note: You must not provide size of array at instance creation step

1. Create array by combining Declaration, Instance Create and Initialization on same line without new operator

int marks[][] = { **{55 , 66, 77, 88, 99}**, **{67, 65, 78, 89, 76}**, **{88, 66, 76, 87, 85}** };

**Task:**

Create an array to store 4 student 5 subject marks.

Print their marks and also the percentage

55 66 77 88 99 = 80%

67 65 78 89 76 = 76%

88 66 76 87 85 = 78%

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

1. In this type of array Rows are fixed but the column length can be change for every row.



**Class, Object and Method**

Class In Java

1. Class is a collection of state (Variable/data member) and behavior (method or member function)
2. Along with variable and methods you can also create constructor inside class and another class which is also known inner/nested class.
3. Syntax:



Method in Java

1. Methods are the collection of variables and executable statement.
2. Methods are used to distribute complex logical code into a smaller chunks(small block).
3. The logic written inside method can be reuse or can be execute from the different location of the program.
4. You have to call method using Object or class name so that the statements written inside method will be executed.
5. Input parameter of the method is used to get the value inside method, you can get more than one value at a time. The values which are passed to a parameter is called as arguments.
6. Method can return any one value at a time, it can be primitive or non-primitive. If method not returning any value, then return type will be void.



Object in Java

1. Object is also known as instance of the class.
2. Object will be represent class properties (method and variable).
3. Every object has an own memory which is allocated inside heap.
4. Properties of object are independent than another object so change done in one object will not affect on another.
5. In Java to create object you can use **new** operator
6. Using object, you can access the properties of class using **dot (.)** operator.





**String in Java**

1. String is an array of character.
2. In a string multiple character can store together.
3. String always store in java inside double-quotes (“Value”)
4. String can be a combination of characters, Symbols or number.
5. String is a non-primitive data type
6. To Create and use string Java has provided multiple build-in (pre provided) classes.
   1. String class
   2. StringBuffer class
   3. StringBuilder class
   4. StringTokenizer class

**String class**

1. String is a build in class in java.
2. String class present inside **java.lang package.**
3. String class is a **final class**.
4. String class is use to create a string value in java.
5. To use a string class, you have to create Object of String class.
6. Object of String class is **immutable** (Once assign a value it remains same/Value does not change once assigned)
7. There are 2 ways to create object of String class
   1. With new Operator

String str = **new** String(“Value”);

* 1. Without new Operator

String str = “Value”;

1. String Constant Pool (SCP)
   1. SCP is one of the location inside Heap Memory.
   2. The String object which is created without new operator will be created inside SCP.
   3. In the SCP before create any object, first it will check whether same object with value is created or not. If Object with same value is present then, it will not create a new object and return the same object. Else it will cerate a new Object and return the address of object.



**StringBuilder Class**

1. This is the build-in (pre-define) class in java.
2. StringBuilder class is present inside **java.lang package**.
3. StringBuilder class is **final class**.
4. To use this class you have to create Object of the class.
5. SCP is not applicable for the StringBuilder.
6. StringBuilder Object are **mutable object**.
7. Syntax:

StringBuilder object = new StringBuilder(“String Value”);

**StringBuffer Class**

1. This is the build-in (pre-define) class in java.
2. StringBuffer class is present inside **java.lang package**.
3. StringBuffer class is **final class**.
4. To use this class you have to create Object of the class.
5. SCP is not applicable for the StringBuffer.
6. StringBuffer Object are **mutable object**.
7. The methods of StringBuffer are synchronized.
8. At a time only one thread can access the object of StringBuffer. So StringBuffer object is thread safe.
9. The performance is slower than StringBuilder.
10. Syntax:

StringBuffer object = new StringBuffer(“String Value”);

**StringTokenizer Class**

1. StringTokenizer class is present inside **java.util package.**
2. This class is use to store the string into a token (Small part of the string).
3. Inside StringTokenizer you have to provide the string value and the Delimiter to convert string into token.
4. Example:

StringTokenizer str = new StringTokenizer("23/11/1998", "/");

**User Input in Java**

1. Command Line Argument
   1. To accept a value from the use at the time of providing an execution command.
   2. These values will be provided before execution of the program.
   3. You cannot accept a user input at the program runtime (in the middle of program execution).
   4. These values will be always in the string format. And it will access using the String array of the main method.
   5. There are following problem in the command line argument
      1. Cannot accept value at run time.
      2. Number of values must be match else exception can be occur in program execution.
      3. The values always in String format, to get it into another format you have to convert it manually.

1. Console Class
   1. This class required the instance of console to accept the value from the user.
   2. Is present inside **java.io package**
   3. Using this class you can accept the value from the user at runtime.
   4. This class can’t be used with the IDEs like eclipse and IntelliJ as they are using the proxy console instead of actual console.
   5. You can accept the value only in string format.
2. Scanner Class
   1. Is present inside **java.util package.**
   2. Scanner class is use to accept input from user(console), String, file.
   3. Scanner class is the easy option to get the user input than other java options.
   4. Using scanner class, you can accept the input from the user at program run time.
   5. In this class there are multiple methods to accepts the put from the user and specific data type has specific methods. Line next(), nextInt(), nextFloat()
3. Buffer Classes

**IDE (Integrated Environment Development)**

For Java Development mostly used IDEs are Eclipse, IntelliJ.

Eclipse Setup

1. Download the Eclipse

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

1. Install the Eclipse

Select “**Eclipse IDE for Java EE developer**”



**Object Oriented Programming (OOPs)**

1. OOPs concepts are use to design and develop application.
2. The oops are also a base of Design pattern.
3. Class and Object is the base of OOPs concept.
4. There are 4 main OOPs concepts.
   1. Encapsulation

Wrapping of data member (variable) and member function (method) into a single unit. Class is an example of Encapsulation.

* 1. Inheritance

Acquiring the properties (data member and member function) of one class into another class.

* 1. Polymorphism

One Object will have a multiple form is called as polymorphism.

* 1. Abstraction

Can hide the complex details/implementation and only show the required functionalities to the user is called Abstraction.

**Encapsulation**

1. Wrapping of data member (variable) and member function (method) into a single unit.
2. Class is also considered as encapsulation as it is a collection of data member and member function.
3. It is always recommended to access the instance variable of the class using method instead of directly accessing it outside class.
4. Instead of using the instance variable directly outside class use the Getter and Setter methods to access the instance variable.
5. Getter methods
   1. Getter Methods is used to return the value of the variable.
   2. Getter method name should be start with “get” word.
   3. Getter method will not accept anything and returns the values.
6. Setter Method
   1. Setter Method is use to set the value.
   2. Setter method name should be start with “set” word.
   3. Setter method will accept the value which has to set for instance variable and will not return any value (void return type).
7. Perfectly encapsulated class is a class which has all instance variable private and providing access to them using getter and setter method.
8. Advantages:
   1. Can achieve the Data Hiding (by make all instance variable private).
   2. Can control who can access what.
   3. This is one of the ways to achieve the loose coupling.

**Inheritance**

1. Acquiring the properties (data member and member function) of one class into another class.
2. Here the parent and child relation will be established once you applied inheritance between the class. This parent and child relation is also known as **IS-A relation in java.**
3. Inheritance can be achieved in java using **extends** keyword.
4. There are total 5 types of inheritance in OOPs. In this only 3 are directly supported in java and 2 types are not directly supported but it can achieve using interfaces.



1. In Java One child class can have maximum one parent class, you cannot create multiple parent classes for a single child class.
2. Object class is a super class of all the java classes. If you are not create any super class manually.
3. Every Java class will get the properties of Object class directly or indirectly.
4. There are some useful function are there in the Object which will be accessible in every java class.

Common Functions of the object

* + 1. equals()
    2. toString()
    3. hashCode()
    4. getClass()
    5. wait(),wait(long), wait(long,int)
    6. notify()
    7. notifyAll()

**Constructor**

1. Constructor is considered as a special type of method, without return type.
2. Constructor is use to initialize the instance variable. It construct the memory by initializing the instance variable.
3. To Create a constructor you have to follow the rules. 
   1. Constructor Name must be same as class name.
   2. Constructor must not have any return data type.
   3. Constructor can be created by any access modifier like private, public, protected, default.
   4. There can be more than one constructor in a class.
   5. If you do not provide any constructor explicitly then, java will provide the default Constructor internally, if user provided any constructor explicitly then java will not provide default Constructor.
4. Constructor Will be called at the time of object creation.
5. You cannot call a constructor using the object and dot operator like method.
6. Every Child/sub class access the default or no-param constructor of the super class. This is happening because of super keyword.

**super Keyword**

1. Using super keyword you can access the properties(variable, method and constructor) of super class.
2. Constructor class using Super keyword
   1. By default, every child class Constructor calls the super class no-param or default Constructor. This is happened because the super keyword will be added as a first line of every sub class Constructor.
   2. This default behavior can be change by adding you own super statement as a first statement in a child class Constructor.
   3. Constructor call must be the first statement in a constructor
3. Super keyword to use variable
   1. If you have a variable in sub class with a same name as super class then you can explicitly call a super class variable using super keyword

super.variableName;

1. Super Keyword to call super class method.

**this keyword**

1. Using this keyword, you can access the properties (constructor, variable and method) of the same class.
2. this is the current class object
3. this keyword to access the constructor
   1. using this keyword you can call the one constructor from another constructor of the same class.
   2. Once class constructor can only call from another constructor.
   3. You cannot call constructor from the method.
   4. Constructor calling line must be a first statement in a constructor.
   5. You cannot use super and this keyword to call constructor at a same time.

**Final Keyword.**

1. Final Keyword is use to create a constant.
2. Constant means the value never change (fixed values).
3. Final keyword can be use for a variable, method, class.
4. Final Variable:
   1. The value of the final variable always fixed. It cannot be change once it is assign.
   2. Final Variables can be create at instance level or also in side method.
5. Final Method
   1. The implementation of the method will be fixed once it is mark as final.
   2. The implementation will not change.
   3. Final methods cannot be override.
6. Final class
   1. Final call cannot be inherit.
   2. You can create an object of final class and access the properties using object.
   3. The methods and variable are not final if you make the class final, to make them final you have to explicitly mark them final.

**Static Keyword.**

1. Static keyword is use make class, method, variable accessible without class Object.
2. Static properties can be access just by using a class Name and without class Object.
3. Static properties will be loaded in side the memory at the time of class loading.
4. Static keyword can be use for variable, method, class and can create a static block.
5. Static variable
   1. Static variables must be created inside class and outside any method..
   2. Static variables cannot be a local variable (inside method)
   3. These variables can be access outside class using class name and without object.
   4. Copy of static variable will not be created inside every class Object, instead all the class objects will use a same copy of static variable.
6. Static Method
   1. Static methods can be access without class object and just by using class name.
   2. Static methods will not override.
   3. Static method will be created inside memory at class loading.
   4. Inside Static methods you cannot use this and super keyword.
   5. Static method cannot access the non-static properties of class directly.
7. Static class:
   1. Static class must be an inner class, you cannot mark outer class as a static.
8. Static block
   1. Static block is use to initialize the static variables.
   2. This block executes only once and before constructor.

**Package, Import and Access Modifier**

1. **Package :**
   1. Package is a group of java classes which is implementing a similar functionality.
   2. Package is a folder in a file system.
   3. Package is a keyword in Java.
   4. To create any class inside package, you have written a package statement as a first statement in the source file.
   5. This package statement must be written only once, and it must be write outside any class.
   6. There can be maximum 1 package statement in a source file.
   7. Syntax:

**package packageName1.packageName2…;**

1. **Import:**
   1. Import statement is used to access the class from one package to another package.
   2. Import statement must be use outside any class and after the package statement (if package statement is present).
   3. There can be multiple import statement in the source file.
   4. You can import a single java class or all the classes from the package.
   5. By default every java class imports the classes from **java.lang** package.
   6. Syntax:

**import package1.package2.ClassName; (To import single java class from package)**

**import package1.package2.\*; (To import all java class from package)**

**Access Modifier**

1. Access modifiers are use to manage the access of properties like variable, method, constructor or class.
2. There are 4 access modifier
   1. public
   2. private
   3. default/package (if you not explicitly provide any access modifier then this is the default modifier )
   4. protected

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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. Restriction Level from Lowest to highest

public --> protected --> default/default --> private

**Polymorphism**

1. Single thing can be used in multiple ways is a polymorphism.
2. Polymorphism is a many forms.
3. There are 2 types of polymorphism
   1. Compile time Polymorphism
      1. A method call resolve at time of compilation and same will be followed at the time of execution.
      2. **Overloading** is an example of compile time polymorphism.
   2. Runtime Polymorphism
      1. A method call resolve at the time of execution.
      2. **Overriding** is an example of runtime polymorphism

**Overloading**

1. In the overloading method name must be same and the input parameter must be different.
2. Overloading can be done in same class or sub-class.
3. To make input parameter different you can follow the following options
   1. Can change the Data type of the parameter.
   2. Can change the number of parameters.
   3. Can change the sequence of parameters.
4. Can change the return data type.
5. Can change the access modifier.
6. Can throw new and broader checked exception.
7. Can overload final method.
8. Can overload static method.
9. The method will be invoked based on the arguments pass at the time of calling method.
10. Overloading is use to improve the readability and usability of the class.
11. StringBuffer and StringBuilder is an example of overloading which is implemented by Java.

**Overriding**

1. In Overriding method name must be same and input parameter also must be same.
2. Overriding must be done in the sub-class.
3. Return Data type must be same (if primitive or void) or it can be sub-type.
4. Access modifier must be same or can be less restrictive.
5. Private method cannot be override.
6. Final Methods cannot be override.
7. Static Method cannot be override.
8. Cannot throw new and broader checked exception.
9. Overriding is use to change the implementation of the function provided by parent class in sub-class.
10. @Override annotation is used in java to make sure that method is overridden correctly at compilation time.

Task:

**Different Between Overloading and Overriding?**

**Different Between Runtime and Compile time polymorphism?**

**Runtime Polymorphism**

1. To access the method dynamically at the run time.
2. To achieve the runtime polymorphism you have to follow steps.
   1. There must be an inheritance between class.
   2. There must be overriding between method.
   3. Call the method using **Polymorphic Object**.

**Polymorphic Object**

1. Polymorphic object as an Object which has parent reference and child object.
2. Every Parent reference variable will hold the object of child class.

****

**Abstraction**

1. Hiding the complexity and showing only the required functionality to the user.
2. There are two ways to achieve the Abstraction
   1. Abstract class
      1. Use to achieve the partial abstraction. That is, you can achieve a 0-100% abstraction.
   2. Interface
      1. Can achieve 100% abstraction.

**Abstract Class**

1. Using abstract class can achieve 0-100% abstraction (partial abstraction)
2. Abstract class are created using an **abstract keyword**.
3. In abstract class you can create an abstract method.
4. Abstract method is a method which has declaration and not provided the implementation.
5. Abstract class are used as a top level (parent) class in a hierarchy.
6. You can use abstract class as a parent class of non-abstract (concrete) class, in this case you have to provided the implementation (override) for all the abstract method.
7. You can use abstract class as a parent class of another abstract class, in this case no need to provide the implementation of the abstract methods into sub class, it will be as it is inherited inside sub class.
8. You cannot create object of abstract class but it can use as a reference (polymorphic object).
9. You can create a constructor inside abstract class. These constructors will be invoked from the Sub class. (When you create an object of sub class of abstract class)
10. Abstract classes can be final or static. (you cannot use abstract and final or static keyword as a combination)
11. Abstract methods cannot be a final or static. But you can create a static and final method inside abstract class but it must be non-abstract method.
12. Abstract method cannot be private. Can be public, protected and default.
13. You can make non-abstract method as private.

**Interface**

1. Using Interface, you can achieve 100% abstraction.
2. Interfaces are not classes, but every interface will get a separate .class file.
3. To create interface, you have to use an **interface keyword**.
4. In interface all the **methods are implicitly/by default public and abstract**.
5. All the **variables of the interface are implicitly public static and final**.
6. Interface are used as a top level in the hierarchy.
7. Interfaces are **implements** on a concrete(non-abstract) class or abstract class.
8. If you implement the interface on concrete class then you have to provide the implementation for all the abstract method.
9. If you implement interface on abstract class then there is no need of providing the implementation of all the abstract method.



1. You cannot create an object of interface, but can be use as a reference (like Polymorphic object)
2. You cannot create constructor inside interface.
3. Only public access modifier is allowed inside interface.
4. One interface can extends another interface or even **one interface can extends more than one interface** which is also known as Multiple inheritance using interface. (fig. A)
5. **One Class can implements more than one interface**. (fig. B)
6. **One class can extends another class and implements more than one interface.** (fig. C)



Task:

1. Difference between interface and abstract class.

**Interfaced In Java 8**

1. Inside interface you can create an implemented method which must be mark as default.
2. These implemented methods in the interface called as **default method**.
3. Inside interface you can create static **implemented/non-abstract method.**
4. Java 8 has introduced **functional interface**
   1. Functional Interface is an interface which must have only one abstract method.
   2. These interfaces can have multiple implemented method (static or default) method but it can have only one abstract method.
   3. These interfaces are used in java for a Lambda expression/functional programming.
   4. @FunctionalInterface annotation can be use to make sure that created interface is functional interface.
   5. There are multiple pre-define functional interface provided by java which is inside java.util.function

**Type Interface**

1. Regular/normal interface
   1. This is the common interface which is used in then normal coding example.
2. Functional Interface
   1. This interface must have only one abstract method.
   2. This type of interface introduce in java 1.8 above
3. Marker/Markable interface
   1. The interface which out any member (variable or method) are called marker interface.
   2. It is also known as blank interface.
   3. Marker interface are use to notify JVM to provide the special implementation while execution of the code.
   4. Example:

Serializable interface

**Exception Handling**

**Exception:** It is an unwanted scenario or situation which occurs at the time of program execution and due to this program gets terminates abnormally.

**Exception handling:** It a process where you can handle the unwanted scenarios and provide the alternative execution which will protect the code from the abnormal termination.

**Read and understand the exception**



**Keyword in Exception handling**

**try:** Try is a block where you can write a statement which may throw exception.

**catch:** catch is a block which gets executed one the exception thrown from the try block. Also you can provide the alternative code for the exception.

**throw:** it is use at a statement level, to throw the object of exception manually.

**throws:** it is used at a method declaration level to declare exception to be thrown

**finally:** finally is a block which execute always irrespective of try and catch execution, this block provides the grantee of execution.

**Exception Hierarchy**



Checked Exception:

1. These are the exceptions which identify at the time of code compilation.
2. If checked exceptions are not handled, the code wont compile.
3. Checked exception has to handle at time of coding and before compilation.

Un-checked exception

1. These are the exception which will ignore by the compiler.
2. If these exception not handled then you wont get any compile time error.
3. But program will get terminated if it is not handled.

Try

1. Try is block in which you can write a statement which may throw exception.
2. From try block the exception object will be thrown.
3. Try block cannot be write alone, it must write with catch block or finally block or both.
4. Syntax:

**try {**

**Statement(s)**

**}**

Catch

1. Catch block is use to handle the exception thrown from the try block.
2. Catch block is also use to provide an alternative way if any exception occurs inside try block.
3. Catch block must be write a try block only.
4. In a catch block you have to specify the type of exception which is to be handle.
5. Syntax:

**catch(Exception\_Class ref)**

**{**

**Statement(s)**

**}**

1. One try block can throw more than one exception. Which can be catch using a multiple catch block.
2. In try with multiple catch you have to handle all parent exception after child exception handling. You cannot write a parent catch block before child catch block.
3. In JDK-7 java provided a way to handle multiple exception in a same catch using OR caluse.
   1. Syntax:

**catch(Exception1 | Exception2 | Exception3 .. ex)**

**{**

**}**