Front End Development

Back End Development

Core Java

Advance Java

Serlvet, JSP

Spring Boot

JPA, REST API, Postman(tool), Swagger(tool)

DB Development

Oracle, MySql

DevOps Tools

Maven

Git

GitHub

Editions of Java Language

1. JSE (J2SE)
   1. **J**ava **S**tandard **E**dition
   2. Also known as core java
   3. All the basic concepts of java language till the advance concepts are included.
   4. Can develop the Desktop and Console based application
2. JEE (J2EE)
   1. **J**ava **E**nterprise **E**dition
   2. Also known as Advance Java
   3. Servlet and JSP technologies.
   4. Can develop dynamic web application
3. JME (J2ME)
   1. **J**ava **M**icro **E**dition
   2. You can develop the Mobile applications and Embedded application using JME.

**Java Setup**

1. Download JDK

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

Download the Installer for you operating System.

1. Install JDK
2. Verify the Installation
   1. You can verify the installation file into C:\Program File\java\<jdk-version-folder>



1. Setup Environment Variable
   1. **Set JAVA\_HOME**
      1. Go to Start and Search for “Environment” word and select the “Edit System Environment Variable” option.
      2. Click on the “Environment Variable” button
      3. Select “New” Button from the System Variable section
      4. Provide the following details into text box
         1. Variable Name: **JAVA\_HOME**
         2. **Variable Value: <JDK-Path>**



* 1. **Set Path**
     1. Check for “path” variable inside system variable section
     2. Select “Path” variable and click on “Edit” Button
     3. Click the “New” button on the new window
     4. And set the variable value as follows

**%JAVA\_HOME%bin**

* + 1. Click on “OK”

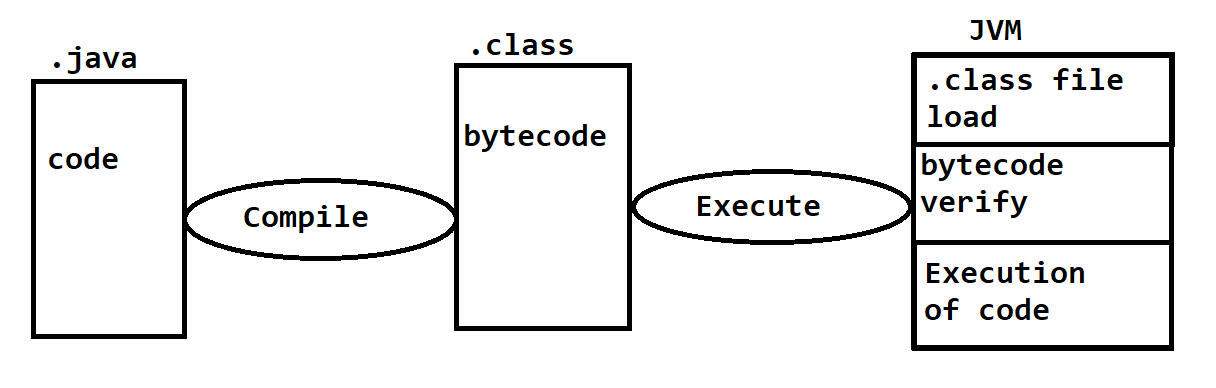


1. Verify the Environment Variable Setup
   1. Open CMD
   2. Try the following commands
      1. “**java -version**”
      2. “**javac**”



Component of Java

1. These components are helpful to develop, compile and execute the java program.
   1. JVM
      1. Java Virtual Machine
      2. JVM is use to execute the java code, it will also allocate and deallocate the memory (on RAM) which is required at the time of program execution.



* 1. JDK
     1. Java Development Kit
     2. This is useful for the developer, it consists of all development tools like javac, javap, Javadoc, java
  2. JRE
     1. Java Runtime Environment
     2. This component is useful to execute the java program and you cannot compile your java using JRE.
     3. Client machine must have a JRE to execute the java programs
  3. API
     1. Application Programming Interface
     2. This is the predefine functionalities provided by java to perform the common task. Most of the functionalities are present inside rt.jar

**Writing and Executing First Java Program**

1. Writing a java code
   1. You can write a using IDE or Notepad or any editor tool.
2. Java Program Syntax
   1. Create Java class
      1. In java everything must be write inside class except import and package statement.
      2. Syntax:

**public class <Class\_Name>**

**{**

**}**

* 1. Cerate Main Method inside java class
     1. Main method is the first method which gets executed.
     2. This method is a start point of java program.
     3. This method will be called internally by java (JVM) when you execute your program.
     4. Syntax:

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

**{**

**}**

* 1. Write an executable code into method.

**System.out.println(“Hello, Welcome to first java program”);**

1. Save File
   1. File Name must be same as public class name.
   2. File extension must be **.java**.
2. Compile the java code
   1. In this sept the java code will be converted into a byte code and the binary file will be created.
   2. As an outcome of the compilation step the .class file will be created.
   3. In this step all the syntax validation will also happens.
   4. To compile a code first Open a CMD
      1. The path of the CMD must be pointing to a location where you save your java class. (to do this you can go to a folder where you save your java class, click on the address bar and type “cmd” and hit enter)
      2. Execute a command to compile

**javac FileName.java**

1. Execute Java code
   1. Use same CMD
   2. Execute a command to run java program

**java className**



**Java Program Rules**

1. One Java file (Source file) can have a multiple classes.
2. Only one class can be a public class.
3. The file name must be a same as public class name.
4. Every class in a file can have a main method.
5. The .class file will be created for the classes created inside the source file and not for the complete java file.

**Keyword, Literals and identifier**

**Keyword**

1. The words which is reverse by the java language.
2. These words cannot be use for own purpose.
3. There are total 52 keywords are present inside java.
4. All keywords are in small letter.
5. Example:

public, class, static, void, int, long, if, else, while, for etc.

**Literals**

1. Literals are the values which is fixed by java language.
2. true, false and null these literals are also consider as a keyword.
3. String values are also known as Literals.
4. Example: **true, false, null**

**Identifier**

1. It is a word which is use by user to identify the code.
2. Identifier are the name of class, method, variable or object.
3. There are rules to create Identifier
   1. Identifier must not be keyword.
   2. Identifier can contain Alphabets or numbers or symbols like $ and \_
   3. Identifier must not be start with number. It must be start with alphabets or symbol.
   4. There no length limit to create identifier.
   5. Cannot contains spaces.
   6. Can be in small or upper cases, and it is case sensitive.

|  |  |
| --- | --- |
| **Identifier** | **Valid/Invalid** |
| firstName | Valid |
| @email | Invalid |
| Number1 | Valid |
| 1stName | Invalid |
| Employee\_Details | Valid |
| $amount | Valid |
| \_$\_ | Valid |
| \_123\_ | Valid |



**Conventions to create and use identifier**

**Conventions for Class**

1. Class Name should be start with capital case.
2. If class name is a combination of more than one word, then every word should start with Capital case.
3. Class name should be short and meaningful.
4. Example:

ProductDetails, Employee

**Convention** **for Method, Variable, Object**

1. The name of method, Variable and Object should be start with small case.
2. If name is a combination of multiple words, then 2nd word onward it should start with a capital case.
3. Example:

main, printEmployeeDetails

**Convention for Constant variable**

1. The name of the constant variable should be in a capital case.
2. If it is combination of multiple words, then there should be ‘\_’ between 2 words.
3. Example:

PI, GRAVITY, COMPANY\_NAME

**Comments in Java**

1. Comments are use to skip the line from the execution process.
2. You can also add the documentation or code level information using a comments.
3. There are 3 types of comments in java
   1. **Single line comment**
      1. Is use to comment the single line
      2. The commented code will not be included inside .class file after compilation.
      3. Syntax:

// line to comment

* 1. **Multi line comment**
     1. To comment out more than one line at a time.
     2. The commented code will not be a part of .class file after compilation.
     3. Syntax:

/\*

Lines to comment

\*/

* 1. **Documentation Comment**
     1. This is also use to comment out more that one line at a time.
     2. The commented code will be a part of .class file.
     3. Using this comment, you can write the code level documentation.
     4. Syntax:

/\*\*

Lines to comment

\*/



**Data Type and Variables**

**Data Type:**

Data type is use to create a different type of Data

There are 2 types of Data Type

1. Primitive
   1. Primitive are not in Object format
   2. Primitive will have a fixed size (Pre define size)
2. Non-Primitive
   1. All Non-primitive data type are in Object format
   2. They do not have fixed size.



**Integer**

1. In this you can store a numeric value which out any decimal point
2. There are 4 option in the integer type and those are byte, short, int and long.
3. Example:

121, 24

**Float**

1. In this type you can store numeric type of value with decimal point
2. There are 2 option in the float type and those are float and double
3. Example:

3453453.45, 5.5

**Textual**

1. In this type you can store a single character, symbol and also numeric value(ASCII value).(Keyboard char)
2. char is the only option in this type of data type
3. All the char type of values must be in single quotes.
4. Can store the ASCII values and their decimal representation.

<https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html>

1. Example:

‘A’, ‘@’, 65, ‘M’

**Boolean**

1. In this type can store a **true** and **false** values only.
2. Internally it will store as a 1 and 0.
3. But in the code you must use true and false and 0,1 is not consider as a Boolean values in java.

**Variables**

1. Variable is a component where you can create a values.
2. Variables are used to identify the values.
3. Variables are use into the math expression.
4. Variables can be used to display the values to the user.
5. One variables value can be assigned to another variable.
6. Syntax to create a variable:



Memory footprints for the variable



The range of the Integer data type can be calculate using following formula

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

n: number of Bits

byte : - 28-1 to 28-1 – 1 => -128 to 127

**Rules to Create Variable**

1. The numeric non-decimal value created without any data type will be assign int memory.
2. The numeric decimal value created without any data type will be assign double memory.

**float :**

1. Float value can be a positive or negative and decimal or non-decimal value.
2. You must use ‘f’ or ‘F’ as a suffix of the float values
3. Example:

float x = 23.32f;

**long**:

1. In the long variable can store non-decimal positive or negative value values
2. The value should be suffix with ‘L’ or ‘l’
3. Example

long l = 8678678667L;

**char**:

1. Can store single character, symbol, numeric value.
2. The character and symbol must be created inside a single quote.
3. Can store a numeric value but it must be a positive and non-decimal number only.
4. Example:

char c = ‘A’;

char c = ‘@’;

char c = 65;

**Primitive Variable Casting**

Casting is a process in which one data type value will be convert into another Data type. There are 2 types of casting in java

1. Implicit Casting
   1. Is a perform internally by java without any warning or error.
   2. Example:

**byte a = 20;**

**int x = a; // Implicit Casting**

1. Explicit Casting
   1. Has to perform manually by writing a code, and java will give an error by default.
   2. Explicit Casting can provides logically incorrect output if it is done in wrong way.
   3. Example:

**long c = 130;**

**byte y = (byte)c; // explicit casting**

****

**Type of Variables**

There are 3 types of variables. The variable type decided on the bases of their creation level. And the scope (till what time it is accessible) of variables are also depend on the type of variable

1. Local Variable
   1. The variable which is created inside method or at method signature (Declaration).
   2. Local variables must have to initialized before use.
2. Instance Variable
   1. The variable which is created inside class and outside any method.
   2. Will be initialized by default value if not provided explicitly.
3. Class/static Variable (Global variable)
   1. The variable which is created inside class and outside any method which static keyword.
   2. Will be initialized by default value if not provided explicitly.

****

**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 values are only appliable for the instance and Static variable**

**Operator**

1. Arithmetic Operator
   1. Arithmetic operator always returns **numeric value**.
   2. Explicit casting may require in this operator.

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

1. Relational Operator
   1. Always returns the Boolean value that is **true or false.**
   2. The expression which is written using this operator also known as Boolean expression or Conditional expression.

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

1. Assignment Operator
   1. This operator returns numeric value only, except = operator.
   2. Equal to (=) operator is use to assign the value, and it can be use for any type of values.
   3. Assignment operators are used to perform the operation and also assigns the value.
   4. Explicit casting will be taken case by java.

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

1. Logical Operator
   1. Logical Operator can return numeric values as well as Boolean value. Except Not operator (!)
   2. Mostly it is use to combine two or more Boolean expressions

**&, I, !**

1. Short circuit Operator
   1. This operator always returns the Boolean value that is **true or false**.
   2. This operator used to combine two or more Boolean expression.

**&&, ||**

1. Increment and decrement Operator
   1. To Increment or decrement value by one.
   2. Pre operation: The Value updated first and then assigns.
   3. Post Operation: The values assigns first and updated later.
   4. Example:

**a++** is same as **a = a + 1**

**a-- is same as a = a – 1**

++ (pre and post increment), -- (pre and post decrement)

**Control Flow Statement**

1. Control flow statement is use to manage the execution flow of the program.
2. Every program executes sequentially.
3. There are three types of control flow statement
   1. Sequential Execution
      1. The program execution will be happened line by line sequentially.
      2. This is the default implementation of any program.
   2. Conditional Statement
      1. This statement is use to execute the specific statement or block of statement only after specific condition is satisfy.
      2. if and its variations and Switch cases are the example of conditional statement
   3. Looping Statement
      1. while, do-while, for is an example This statement is use to execute the specific statement or block of statement multiple time.
      2. of looping statement.

**If Condition**

Syntax:

if(condition/Boolean Expression)

{

Statement(s) // this statements only executes if the condition is true.

}

**If-else**

1. on the condition true it executes the statement of if block
2. On the false condition it executes the statement of else block

Syntax:

if(condition/Boolean Expression)

{

Statement(s) // this statements only executes if the condition is true.

}

else

{

Statement(s)

}

**else-if (else-if ladder)**

1. Condition check can be apply for the else block.
2. There can be multiple else-if block present.
3. If none of the condition satisfy then it will execute the else block.
4. Else block is not mandatory here.

Syntax:

if(condition/Boolean Expression)

{

Statement(s) // this statements only executes if the condition is true.

}

else if(condition/Boolean Expression)

{

Statement(s)

}

else if(condition/Boolean Expression)

{

Statement(s)

}

else

{

Statement(s)

}

**Nested If**

1. One if can have another if condition (one if structure can have another if structure).
2. Syntax:

if(condition)

{

Statement(s)

if(condition)

{

Statement(s)

}

}

**Task-1**

Create int variable Age and assign some value.

age between 0 to 17 -> Teen Age

age between 18 to 60 -> Adult Age

age between 60 to 110 -> Old Age

other than this age -> Invalid Age

**Switch Case**

1. It is also used to relace the else-if conditions.
2. Switch is faster as compare to else-if
3. It can be used when there is exact match check.
4. Syntax:

switch(value)

{

case label (value):

statement(s)

break;

case label:

statement(s)

break;

case label:

statement(s)

break;

default:

statement(s)

break;

}

1. Inside Switch case there will be no condition checks.
2. Switch execute the matching case directly and it keep on executing until reaches to an end of switch or till the time break statement.
3. Default is not mandatory inside switch case.
4. Break statement is not mandatory in the case statement, but if you don’t provide the break then you may get a logically incorrect output.
5. Rule to use Switch Case
   1. As a switch value you can only use the **byte, short, int, char, enum (jdk1.5), String (Jdk1.7)**
   2. Data type of the switch value and the case label must be same
   3. Every case label must be unique.
   4. You can combine multiple cases if their execution is same.

**Task:**

1. Create a variable month and assign value to month.

It should print following output

If month = 1 then Print “Jan”

If month = 2 then Print “Feb”

If month = 3 then Print “March”

.

.

.

If month = 12 then Print “Dec”

Other than 1-12 print “Invalid Month”

1. Create a variable Day and assign value to day variable

If Day value is 1,2,3,4,5 then print “It’s a Working Day”

If Day value is 6,7 then print “It’s a weekend”

Other than 1-7 print “Invalid Day..”

Hint: Use Switch and combine cases for the values having similar output

**While Loop**

1. To execute the statement or block of statement multiple times.
2. This loop is a pre check loop.
3. Syntax:

Declaration and Initialization

while(condition)

{

Statement(s)

Increment/Decrement

}

**do-while loop**

1. To execute the statement or block of statement multiple times.
2. This is a post check loop.
3. This loop executes at least once even the condition is false.
4. Syntax:

Declaration and Initialization

do

{

Statement(s)

Increment/Decrement

}

while(condition);

**for loop**

1. This loop is recommended to execute when you know the total time of execution.
2. Syntax:

for(declaration and initialization **;** condition **;** increment/decrement/statement)

{

Statement(s)

}

1. In the for loop declaration and initialization, condition and increment/decrement is optional.
2. If you not provide a condition then by default it will consider as true and execute the loop infinite times.



Task:

1. Print the table of any number using while loop

Output : 5

10

15

20

.

.

.

50

1. Print only event numbers between 1-100 using for loop
2. Print the Odd numbers between 1-100 using do while loop

**Nested Loop**

1. One for loop inside another for loop is known as nested looping
2. Is mostly used for working with row and column format.
3. You can use a nested looping when you wants to handle a table kind of structure.
4. Syntax: 

Task:

WAP to print following pattern

1. \* \* \* \* \* \*

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

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

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

1. 1 1 1 1 1

2 2 2 2 2

3 3 3 3 3

4 4 4 4 4

**Array**

1. Array is a group of values/Objects.
2. Array is a group of similar type of values/object.
3. Array is of fixed in size.
4. You should know the size and the data type of values at the time of using array.
5. Array is a non-primitive Data type.
6. Array is indexed based.
7. Index will be created and maintain internally.
8. Index are always start from 0.
9. If you want to get or set (access) the values from array you have to use index.
10. Creating Array. Follow 3 steps to create array.
    1. Array Declaration.
    2. Array Instantiation. (Object creation)
    3. Array Initialization.
11. There are different type of arrays available in java such as 1D Array, 2D Array, Multi-dimensional (Jagged Array).

**1-D Array**

1. The values will be store in the form of row.
2. Syntax:

**Array Declaration**

**DataType identifier[];**

**Array Instantiation (Object creation)**

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

**Array Initialization**

**Identifier[index] = value;**



1. While creating object of an array you must have to assign size of the array.
2. All the array blocks will be initialized by default values after object creation.
3. Declaration square bracket can be write at the end or in between datatype and identifier.
4. To get the total number of values from the array you can use length function.
5. **Array.length** will return the total number of values present inside array.
6. **Array.length-1** will return the last index of the array.
7. There are different ways to declare, instantiate and initialize array.

double []percent; // Declaration of Array

percent = new double[6]; // Array Instantiation (Object creation)

percent[0] = 76.13; // Array Initialization

percent[1] = 77.23;

percent[2] = 71.23;

percent[3] = 68.43;

percent[4] = 88.87;

percent[5] = 78.33;

double []percent = new double[6]; // Declaration of Array and Array Instantiation (Object creation)

percent[0] = 76.13; // Array Initialization

percent[1] = 77.23;

percent[2] = 71.23;

percent[3] = 68.43;

percent[4] = 88.87;

percent[5] = 78.33;

double []percent = new double[] {76.13, 77.23, 71.23, 68.43, 88.87, 78.33};

double []percent = {76.13, 77.23, 71.23, 68.43, 88.87, 78.33};

**2-D Array**

1. In this array the records will be store in the form of row and column.
2. This array is also use to create a matrix.
3. Syntax

**Array Declaration**

**DataType identifier[][];**

**Array Instantiation**

**Identifier = new DataType[Row-Size][Column-Size];**

**Array Initialization**

**Identifier[Row-Size][Column-Size] = value;**



1. Length function in 2-D Array

array.length = returns the total number of rows in array

array[index].length = returns the total number of column in provided row index of array

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

double percent[][]; // Declaration

percent = new double[3][6]; // instantiation

percent[1][1] = 67.76; // Initialization

percent[0][3] = 77.23;

percent[2][2] = 81.12;

double percent[][] = new double[3][6]; // declaration and instantiation

percent[1][1] = 67.76; // Initialization

percent[0][3] = 77.23;

percent[2][2] = 81.12;

double percent[][] = new double[][] { {67.76, 77.56, 77.66, 78}, {67.66, 56.54, 67.88, 89.88}, {66.76, 87.65, 69.66, 97.66} };

double percent[][] = {{67.76, 77.56, 77.66, 78}, {67.66, 56.54, 67.88, 89.88}, {66.76, 87.65, 69.66, 97.66}};

**Multi Dimensional Array (Jagged Array)**

1. In the Multi-D array the number of rows will be fixed and the column number can be different for every row.
2. Syntax:

int studentId[][] = {

{45, 5, 41, 57, 86, 45},

{45, 7, 6},

{33, 44, 66, 77}

};

**Class, Method and Objects**

**Class**

1. Class is collection of state and behavior.
2. Class is a collection of Data Member (State/Variable) and Member function (behavior/Method)
3. Variables and methods created inside class can be access/used with object in to another class.
4. Along with Variable and methods you can also create the constructor and another class which is known as inner class.

Syntax

**Method**

1. Methods are the collection of variable (local variable) and the statement.
2. In side method you can write a logical code.
3. Methods has to call manually using Objects to execute it.
4. Main method will execute/call by JVM internally.
5. Methods are used to divide the logical code into a smaller chunk of code.
6. Return Data type is use to return the output of the method, you can return at a time only one value. If You mark return type as void then, that method will not return any value.
7. Input **parameter** is used to accept the values from the caller of the method. The input parameter are the local variables. You can accept more that one values of different data type at a time.
8. Th values passed to an input parameter during method call is known as **arguments.**
9. Syntax:



**Object**

1. Object is a representation of class using which you can access the properties (data member/variable and member function/methods) of the class.
2. After creating object the memory will be allocated inside the Heap memory.
3. Object creation process is also known as instantiation and Object is also known as instance of the class.
4. In Java there are multiple ways to create object of the class, but the mostly used option is using **new operator**.
5. Syntax:



1. Object can access the properties (call the method and access variables) of the class using a **dot (.) operator**.





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 an array of character or combination of multiple character.
2. String is a non-primitive data type.
3. In java String has to create in double quotes (“Value”).
4. Java provided multiple class to work with string
   1. String class
   2. StringBuffer
   3. StringBuilder
   4. StringTokenizer
5. These classes are the build-in (already provided) classes. In these classes there are multiple method/functionalities to store and perform operation on string.

**String class**

1. String class is build-in class.
2. String class is present inside java.lang package.
3. String class is a final class.
4. String Objects are immutable. Once you assign a String value it never change by any function of the string class.
5. Using string class you can store a values which has multiple characters, and internally it store in the form of array.
6. String class provides multiple methods/functions using which you can perform the operation on string value.
7. To store the value, or to perform the operation on string you have to create an object of string.
8. There are 2 ways to create object of String class
   1. With new Operator
      1. The object will be created inside a Heap memory.
      2. Example: String str = **new** String(“Value”);
   2. Without new operator
      1. The Object will be created inside SCP (String constant pool) which is a part of Heap.
      2. Example: String str = “Value”;



1. SCP is a String Constant Pool. This is use to store the String objects created without new operator.
2. In SCP before creating an object it will first check if same value is present inside the SCP or not, If same value is present then it will not create new object other wise it will create a new Object.



**StringBuilder**

1. StringBuilder class is also used to store String value in java
2. This class is present inside java.lang package.
3. This is a build-in class provided by java.
4. To store the string value and to perform operations on the string you can create object of StringBuilder and can set the value.
5. StringBuilder objects are created inside Heap and SCP is not applicable for StringBuilder.
6. The Object of StringBuilder is mutable. Mutable means the original value changes by applying the method of StringBuilder.

**StringBuffer**

1. StringBuffer class is also used to store String value in java
2. This class is present inside java.lang package.
3. This is a build-in class provided by java.
4. To store the string value and to perform operations on the string you can create object of StringBuffer and can set the value.
5. StringBuffer objects are created inside Heap and SCP is not applicable for StringBuffer.
6. The Object of StringBuffer is mutable. Mutable means the original value changes by applying the method of StringBuffer.
7. StringBuffer Objects are thread safe. Methods of StringBuffer are synchronized.
8. That is StringBuffer object can be access by one thread at a time only and if one thread performing any operation then other threads has to wait.
9. Due to this the execution of the StringBuffer is slower than the StringBuilder.

**StringTokenizer**

1. Using this class you can convert your string values into a tokens.
2. This class is present inside java.util package.
3. You can provide a string into the object and the Delimiter using which you can tokenize the string.
4. Example

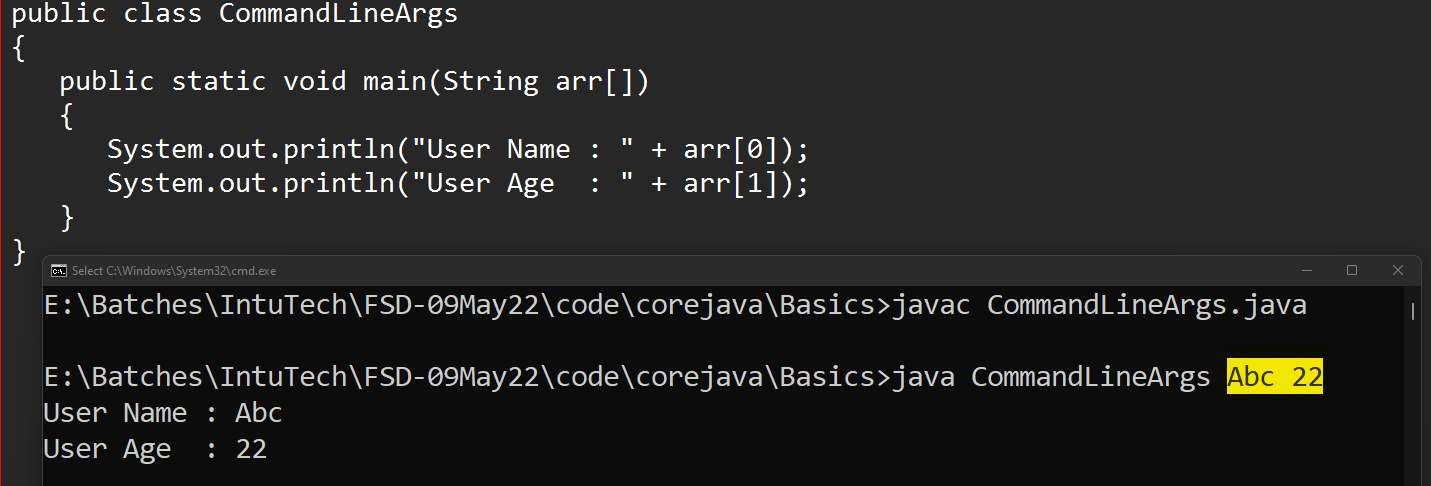
String: “This\_is\_Java\_Program”

Delimiter: “\_”

Tokens: This, is, Java, Program

**User Input in Java**

1. **Using Command Line Argument**
   1. In this option you can accept the values from the user at the time of writing command for the execution.
   2. That is passing a values from the command (execution command) is known as command line arguments.
   3. These values will be accessible inside java program from the String array parameter of the main method.
   4. The values which you pass from the command line will always be in the string format.



* 1. Issues with command line argument
     1. The values have to provide at the time of writing command.
     2. All the values are in string format, you have to convert it into a specific data type manually.
     3. User has to pass the values of all the parameter used in the program and also in the same sequence which is used in the code.

1. **Scanner Class**
   1. This class is the build-in class.
   2. Scanner is present inside java.util package.
   3. Using this class you can accept the values from the user at run time.
   4. In this class there are multiple methods provided to accept the value of different data type from the user.
   5. There is no need of converting values from one data type to another.
   6. There are method with name **nextXXX()** which are used to accept values from the user.

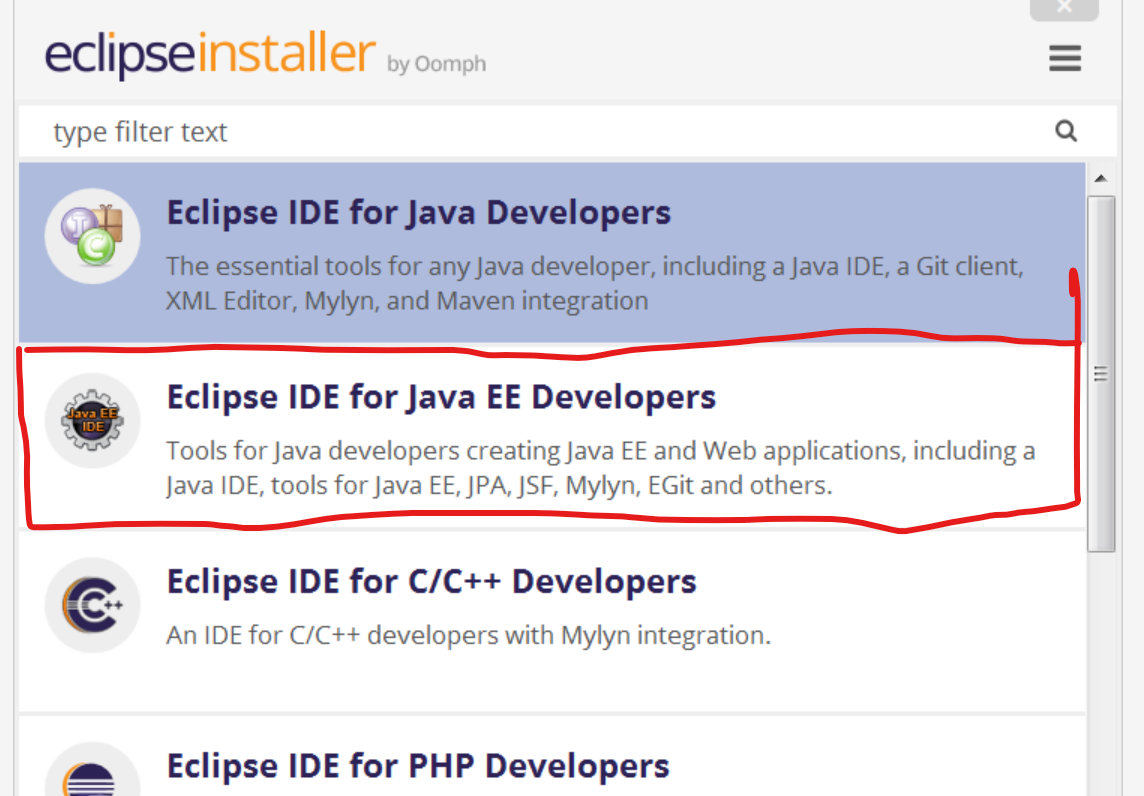
**Eclipse Setup**

1. Download Eclipse

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

1. Install Eclipse

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



**Object Oriented Programming Concepts (OOPs)**

1. Encapsulation

Wrapping of data member (variable) and member function (method) into single unit.

1. Inheritance

Accessing/inheriting the properties (variables & methods) of parent class into child class.

1. Polymorphism

One object will have multiple forms (single thing can be used in a multiple ways).

1. Abstraction

Is a way to hide the complex implementation and display only required functions.

**Encapsulation**

1. Wrapping of data member and member function into single unit.
2. Class is also consider as a encapsulation.
3. In Encapsulation, all instance variable should be private and provide access to it using getters and setters method.
4. **Getter Method** is use to get the values of the instance variable. Use **get** word as a prefix of the method name. This method always returns the value.
5. **Setter method** is used to set the values for the instance variable. Use **set** word as a prefix of the method name. This method always accepts from the input parameter and do not return anything.
6. The encapsulated class is also known as DTO (Data Transfer Object) class or Entity class in the Hibernate, JPA, EJB framework or bean classes in the Spring framework.
7. Advantages of Encapsulation
   1. It is one of the way to achieve loose coupling.
   2. Code is flexible so that if you change one part of code it will not affect on another part.
   3. You can easily add new requirements and update the existing requirement.
   4. Can control who can access what using encapsulation.

**Inheritance**

1. Is a way to access the properties (data member and member function) from one class (parent class) into another class (child class).
2. In Java inheritance can be achieved by **extends** keyword.
3. There are 5 types of inheritance in OOPs, in Java 3 types are directly supported and 2 are not directly supported in java but it can be achieve using interface.
4. In Java, One class can have only one parent class at a time.



1. By Inheritance IS-A relation will be established between two class.
2. While creating Object in case of inheritance you can use a reference of parent class and Object of Sub class which is called as Polymorphic Object.
3. Object class is a super class of all the Java classes if you not provided parent class explicitly.
4. There are some common methods present inside Object class which can be used for different implementations.
   1. toString() : This method gets called whenever you try to print the Object of class. By default this method return String “ClassName@HashCode”
   2. hashCode() : is the proxy location of the object.
   3. equals() : it Used to compare 2 java objects.
   4. wait()
   5. wait(long)
   6. wait(long, int)
   7. notify()
   8. notifyAll()
   9. finalized() : is called before Garbage collection (GC) process.

**Constructor**

1. Constructor are used to create a memory by assigning the values for instance variable at the time of object creation.
2. By default every class has a constructor, if you not created constructor manually then java will add a default constructor inside class. And if you create constructor explicitly then java is not responsible to provide any constructor.
3. Rules to create constructor
   1. constructor name must be same as class Name
   2. constructor must not have a return data type.
   3. Constructor can be create using any access modifier such as private, public, protected and default.
   4. There can be more than one constructor created inside a class.
   5. Constructors gets called at the time of object creation only or one constructor calls another constructor using this or super keyword.
4. All the sub class constructor, calls super class default or no parameterized constructor by default using super keyword.

**super, this, static, final Keywords in Java**

**super Keyword**

1. Is used to access the properties of super class inside sub class.
2. You can access the variables, methods and constructor of the super class.
3. Super class constructor must be call from the sub class constructure as a first line using super keyword.

**this keyword**

1. Is used to access the properties of same class.
2. this keyword is the current object of same class.
3. You can access the Variable, method and constructor of the same class.
4. One constructor of the class can be called form another constructor of the same class using this keyword. Use this keyword as a first line in a constructor.

**final Keyword**

1. Final Keyword is used to create constants.
2. Constant are the fixed values. Which is once assigned it can never be changes.
3. Final keyword can be used for a variable, method and class.
4. Final variable has a fixed values, once assign a values for final variable it never changed.
5. Final methods cannot be override.
6. Final class cannot be used as a super class (cannot be inherit)

Static

1. Static keyword can be used for variable, method, class (inner class) and static block.
2. Static keyword is used to access the properties of the class using class name and without creating object of class.
3. Static properties are loaded inside a memory at the time of class loading.
4. Static Variable.
   1. Static variables are also known as class variable.
   2. Static variables can be created inside class and outside any method,
   3. You cannot create static variable inside method.
   4. Static variables can be access outside class using class name.
5. Static method
   1. Static methods are known as class methods.
   2. Static methods can be access outside class using class name without creating object of class.
   3. Inside static method you can only access the static properties of the class
   4. You cannot used super and this keyword inside static method.
6. Static class
   1. Classes can static but it must a inner class (one class inside another class).
   2. Outer class cannot be static.
7. Static block
   1. Static blocks are used to initialized the static properties.
   2. Static blocks gets called before constructor.

**Package and Import Keyword**

Package:

1. Is a group of java classes.
2. In a package the classes of the similar functionalities will be placed.
3. In the file system packages are nothing but the folder.
4. To create any class inside package you have to use package statement inside class.
5. Rules to use package statement
   1. Package statement must be present as a first line of the java file.
   2. There can be only one package statement for a one java file.
   3. Package statement is applicable for all the classes in the source file.
   4. You can create a package and also use a sub packages in the package statement.
   5. Syntax:

**package pack1.pack2.pack3;**

Import:

1. Import statement is use to import a java class from one package to another package.
2. Using import statement the class form one package is accessible inside another class from another package.
3. Rules to use import statement
   1. Import statement must be write after the package statement (if package statement is available)
   2. There can be a multiple import statement in a class.
   3. By Using a import statement you allow to use the external class in the current java files.
   4. Syntax:

**import pack1.pack2.pack3.ClassName**

Above syntax import a single class from the package

**import pack1.pack2.pack3.\***

Above syntax imports all the classes from the package.

Access Modifiers

1. Access Modifiers are used to manage the access of variables, methods, classes.
2. There are total 4 access modifiers present in java
   1. public
   2. protected
   3. default/package
   4. private

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Access Modifier** | **Same Class** | **Same package different class using object** | **Same package different class using Inheritance** | **different package different class using Inheritance** | **different package different class 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** |

Restriction Level from High to Low

**private -> default/package -> protected -> public**

**Polymorphism**

1. One object will have multiple forms (single thing can be used in a multiple ways).
2. In Java Polymorphism has 2 types
   1. Compile time polymorphism
      1. The method call resolve at the time of compilation and same will be followed at execution time.
      2. The example of compile time polymorphism is **Overloading**
   2. Runtime polymorphism
      1. The method call resolve at the time of execution. The method linking happened at the time of compilation but the different linked followed at execution time.
      2. The example of run time polymorphism is **Overriding**

**Overlading**

1. In the overloading method name is same and the input parameter must be different.
2. Return data type of the methods may or may not be same.
3. Access modifier can be changed.
4. Overloading can be done in same class or in sub class.
5. Can throw new and broader checked exception.
6. Method call is decided on the bases of the type of arguments you passed to an overloaded method.
7. In Java StringBuffer and StringBuilder is an example if overloading.
8. Using Overloading you can make you program more readable.
9. Final methods can be overload.
10. Static methods can be overload.

**Overriding**

1. The method name must be same with same input parameter list.
2. Overriding must be done in sub class.
3. Return data type must be same if it is primitive or void.
4. Return data type can be change to sub-type in case of non-primitive data type.
5. Access modifier must be same or it can be less restrictive.
6. Cannot throw new and broader checked exception.
7. You cannot override static method.
8. You cannot override a final method.
9. Overriding is used to change the default implementation provided inside super class in sub class.

**Runtime Polymorphism**

1. The method call will be dynamically/runtime decided on the bases of the object.
2. To Achieve runtime polymorphism you have to follow the steps
   1. There must be inheritance between classes.
   2. There must be method overriding.
   3. Using polymorphic object access the overridden method.
3. Polymorphic Object: The object which has a reference of parent class and Object of Sub class which is called as Polymorphic Object.
4. Example:

SuperClass ref = new SubClass();

**Abstraction**

1. Hiding the complex implementation from the user and display only the required functionality.
2. To achieve abstraction you can used 2 ways.
   1. **Interface**: you can achieve 100% abstraction. Full abstraction can be achieved.
   2. **Abstract class**: You can achieve 0-100% abstraction, Partial abstraction can be achieve.

**Abstract Class**

1. It is used to achieve 0-100% abstraction.
2. Abstract classes are created using an abstract keyword.
3. Abstract classes can have an abstract method and non abstract method also.
4. The abstract method is the method which do not have any implementation, and just the declaration of the method is in the class.
5. Abstract classes are mostly used as a top-level class in the hierarchy.
6. Abstract class can be inherited by non-abstract(concrete) class, and the child class has to provide implementation for all the abstract methods.
7. You can also inherit one abstract on another abstract class, in this case it is not necessary to implement all abstract methods.
8. Abstract class and Abstract methods cannot be static or final.
9. Abstract methods cannot be private.
10. You can create static/final/private method inside abstract class which must be non-abstract.
11. You cannot create Object of abstract class but it can be used as an reference.
12. Can Create a constructor inside abstract class. Abstract class constructor will invoked when you create an Object of its sub class.

**Interface**

1. Interfaces are used to achieve 100% abstraction.
2. Interfaces can be create using interface keyword.
3. Interfaces are not the classes but it will get a .class file after compilation.
4. All the variables inside interface are by default public static and final.
5. In the interfaces all the methods are by default public and abstract
6. Interfaces are implements on an abstract class or concrete class.
7. If you implement interface on concrete class, then all the abstract methods has to implement inside concrete class.
8. If you implement the interface on abstract class then it is not necessary to implement abstract method inside abstract class.
9. You cannot create object of interface but it can be used as reference.
10. You cannot create static and final method inside interface till JDK1.7.
11. You cannot create constructor inside interface.
12. One interface can extends another interface or more than one interface. Using this you can achieve multiple inheritance.
    1. One Interface can extends one or more interface.
    2. One Class can implements one or more interface.
    3. One class can extends another class and implements more than one interface.

**Types of interface**

1. Simple/Regular Interface.
   1. This is the normal interface.
2. Marker Interface
   1. The interface without any members.
   2. This are the blank interface.
   3. Marker interfaces are use to notify JVM to provide a special execution at the program runtime.
   4. Serializable is an example of marker interface.
3. Functional Interface
   1. The interface with single abstract method is called as functional interface.
   2. This concept is introduce in java 1.8
   3. Runnable interface is an example of functional interface.
   4. These interfaces are used to achieve functional programming in java.
   5. @FunctionalInterface annotation id used to specify the interface is functional interface

**Interface Changes in Java 8**

1. In Java 8 functional interface is introduced. It is used for a functional programming and foe Lambda expression.
2. Can Create an implemented methods. These methods are called as a default methods.
3. You can create static method inside interface. Jdk 1.8 onwards you can create a main inside interface and run as a normal class.

**Exception Handling**

**Exception** is an unwanted scenario which occurs during the execution of the program due to which program may terminate abnormally.

**Exception Handling** is a way using which you can handle the exception so that you can avoid abnormal termination of the program.

**Exception Details**



**Exception Handling Keywords**

**try**: Try is a block which is use to write a statement which may throws an exception. Exception are thrown from the try block (from the try block the exception will be raised).

**catch**: catch is block where you can provide the exception handling logic that is, the logic which has to execute after getting an exception.

**throw**: it is use to raise an exception manually.

**throws**: It used to propagate exception out side method at the calling level or at the VML.

**finally**: Finally is a block which give you a guarantee of execution and this block executes always irrespective of try catch execution.

**Exception Hierarchy**



**try**

1. Try is a block of statements which may throw exception.
2. Every try block must be comes with a catch or finally or both.
3. One try block can have multiple catch block.
4. From the try block object exceptions are thrown and it will catch by appropriate catch block.
5. Syntax:

**try** {

**// statement(s)**

**}**

**catch**

1. Catch block is used to catch the exception thrown from the try block.
2. Catch block you must have to specify the exception.
3. The exceptions are catch with the help of Exception object.
4. Catch Block has to followed with try block .
5. Syntax:

**catch(ExceptionType refVariable) {**

**Ststement(s)**

**}**

**How exception process internally?**

1. First the exception scenario will be identified.
2. The type (class) of exception will be identified for that exception and creates an object of exception class.
3. The Object of the exception thrown from the try block.

**Try with multiple catch**

1. One try block can have multiple catch block.
2. You must not handle the parent exception before the child exception.
3. First you have to handle all the child exception and then you can add the parent exception else you will end up with a compile time error.
4. You can add multiple exceptions into a single catch block separated by ‘|’ OR. This feature introduce in java 1.7 onwards.
5. To use this feature you have to make sure that you are not writing exceptions from the same hierarchy.

**Throw**

1. Throw keyword is used to throw an object of exception manually.
2. Can throw the object of build-in (pre-define) exception or custom exception.
3. Throw always used at a statement level.
4. Throw must be followed with Exception Object.
5. Throw is used to raise exception manually.

**Throws**

1. Throws keyword is use to propagate exception to the caller level.
2. Throws keyword is used at a methos signature.
3. Throws always followed with the Class Name.
4. Throes can be used with multiple class name.

**Finally**

1. Finally keyword is used to close a resources.
2. Finally is a block which provides a guarantee of execution.
3. This block will execute always irrespective of try and catch execution.
4. This block comes with try or try catch.