**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>

**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

* **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

{

}

}



**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.

**Enhance For**