|  |  |
| --- | --- |
| Week1 | Java Fundamentals-I -- features, tokens, control statements, class, object |

**INTERVIEW QUESTION:-**

Give a few reasons for using Java?

* Object Oriented (OO).
* Better portability than other languages across operating systems.
* Built-in support for multi-threading, socket communication, and memory management (automatic garbage

collection).

* Supports Web based applications (Applet, Servlet, and JSP), distributed applications (sockets, RMI, EJB etc)

and network protocols (HTTP, JRMP etc) with the help of extensive standardized APIs (Application

Programming Interfaces).

* Robustness, through Exception Handling

**INTERVIEW QUESTION:-**

What is the main difference between the Java platform and the other software platforms?

Java platform is a software-only platform, which runs on top of other hardware-based platforms like UNIX, NT etc.

**INTERVIEW QUESTION:-**

What is the difference between C++ and Java?

* Java does not support pointers.
* Java does not support multiple inheritances instead Java supports multiple interface inheritance.
* Java does not support destructors but adds a finalize() method.
* Java does not include structures or unions because the traditional data structures are implemented as Java Collections Framework
* Java does not have global variables or functions.
* Java includes automatic garbage collection

**INTERVIEW QUESTION:-**

What are class loaders, types & differentiate static & dynamic class loading ?

The **Java ClassLoader** is a part of the [**Java Runtime Environment**](https://www.geeksforgeeks.org/differences-jdk-jre-jvm/) that dynamically loads Java classes into the [**Java Virtual Machine**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/).

All classes are loaded based on their names and if any of these classes are not found then it returns a [**NoClassDefFoundError**](https://www.geeksforgeeks.org/classnotfoundexception-vs-noclassdeffounderror-java/) or [**ClassNotFoundException**](https://www.geeksforgeeks.org/classnotfoundexception-vs-noclassdeffounderror-java/).

A Java Classloader is of **three types**

1. **Bootstrap or** **Primordial ClassLoader** loads classes from the location of **JDK**
2. **Extension ClassLoader** loads classes from the location **jre/lib/ext**
3. **System ClassLoader** loads classes from the location of **CLASSPATH**

|  |  |
| --- | --- |
| **Static class loading** | **Dynamic class loading** |
| Classes are statically loaded with Java’s “new” operator.  Car c = new Car(); | Dynamic loading is a technique for programmatically invoking the functions of a class loader at run time.  String myClassName = "au.com.Jeep" ;  Class vehicleClass = **Class.forName**(myClassName) ;  (Or)  myJeep = (Jeep) vehicleClass.**newInstance**(); |

**INTERVIEW QUESTION:-**

# Difference between JDK, JRE, and JVM

**Java Development Kit (JDK)** is a software development environment that offers a collection of tools and libraries necessary for developing Java applications.

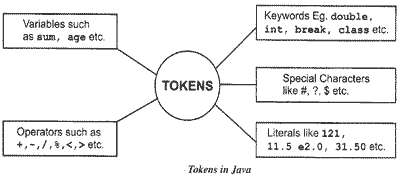
**Java Runtime Environment**(JRE) provides a platform to run and execute the source code. JRE contains JVM

**Java Virtual Machine**(JVM) provides the libraries to execute the program. The JVM, in turn, contains the JIT compiler.

**Just-In-Time(JIT) compiler:** It is used to improve the performance.JIT compiles parts of the bytecode that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.



**Java Tokens**



There are rules for naming identifiers.

* The characters allowed are **[A-Z], [a-z], [0-9], \_ and $.**
* Identifiers are case-sensitive. That is, “var” is not the same as “VAR”.
* Identifier names should not start with a digit. For example, “1var” is an invalid identifier.
* Whitespace is not allowed inside an identifier.
* Keywords can’t be used as an identifier.

**INTERVIEW QUESTION:-**

**Identify whether variable names are valid or not**

* te-st
* 2var
* $2
* \_var\_

**Welcome Program**

**Welcome.java**

public class Welcome

{

public static void main(String args[])

{

System.out.println ("welcome to Java Coding");

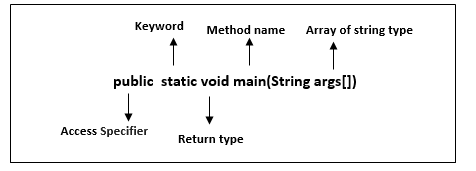
}

}

**public, class, static, void ---**are keywords, all keywords strat with small letters

**main()------** is a function, starting point of execution of a program

**println()---**is a predefined function used for printing on standard output device i.e, monitor



**Note:-**

Class name and java file name should be same.

Function names follow camelcap’s

Class names follow initial cap’s

**main method----**

main() method is the starting point of execution of a program.

main() method can take String array as argument

main() method can be overloaded but not overridden

**INTERVIEW QUESTION:-**

Why main() method is static & public ?

The main() method is static so that JVM can invoke it without instantiating the class

It is made public so that JVM can invoke it from outside the class as it is not present in the current class.

The java command-line argument is an argument i.e. passed at the time of running the java program. These command line arguments can be received in the java program by using the argument of main() method “String args[]”

**Task:**

Swapping 2 numbers without using a 3rd variable

**Method1:**Using Temporary Variable

To swap two numbers using third variable we have to declare a temp variable.

temp=x; //contains of x will be assigned to temp

Again y value we will assign to x .x value will be replaced with y value.

x=y;

Now temp value we will assign to y .

y=temp;

**Method2:**Without Using the Temporary Variable – Addition & Subtraction Operators

Way 1:

x=x+y;

y=x-y;

x=x-y;

Way 2:

a = (a + b) - (b = a);

**Method3:**Without Using the Temporary Variable – Bitwise XOR Operator (^)

x=x^y

y=x^y

x=x^y

**Method4:**Without Using the Temporary Variable – Multiplication & Division Operators

Way 1:

a = a \* b;

b = a / b;

a = a / b;

Way 2:

a = (a \* b) / (b = a);

**Task:**

Print all the arguments passed from the command-line

Input:-java CmdLineArgs This is command line argument program

class CmdLineArgs{

public static void main(String args[]){

for(int i=0;i<args.length;i++)

System.out.println(args[i]);

}

}

Output:-

This

is

command

line

argument

program

**INTERVIEW QUESTION:-**

Can a java program be executed without main() method?

Prior to JDK 7, the main method was not mandatory in a java program.

However, from JDK7 main method is mandatory.

JVM executes a static block on the highest priority basis. It means JVM first goes to static block even before it looks for the main() method in the program.

yes,one of the way is static block but in previous version of jdk1.7

**public** **class** WithoutMain

{

**static** {

System.*out*.println("hai");

System.*exit*(0); // prevents “main method not found” error

}

}

The order of code execution in a Java class is as follows.

1. Static variables are initialized.
2. Static blocks are executed.
3. Instance variables are initialized.
4. Instance blocks are executed.
5. Constructor is executed.
6. Static methods can be called without creating an instance.

**System----**

***System*** is a part of *java.lang* package. System indicates the current computer system.

and one of its main features is to give us access to the standard I/O streams, one for each stream:

* **out--** indicates standard output device, a static data memberof type output stream class *OutputStream*.
* **err**-- represents standard error and we use that specifically to output error messages
* **in**--- indicates standard input device, a static data memberof type input stream class *InputStream.*

Ways to read input from console in Java

1. Using System.in.read()
2. Using Buffered Reader Class
3. Using Scanner Class
4. Using Console Class

|  |  |  |  |
| --- | --- | --- | --- |
| System.in.read() | BufferedReader | Scanner | console |
| For reading I/P in bytes form.  abstract method of abstract class InputStream  Using low level method may be prone to errors. | For reading all kinds of data  Synchronized  Efficient reading as I/P is buffered & faster as it just reads the stream doesn’t parse the tokens. | For reading primitive data types  Not Synchronized  Simple way of reading I/P & slower since it parse the tokens. | For reading password like I/P without echoing  Synchronized  But works with only non-interactive environment i.e, command line |
| int ch= System.in.read(); | BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));    String name=reader.readLine(); | Scanner in = new Scanner(System.in);  String s;  s=in.nextLine();      int a=in.nextInt();  float b = in.nextFloat(); | String name = System.console().readLine(); |

**Task:**

Guess the Output:-

import java.io.IOException;

class System\_In\_Read {

public static void main(String[] args)throws IOException {

char ch, ch2;

System.out.print("Enter two characters: ");

ch = (char) System.in.read();

ch2 = (char) System.in.read();

int x1=ch;

int x2 = ch2;

System.out.println("x1="+x1+" | x2="+x2);

if (ch == ch2)

System.out.println("Same");

else

System.out.println("different");

}

}

Input:-

a

a

Output:-

x1=97 | x2=10

different

**Explanation:-**

read() method is a type of input stream that reads the next byte of data from input stream . However, pressing the enter key adds extra byte to the input stream, which is always stored in ch2.

**Task:**

Their exist two lines of input each consist of single integer, read these two integers using BufferedReader Class and then print both of them on a single line in a space-separated manner.

**Input Format**

Input consist of two lines.

Both of the lines contains a single integer.

**Output Format**

Print both of the integers on a single line in a space-separated manner.

Example Input

Input 1:

12

1

Output:

12 1

**Solution:-**

import java.lang.\*;

import java.util.\*;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

class Test {

public static void main(String[] args) throws IOException {

**BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));**

String s1=reader.readLine();

String s2=reader.readLine();

System.out.println(s1+" "+s2);

}

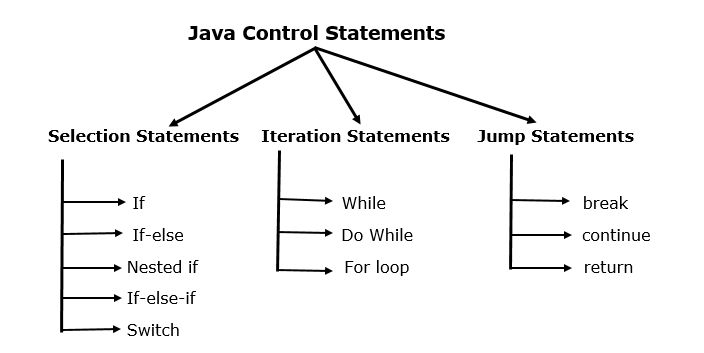
}

**INTERVIEW QUESTION:-**

**What value does read() return when it reaches the end of a file?**  
**Ans:** The read() method returns -1 when it reaches the end of a file.

**What value does readLine() return when it has reached the end of a file?**  
**Ans:** The readLine() method returns null when it has reached the end of a file.

**Control statements**



**Switch:--**

|  |  |
| --- | --- |
| The Java switch statement executes one statement from multiple conditions  The case value must be literal or constant.  **A switch works only with four primitives and their wrappers as well as with the enum type and the String class**   * *byte* and *Byte* * *short* and *Short* * *int* and *Integer* * *char* and *Character* * *enum* * *String* | switch(expression){  case value1:  //code to be executed;  break; //optional  case value2:  //code to be executed;  break; //optional  ......    default:  code to be executed if all cases are not matched;  } |

**INTERVIEW QUESTION:-**

**Guess the O/P:-**

class Test {

public static void main(String[] args) {

String cat="CAT";

String animal="CAT";

switch (animal) {

case cat: //does not compile

System.out.println("cat");

}

}

}

**Hint:-** The case value must be literal or constant, so it should be made final

**Task:**

Write a Java program to print the Climatic Season of your place.

Months Season

12,1,2 Winter

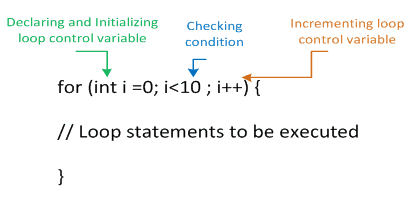
3,4,5 Spring

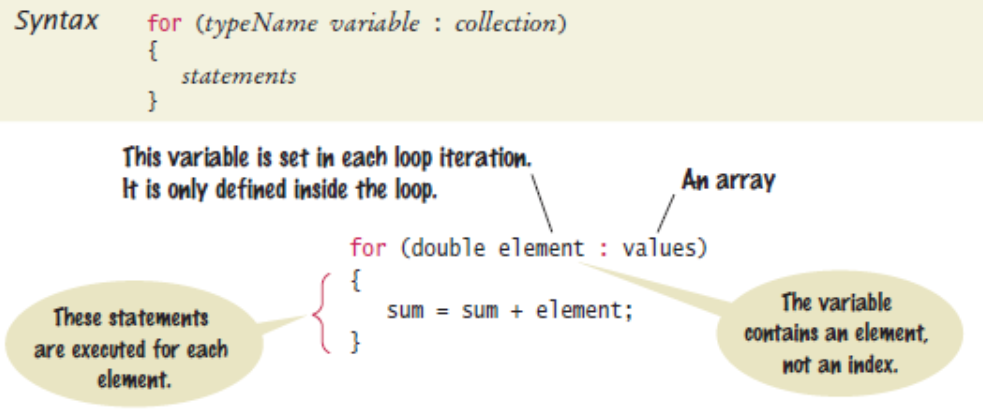
6,7,8 Summer

9,10,11 Monsoon

Anything else entered - "Invalid Month"

**LOOPS:-- for, while & do while**



****

[**break statement**](https://www.javatpoint.com/java-break)**,** it jumps the control out of loop or switch block.

**continue statement,** will skip the execution of current loop iteration & continue with next iteration

**INTERVIEW QUESTION:**

Guess the O/P:-

class Test {

public static void main(String[] args) {

for (int i = 0; ; i++) {

System.out.println(i);

}

}

}

**INTERVIEW QUESTION:**

Guess the O/P:-

class Test

{

public static void main (String args[])

{

for(int i=0; 0; i++)

{

System.out.println("Hello Javatpoint");

}

}

}

**Task**

### **Pyramid Program**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

**Solution:-**

public class Pyramid

{

public static void pyramidPattern(int n)

{

for (int i=0; i<n; i++) //outer loop for number of rows(n) {

for (int j=n-i; j>1; j--) //inner loop for spaces

{

System.out.print(" "); //print space

}

for (int j=0; j<=i; j++ ) //inner loop for number of columns

{

System.out.print("\* "); //print star

}

System.out.println(); //ending line after each row

}

}

public static void main(String args[]) //driver function

{

int n = 5;

pyramidPattern(n);

}

}

**Task**

### **Number Pattern Program in java**

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

**INTERVIEW QUESTION:-**

**Differentiate continue & break (or) Guess the O/P:-**

|  |  |
| --- | --- |
| class Test {  public static void main(String[] args) {  for (int i = 0; i<50 ; i++) {  if (i%2==0 && i >= 10) {  continue;  }  System.out.println(i);  }  }  } | class Test {  public static void main(String[] args) {  for (int i = 0; i<50 ; i++) {  if (i%2==0 && i >= 10) {  break;  }  System.out.println(i);  }  }  } |

**INTERVIEW QUESTION:-**

Difference between while & do while loop

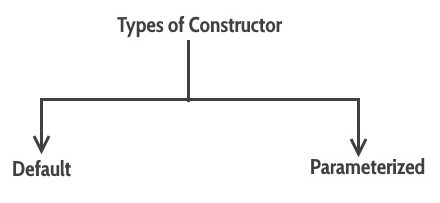
Differentiate between break & continue

**Task:**

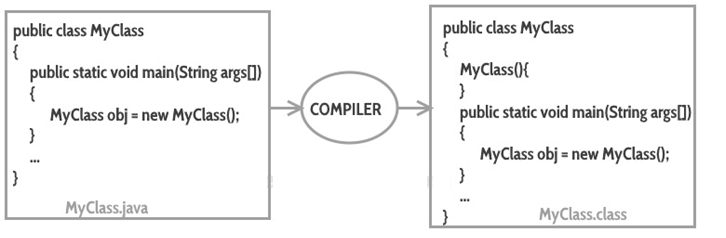
Write a Java program to read the following inputs from the user - Name, Rollno, height and weight. And check whether the height is greater than 170 and weight is less than 60kgs. If these conditions are satisfied the candidate is eligible to become an NCC cadet, otherwise print not eligible.

**Class, Object, Constructors**

|  |  |
| --- | --- |
| **Real world Terms** | **Programming Terms** |
| Entity | Class, is a single unit of combination of variables & methods(logical entity)  Object, is an instance of class (a physical entity) |
| Properties | Variables, stores data of an entity |
| charactertics | Methods, gives the functionality of an entity |

****

**NOTE**:- If you do not implement any constructor in your class, Java compiler inserts a [default constructor](https://beginnersbook.com/2014/01/default-constructor-java-example/) into your code on your behalf. This constructor is known as default constructor.

****

**Task :**

Create a student class with roll number, name & address as data members, initialize and display student details.

**class** Student{

**int** rollno; // Instatnce variables

String name;

String address;

Student(){ // Default Constructor

**this**.address="MRUH";

}

Student(**int** r,String n){ // Parameterized Constructor (2 argument)

**this**(); // refers the default constructor

**this**.rollno=r;

**this**.name=n;

}

Student(**int** r,String n, String a){ // Parameterized Constructor (3 argument)

**this**.rollno=r;

**this**.name=n;

**this**.address=a;

}

**void** addStudentRecord(**int** r, String n,String a){ // Instance method

rollno=r;

name=n;

address=a;

}

**void** displayDetails() // Instance method

{

System.***out***.println(rollno+" "+name+" "+address);

}

}

**class** Test{

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

Student s1=**new** Student(101,"Akhil","HYD");

Student s2=**new** Student(102,"Bindhu");

Student s3=**new** Student();

s3.addStudentRecord(103,"Chanran","SEC");

s1.displayDetails();

s2.displayDetails();

s3.displayDetails();

}

}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Student Class** | | Rollno  name  address | | addStudentRecord()  displayDetails() | | |  |  |  | | --- | --- | --- | | **Test Class** | | | | S1 | S2 | S3 | | 101  Akhil  HYD | 102 Bindhu MRUH | 103 Chanran SEC | | main() | | | |

**Explanation:-**

**Student s1=new Student(101,"Akhil","HYD");** invokes the 3 argument parameterized

Constructor to initialize all 3 variables

**Student s2=new Student(102,"Bindhu");** invokes the 2 argument parameterized

Constructor initializing rollno & name. And

this() function call invokes the default

constructor to initialize address varaiable

**Student s3=new Student();**

**s3.addStudentRecord(103,"Chanran","SEC");** invokes addStudentRecord() method

to initialize all variables

**this** keyword refers to the current object

Ex:- **this.rollno=r;**

**this() ,** invokes another constructor of the same class & it **should be the first statement in the constructor code**

**INTERVIEW QUESTION:-**

**this vs this()?**

**INTERVIEW QUESTION:-**

**When do we don’t use this?**

If local variables(formal arguments) and instance variables are different,there is no need to use this keyword.

int i,j; int i,j;

Void add(int i,int j){ Void add(int a,int b){

this.i=i; i=a;

this.j=j; j=b;

} }

**INTERVIEW QUESTION:-**

**Anonymous object**

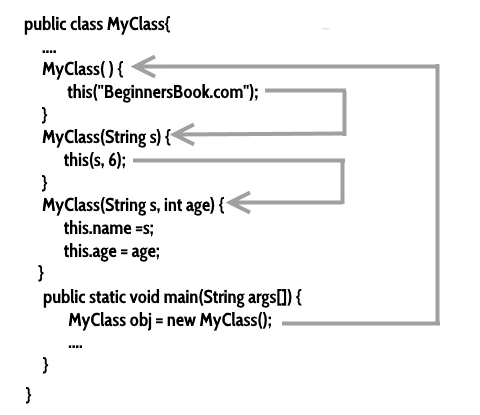
* Anonymous simply means nameless.
* An object which has no reference is known as an anonymous object.
* It can be used at the time of object creation only.
* If you have to use an object only once, an anonymous object is a good approach.

**Example:-**

**new** Student(101,"Akhil","HYD"). displayDetails();

**INTERVIEW QUESTION:-**

**Constructor chaining,** When A constructor calls another constructor of same class then this is called constructor chaining.



**toString() method**

**------------------------**

toString() method is the string representation of class

the toString() method’s default implementation gives a String that describes the

class name + ‘@’+ hashcode value of the object

toString() method can be overriden

public String toString() { } giving your own string representation of class.

**INTERVIEW QUESTION:-**

**Shallow vs deep comparision**

**equals() method**

**---------------------**

In java equals() method is used to compare equality of two Objects. The equality can be compared in two ways:

**Shallow comparison:** The default implementation of equals method is defined in Java.lang.Object class which simply checks if two Object references (say x and y) refer to the same Object. i.e.

It checks if x == y without comparing the data members that define its state, it is also known as shallow comparison.

**Deep Comparison:** Suppose a class provides its own implementation of equals() method in order to compare the Objects of that class w.r.t state of the Objects.

That means data members (i.e. fields) of Objects are to be compared with one another. Such Comparison based on data members is known as deep comparison.

**equals() and hashCode() methods in Java**

-----------------------------------------------------

hashCode() method returns the hashcode value as an Integer. Hashcode value is mostly used in hashing based collections like HashMap, HashSet, HashTable….etc.

This method must be overridden in every class which overrides equals() method.

Syntax :

public int hashCode(){}

**INTERVIEW QUESTION**

**NOTE**:-hashCode() method must be overridden in every class which overrides equals()

contract of hascode with equals()

1. If two Objects are equal, according to the equals(Object) method, then hashCode() method must produce the same Integer on each of the two Objects.
2. If two Objects are unequal, according to the equals(Object) method, It is not necessary the Integer value produced by hashCode() method on each of the two Objects will be distinct.

@Override

public boolean equals(Object obj)

{

// checking if both the object referring to the same object.

if(this == obj)

return true;

if(obj == null || obj.getClass()!= this.getClass())

return false;

// type casting of the argument.

Student stud = (Student) obj;

// comparing the state of argument with the state of 'this' Object.

return (stud.name == this.name && stud.rollNumber == this.rollNumber);

}

@Override

public int hashCode()

{

return this.rollNumber;

}

}

**INTERVIEW QUESTION**

**equals() vs ==**

**INTERVIEW QUESTION:-**

**How to copy an object in Java?**

1) Shallow copy: You can create a new object and copy the reference of the original object to the new object. This is known as a shallow copy.

Student student1 = new Student(101,"Naresh");

Student student2=student;

2) Deep copy:

1. Copy constructor
2. overriding the Cloneable **interface's clone()** function

**Copy Constructor ,** creates an exact duplicate of an object it initializes new instances with the same values as existing ones, facilitating deep copies and value swapping without affecting the original object's state.

To create a copy constructor, we need to take the existing object as an argument and initialize the values of instance variables with the values obtained in the object.

public class Student

{

private int roll;

private String name;

// Copying each field of the existing object into the newly created object

public Student( Student student )

{

this.id = student.roll;

this.name = student.name;

}

}

**overriding the Cloneable interface's** **clone() function**

class Student implements clonable{

………

}

try {

Student S1= new Student(1,”Hari”);

Student S2= (Student)S1.clone();

**}**catch (CloneNotSupportedException e) {

e.printStackTrace();

}