**Java concepts**

-Difference between method overriding and method overloading

- Constructor overloading – using **this** keyword in overloaded constructor to call constructor.

- Need of creating objects

- Polymorphism

- Encapsulation

- Static/Non static variables

-Modifiers

- Interfaces - **@Override annotation describes that methods are overridden on interface method.**

- Final keyword

-Super keyword

**Programs:-**

- Prime number program

- Fibonacci series

- print odd and even

- Reverse string using stringbuilder

- Reverse string wo stringbuilder

- Program to count occurrence of a given character in a string

- Program to count occurrence of all characters in a string using hashmap

- Triangle pattern

- Pyramid

- \*

\* \*

\* \* \*- print program.

- Find top 2 maximum numbers from array

- Find duplicate characters from a string

### **capitalize the first letter of a String is:**

String name = "stackoverflow";

name = name.substring(0,1).toUpperCase() + name.substring(1).toLowerCase();

**OOPS concepts-** Inheritance, encapsulation, polymorphism, abstraction

Difference between protected and no modifier-

**Protected**- can be accessed by all classes within same package and also by child classes.( form by importing and extending parent class)

**No modifier**- Can be accessed by all classes within same package.

To create a **sub class** (child) from a Java **super class** (parent), the keyword **extends** is used.

**Diff bet implementing and extending class**- The distinction is that implements means that you're using the elements of a **Java** Interface in your class, and **extends** means that you are creating a subclass of the class you are **extending**. You can only **extend** one class in your new class, but you can implement as many interfaces as you would like.

Public class certificates extends StudentResults{}

The StudentResults class will be the super class and the Certificates class will be the sub class. You have now created a sub class that inherits the code from the StudentResults class.

**Can access instance(non static) variable of parent class using object reference of child class Inside static methods.**

**Static/ Non static method and variables**- we can access only static stuff Inside any static methods directly. If you wants to access static method or variable Inside different class then you can access It using simply class name.  You must have to create object of class to access non static method or variable Inside static method of same class or different class. However, we can access static and non static methods and variables directly inside non static method .There is not any such access restrictions.

Static(Class variables) are used In rare case when It Is predefined that value of variable will never change.

* If you wants to access Instance(non static) variables Inside static method, You needs to create object of that class.

To access non static(instance) variables outside class, you have to provide object reference with variable name.

To access static (class) variables outside class, you need to use class name with variable.

**Method Overriding (run time polymorphism, dynamic binding)** Is useful to change to the behavior of parent class methods. When you create a method with same signature, return types and arguments of parent class's method then that sub class's method Is known as overridden method

When class have two or more methods with same name but different parameters, it is called **method overloading (Compile time polymorphism, static binding)**". We can achieve method overloading by changing number of arguments or by changing data types of arguments for same name methods in single class.

**final** keyword can be used with variable, method or class to make them final. i.e. you cannot change value of variable, override method or extend class.

|  |  |
| --- | --- |
| **Interface** | **Abstract** |
| Using keyword interface | Using keyword abstract |
| Implement interface | Extend abstract |
| 100% abstraction | Partial abstraction |
| Only static or final variables and mandatory to initialize them. | Hold Static, non static, final or non final and not mandatory to initialize them |
| Multiple inheritance--, class A implement B,C | More than 1 cannot be extended-class A extends class B |
| If add method in interface, difficult to implement in all | Method can be added when required in abstract class as concrete. |
| Cannot hold constructors, main and static method | Can hold constructors, main or static method |
| Can write body less methods Inside Interface. | Abstract method cannot have a body. |

Similarity- They both cannot be Instantiated i.e. objects cannot be created.

difference between constructor overloading and method overloading

-use of this keyword in constructor overloading

- constructor has same name as classname.

- Constructor do not Support return type as Methods do,

- to access constructor we have to create object each time

|  |  |
| --- | --- |
| class 1 | class 1 |
| **Static method 1** | **Non static method 1** |
| Static\_method() directly | Static method() directly |
| Static\_variable directly | Static variable directly |
| a.non static() ( by object reference) | Non Static method() directly |
| a.non\_static (by object reference) | Non Static variable directly |
|  |  |
| class 2 | class 2 |
| **Static method 2** | **Non static method 2** |
| class 1.Static\_method() |  |
| class1.Static\_variable | class1.Static\_variable |
| b.non static() ( by object reference) |  |
| b.non\_static (by object reference) | b.non\_static (by object reference) |
|  |  |
| class 3 extends class 1 | class 3 extends class 1 |
| **Static method 2** | **Non static method 2** |
|  | Static method() directly |
|  | Static variable directly |
| c.non static() ( by object reference) | Non Static method() directly |
| c.non\_static (by object reference) | Non Static variable directly |

break is used to exit (escape) the for-loop, while-loop, switch-statement that you are currently executing. **break** jumps to the first instruction after the switch block/ loop.

return will exit the entire method you are currently executing (and possibly return a value to the caller, optional).

**return** is used to go back to the step where it was called **or** to stop further execution,

continue- The continue statement stops the current execution of the iteration and proceeds to the next iteration. When a continue statement is encountered inside the body of a loop, remaining statements are skipped and loop proceeds with the next iteration

***Casting :*** taking an object of one type and assigning it to reference variable of another type.

***Upcasting***-Object of child class is assigned to reference variable of parent type.

Example- Animal anim = new Dog();

Where Animal is super class and Dog is sub class. here we can call the methods defined/declared in A but during runtime it will call class B’s overridden methods.  
if the method is not overridden in child’s class then only parent’s method which will be inherited to child will be called.

but same is not applicable to variables because variables decision happens at a compile time so always class A’s variables (not child’s inherited variables) will be accessed.

***Downcasting***is casting to a subtype, downward to the inheritance tree. no direct downcasting is supported in Java Child c = **new** Parent(); // it gives compile time error. because all the members of child class are not available in parent

but we can do upcasting and then we can use that variable for downcasting

Parent p = **new** Child();

Child c =(Child) p;

**JVM (java virtual machine) –** JVM is a virtual machine which work on top of your operating system to

provide a recommended environment for your compiled Java code. JVM only works with bytecode. Hence

you need to *compile your Java application (.java) so that it can be converted to bytecode format (also*

*known as the .class file*). Which then will be used by JVM to run application. JVM only provide the

environment It needs the Java code library to run applications.

.  Once a Java virtual machine has been implemented for a given platform, any Java program (which, after compilation, is called bytecode) can run on that platform.

 It is a specification that provides runtime environment in which java bytecode can be executed.

 JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

what happens to java code from start to finish:-

1- Java Source Code (Written by Developer) (Machine Neutral)

2- Compiled Code / Byte Code (Compiled by java compiler- javac) . (Machine Neutral)

3- Byte Code executed (Executed by JVM) (Machine Specific)

**JRE** is an acronym for **Java Runtime Environment**. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

JRE contains everything required to ***run*** Java application which has already been compiled. It doesn’t contain the code library required to develop Java application.

JRE= JVM+ set of libraries + other files that JVM uses

**JDK** is an acronym for **Java Development Kit**. It physically exists. It contains JRE + development tools. JDK contains everything that will be required to***develop and run*** Java application.

JDK= JVM+ set of libraries + other files that JVM uses [JRE]+ development tools e.g. javac,java.

byte code generated by javac is universal and can be converted to machine code on any operating system, this is the reason why java is platform independent.

**Exceptional handling-**

**Checked Exception-** checked at compile-time e.g.IOException, SQLException etc.

### **Unchecked Exception-** checked at run-time e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

**Error-** Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Constructor is used for initialising any parameter.

**This keyword**

1. This.parameter- This is used to refer to current instance/class variable (global) in case we have 2 variables with same name.

Public Class test{

Int param;

Public test(int param)

{

this.param=param;

}

This is referring to param declared in class and assigning it to param defined in constructor.

### 2. to invoke current class method

### 3. to invoke current class constructor

**Difference between StringBuffer and String**

**-----------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*printout till here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-----------**

**Exception handling-**

Exception Handling is mainly used to handle the checked exceptions as unchecked exception is programmers fault. Throws is used is used to declare an exception.  It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code.

There are two cases:

1. **Case1:**You caught the exception i.e. handle the exception using try/catch.
2. **Case2:**You declare the exception i.e. specifying throws with the method.-  if exception occurs, an exception will be thrown at runtime because throws does not handle the exception.

Method overriding and super concept is used in case of inheritance.

**Main method**

* **Cannot be non static**- main() method is called by the JVM even before the instantiation of the class hence it is declared as static. And non-static members can be called after creating the object
* **overload** **main method**- Yes,  there should be atleast one main function with "**String args[]** "

public class Test {

public static void main(String[] args) {

System.out.println("main(String[] args)");

}

public static void main(String arg1) {

System.out.println("main(String arg1)");

}

* **override main method-** No, because its a static method. and [static method cannot be overridden in Java](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html). it will compile and run smoothly but it will lose Polymorphism which is the basic property of Java.