01/02/21

CORE JAVA INTERVIEW QUESTIONS

1) Difference between static & Non-static

Static	Non-Static
→ Static members are loaded during class loader.	→ Non-static mem. are loaded during object creation.
→ At any pt of time we have only one copy of static members bcoz class loader will get executed only 1 time.	→ At any pt. of time we have multiple copies of non-static members bcoz multiple objects will be created by developer.
→ We go for static Variable when we feel that Variable is same for entire class.	→ We go for Non-Static Var. when value keeps on changing from one object to another object
→ Static Variables are also called as class Variables bcoz there is one copy & that copy belongs to entire class.	→ Non-static Var. are also called as instance Var. bcoz there are multiple copies, each copy belongs to an instance.
→ Static mem are accessed directly within same class and	→ Non-static members are accessed within Constructor & NSM

In another class we call by class Name.	cohese in SM we should create an object as well as in another class too.

2) Difference b/w method overloading and method overriding.

Method Overloading	Method Overriding
1) Method overloading is used to increase the readability of the program.	1) It is used to provide specific implementation of method that is already provided by its super class
2) Developing multiple methods with same name but may differs in method differs in method signature and having diff- arg	2) Developing multiple methods with same method signature but having different implementation
3) overloading methods may having diff access specifier, access modifiers & return type.	3) over-riding methods must contain same as AM & return type as in super class
4) Inheritance is not compulsory.	4) We can only over-ride & Non-static methods
5) We can over-load static & non-static methods	5) We can only override non-static methods

3) Difference b/w method & Constructor

Method	Constructor
1) Methods will have Access specifier, Access modifier, return type method name, arguments or no arguments.	1) Constructor will have only Access specifier & it may be with or without args.
2) Method names can be anything i.e they may or may not be same.	2) Construct name must be same as the class Name.
3) Methods must be called by the developer for the execution.	3) Constructor gets executed as soon as the object is created
4) Any operation that needs to be performed repeatedly then that logic comes under the method.	4) Any operation that we want to execute only once during object creation then that comes unde Constructo
5) Methods can be over-riden.	5) Constructor cannot be over-riden

4)

Abstract class	Interface
1) Abstract class can have both complete method & incomplete method.	1) Interface will have only incomplete method.
2) In abstract class we may explicitly use the keywords abstract and public for methods	2) In interface all the methods are by default abstract & public
3) Abstract class will be extended to the sub class	3) Interface will be implemented to the implementation class
4) Abstract class can have a constructor	4) Interface cannot have a constructor
5) Using Abstract class we can achieve 0-100% abstraction	5) Using interface we can always 100% achieve abstraction.

5) Difference between upcasting and downcasting

Upcasting	Downcasting
1) Converting sub class object to super class object is called as upcasting.	1) Converting super class object to sub class object is called as Downcasting.
2) Inheritance is compulsory.	2) Inheritance is compulsory.
3) When object is upcasted sub class properties are hidden.	3) When the object is downcasted hidden objects become visible. Downcasting is possible only when there is upcasting.
4) Upcasting is both implicit & explicit.	4) Downcasting is always explicit.
5) Upcasting and Downcasting can not only happen b/w super class & sub class, it can happen b/w abstract class & Subclass interface and the implementation class.	
6) Upcasting & downcasting plays major role in polymorphism & abstraction	

5) Checked Exception	Unchecked Exception
1) Checked exception statements are identified by the developer Compiler.	1) Unchecked Exception statements are identified by the developer.
2) Its checked exception super class is Exception class.	2) Its unchecked exception super class is run-time exception class
3) Both checked & unchecked are handled by using try & catch block	
4) So if we want to perform checked exception propagation we have to explicitly use the keyword throws	4) Here the exception propagation is implicit.

6) Throw	Throws	Throwable
1) It is a keyword	1) keyword	1) It is a class
2) Throw keyword is used when we want to throw exception object	2) Throws keyword is used when we want to perform exception propagation on checked Exception	3) Throwable is a super class has all the type of exceptions If we want to create any exception class it must be sub class of Throwable class

8) Final	Finally	Finalize
1) It is a keyword.	1) It is a block.	1) It is a method present in object class
2) Applies restrictions on class, methods, variables.	2) used to place an important code.	2) used to perform clean up processing just before the object is garbage collected. to clean the heap area.
3) Final class can't be inherited final method can't be overridden Final variable can't be changed	3) It will be executed whether the exception is handled or not	

9) List	Queue	Set
1) It is index-based.	1) It is not index-based	1) It is not index based
2) It allows duplicate values	2) It allows duplicate values	2) It does not allow duplicate values
3) It allows null-values	3) It does not allow null value	3) It allows only one null value

10) Method Overloading :-

→ Developing multiple methods with same method name but diffn in arguments is called as Method Overloading

→ We can achieve method overloading by maintaining same method name and differ in arguments. Args. diffn in 3 ways

i) Different no. of arguments
ii) Diff. arg. data type
iii) Diff in position of arg.

→ We go for method overloading when we want to perform same operation in multiple ways.

Real world ex:

operation :- listening to music

multiple ways :- i) Youtube
ii) Wynk
iii) Gaana
iv) Spotify

Real Time Example:

```
class Display
{
 public static void disp(char c)
 {
  sopln(c);
 }
 public static void disp(char c, int num)
 {
  sopln(c+" "+num);
 }
 public static void main(String[] args)
 {
  disp('a');
  disp('a',10);
 }
}
```

11) Constructor :-

→ A special method that is used to initialise a newly created object is called as Constructor.

→ Constructor can be achieved by maintaining Constructor name same as class Name and it may be may not have args.

→ Too easy re-initialization we go for Constructor.

Real World Example

Human - class

Birth of child - initialising object

Real Time Example:

```
Class Human
{
  public string name;
  public string birthMonth;

  public Human(string name, string birthMonth)
  {
    this.name = name;
    this.birthMonth = birthMonth;
  }
  public void detailsOfHuman()
  {
    Sopln(this.name);
    Sopln(this.birthMonth);
  }
  public static void main (String[] args)
  {
    Human hi = new Human("Spadara", "Feb");
    hi.detailsOfHuman();
  }
}
```

12) Constructor chaining:

→ Sub class constructor calling the super class constructor is called as Constructor chaining. It can happen b/w same class and b/w super class & sub class.

→ We go for this (super()) to reduce the repetation of code & implementing the reusability of code.

→ We need to call this() or super() statement inside Constructor in first statement. It can be used in 2nd or 3rd statement. If first we have to perform Constructor chaining then other tasks.

Real time example

```
Class Multiplication
{
  Multiplication(int a, int b)
  {
    this(5);
    Sopln(x * y);
  }
  Multiplication(int a)
  {
    this(); Sopln(x);
  }
  public static void main(String[] args)
  {
    Multiplication m = new Multiplication();
  }
}
```

13) Inheritance:

→ The process of acquiring the properties of state & behaviour of super class & sub class (or) acquiring the properties of from one class to other class is called as Inheritance.

→ Inheritance relationship is also called as "is a" relationship

→ We can achieve inheritance by using the keyword extends.

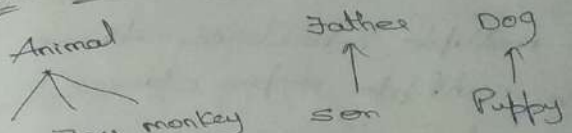
There 4 types of inheritance

i) single level inheritance
ii) multi-level inheritance
iii) Hierarchical inheritance
iv) Multiple inheritance

→ Main purpose of inheritance is to achieve code reusability.

→ With the help of inheritance we can achieve upcasting, downcasting, overriding, abstraction

Real World Example



Real Time Example

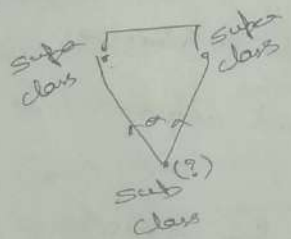
```
Class Running
{
 public static Void run()
 {
  sopln("run 5 kms");
 }
}
class singing extends Running
{
 public static Void sing()
 {
  run();
  sopln("sing for 5 min");
 }
 public static Void main (String[] args)
 {
  sing();
 }
}
```

14) Why multiple inheritance not possibly?

→ In multiple inheritance there are multiple super classes & one sub-class.

→ there is ambiguity to perform Constructor chaining & method execution.

→ Both together called as Diamond problem.



15) Method Overriding:

→ During inheritance process sub class will inherit members from super class in case sub class wants to maintain same functionality but different implementation that is when we go for method overriding.

→ In order to achieve inheritance method overriding inheritance is compulsory.

→ Method overriding can be achieved by maintaing same method signature but provide different implementation.

→ When we want to makes changes in one particular sub class without effecting other sub classes we go for method overriding.

→ In case of. method (Complete), method overriding is optional and in case of abstract method overriding is Compulsory.

Real World Ex:

what's app status (or) settings; Company employees Student Id

Real time Ex:

```
class Status
{
  public Void taaan()
  {
    sopln("Visible");
  }
}
```

```
class Hidden extends Visible
{
  public Void taaan()
  {
    sopln("Invisible");
  }
  public static Void main
      (String[] args)
  {
    taaan();
  }
}
```

16) ~~Feature~~ 02/02/21

16) Abstract:

→ Abstract is a keywood which represents incompleteness.

→ Abstract keywood can be used with class & method. If class contains atleast one incomplete method then class is declared as Abstract. For abstract method overriding is compulsory

→ By declaring method as abstract inheritance is made compulsory & method need to be over-riden by sub-class

→ By using abstract class we can achieve abstraction

Real Time Example:

```
abstract class Animal
{
abstract public Void Lion();
}
```

```
class Animal1 Extends Animal
{
public Void Lion()
{
  sopln("King of Jungle");
}
class Mainclass
{
public static Void main(String[] args)
{
  Animal1 m1 = new Animal1();
  m1.Lion();
}
}
```

Real World Example

Animal, student etc.

17) Interface:

→ Interface is one of the type definition block, using interface we can create derived data type.

→ Within interface only incomplete methods are allowed. All the methods are by default public and abstract.

→ Using interface we can achieve 100% abstraction.

→ Interface methods has to be implemented inside class this class is called implementation class

Real World Example

lights (tubelight, led light)

Real time Example

```
Interface Light
{
   void tubeLight();
   void ledLight();
}
class Functionality implements Light
{
   public void tubeLight()
   {
      sopln("we can on using switch")
   }
   public void ledLight()
   {
      sopln("we can on using detection
             of motion");
   }
class Mainclass
{
   public static void main(String[] args)
   {
      Functionality fn = new Functionality();
      fn.tubeLight();
      fn.ledLight();
   }
}
```

18) Upcasting & Downcasting

→ Converting sub class object to super class object is called as upcasting and Converting super class object to sub class object is called as Downcasting.

→ When the object is upcasted its sub class properties are hidden. When object is downcasted hidden properties becomes visible.

→ Every subclass will have only one superclass so upcasting is ~~se~~ implicit and downcasting is explicit because

one super class can have multiple sub classes.

→ Downcasting is possible only when the object is already upcasted.

Real World Example

Aparichithudu

Aeroplane

Amphibious Vehicle

Real Time Example

```
class Amphibious
{
  public void function1()
  {
    sopln("moves on the land");
  }
}
class AmpVehicle extends Amphibious
{
  public void function2()
  {
    sopln("moves under the water");
  }
}
```

```
class Mainclass
{
  public static void main(String[] args)
  {
    Amphibious a1 = (Amphibious) new AmpVehicle();
    a1.function1();
    AmpVehicle av = (AmpVehicle) a1;
    av.function1();
    av.function2();
  }
}
```

19) Method Binding :-

→ Method has 2 parts ie method signature and method implementation.

→ During coding state we just define both of them.

→ Connecting method signature & method implementation is called Method Binding

→ Method Binding 2 types

i) Compile Time Binding

ii) Runtime Binding

→ For all static methods binding will happen during Compile time because static methods will have only one method signature & implemented & static methods cannot be overrided. Compile time binding is also called as static binding & early binding.

→ It is called static binding because once the Compiler binds, it cannot be broken. It is called early binding because before execution binding is done & it is ready for execution.

→ For non-static methods usually there is one method signature & multiple implementations, hence Compiler cannot decide to bind them rather JVM will decide to bind based on object creating. Hence, it is called Run-time binding.

→ It is also dynamic binding because it (binding) changes from one implementation to other & it is also called normal late binding because non-static methods are not ready for execution, those methods are binded after execution.

* Real time Example

static Binding

```
class Human
{
 psv walk()
  {
   sopln("Human walks");
  }
}
class Boy extends Human
{
 psv walk()
  {
   sopln("Boy walks");
  }
 psvm(string[] args)
  {
   Human obj1 = new Boy();
   Human obj2 = new Human();
   obj1.walk();
   obj2.walk();
```

o/p:- Human walks
Human walks

* Real World Example

eg: mobile with battery inbuilt; pen with refill

eg: mobile charger; Earphones; pen without refill

20) Polymorphism:

→ Poly means many & morphism means forms i.e many forms. Hence, polymorphism means single entity multiple forms.

→ It is of 2 types

i) Run-time polymorphism

ii) Compile-time polymorphism

→ In order to achieve Compile time polymorphism we must follow 2 rules i.e methods must be overloaded and overloaded methods must be static only when static we can achieve binding during Compile time. Hence, it is called as Compile time polymorphism.

→ In order to achieve to Run time polymorphism we must have 3 things i.e generalisation, upcasting & overriding.

Real World Example

Light [tubelight, led, candle, torch]

Real time Example

1) Compile-Time

Example for Compile-time polymorphism is method overloading.

2) Run-time

```
Interface Flower
{
  void Colour();
}
class Rose implements Flower
{
  public void Colour()
  {
    Sopln("red in Colour");
  }
}
class Jasmine implements Flower
{
  public void Colour()
  {
    Sopln("white in Colour");
  }
}
class Mainclass
{
  psv getColour(Flower f1)
  {
    f1.Colour();
  }
```

```
psvm(String[] args)
{
  getColour(new Rose());
  getColour(new Jasmine());
 }
}
```

21.) Generalization & Specialization

→ Developing a method which can handle single type of object is called specialized method & process is called as specialization.

→ In order to achieve specialization we must have method with argument and it must be reference type of same class.

→ Developing a method which can handle multiple type of objects is called as generalized methods & this process is called as Generalization

→ In order to achieve generalization we must have method with arguments & it must be reference type of super class or interface.

→ Soo both arguments must be of derived data type.

→ Generalization helps to achieve polymorphism.

Real World Example

Vehicle (Bus, Car, Truck)
- park at station
- park at home
- load & unload Goods

Real Time Example

```
class Bus
{
  public void stop()
  {
    sopln("stops at stations");
  }
}
class Car
{
  public void park()
  {
    sopln("park at home");
  }
}
```

```
Class Mainclass
{
  psvm(string[] args)
  {
    busStop(new Bus());
    parking(new Car());
  }
  psv busStop(Bus bi)
  {
    bi.stop();
  }
  psv parking(Car ci)
  {
    ci.park();
  }
}
```

22) Access specifiers:

→ Access specifiers will help us to control the visibility of java components (class, interface, SV, NSV, SM, NSM, Constructor).

→ There are 4 types

i) Public
ii) Private
iii) Protected
iv) Default

→ If we declare any member or component as public then we can access them in same class, another class of same package & another class of diff. package after import statement.

→ Protected is same as public but here it is possible to access after import and inheritance in case of another class of different package.

→ Default is also called as package level AS. They can be accessed in same class & another class of same package. If we don't mention public, private & protected then it means it is default. There is no keyword for default AS.

→ If we declare any member as private then we can access them only in the same as

Real World Example

1) Public :- Public washrooms
2) Private :- Self mobile
3) Protected :- TV among family

Real Time Example

1) Public

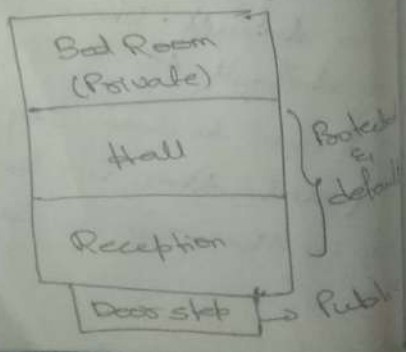
```
public class Demo
{
  public x, y, size;
}
```

2) Private

```
public class Demo
{
  private double x, y;
}
```

3) default

```
Class Demo
{
  int i;
}
```



23) Java Bean Class

→ Developing a class where every data members need to be made private & providing access to that particular variables through getters & setters. This is called as Java Bean Class.

→ The main purpose of Java Bean class is to make sure that to protect our data members from invalid values.

→ When we give access through the methods we can control the flow of data i.e we can allow only valid data & block invalid data.

Real World example

Vegetable Bean
Switch Board
Keyboard Keys

Real Time Example

```
package mypack;
```

```
public class Test
{
  psvm(String[] args)
  {
    Employee e = new Employee();
    e.setName("Arjun");
    sopln(e.getName());
  }
}
```

24) Encapsulation:

→ Wrapping data members & member functions into a single unit is called as Encapsulation.

→ class & method are examples for encapsulation.

→ class encapsulate variables, methods & Constructor where a method encapsulates logic & statements or instructions.

Real World Example

chocolate wrapper

Human Body (inner parts covered by skin)

Mobile

Real time Example

```
class Student
{
  private String name;
  public String getName()
  {
    return name;
  }
  public void setName(String name)
  {
    this.name = name;
  }
}
class Test
{
  psvm(String[] args)
```

```
Student s = new student();
s.setName("vijay");
sopln(s.getName());
}
}
```

25) Abstraction

→ Hiding implementation details & proving access to the method signature.

→ Abstraction can be achieved by abstract class & interface.

→ Using abstract class we can achieve 0-100% abstraction because abstract class allows Complete meth

→ using interface we can achieve always 100% abstraction.

→ steps to be followed to achieve abstraction are

- Creat an interface
- Develop method signature inside interface
- create implementation class and implement the method.
- Develop helpa class in which the helpa method is created in which object of implementation class is created and upcast to interface.
- Return the object in the form of interface

Real World Example

Human Body; Ink flow of pen; water tap; shower; light; fan etc

Real time Example

```
interface Pencil
{
  void board();
}
class Natraj implements Pencil
{
  public void board()
```

```
    {
      soPln("top board in India");
    }
}

class Run
{
  public static Pencil getObject()
  {
    Pencil p1 = new Nataraj();
    return p1;
  }
}
class Mainclass
{
  psvm(String[] args)
  {
    Pencil p2 = Run.getObject();
    p2.board();
  }
}
```

26) Object class

→ Object class is an inbuilt class present in java.lang package and it is supa class to all the classes in java.

→ Every class inherits methods from object class

→ Three imp. methods in object class are

i) toString()

ii) equals()

iii) hashCode()

→ hashCode() is mainly used in a Collection called as Set type of collection.

→ toString() : By default the o/p of this method is PN.CN@obj address. We can override toString() to return the useful info. of our class or object

→ By default equals() compares object address i.e current object address is compared with the given object address.

→ We can override equals() based to compare based on features and properties.

27) String class

→ String is a class present in java.lang package. String is a final class i.e it cannot be further inherited. It comes under Derived Data type.

→ We can create String object in 2 ways
i) new operator
ii) " " (literals)

→ new operator will create String object in non-constant pool area & literal will create string object in constant pool area.

→ Non-constant pool area allows duplicate string objects whereas constant pool area does not allow duplicate string objects.

→ The main property of string is it is immutable i.e the information present in it cannot be modified.

→ String object is immutable because single objects can have multiple reference variables. If one ref. var. manipulates string object then other ref. var. get affected.

→ String methods(explain)

28) Wrapper class

→ Java is not pure object oriented programming language because it allows primitive data types.

→ To represent primitive data types in the form of object we go for Wrapper classes.

→ Wrappa class says ... primitive data type there is a corresponding Wrappa class.

→ Representing primitive data in the form of object we call the process as Boxing and representing same object back to primitive data we call this process as Unboxing.

→ From JDK 1.0 to 1.4 version boxing and unboxing is explicit whereas from JDK 1.5 to latest version i.e Java is both boxing & unboxing is implicit & explicit.

→ All Wrappa classes are present in java.lang package.

→ All Wrappa classes are final classes.

→ Wrappa classes plays major role in Collections bcoz Collections does not allow primitive data type it says in and out everything is object

29) Exception:

→ There are 2 types of statements i, normal statement ii, Dangerous Statement

→ Exceptions are created when these dangerous statements behaves abnormally.

→ In Java, exception is an object created by JVM whenever it is executing dangerous statement which behaves abnormally.

→ Abn exception object is created JVM will throw the exception object.

→ If program does not catch the exception then programs gets terminated. In order to handle this exception we go for try & catch block.

→ Within try block we have the dangerous statements & within catch block we have recovery statements. Catch block will catch the exception and recover from exception.

→ Exception is Categorized into 2 types

i) check Exception

ii) Uncheck Exception.

→ The dangerous statements identified by the developers come under unchecked exception. whereas the dangerous statements identified by the Compiler comes under checked exception.

→ For unchecked exception super class is run-time exception & for checked exception super class is exception.

→ Ultimate super class for exception Category is Throwable.

→ If Current method is unable to recover from the exception caused method can propagate the exception to calling method this is called Exception propagation.

→ In Case of unchecked exception exception propagation is implicit.

→ In Case of checked exception exception propagation has to be done by using keyword "throws."

→ Not only inbuilt exceptions we can create our own exceptions we call this as user defined exceptions.

→ After creating user defined exception we can throw our exception object by using keyword "throw"

→ An exception handling there 5 keywords

i) try() :- dangerous statements

ii) Catch() :- recovery statements

iii) finally :- Termination statements

iv) throws :- used for exception propagation in Case of checked exception

v) throw :- used for throwing actual exception object.

30) Threads:-

→ Thread is a path given for execution. Thread is another name for stack area.

→ Whenever we run any java program an inbuilt thread is created for main method, when we want to run all our methods in seperate path that is when we create user defined thread.

→ There are 2 ways to create user-defined thread

i) Extending from thread class

ii) By implementing runnable interface

→ Steps to create thread

i) Inherit from thread class

ii) override run() and all that method which needs to be executed in seperate thread.

iii) We must create object of our class & start the thread.

→ start() functionality has 2 jobs

i) sends request to thread scheduler to ask for seperate thread

ii) invoke run() which executes in seperate path.

→ This is how we create user defined thread using thread class.

→ We have another way that is by implementing runnable interface.

→ Steps

i) ~~create~~ implement runnable interface

ii) It has only one method ie run() so override run()

iii) Here we cannot directly start the thread because it does not have start()

→ so, we must create thread class object & pass runnable type object as input for thread class object.

→ Thread will start at thread on behalf of runnable interface.

→ Every thread has 3 properties

i) id
ii) name
iii) priority

→ All three are private we can access through getters and setters.

→ We can get the id but we cannot set the id.

→ We can get & set for both name and priority.

3) Difference between class & Object

Class	Object
→ It is logical entity.	→ It is real entity.
→ A template for creating or instantiating objects within a program	→ An instance of a class
→ Declared with "class" keyword	→ Declared using "new" keyword
→ A class is declared once.	→ Multiple objects are created using class
→ Class does not get any memory when it is created.	→ Object gets memory when they are created

32) Advantages of Encapsulation

i) Reusability:

→ You can reuse the code and implement new requirements in your program.

ii) Easy debugging and testing

→ The code which is encapsulated is easy to debug and enables unit testing.

33) Advantages of Abstraction

1) Helps to increase security of an application or program as only important details are provided to the user.

2) It avoids code duplication.

3) Increases reusability.

4) The design & implementation gets benefitted as it reduces the complexity of the codebase.

---

34) Advantages of Polymorphism

1) It helps programmers reuse the code and classes. Once written they can be used in many ways.

2) Polymorphism helps in reducing the coupling between diff functionalities

3) Increased code extensibility