Java Interview Question and Answers

➢ Topic – Java OOP'S Concepts:

Frequently Asked OOPs Interview Questions in Java

Java Encapsulation Interview Questions Answers

1. What is Encapsulation in Java? Why is it called Data hiding?
Ans: The process of binding data and corresponding methods (behavior) together into a single unit is called encapsulation in Java.

In other words, encapsulation is a programming technique that binds the class members (variables and methods) together and prevents them from being accessed by other classes, thereby we can keep variables and methods safes from outside interference and misuse.

If a field is declared private in the class then it cannot be accessed by anyone outside the class and hides the fields within the class. Therefore, Encapsulation is also called data hiding.

2. What are the important features of Encapsulation?
Ans: Encapsulation means combining the data of our application and its manipulation in one place. It allows the state of an object to be accessed and modified through behavior. It reduces the coupling of modules and increases the cohesion inside them.

3. What is the advantage of Encapsulation?
Ans: There are the following advantages of encapsulation in Java. They are as follows:

- The encapsulated code is more flexible and easy to change with new requirements.
- It prevents the other classes to access the private fields.
- Encapsulation allows modifying implemented code without breaking other code who have implemented the code.

- It keeps the data and codes safe from external inheritance. Thus, Encapsulation helps to achieve security.
- It improves the maintainability of the application.

4. What are the main benefits of using encapsulation in Java?

Ans: The main benefits of using encapsulation are:

- The main benefit of encapsulation is the ability to modify the implemented code without breaking the others code who have implemented the code.
- It also provides us with maintainability, flexibility, and extensibility to our code.
- The fields of a class can be made read-only or write-only.
- A class can have total control over what is stored in its fields.

5. How to achieve encapsulation in Java? Give an example.

Ans: There are two key points that should be kept in mind to achieve the encapsulation in Java. They are as follows:

- Declare the variable of the class as private.
- Provide public setter and getter methods to modify the values of variables.

Let's understand it with the help of an example program.

```
public class EncapsulationTest{

 private String name;

 private String idNum;

 private int age;



public int getAge() {

  return age;

}

public String getName() {

  return name;
```

```
}

public String getIdNum() {

   return idNum;

}

public void setAge( int newAge) {

   age = newAge;

}

public void setName(String newName) {

   name = newName;

}

public void setIdNum( String newId) {

   idNum = newId;

 }

}
```

6. What is data hiding in Java?

Ans: An outside person cannot access our internal data directly or our internal data should not go out directly. This oops feature is called data hiding n Java. After validation or authentication, the outside person can access our internal data.

7. How to achieve Data hiding programmatically?

Ans: By declaring data members (variables) as private, we can achieve or implement data hiding. If the variables are declared as private in the class, nobody can access them from outside the class.

The biggest advantage of data hiding is we can achieve security.

8. What is a Tightly encapsulated class in Java?

Ans: If each variable is declared as private in the class, it is called tightly encapsulated class in Java. For tightly encapsulated class, we are not required to check whether class contains getter and setter method or not and whether these methods are declared as public or not.

9. What is the difference between Abstraction and Encapsulation?
Or, how abstraction is different from encapsulation in Java?
Ans: There are the following differences between Abstraction and Encapsulation:

a) Abstraction solves the problem at the design level whereas encapsulation solves the problem at the implementation level.

b) Abstraction is implemented in Java using Interface and Abstract class whereas encapsulation is implemented using private and protected access modifiers.

c) Abstraction is used to hide the unwanted data and giving relevant data whereas encapsulation is used for hiding data and code in a single unit to prevent access from outside.

d) The real-time example of Abstraction is TV Remote Button whereas the real-time example of Encapsulation is medical medicine.

10. Can we achieve abstraction without encapsulation in Java?
Ans: Yes, we can achieve abstraction without encapsulation because both are different things and different concepts.

11. What are getter and setter methods in Java?
Ans: In Java, setter method is a method that is used for updating the values of a variable. This method is also known as mutator method.

Getter method is a method that is used to retrieve the value of a variable or return the value of the private member variable. This method is also known as an accessor method.

12. In the following code, radius is declared as private in the class Circle, and myCircle is an instance of class Circle. Does the code cause any error problems? If so, explain why?

```
class Circle {

 private double radius = 1;
```

```
/** Find area of the circle */


 public double getArea() {

            return radius * radius * Math.PI;

}

public static void main(String[] args) {

  Circle myCircle = new Circle();



  System.out.println("Radius is " + myCircle.radius);

  System.out.println("Area of cirle: " +myCircle.getArea());

 }

}
```

Ans: No, the above code will not create any problem. The code will be compiled successfully. The output is: Radius is 1.0, Area of cirle: 3.141592653589793.

13. Does reflection violates encapsulation in Java?

Ans: Reflection violates encapsulation because it reveals the internal data structures.

14. Explain design pattern based on encapsulation in java?

Ans: In many design patterns, Java uses the encapsulation technique and one of them is Factory pattern which is used to create the objects.

Factory pattern is a better choice in creating the object of those classes whose creation logic can vary. It is also used for creating different implementations of the same interface.

'BorderFactory class' of JDK is a good example of encapsulation in Java which creates different types of 'border' and encapsulates creation logic of border.

15. How can the variable of the EncapsulationTest be accessed by using getter and setter methods?

Ans: The public setXXX() and getXXX() methods are access points of the instance variable of EncapsulationTest class. Basically, these methods are known as getter and setter methods.

Therefore, any class that wants to access variable should access them through these getters and setters. The variables of the EncapsulationTest class can be accessed as shown in the following code:

```
public class RunEncapTest {

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

{

 EncaptulationTest encap = new EncapsulationTest();

  encap.setName("John");

  encap.setAge(22);

  encap.setId(123456);



 System.out.println("Name: " +encap.getName()); System.out.println("Age: "
+encap.getAge());

 System.out.println("Id: " +encap.getId());

}

}
```

This would produce following output: Name: John Age: 22 Id: 123456.

Java Inheritance Interview Questions Answers

Java Inheritance Interview Questions

1. What is Inheritance in Java?

Ans: The technique of creating a new class by using an existing class functionality is called inheritance in Java. In other words, inheritance is a process where a child class acquires all the properties and behaviors of the parent class.

2. Why do we need to use inheritance?

Or, What is the purpose of using inheritance?

Ans: Inheritance is one of the main pillars of OOPs concept. Some objects share certain properties and behaviors. By using inheritance, a child class acquires all properties and behaviors of parent class.

There are the following reasons to use inheritance in java.

- We can reuse the code from the base class.
- Using inheritance, we can increase features of class or method by overriding.
- Inheritance is used to use the existing features of class.
- It is used to achieve runtime polymorphism i.e method overriding.

3. What is Is-A relationship in Java?

Ans: Is-A relationship represents Inheritance. It is implemented using the "extends" keyword. It is used for code reusability.

4. What is super class and subclass?

Ans: A class from where a subclass inherits features is called superclass. It is also called base class or parent class.

A class that inherits all the members (fields, method, and nested classes) from other class is called subclass. It is also called a derived class, child class, or extended class.

5. How is Inheritance implemented/achieved in Java?

Ans: Inheritance can be implemented or achieved by using two keywords:

1. **extends:** extends is a keyword that is used for developing the inheritance between two classes and two interfaces.

2. **implements:** implements keyword is used for developing the inheritance between a class and interface.

6. Write the syntax for creating the subclass of a class?

Ans: A subclass can be created by using the "extends" keyword. The syntax for declaring a subclass of class is as follows:

```
class subclassName extends superclassName

{

  // Variables of subclass

  // Methods of subclass

}
```

where class and extends are two keywords.

7. Which class in Java is superclass of every other class?

Ans: In Java, Object class is the superclass of every other class.

8. How will you prove that the features of Superclass are inherited in Subclass?

Ans: Refer to this tutorial: Inheritance in Java with Realtime Example

9. Can a class extend itself?

Ans: No, a class cannot extend itself.

10. Can we assign superclass to subclass?

Ans: No.

11. Can a class extend more than one class?

Ans: No, one class can extend only a single class.

12. Are constructor and instance initialization block inherited to subclass?

Ans: No, constructor and instance initialization block of the superclass cannot be inherited to its subclass but they are executed while creating an object of the subclass.

13. Are static members inherited to subclass in Java?

Ans: Static block cannot be inherited to its subclass.

A static method of superclass is inherited to the subclass as a static member and non-static method is inherited as a non-static member only.

14. Can we extend (inherit) final class?
Ans: No, a class declared with final keyword cannot be inherited.

15. Can a final method be overridden?
Ans: No, a final method cannot be overridden.

16. Can we inherit private members of base class to its subclass?
Ans: No.

17. What is order of calling constructors in case of inheritance?
Ans: In case of inheritance, constructors are called from the top to down hierarchy.
18. Which keyword do you use to define a subclass?
Or, which keyword is used to inherit a class?
Ans: extends keyword.

19. What are the advantages of inheritance in Java?
Ans: The advantages of inheritance in java are as follows:

- We can minimize the length of duplicate code in an application by putting the common code in the superclass and sharing it amongst several subclasses.
- Due to reducing the length of code, the redundancy of the application is also reduced.
- Inheritance can also make application code more flexible to change.

20. What are the types of inheritance in Java?
Ans: The various types of inheritance are as follows:

a. Single inheritance
b. Multi-level inheritance
c. Hierarchical inheritance
d. Multiple inheritance
e. Hybrid inheritance

21. What are the various forms of inheritance available in Java?
Ans: The various forms of inheritance to use are single inheritance, hierarchical inheritance, and multilevel inheritance.

22. What is single inheritance and multi-level inheritance?

Ans: When one class is extended by only one class, it is called single level inheritance. In single-level inheritance, we have just one base class and one derived class.

A class which is extended by a class and that class is extended by another class forming chain inheritance is called multilevel inheritance.

23. What is Multiple inheritance in Java?
Ans: A class that has many superclasses is known as multiple inheritance. In other words, when a class extends multiple classes, it is known as multiple inheritance.

24. Why multiple inheritance is not supported in java through class?
Ans:
Multiple inheritance means that one class extends two superclasses or base classes but in Java, one class cannot extend more than one class simultaneously. At most, one class can extend only one class.

Therefore, to reduce ambiguity, complexity, and confusion, Java does not support multiple inheritance through classes.

For more detail, go to this tutorial: Types of Inheritance in Java
25. How does Multiple inheritance implement in Java?
Ans: Multiple inheritance can be implemented in Java by using interfaces. A class cannot extend more than one class but a class can implement more than one interface.

26. What is Hybrid inheritance in java? How will you achieve it?
Ans: A hybrid inheritance in java is a combination of single and multiple inheritance. It can be achieved through interfaces.

27. How will you restrict a member of a class from inheriting its subclass?
Ans: We can restrict members of a class by declaring them private because the private members of superclass are not available to the subclass directly. They are only available in their own class.

28. Can we access subclass members if we create an object of superclass?
Ans: No, we can access only superclass members but not the subclass members.

29. Can we access both superclass and subclass members if we create an object of subclass?

Ans: Yes, we can access both superclass and subclass members.

30. What happens if both superclass and subclass have a field with the same name?

Ans: Only subclass members are accessible if an object of subclass is instantiated.

31. Will the code successfully compiled? If yes, what is the output?

```
public class A {

  int x = 20;

}

public class B extends A {

  int x = 30;

}

public class Test {

public static void main(String[] args)

{

  B b = new B();

  System.out.println(b.x);



  A a = new A();

  System.out.println(a.x);



  A a2 = new B();

  System.out.println(a2.x);
```

```
 }

}
```

Ans: Yes, code will be successfully compiled. The output is 30, 20, 20.

32. Which of the following is correct way of inheriting class A by class B?

a) class B + class A { }
b) class B inherits class A { }
c) class B extends A { }
d) class B extends class A { }

Ans: c

33. Is interface inherited from the Object class?

Ans: No.

34. What is the output of executing code of class Test?

```
public class A {

 void m1() {

     System.out.println("m1 in class A");

 }

}

public class B extends A {

 void m1() {

     System.out.println("m1 in class B");

 }

}

public class Test {

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

```
{
   B b = new B();
    b.m1();

   A a = new A();
    a.m1();

   A a2 = new B();
    a2.m1();
 }
}
```

Ans: The result is m1 in class B, m1 in class A, m1 in class B.

34. Will this code run successfully? If yes, what will be the output?

```
public class A
{
  private int x = 50;
  void m1() {
     System.out.println(x);
 }
}
public class B extends A {
```

```
}

public class Test {

public static void main(String[] args)

{

  A a = new B();

   a.m1();

 }

}
```

Ans: Yes, code will be successfully compiled. The output will be 50.

35. Will this code compile successfully? If yes, what is the output? If no, identify the errors.

```
package pack1;

public class A

{

  private int x = 50;

  protected int y = 100;

   int z = 200;

}

package pack2;

import pack1.A;

public class B extends A {
```

```
}

import pack2.B;

public class Test {

public static void main(String[] args)

{

  B b = new B();

  System.out.println(b.x);



  System.out.println(b.y);

  System.out.println(b.z);

 }

}
```

Ans: No, the code will not compile successfully because of two compile-time errors.

- First error is in line System.out.println(b.x); because private members cannot be accessed in the subclass.
- Second error is in line System.out.println(b.y); because default members of superclass can be accessed in the subclass within the same package only.

36. How will you prohibit inheritance in Java?

Ans: If you declare a class as final, it cannot be extended. Thus, we can prohibit the inheritance of that class in Java.

37. In the following code, we have created 3 Classes A, B, and C. Class C extends Class B and Class B extends Class A.
Each class has a method m1(), is there any way to call A's m1() method from Class C?

```
public class A
{
  void m1(){
    System.out.println("m1 in class A");
 }
}
public class B extends A
{
  void m1() {
     System.out.println("m1 in class B");
 }
}
public class C extends B
{
  void m1() {
              System.out.println("m1 in class C");
   }
 }
public class Test {
public static void main(String[] args)
{
```

```
  C c = new C();

   c.m1();

 }

}
```

Ans: In the above code, every class has m1() method with the same signature thus Class B is overriding A's m1() method and Class C is overriding Class B's m1() method.

Now, it is possible for class B and class C to call their super class's m1() method by using super.m1() call.

But here, the question is asking to invoke A's m1() method from Class C, which is not possible because it violates the OOPs concept in Java and also not super.super in java.

Since Java does not allow multiple inheritance, that means C can see only one superclass which will have just one m1() method implementation. C can never see A's m1() method.

We also call this scenario as the Diamond Problem of Multiple Inheritance. The only way to call A's m1() method from Class C is if Class B and class C call super.m1() method in its m1() implementation.

38. What will be the output of the following program?

```
public class A

{

 void m1() {

    System.out.println("m1 in class A");

 }

}

public class B extends A

{
```

```
 void m1() {
     super.m1();
     System.out.println("m1 in class B");
  }
}
public class C extends B
{
   void m1() {
     System.out.println("m1 in class C");
   }
 }
public class Test {
public static void main(String[] args)
{
   C c = new C();
    c.m1();
  }
}
```

Ans: Output: m1 in class C

39. Find out the output of the following program.

```
public class A
{
```

```
  void m1() {
     System.out.println("m1 in class A");
 }
}
public class B extends A
{
void m1() {
    super.m1();
    System.out.println("m1 in class B");
  }
}
public class C extends B
{
 void m1() {
   System.out.println("m1 in class C");
   super.m1();
  }
 }
public class Test {
public static void main(String[] args)
{
```

```
  C c = new C();

   c.m1();

 }

}
```

Ans: Output: m1 in class C, m1 in class A, m1 in class B

40. Identify the error in the following code.

```
final class A

{

 void m1() {

    System.out.println("m1 in class A");

 }

}

public class B extends A

{

public static void main(String[] args)

{

   B b = new B();

 }

}
```

Ans: A class declared with final keyword cannot be extended.

41. Will the code compile successfully? If not, identify the error in the following code.

```
class A
```

```
{
  int x = 50;
 private A() {
   System.out.println(x);
  }
}
public class B extends A
{
public static void main(String[] args)
{
   B b =new B();
 }
}
```

Ans: No, the code will not compile successfully because the constructor is private, the class cannot be inherited.

42. What problem will arise when the following code is compiled?

```
class A {
 public A(int a) {


  }
}
public class B extends A {
```

```
public B() {



  }

}

public class Test {

public static void main(String[] args)

{

   B b = new B();

  }

}
```

Ans: Java compiler will generate compile-time error: Implicit super constructor A() is undefined. Must explicitly invoke another constructor

43. How many ways to implement relationships among classes in Java?
Ans: There are four ways to make relationships among classes. They are as follows:

a. Association
b. Aggregation
c. Composition
d. Inheritance

44. What is Association in OOPs concept?
Ans: Association in Java is one of the core concepts of OOPs. It establishes the relation between two classes that are independent of one another.

In association, all objects have their own life cycle and there is no ownership between objects. Association can be unidirectional or bidirectional.

45. In the OOPs concept, what is meant by Aggregation?
Ans: Aggregation in java represents has-a relationship. It is a special form of association that represents an ownership relation between two objects.

In aggregation, two aggregated objects have their own life cycle but one of the objects is the owner of has-a relationship.

46. What is meant by Composition in OOPs?
Ans: Composition in java is one of the core concepts of OOPs and a more restrictive case of aggregation. It represents has-a relationship that contains an object and cannot exist on its own.

In simple words, if a class object holds an object of another class, it is called composition. It establishes strong relationship between two objects than aggregation.

47. Which is more tightly bound? Inheritance or Composition?
Ans: Composition is more tightly bound than inheritance.

48. What is the difference between Inheritance and Composition?
Ans: Inheritance represents Is-A relationship between subclass and its superclass whereas, composition represents Has-A relationship between classes.

49. When will you prefer composition over inheritance?
Ans: Inheritance can be used only when there is a perfect IS-A relationship between the superclass and subclass definitions. But in case of any confusion, prefer composition over inheritance.

50. How aggregation and composition both are different concepts?
Ans: In the OOPs concept, aggregation and composition both are types of association relations. A composition establishes a strong relationship between classes.

If the composite object is destroyed, all its parts are destroyed. For example, a car has a steering wheel. If car object is destroyed, there is no meaning to be the existence of steering wheel.

Aggregation establishes a weaker relationship between classes than composition. For example, a library has students. If a library is destroyed, students still exist. Hence, library and student are related by aggregation.

Java Abstract Class Interview Questions Answers

Abstract Class Interview Questions and Answers for best Practice

1. What is Abstraction in Java?

Ans: Abstraction in Java is a technique by which we can hide the data that is not required to users. It hides all unwanted data so that users can work only with the required data.

2. How to achieve or implement Abstraction in Java?

Ans: There are two ways to implement abstraction in java. They are as follows:

a) Abstract class (0 to 100%)
b) Interface (100%)

3. What is Abstract class in Java? How to define it?

Ans: An abstract class in java is a class that is declared with an abstract keyword.

Example of Abstract class:

```
abstract class Test {

abstract void show();

}
```

4. What is the difference between abstract class and concrete class?

Ans: There are mainly two differences between an abstract class and concrete class. They are:

a) We cannot create an object of abstract class. Only objects of its non-abstract (or concrete) sub classes can be created.

b) It can have zero or more abstract methods that are not allowed in a non-abstract class (concrete class).

5. What is Abstract in Java?

Ans: Abstract is a non-access modifier in java that is applicable for classes, interfaces, methods, and inner classes.

6. Can abstract modifier applicable for variables?

Ans: No.

7. What is Abstract method in Java?

Ans: A method which is declared with abstract modifier and has no implementation (means no body) is called abstract method in java.

It does not contain any body. It has simply a signature declaration followed by a semicolon. It has the following general form as given below.

```
Syntax:

    abstract type MethodName(arguments); // No body



For example:

    abstract void msg(); // No body.
```

8. Can an abstract method be declared as static?
Ans: No.

9. Can an abstract method be declared with private modifier?
Ans: No, it cannot be private because the abstract method must be implemented in the child class. If we declare it as private, we cannot implement it from outside the class.

10. What is Concrete method in Java?
Ans: A concrete method in Java is a method which has always the body. It is also called a complete method in java.

11. When to use Abstract class in Java?
Ans: An abstract class can be used when we need to share the same method to all non-abstract sub classes with their own specific implementations.

12. When to use Abstract method in Java?
Ans: An abstract method can be used

a) When the same method has to perform different tasks depending on the object calling it.
b) When you need to be overridden in its non-abstract subclasses.

13. Is abstract class a pure abstraction in Java?
Ans: No, It provides 0 to 100% abstraction.

14. Is it possible to create an object of abstract class in Java?
Ans: No. It is not possible but we can create an object of its subclass.

15. Is it possible that an abstract class can have without any abstract method?
Ans: Yes.

16. Can an abstract class have constructor?
Ans: Yes.

17. Is abstract class allow to define private, final, static, and concrete methods?
Ans: Yes.

18. Is it possible to achieve multiple inheritance through abstract class?
Ans: No.

19. Can we make an abstract class without abstract keyword?
Ans: No, a class must be declared with abstract keyword to make an abstract class.

20. Can we define an abstract method inside non-abstract class (concrete class)?
Ans: No, we cannot define an abstract method in non-abstract class.

For example:

```
class Test {

  abstract void show();

}
```

The above code will generate a compile-time error.

21. Which among the following code have errors?

```
a) abstract class A {

    void m1();

  }

b) public class A {

    abstract void m1();

  }
```

```
c) abstract public class A {
   abstract void m1();
  }
d) abstract public class A {
    void m1() { }
  }
e) public abstract class A {
   abstract void m1();
   A(){ }
   void m2() { }
   }
f) public abstract class A {
    abstract int x = 100;
    abstract void m1();
    abstract void m2();
   }
g) public abstract class A {
   abstract void m1();
  }
  public class Test {
  public static void main(String[] args) {
```

```
  A a = new A();
  }
 }
h) public abstract class A {
  abstract void m1();
  A(){}
  static void m2() {System.out.println("Hello Java!"); }
  }
  public class B extends A {
   void m1(){
     A.m2();
   }
  }


i) public abstract class A {
  abstract void m1();
 private A(){}
 }
 public class B extends A {}
```

Ans: a, b, f, g, i.

22. Will the following code compile successfully? If yes, what will be the output of program?

```
public abstract class A {

  abstract void m1(A a);

}

public class B extends A {

void m1(A a) {

  System.out.println("One");

 }

}

public class C extends B {

void m1(B b) {

 System.out.println("Two");

 super.m1(new B());

 }

}

public class Test {

public static void main(String[] args) {

C c = new C();

 c.m1(new B());

  }

}
```

Ans: Yes, the above code will be compiled successfully. The output of above program is Two, One.

23. What will happen if we do not override all abstract methods in subclass?

Or, what will happen if we do not provide implementation for all abstract methods in subclass?

Ans: Java compiler will generate compile time error. We will have to override all abstract methods in subclass.

24. What is the difference between Abstraction and Encapsulation?

Ans: Abstraction hides the implementation details from users whereas, encapsulation wraps (binds) data and code into a single unit.

25. Why abstract class has constructor even though you cannot create object?

Ans: We cannot create an object of abstract class but we can create an object of subclass of abstract class. When we create an object of subclass of an abstract class, it calls the constructor of subclass.

This subclass constructor has a super keyword in the first line that calls constructor of an abstract class. Thus, the constructors of an abstract class are used from constructor of its subclass.

If the abstract class doesn't have constructor, a class that extends that abstract class will not get compiled.

26. Why should we create reference to superclass (abstract class reference)?

Ans: We should create a reference of the superclass to access subclass features because superclass reference allows only to access those features of subclass which have already declared in superclass.

If you create an individual method in subclass, the superclass reference cannot access that method. Thus, any programmer cannot add their own additional features in subclasses other than whatever is given in superclass.

27. What is the advantage of Abstract class in Java?

Ans: The main advantages of using abstract class are as follows:

- Abstract class makes programming better and more flexible by giving the scope of implementing abstract methods.
- Programmer can implement abstract method to perform different tasks depending on the need.

- We can easily manage code.

28. Will the code compile successfully? If yes, what will be the output?

```
public abstract class A {

abstract void m1();

}

public class B extends A {

 void m1(){

  System.out.println("m1 in class B");

 }

}

public class Test {

public static void main(String[] args) {

B b = new B();

 b.m1();

  }

}
```

Ans: Yes, the code will be compiled successfully. The output is m1 in class B.

29. Consider the below given code.

```
public abstract class A {

abstract void m1();

void m2(){

 System.out.println("One”);

 }
```

```
}
```

How to call m2() method in the above code?
Ans: Make it static and call as A.m2();

30. Identify the errors in the following code.

```
public abstract class A {

abstract void m1();

void m2(){

 System.out.println("One");

 }

}

public class B extends A {

 void m2(){

  System.out.println("Two");

 }

}

public class Test {

public static void main(String[] args) {

B b = new B();

 b.m2();

  }

}
```

Ans: The abstract method m1() has been not implemented (overridden) in subclass B. Therefore, we will get a compile-time error.

31. Will the given code compile successfully? If yes, how?

a)

```
public abstract class A {

abstract void m1();

}

public class B extends A {

 private void m1(){

  System.out.println("One");

 }

}

public class Test {

public static void main(String[] args) {

B b = new B();

 b.m1();

  }

}
```

Ans: No, the code will not be compiled successfully because we cannot reduce the visibility of the inherited method from A.

b)

```
public abstract class A {

abstract void m1();

}

public class B extends A {
```

```
 protected void m1(){
  System.out.println("One");
 }
}
public class Test {
public static void main(String[] args) {
B b = new B();
 b.m1();
  }
}
```

Ans: Yes, the code will be compiled successfully. The output is One.

c)

```
public abstract class A {
public abstract void m1();
}
public class B extends A {
 protected void m1() {
    System.out.println("One");
 }
}
public class Test {
public static void main(String[] args) {
```

```
B b = new B();

 b.m1();

  }

}
```

Ans: No, compile-time error because we cannot reduce the visibility of inherited method m1() from A.

d)

```
public abstract class A {

private abstract void m1();

}

public class B extends A {

 protected void m1(){

  System.out.println("One");

 }

}

public class Test {

public static void main(String[] args) {

B b = new B();

 b.m1();

  }

}
```

Ans: Abstract method m1() in class A cannot be private.

e)

```
public abstract class A {

protected abstract void m1();

}

public class B extends A {

 void m1(){

  System.out.println("One");

 }

}

public class Test {

public static void main(String[] args) {

B b = new B();

 b.m1();

  }

}
```

Ans: No, compile time error.

f)

```
public abstract class A {

protected abstract void m1();

}

public class B extends A {
```

```
 protected void m1(){
  System.out.println("One");
 }
}
public class Test {
public static void main(String[] args) {
B b = new B();
 b.m1();
  }
}
```

Ans: Yes, the code will be compiled successfully. The output is One.

g)

```
public abstract class A {
static abstract void m1();
}
public class B extends A {
 void m1(){
  System.out.println("One");
 }
}
public class Test {
public static void main(String[] args) {
```

```
B b = new B();

 b.m1();

  }

}
```

Ans: No, compile-time error because we cannot set non-access modifiers with abstract method m1() in class A.

32. Will the below code compiled successfully? If yes, what will be the output of the program?

```
public abstract class A {

abstract void m1();

abstract void m2();

}

public class B extends A {

 void m1(){

  System.out.println("One");

 }

 void m2(){

  System.out.println("Two");

 }

 void m3() {

  System.out.println("Three");

 }

}
```

```
public class Test {

public static void main(String[] args) {

A a = new B();

 a.m1();

 a.m2();

 a.m3();

  }

}
```

Ans: No, compile-time error because we cannot call m3() method defined in class B.

33. Will the below code be compiled successfully? If yes, what will be the output of the following program?

```
public abstract class A {

abstract void m1();

 A(){ System.out.println("Hello"); }

 static void m2() {System.out.println("Hello Java!"); }

}

public class B extends A {

 void m1(){

  A.m2();

 }

}

public class Test {
```

```
public static void main(String[] args) {
 B b = new B();
  b.m1();
 }
}
```

Ans: Yes, Output is Hello, Hello Java!

34. What will be the output of the following program?

```
public abstract class A {
abstract void m1();
 A(){ System.out.println("Red"); }
 static void m2() {System.out.println("Orange"); }
}
public class B extends A {
 void m1(){
  A.m2();
  System.out.println("Pink");
 }
}
public class C extends B {
 void m3(){
    System.out.println("Green");
  }
```

```
}

public class Test {

public static void main(String[] args) {

C c = new C();

 c.m3();

 c.m1();

  }

}
```

Ans: Output: Red, Green, Orange, Pink.

35. Identify the error in the below code and correct the error.

```
public abstract class A {

abstract void m1(int x, int y);

}

public class B extends A {

 void m1(){

  System.out.println(" ");

 }

}
```

Ans: The class B must implement inherited abstract method as void m1(int x, int y) { System.println.out(" ");}

36. Correct the given code and find out the output?

```
public abstract class A {

abstract void m1(int x, double y);
```

```
}

public class B extends A {

 void m1(double y, int x){

  System.out.println("Hello Java!");

 }

}

public class Test {

public static void main(String[] args) {

A a = new B();

 a.m1(20, 20.50);

 }

}
```

Ans: Correction code is:

```
public class B extends A {

 void m1(int x, double y){

  System.out.println("Hello Java!");

 }

}
```

The output is “Hello Java!”.

37. What will be the output of the below program?

```
public abstract class A {

 private int x;
```

```
 A(int x){
  System.out.println("Value of x: " +x);
 }
  abstract void m1(int x, double y);
}
public class B extends A {
 private int y;
 B(int y){
   super(10);
   System.out.println("Value of y: " +y);
}
void m1(int x, double y){
  System.out.println("One");
 }
void m2(){
  System.out.println("Two");
  this.m1(5, 10.50);
 }
}
public class C extends B {
 C(){
```

```
 super(30);

 }

 void m1(int x, double y){

   super.m1(10, 15.15);

   System.out.println("Three");

  }

}

public class Test {

public static void main(String[] args){

B b = new C();

 b.m1(10, 20.50);

 b.m2();

  }

}
```

Ans: Output is:

Value of x: 10
Value of y: 30
One
Three
Two
One
Three

38. What will be the output of the below program?

```
public abstract class A {

 abstract void m1(int x, double y);
```

```
 abstract void m2(String name);

}

public class B extends A {

void m1(int x, double y) {

  System.out.println("One");

}

void m2(String name){

  System.out.println("Two");

 }

}

public class C extends B {

static void m1(){

 super.m1(20, 30);

  }

 }

public class Test {

public static void main(String[] args){

 C.m1();

  }

}
```

Ans: The code cannot be compiled because super keyword cannot be used in static method.

39. What will be the output of the following program?

```
public abstract class A {

abstract void m1(int x, double y);

abstract void m2(String name);

}

public class B extends A {

void m1(int x, double y){

  System.out.println("One");

}

void m2(String name){

  System.out.println("Two");

}

void m3(){

 System.out.println("Three");

}

void m4(){

 System.out.println("Four");

 }

}

public class C extends A {
```

```
 B b;

void m1(int x, double y){

 b = new B();

 b.m3();

}

void m2(String name){

  b = new B();

  b.m4();

  }

}

public class Test {

public static void main(String[] args){

A a = new C();

 a.m1(20, 30.0);

 a.m2("abc");

  }

}
```

Ans: Output is Three, Four

40. What will be the output of the below program?

```
public abstract class A {

abstract void m1(A a);

abstract void m2(A a);
```

```
}
public class B extends A {
void m1(A a) {
  System.out.println("One");
}
void m2(A a) {
  System.out.println("Two");
 }
}
public class C extends B {
void m1(B b) {
 System.out.println("Three");
 super.m1(new B());
}
void m2(B b) {
 System.out.println("Four");
 super.m2(new B());
  }
 }
public class Test {
public static void main(String[] args){
```

```
C c = new C();

 c.m1(new B());

 c.m2(new B());

  }

}
```

Ans: Output: Three, One, Four, Two.

Interview Questions on Polymorphism in Java

Java Polymorphism Interview Questions and Answers

1. What is Polymorphism in Java OOPs?

Ans: Polymorphism in java is one of the core concepts of object-oriented programming system. Polymorphism means “many forms” in Greek. That is one thing that can take many forms.

Polymorphism is a concept by which we can perform a single task in different ways. That is, when a single entity (object) behaves differently in different cases, it is called polymorphism.

In other words, if a single object shows multiple forms or multiple behaviors, it is called polymorphism.

2. What are the types of Polymorphism in Java?

Ans: There are two types of polymorphism in java. They are:

- Static polymorphism (Compile time Polymorphism)
- Dynamic polymorphism (Runtime Polymorphism)

3. What are different ways to achieve or implement polymorphism in Java?

Ans: Polymorphism in Java can be primarily achieved by subclassing or by implementing an interface. The subclasses can have their own unique

implementation for certain features and can also share some of the functionality through inheritance.

4. How is Inheritance useful to achieve Polymorphism in Java?
Ans: Inheritance represents the parent-child relationship between two classes and polymorphism take the advantage of that relationship to add dynamic behavior in the code (or to make the program more dynamic).

5. What are the advantages of Polymorphism?
Or what is the use of polymorphism?
Ans: There are the following advantages of polymorphism in java:

a. Using polymorphism, we can achieve flexibility in our code because we can perform various operations by using methods with the same names according to requirements.

b. The main benefit of using polymorphism is when we can provide implementation to an abstract base class or an interface.

6. What are the differences between Polymorphism and Inheritance in Java?
Ans: The differences between polymorphism and inheritance in java are as follows:

a. Inheritance represents the parent-child relationship between two classes. On the other hand, polymorphism takes the advantage of that relationship to make the program more dynamic.

b. Inheritance helps in code reusability in child class by inheriting behavior from parent class. On the other hand, polymorphism enables child class to redefine already defined behavior inside parent class.
Without polymorphism, it is not possible for a child class to execute its own behavior.

7. What is Compile time polymorphism (Static polymorphism)?
Ans: A polymorphism where object binding with methods happens at compile time is called static polymorphism or compile-time polymorphism.

In static polymorphism, the behavior of method is decided at compile-time based on the parameters or arguments of method.

8. How to achieve or implement static polymorphism in Java?

Ans: Static polymorphism can be achieved by method overloading. Other examples of compile time polymorphism are constructor overloading and method hiding.

9. What is the output of the following program if no error?

```
class A {

 void sum(int x, int y){

  System.out.println("Sum of two numbers: " +(x+y));

 }

 void sum(int x, int y, int z){

  System.out.println("Sum of three numbers: " +(x+y+z));

 }

public static void main(String[] args){

 A a = new A();

  a.sum(20, 30);

  a.sum(30, 40, 50);

 }

}
```

Ans: The output is:

Sum of two numbers: 50
Sum of three numbers: 120

10. Identify the errors in the following code.

```
class A {

 void sum(int x, int y){

  System.out.println("Sum of two numbers: " +(x+y));
```

```
}
 void sum(int y, int x){
  System.out.println("Sum of three numbers: " +(x+y));
 }
public static void main(String[] args){
 A a = new A();
  a.sum(20, 30);
 }
}
```

Ans: Duplicate method error.

11. What is the output of the following program if no errors?

```
class A {
void m1(String x){
 System.out.println("One");
}
protected void m1(String x, String y){
 System.out.println("Two");
}
public static void main(String[] args){
 A a = new A();
 a.m1("ABC");
 a.m1("PQR", "XYZ");
```

```
}

}
```

Ans: The output is One, Two.

12. How Java compiler differentiate between methods in Compile time Polymorphism?

Ans: During compilation, Java compiler differentiates multiple methods having the same name by their signatures.

13. Is there any errors in the following code? Will the code compile successfully?

```
class A {

String m1(String x){

 System.out.println("One");

 return "ABC";

}

String m1(String y){

 System.out.println("Two");

 return "PQR";

}

public static void main(String[] args){

 A a = new A();

 a.m1("ABC");

 }

}
```

Ans: Duplicate method error in the above code but the code will be compiled successfully. The output is One.

14. What is Runtime Polymorphism (Dynamic Polymorphism)?

Ans: A polymorphism where object binding with methods happens at runtime is called runtime polymorphism. In runtime polymorphism, the behavior of a method is decided at runtime.

JVM (Java Virtual Machine) binds the method call with method definition/body at runtime and invokes the relevant method during runtime when the method is called. This happens because objects are created at runtime and the method is called using an object of the class.

15. How to achieve/implement dynamic polymorphism in Java?

Ans: Dynamic or runtime polymorphism can be achieved or implemented via method overriding in java. It is another way to implement polymorphism and a common approach when we have an IS-A relationship.

16. Is it possible to implement runtime polymorphism by data members in Java?

Ans: No, we cannot implement runtime polymorphism by data members in java.

17. Will the code compile successfully? If yes, what will be the output of the program?

```
class A {

 void m1(A a){

  System.out.println("m1 method in class A");

 }

}

class B extends A {

 public void m1(A a){

  System.out.println("m1 method in class B");

 }
```

```
}

public class Test{

public static void main(String[] args){

 A a = new A();

 a.m1(a);

 a.m1(new B());


B b = new B();

  b.m1(null);


 a = b;

 a.m1(null);

 a.m1(new A());

  }

}
```

Ans: Yes, the code will be compiled successfully. The output is given below:

m1 method in class A
m1 method in class A
m1 method in class B
m1 method in class B
m1 method in class B

17. What is the output of the following program if no errors?

```
class A {

 void m1(String x){
```

```
 System.out.println("One");
 }
}
class B extends A {
 public void m1(String x){
  System.out.println("Two");
  super.m1(null);
 }
}
public class Test{
public static void main(String[] args){
 A a = new B();
 a.m1(null);
  }
}
```

Ans: The output is Two, One.

18. Identify the errors in the following code.

```
class A {
 void m1(String x){
  System.out.println("One");
 }
}
```

```
class B extends A {



}

public class Test{

public static void main(String[] args){

 A a = new B();

 a.m1(new A());

  }

}
```

Ans: The method m1(String) in type A is not applicable for the arguments (A).

19. What is the output of the below program if no errors?

```
class A {

 void m1(Object obj){

  System.out.println("One");

 }

}

class B extends A {

 void m1(Object obj){

  super.m1(null);

  System.out.println("Two");

 }
```

```
 void m2(Object obj){

  System.out.println("Three");

  this.m1(null);

 }

}

public class Test{

public static void main(String[] args){

 A a = new B();

 a.m1(new A());


B b = new B();

 b.m2(new B());

  }

}
```

Ans: No error. The above code will be compiled successfully. The output is given below:

One
Two
Three
One
Two

20. What are the differences between compile-time polymorphism and runtime polymorphism in java?

Ans: There are three main differences between compile-time polymorphism and runtime polymorphism that are as follows:

a) In the compile-time polymorphism, the behavior of a method is decided at compile-time. Hence, Java compiler binds method calls with method definition/body during compilation.

In runtime polymorphism, the behavior of a method is decided at runtime, JVM binds the method call with method definition at runtime and invokes the relevant method during runtime when the method is called.

b) Compile time polymorphism is also known as early binding because the binding is performed at compile time.

Runtime polymorphism is also known as late binding because the binding is performed at runtime.

c) Compile time polymorphism can be achieved via method overloading.

Runtime polymorphism can be achieved via method overriding.

21. What is Binding in Java?
Ans: The connecting (linking) between a method call and method definition is called binding in java.

22. What are the types of binding in Java?
Ans: There are two types of binding in java. They are as follows:

a. Static Binding (also known as Early Binding).
b. Dynamic Binding (also known as Late Binding).

23. What is Static binding in Java?
Ans: The binding that happens during compilation is called static binding in java. This binding is resolved at the compiled time by the compiler.

24. How Java compiler performs static binding?
Ans: Java compiler just checks which method is going to be called through reference variable and method definition exists or not.

It does not check the type of object to which a particular reference variable is pointing to it.

25. Why static binding is also called early binding in Java?
Ans: Static binding is also called early binding because it takes place before the program actually runs.

26. Give an example of static binding.
Ans: An example of static binding is method overloading.

27. What is Dynamic binding in Java?

Ans: The binding which occurs during runtime is called dynamic binding in java. This binding is resolved based on the type of object at runtime.

28. How JVM performs dynamic binding in Java?
Ans: In the dynamic binding, the actual object is used for binding at runtime. JVM resolved the method calls based on the type of object at runtime. The type of object cannot be determined by the compiler.

29. Why Dynamic binding is also called late binding in java?
Ans: Dynamic binding is also called late binding or runtime binding because binding occurs during runtime.

30. Give an example of dynamic binding in Java.
Ans: An example of dynamic binding is method overriding.

31. What is the difference between static binding and dynamic binding in Java?
Ans: Refer to this tutorial: Polymorphism in Java OOPs with Example

32. Why binding of private, static, and final methods are always static binding in Java?
Ans: Static binding is better performance-wise because java compiler knows that all such methods cannot be overridden and will always be accessed by object reference variable.

Hence, the compiler doesn't have any difficulty in binding between a method call and method definition. That's why binding for such methods is always static.

Hope that this tutorial has covered almost all the important interview questions on polymorphism in java with exact answers. I hope that you will have understood the level of questions asked by the interviewer in the company.
Thanks for reading!!!
