Abstraction

Interview Questions: Abstraction

1. What is Abstraction?

Definition:

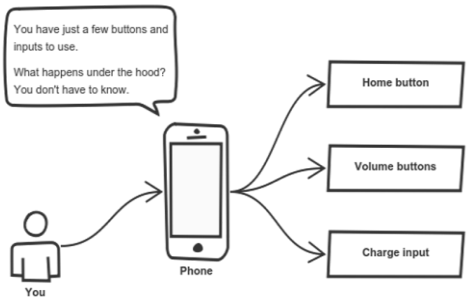
Abstraction *is the concept of object-oriented programming that "**represents**" only essential attributes and "**hides**" unnecessary information.*

- **Abstraction** is all about representing the simplified view and avoid complexity of the system.
- It only shows the data which is **relevant** to the user.
- In object-oriented programming, it can be implemented using *Abstract Class*.
- As shown in the image, user can only access interface provided by ATM machine while internal process is hidden.
- Abstraction is the **design concept** where only functionality is declared and does not define it. In other words, Abstraction only focuses on the **object** and not how it is **implemented**.

2. Mention Real world example for Abstraction

Example 1: Mobile Phone

A mobile user uses various mobile phone functionalities, such as calls, SMS, etc. But the actual implementation details of these functions are hidden from the mobile user. This is an example of abstraction.



Example 2: Car

Your car is a great example of abstraction. You can start a car by turning the key or pressing the start button. You don't need to know how the engine is getting started, what all components your car has. The car internal implementation and complex logic is completely hidden from the user.

Example 3: Microwave

If user is trying to use a microwave to heat up your food. It can easily heat up your food by just reading and pressing the buttons on the microwave accordingly. These buttons abstracts the inner mechanism of how the microwave works so that anyone can use the microwave without knowing how the microwave is able to heat up your food. The buttons hide the details of the microwave processes and it acts as a representation of the inner mechanism of the microwave.

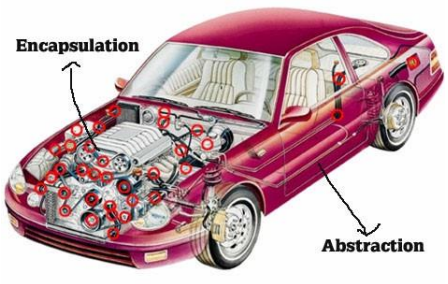
Abstraction

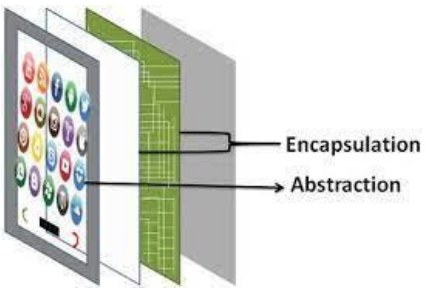
3. What are the advantages of Abstraction?

- It is helpful in reducing the complexity of data.
- It derives the result for a specific class.
- It is helpful in hiding unwanted details from users.
- The best example of Abstraction is Television. To use the Television, we do not need to understand how it works internally.

4. Abstraction vs. Encapsulation

Abstraction	Encapsulation
It means act of removing/ withdrawing something **unnecessary**.	It is act of binding code and data together and keep the **data secure** from outside interference.
It is the process of gaining information.	It is a method that helps wrap up data into a single module.
Applied at **Designing** stage.	Applied at **Implementation** stage.
E.g. Interface and Abstract Class	E.g. Access Modifier (public, protected, private)
Purpose: Reduce code complexity	**Purpose**: Data protection
The objects that help to perform abstraction are encapsulated.	Whereas the objects that result in encapsulation need not be abstracted.
Abstraction focus is on "what" should be done.	Encapsulation focus is on "How" it should be done.





5. How to achieve or implement Abstraction in Java?

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

1. Abstract class (0 to 100%): No fully abstract as Abstract class may contain abstract as well as non-abstract method.
2. Interface (100%): All the methods are abstract here.

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

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

Syntax:

```
abstract class ClassName{
      abstract return_type method_1(); //abstract method (method without body)
      abstract return_type method_2();//abstract method (method without body)

               return_type method_3(){ //Non-abstract or concrete method
                  …
            }
}
```

Abstraction

Example:

```
1. abstract class Car {
2.       public abstract double getAverage();
3. }

4. class Benz extends Car{
5.       public double getAverage(){
6.            return 15.5;
7.       }
8. }

9. class Audi extends Car{
10.           public double getAverage(){
11.                 return 13.2;
12.           }
13. }
```

7. What are the important points to remember while implementing and abstract class.

- A class which contains the **abstract** keyword in its declaration is known as abstract class.
- Abstract classes may or may not contain **abstract methods**, i.e., methods without body (public void get();)But, if a class has at least one abstract method, then the class must be declared abstract.
- If a class is declared abstract, it **cannot** be **instantiated**.
- There can be no objects of an abstract class. That is, an abstract class cannot be directly instantiated with the **new** operator. Such objects would be useless, because an abstract class is not fully defined.
- To use an abstract class, we have to inherit it to another class and provide **implementations** of the abstract methods in it.
- Any subclass of an abstract class must either implement all of the abstract methods in the superclass, or be itself declared **abstract**.

8. Why to implement an Abstract Class?

- Sometimes, we need to define a superclass that declares the structure of a given abstraction without providing a complete implementation.
- The superclass will only define a generalized form, that will be shared by all the subclasses.
- The subclasses will fill the details of every method.
- When a superclass is unable to create a meaningful implementation for a method.

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

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

- We cannot create an object of abstract class. Only objects of its non-abstract (or concrete) sub classes can be created.
- It can have zero or more abstract methods that are not allowed in a non-abstract class (concrete class).

10. What is Abstract in Java?

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

11. Can abstract modifier applicable for variables?

No, abstract modifiers are only applicable for classes, interfaces, methods, and inner classes.

12. Can an abstract method be declared as static?

No, abstract method cannot be declared as static.

Abstraction

13. **Can an abstract method be declared with private modifier?**
    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.

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

15. **When to use Abstract class in Java?**
    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.

16. **Is abstract class a pure abstraction in Java?**
    No, It provides 0 to 100% abstraction. If abstract class has non-abstract method then 100% abstraction cannot be achieved. 100% abstraction can be achieved using Interface in java.

17. **Can an abstract class have constructor?**
    Yes. But there can be no objects of an abstract class. That is, an abstract class cannot be directly instantiated with the new operator. Such objects would be useless, because an abstract class is not fully defined.

18. **Why modifiers such as final, static & private are not allowed for abstract methods declared in abstract class?**
    Final: as sub-class needs to provide method implementation for all abstract methods inside an abstract class, therefore abstract cannot be marked as final
    Static: abstract methods belong to instance not a class, therefore it cannot be marked as static
    Private: abstract methods need to be overridden in the sub-class for this we need wider accessibility
    By marking abstract method declaration with final or static or private modifier –> results in a compilation error
    Compile-time error: The abstract method display in type <abstract-class-name> can only set a visibility modifier, one of public or protected

19. **Is it possible to achieve multiple inheritance through abstract class?**
    No. Multiple Inheritance can be implemented using Interface in java.

20. **Can we make an abstract class without abstract keyword?**
    No, a class must be declared with abstract keyword to make an abstract class.

21. **Can we define an abstract method inside non-abstract class (concrete class)?**
    No, we cannot define an abstract method in non-abstract class.

```
class Demo {
abstract void show();//Compile time ERROR! Abstract method w/o Abstract class
}
```

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

    Java compiler will generate compile time error. It is mandatory to override all abstract methods in subclass.

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

    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.

Abstraction

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

- 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.

```
1. public abstract class A {
2.       abstract void m1();
3.       void m2(){
4.             System.out.println("One”);
5.  }
6.  }
```

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

25. Whether abstract class compiles successfully if it contains both concrete & abstract methods together?
Yes, abstract class compile successfully as it can contain both abstract methods and concrete.

26. Can abstract class implements interface?
Yes, an abstract class can implement an interface and this is valid.

27. Can an abstract class be defined without any abstract methods?
Yes, a class can be declared with abstract keyword even if it doesn't get 1 abstract method. But vice-versa is not true; means if a class contains abstract methods then the class has to be declared with abstract keyword.

28. It is mandatory to have abstract methods in an abstract class? If not, why such a design is required?

- It's not mandatory to have abstract methods in an abstract class
- Even without a single abstract method in a class can be declared as abstract
- This is to flag compiler that this class is not for instantiation

29. Can an abstract class be final?

- No, an abstract class cannot be final
- Abstract methods need to be implemented; therefore it is overridden in the subclass
- But by marking final, we are restricting it to override
- A compile-time error will be thrown: The abstract method display in type <abstract-class-name> can only set a visibility modifier, one of public or protected

30. Can we declare a concrete (non-abstract) method with the final modifier inside an abstract class?
Yes, the concrete method can be declared with the final modifier

31. What are all modifiers allowed for abstract method declaration?
public and protected access modifiers are allowed for abstract method declaration

32. Can we define private constructor inside an abstract class?
Yes, it is allowed to have private constructor inside an abstract class.

33. Does abstract method inside the abstract class can throw exceptions?
Yes, abstract methods can throw an exception

Abstraction

34. Can abstract class contain the main() method and starts the execution?

Yes, the main() method allowed inside the abstract class; also we can execute it.

```
public abstract class AbstractExampleMain
 {
      // abstract method throwing exception
      abstract void display() throws ClassCastException;
      static void staticDisplay() {
            System.out.println("main() executed");
      }
      public static void main(String arg[]) {
            staticDisplay();
      }
 }
```

Output: main() executed

35. Can we an overload abstract method in Java?

Yes, abstract methods can be overloaded and its extending class will provide an implementation for all abstract methods.

36. What is the purpose of an Abstract Class?

- Abstract Classes enforce abstraction.
- A class may be declared abstract even if it has no abstract methods. This prevents it from being instantiated.
- Abstract class must be extended/subclassed (to be useful).
- It serves as a template. A class that is abstract may not be instantiated (ie. you may not call its constructor), abstract class may contain static data.

37. Can we declare a variable as Abstract?

There is nothing called Abstract Variables as variables can't be declared as abstract. Only classes and methods can be declared as abstract.

38. Can you create an object of an abstract class?

No. Abstract classes can't be instantiated.

39. What is 'final' Keyword in Java?

The final keyword is used for restriction. final keyword can be used in many context.
Final can be:

1. **Variable**

   If you make any variable as final, you **cannot change the value** of final variable(It will be constant).

```
public class FinalDemo {
      final int speedlimit=90;//final variable
      void run(){
            speedlimit=400;  //Compiler Error!
      }
      public static void main(String args[]){
            FinalDemo obj=new  FinalDemo();
            obj.run();
      }
}
```

2. **Method**
   If you make any method as final, you **cannot override** it.

```
class BikeClass{
  final void run(){
      System.out.println("Running Bike");
  }
}
class Pulsar extends BikeClass{
   void run(){//Compiler Error!: can't override final method
      System.out.println("Running Pulsar");
   }

   public static void main(String args[]){
      Pulsar p= new Pulsar();
      p.run();
   }
}
```

3. **Class**
   If you make any class as final, you **cannot extend** it.

```
final class BikeClass{
   void run(){
      System.out.println("Running Bike");
   }
}
class Pulsar extends BikeClass
{  //Compiler Error!: can't inherit final class
   void run(){
      System.out.println("Running Pulsar");
   }

   public static void main(String args[]){
      Pulsar p= new Pulsar();
      p.run();
   }
}
```

40. What if the main() method is declared as private?
The program compiles properly but at runtime, it will give the "main() method not public." message.

41. What is Interface?
Definition:An interface in Java is a mechanism that is used to achieve complete abstraction. It is basically a kind of class that contains only constants and abstract methods.

- An interface is **similar** to an **abstract class** with the following exceptions
- All methods defined in an interface are **abstract**.
- Interfaces doesn't contain any **logical implementation**
- Interfaces cannot contain **instance variables**. However, they can contain **public static final variables** (ie. constant class variables)
- Interfaces are declared using the "***interface***" keyword
- Interfaces are more **abstract** than abstract classes
- Interfaces are implemented by classes using the "***implements***" keyword
- Interfaces are syntactically similar to **classes**, but they **lack** instance variables, and their methods are declared without any body.

Abstraction

42. How to implement Interface in java?

Syntax : Interface

```
access interface interface_name
{
      return-type method-name1(parameter-list);
      return-type method-name2(parameter-list);
      type final-varname1 = value;
      type final-varname2 = value;
      // ...
      return-type method-nameN(parameter-list);
      type final-varnameN = value;
}
```

- Once an interface has been defined, one or more classes can implement that interface.
- To implement an interface, include the ***implements*** clause in a class definition, and then create the methods defined by that interface.

Syntax: Implementing Interfaces in a class

```
access class classname
                  [extends superclass]
                       [implements interface [,interface...]]
{
// class-body
}
```

43. Differentiate Abstract class and Interface

Abstract class	Interface
Abstract classes allow you to create blueprints for concrete classes. But the inheriting class should implement the abstract method.	The interface is a blueprint that can be used to implement a class. The interface does not contain any concrete methods (methods that have code). All the methods of an interface are abstract methods.
Abstract class **doesn't** support **multiple inheritance.**	Interface **supports multiple inheritance.**
Abstract class can have **abstract and non-abstract/concrete** methods.	Interface can have **only abstract** methods.
Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only public, static and final variables**.
An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.
A class inherits only **one abstract class**.	A class may implement **numerous interfaces**.
An abstract class can **inherit** a class and multiple interfaces.	An interface can **inherit** multiple interfaces but cannot inherit a class.
An abstract class can declare **constructors** and **destructors**.	An interface cannot declare **constructors** or **destructors**.

44. Suppose A is an interface. Can we create an object using new A()?

No, we cannot create an object of interface using new operator. But we can create a reference of interface type and interface reference refers to objects of its implementation classes.

Abstraction

45. Can we define an interface with a static modifier?
Yes, from Java 8 onwards, we can define static and default methods in an interface. Prior to Java 8, it was not allowed.

46. Suppose A is an interface. Can we declare a reference variable a with type A like this:
`A a;`
Yes, we can create such reference variable of type A.

47. Can an interface extends another interface in Java?
Yes, an interface can extend another interface.

```
1. interface A{
2.       public void method1();
3.       public void method2();
4. }
5. interface B extends A{
6.       public void method3();
7. }
8. interface C extends A{
9.       public void method4();
10. }
```

48. Can an interface implement another interface?
No, an interface cannot implement another interface. Only a class can implements an Interface.

49. Is it possible to define a class inside an interface?
Yes, we can define a class inside an interface.

50. What happens if a class has implemented an interface but has not provided implementation for that method defined in Interface?
The class has to be declared with an **abstract** modifier. This will be enforced by the Java compiler.
If a class includes an interface but does not fully implement the methods defined by that interface, then that class must be declared as abstract.

```
1. interface VehicleInterface {
2.       int a = 10;
3.       public void turnLeft();
4.       public void turnRight();
5.       public void accelerate();
6.       public void slowDown();
7. }
8. abstract class CarClass implements VehicleInterface
9. {
10.      public void turnLeft() {
11.            System.out.println("Left");
12.      }
13.      public void turnRight() {
14.            System.out.println("Right");
15. }
```

Abstraction

51. Why an Interface method cannot be declared as final in Java?
Because final methods cannot be overridden. Doing so will result the compilation error problem. interface method should be implemented by another class.

52. Why an interface cannot have a constructor?
Inside an interface, a constructor cannot be called using super keyword with hierarchy.

53. Why an Interface can extend more than one Interface but a Class can't extend more than one Class?
We know that Java doesn't allow multiple inheritance because a class extends only one class. But an Interface is a pure abstraction model. It does not have inheritance hierarchy like classes. Therefore, an interface allows to extend more than one Interface.

54. What is the use of interface in Java?
There are many reasons to use interface in java. They are as follows:
a. An interface is used to achieve **fully abstraction**.
b. Using interfaces is the best way to **expose** our project's API to some other project.
c. Programmers use interface to **customize** features of software differently for different objects.
d. By using interface, we can achieve the functionality of **multiple inheritance**.

55. Is it necessary to implement all abstract methods of an interface?
Yes, all the abstract methods defined in interface must be implemented.

56. Can we define a variable in an interface? What type it should be?
Yes, we can define variable in an interface that must be implicitly static and final.

```
Interface A{
      int i=10;//implicity static and final
      }
```

57. How is an interface different from a class?
An interface is different from a class in several ways:

1. An interface can only have abstract methods, while a class can have both abstract and non-abstract methods.
2. An interface cannot have constructors, while a class can have constructors.
3. An interface cannot have instance variables, while a class can have instance variables.
4. An interface can be implemented by a class, while a class can only be extended by another class.

58. What is the difference between a functional interface and a regular interface in Java?
A functional interface is a type of interface that has only one abstract method. This method is called the functional method, and it defines the behavior of the interface. A regular interface can have multiple abstract methods. Functional interfaces are used to implement lambda expressions and method references, which were introduced in Java 8.

59. When would you use an interface over an abstract class?
You would use an interface over an abstract class when you want to define a contract that classes must follow without specifying how they should implement it. This allows for greater flexibility and code reuse, as any class that implements the interface can be used in place of another. An abstract class, on the other hand, is more appropriate when you want to provide a base implementation for its subclasses, or when you want to define common behavior or state.

60. What is the difference between an interface and a marker interface in Java?
A marker interface is a type of interface that does not contain any methods. Its purpose is to provide a tag or marker to a class, indicating that the class has some special behavior or characteristic. An interface, on the other hand, contains one or more abstract methods and is used to define a contract between a class and its user.

Encapsulation

1. **Define Encapsulation**

Dictionary Meaning: *"the action of enclosing something in or as if in a capsule."*
OOP Definition1: *"Encapsulation refers to the bundling of fields and methods inside a single class". It prevents outer classes from accessing and changing fields and methods of a class. This also helps to achieve data hiding.*
OOP Definition2: *"The process of binding data and corresponding methods (behavior) together into a single unit is called encapsulation in Java."*

```
Class
{
  Data Members(e.g. int a=10)
        +
  Methods(e.g. add() )
}
```

*The wrapping up of data and functions into a single unit is known as **encapsulation***

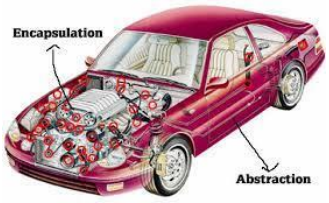
2. **Define Encapsulation with real world example**

Example 1: Capsule

A **Capsule** is wrapped or encapsulates various combinations of medicine. A Java Class is an example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Example 2: Car

In a car engine, encapsulation refers to the concept of bundling the different components of the engine together and hiding their implementation details from the outside world. As a driver you know how to start the car by pressing the start button and internal details of the starting operations are hidden from you. So the entire starting process is hidden from you otherwise we can tell starting operation is encapsulated from you.



Example 3: Mobile Phone

You have a Mobile Phone, you can dial a number using keypad buttons. Even you don't know how these are working internally. This is achieved in JAVA using **Abstraction**. But how the Mobile Phone internally working? How keypad buttons are connected with internal circuit? This is achieved using **Encapsulation**.

3. **How to Achieve or Implement Encapsulation in Java**

There are two important points whereby we can achieve or implement encapsulation in Java program.

i. Declaring the **instance variable** of the class as **private** so that it cannot be accessed directly by anyone from outside the class.
ii. Provide the public **setter** and **getter** methods in the class to set/modify the values of the variable/fields.

4. **Advantage of Encapsulation in Java**

There are the following advantages of encapsulation in Java:

i. The encapsulated code is more **flexible** and easy to change with new requirements.
ii. It prevents the other classes from accessing the **private** fields.
iii. Encapsulation allows **modifying** the implemented code without breaking other code that has implemented the code.
iv. It keeps the data and codes **safe** from external **inheritance**. Thus, encapsulation helps to achieve **security**.
v. It improves the **maintainability** of the application.
vi. If you don't define the setter method in the class, then the fields can be made **read-only**.
vii. If you don't define the getter method in the class, then the fields can be made **write-only**.
viii. If you define both getter and setter methods in the class, then the fields can be made both read-write.

Encapsulation

5. What are the important features of Encapsulation?

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 behaviour. It **reduces** the **coupling** of modules and increases the **cohesion** inside them.

6. What is data hiding in Java?

The insulation of the data from direct access by the program is called data hiding or information hiding. It is the process of enclosing one or more details from outside world through access right.

```
Encapsulation = Data Hiding + Abstraction
```

7. Advantages of Data Hiding.

i. Protects an object from unwanted access
ii. It reduces implementation errors.
iii. Simplifies the maintenance of the application and makes the application easy to understand.
iv. Protection of data from accidental corruption.

8. What do you mean by Tightly Encapsulated Class in Java

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

Example

```
public class Bank{
 private double balance;
 private String accType;
 public double getbalance()
 {
   return balance;
 }
}
```

- If the parent class is not tightly encapsulated, no child class is tightly encapsulated because the parent class's non-private data members by default are available to every child class.
- Thus, we can say that data hiding, encapsulation, and tightly encapsulated class concepts are used for security purposes.

9. What is the difference between Abstraction and Encapsulation?

- Abstraction solves the problem at the design level whereas encapsulation solves the problem at the implementation level.
- Abstraction is implemented in Java using Interface and Abstract class whereas encapsulation is implemented using private and protected access modifiers.
- 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.
- 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?

Yes, we can achieve abstraction without encapsulation because both are different things and different concepts.

11. Differentiate data hiding and Encapsulation.

Data hiding	Encapsulation
Data hiding means protecting the variable of a class from outside world access.	Encapsulation means wrapping or binding data in a single unit.
Focuses on securing the data along with hiding.	Focuses more on wrapping data to make classes and the methods in the simplest form for the system.
It is a technique and process both.	It is a sub-process in data hiding.
Here, data is always private and inaccessible.	Here, data may be private or public.

Encapsulation

12. Where Encapsulation is used?

Encapsulation is used in various design patterns and real-life problems that make use of the Encapsulation object-oriented concept. It is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties direct access to them.

13. How does getter and setter functions help in encapsulation?

In encapsulation, the getters and setters method is because to hide the attributes of an object class from other classes so that no accidental modification of the data happens by methods in other classes.

Example:

Declare class variables/attributes as private.

Provide public get and set methods to access and update the value of a private variable.

```
public class Student {
  private String name;
  // Getter
  public String getName() {
    return name;
  }
  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

14. Is Encapsulation violating reflection?

Encapsulation wants to give modules a safe space and Reflection wants to break into all code. Thus the manipulation of the data is done at one place unknowing to other members of variables. Encapsulation principles are broken using the reflection as it can access the private classes without setting getter and setter methods.

15. Differentiate Encapsulation and Polymorphism.

Packaging data and methods in a single unit are Encapsulation. Their own methods are used for accessing the data which are hidden in the objects.

The word polymorphism means Poly means many and morph means form.

Polymorphism means the ability to take more than one form by using a single interface. When single or many types are not defined by name it represents any type by using abstract symbols. Polymorphism is a process used for implementing inherited data.

16. What is a final class as it relates to encapsulation?

A final class as it relates to encapsulation is a class that a programmer cannot extend. No subclass within the code can inherit any variables or methods from a final class. It is advisable to write a final class when you want to create a class that is impossible to extend.

Example:

Consider you have a class that has sensitive data relating to a customer's payment and billing information. You can establish this class as a **final** class so that no subclasses can access it.

17. Which access modifier is used for encapsulation in Java?

The access modifier "***private***" is used for encapsulation in Java.

18. What do you understand by interfaces and abstract classes? How do they help with encapsulation?

Interfaces and abstract classes help with encapsulation by allowing you to specify the behaviour of a class without having to provide a concrete implementation. This means that you can hide the details of how a class works from the outside world, and only expose the interface that you want other classes to use. This makes it much easier to change the internals of a class without breaking any code that depends on it.

Encapsulation

19. Why to use private constructors?

Private constructors are used in order to prevent a class from being instantiated. This is often done in order to enforce the Singleton pattern, where only one instance of a class is allowed to exist.

20. What's your opinion on public static final members (constants) in a Java class?

Public static final members are a great way to create constants in a Java class. By making the members public, you are ensuring that they can be accessed by any other class that needs to use them. By making them static, you are ensuring that there is only one copy of the constant that is shared by all instances of the class. And by making them final, you are ensuring that the value of the constant cannot be changed.

21. What are some advantages of using constants instead of variables or hardcoded values?

i. Constants are immutable, meaning they cannot be changed once they are defined. This can be helpful in ensuring that your code always uses the same value for a given quantity, which can make your code more reliable.
ii. Additionally, constants can improve the readability of your code by making it clear what values are being used.
iii. Finally, constants can make your code more flexible, as they can be easily changed if the need arises.

22. Can you give me some examples of where the object state should be hidden from the outside world?

Example-1: you might want to hide an object's state is when you are working with sensitive data that you don't want to be tampered with.

E.g. Bank balance, Exam Result etc.

Example-2: Another example might be when you are working with an object that is in a delicate state and you don't want outside forces to be able to change it and potentially break it.

E.g. Military data.

23. What are some best practices for writing production-quality code that makes use of encapsulation?

i. Always make sure that your data is properly encapsulated and hidden from outside access.
ii. Be sure to write code that is easy to understand and follow, as this will make it easier for others to work with your code.
iii. Thoroughly test your code before putting it into production, as this will help to ensure that it is working as intended.

24. What is coupling?

Coupling is the degree of interdependence between software modules; a measure of how closely connected two modules are. When module A calls module B, they are said to be coupled. The more modules that module A calls, the higher its coupling.

25. What is cohesion?

Cohesion is a measure of how well the elements of a module work together to achieve a specific goal. A module with high cohesion is one where the elements are all focused on a single task, while a module with low cohesion is one where the elements are more general purpose and can be used for multiple tasks.

INHERITANCE

Interview Questions: Inheritance

1. **Which class in Java is superclass of every other class?**
   In Java, **Object class** is the superclass of every other java class.
   i.e.: java.lang.Object is the parent class of all the java classes

2. **Can a class in java extends itself?**
   No, a class cannot extend itself

3. **Can a class extend more than one class?**
   No, a class cannot extend more than one class

4. **Differentiate 'extends' and 'implements' keyword in Java.**
   Extends: extends is a keyword that is used for developing the inheritance between two classes and two interfaces.
   Implements: implements keyword is used for developing the inheritance between a class and interface.

5. **Can we assign superclass to subclass?**
   No, we can't assign superclass to subclass.

6. **Can a class extend more than one class?**
   No, one class can extend only a single class.

7. **Are static members inherited to subclass in Java?**
   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.

8. **Can we extend (inherit) final class?**
   No, a class declared with final keyword cannot be inherited.

9. **Can a final method be overridden?**
   No, a final method cannot be overridden.

10. **Can we inherit private members of base class to its subclass?**
    No, all the members (public, protected and default) of superclass, except private members are accessible by sub-class.

11. **What are the advantages of inheritance in Java?**
    - Promotes reusability
      When an existing code is reused, it leads to less development and maintenance costs.
    - It is used to generate more dominant objects.
    - Avoids duplicity and data redundancy.
    - Inheritance makes the sub classes follow a standard interface.

12. **What is Multiple inheritance in Java?**
    A class that has many super classes is known as multiple inheritance. In other words, when a class extends multiple classes, it is known as multiple inheritance.

13. **Why multiple inheritance is not supported in java through class?**
    Multiple inheritance means that one class extends two or more super classes or base classes but in Java, one class cannot extend more than one class simultaneously. At most, one class can extend only single class.
    Therefore, to reduce ambiguity, complexity, and confusion, Java does not support multiple inheritance through classes.Multiple inheritance can be achieved in java by implementing interfaces instead of class.

INHERITANCE

14. How does Multiple inheritance implement in Java?

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.

Example:

```
class B extends A
{
        …
}
```

```
class B extends A implements interface1,interface2,…
{
        …
}
```

15. What is Hybrid inheritance in java? How will you achieve it?

A hybrid inheritance in java is a combination of single and multiple inheritance. It can be achieved through interfaces.

16. How will you restrict a member of a class from inheriting its subclass?

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.

17. Can we access subclass members if we create an object of superclass?

No, we can access only superclass members but not the subclass members.

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

Yes, we can access both superclass and subclass members.

19. Is interface inherited from the Object class?

No.

20. Can you override a final method?

No, method declared as 'final' cannot be overridden.

21. What does Java's Super Keyword mean?

A reference variable used to refer to the immediate parent class object in Java is the super keyword.
When you create an instance of a subclass, an implicit instance of the parent class is also produced and referred to by the super reference variable.

22. Differentiate inheritance from encapsulation.

Inheritance: hers one class acquires another class to promote reusability.
Encapsulation: Binds code and data together for securing data access.

23. Differentiate inheritance from composition.

Inheritance and composition are two programming techniques developers use to establish relationships between classes and objects. Whereas inheritance derives one class from another, composition defines a class as the sum of its parts.

24. super vs. this keyword

'super' keyword

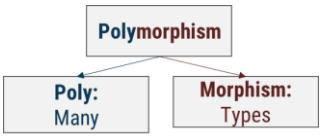
- is used to access methods of the parent class while **'this'** is used to access methods of the current class.

POLYMORPHISM

Interview Questions: Polymorphism

1. What is Polymorphism in Java OOPs?

Dictionary Meaning1:
It is a Greek term means,**"*<u>One Name, Many Forms</u>*"**.
One entity that can take many forms.



Dictionary Meaning2:
"*The condition of occurring in **several different forms**.*"

OOP Definition:
*"**Polymorphism*** is the ability of an object to take on many forms."
In other words, *if a single object shows multiple forms or multiple behaviour, it is called polymorphism*.

2. Define Real-world example of Polymorphism.

Example-1: Smart Phone
The smartphone can act as phone, camera, music player, alarm clock, calculator, remote control, torch and what not, taking different forms and hence it is polymorphism.

Example-2: Water
Another real time example of polymorphism is water. Water is a liquid at normal temperature, but it can be changed to solid when it frozen, or same water changes to a gas when it is heated at its boiling point. Thus, same water exhibiting different roles is polymorphism.

Example-3: Human Behaviour
Man is only one, but he takes multiple roles like - he is a dad to his child, he is an employee, a salesperson, a customer in a shopping mall and many more. This is known as Polymorphism.

3. What is Static Polymorphism?

Static polymorphism (**static** binding) is a kind of polymorphism that occurs at compile time. An example of compile-time polymorphism is **Method Overloading** (*A class with multiple methods having same name but different in parameters is known as **<u>Method Overloading</u>***).

Example:

```
public void draw()
public void draw(int height, int width)
public void draw(int radius)
```

- Overloaded methods must **differ** in the **type and/or number** of their parameters.
- While in overloaded methods with different return types and same name & parameter are not allowed, as the return type alone is insufficient for the compiler to distinguish two versions of a method.

POLYMORPHISM

4. What is Dynamic Polymorphism?

Runtime polymorphism or dynamic polymorphism (dynamic binding) is a type of polymorphism which is resolved during runtime. An example of runtime polymorphism is method overriding (If child class has the same method as declared in the parent class with same parameters and signature, it is known as **method overriding in Java**).

5. Differentiate between overloading and overriding.

Overloading	Overriding
Used to implement **Compile-time polymorphism(Static binding)**	Used to implement **Run-time polymorphism(dynamic binding)**
Mehtod with same name and different signature is method overloading	Mehtod with same name and same signature is method overriding
Implemented within a class	Implemented with in two/more classes with inheritance relationships.

Example:
`class OverloadingDemo{`
`public void draw(){…}`
`public void draw(int height, int width) {…}`
`public void draw(int radius) {…}`
`}`
	Example:
	`class Shape{`
	`void draw(){`
	`System.out.println("Draw Shape");`
	`}`
	`}`
	`class Circle extends Shape{`
	`void draw(){`
	`System.out.println("Draw Circle");`
	`}`
	`}`

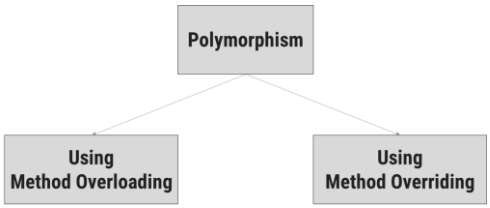
In method overloading, call will be referred according to the parameter passed and will be decided at compile time.	When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the superclass will be hidden.

6. What is operator overloading?

Operator overloading refers to implementing operators using user-defined types based on the arguments passed along with it. Operator overloading is concept in C & C++ and not supported in Java. Java does not allow you to design your own operators.

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

Polymorphism can be implemented by Method Overloading(Compile-time polymorphism) , method Overriding(Run-time polymorphism) and by implementing an interface.



8. How is Inheritance useful to achieve Polymorphism in Java?

Inheritance represents the parent-child relationship between two classes and polymorphism take the advantage of that relationship to add dynamic behavior in the code.
Example: Dynamic Method Dispatch

POLYMORPHISM

9. What are the advantages of Polymorphism?

- 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.
- The main benefit of using polymorphism is when we can provide implementation to an abstract base class or an interface.
- A single variable can be used to store multiple data values. ...
- With lesser lines of code, it becomes easier for the programmer to debug the code.

10. Differentiate Polymorphism and Inheritance

- 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.
- 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.
- Inheritance is basically applied to classes while polymorphism is basically applied to functions or methods.

11. How runtime polymorphism is achieved in Java?

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. The type of object cannot be determined by the compiler.

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

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

13. What is Binding in Java?

The connecting (linking) between a method call and method definition is called binding in java.
E.g.:Static and dynamic binding

14. Why static binding is also called early binding in Java?

Static binding is also called early binding because it takes place before the program actually runs.
e.g.: Method Overloading

15. Why binding of private, static, and final methods are always static binding in Java?

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.

16. What is covariant return type in Java?

Covariant return type is a feature in Java that allows a subclass to return a type that is a subtype of the type returned by the overridden method in its parent class.Example, if a method in a parent class returns an Animal type, the overridden method in the subclass can return a more specific type like a Dog or a Cat, as long as it is a subtype of Animal.

17. What happens if two methods have different return types but share the same name and signature?

In this case, the method with the more specific return type will be chosen. For example, if one method returns an int and the other returns a double, the method returning the double will be chosen.

POLYMORPHISM

18. Can you explain the difference between early binding and late binding? Which one provides better performance?

Early binding is when the compiler knows exactly which method to call at compile time, while late binding is when the compiler only knows the general type of object that it will be dealing with. Early binding is generally faster because the compiler can optimize the code better, but late binding can be more flexible.

19. What do you understand about virtual functions?

A virtual function is a member function of a base class that is declared using the virtual keyword. When a virtual function is called, the compiler will automatically choose the correct function to call based on the type of the object that is calling the function. This allows for polymorphism, or the ability to treat objects of different types in the same way. Virtual Function is not supported in Java.

20. Why can't a class be both abstract and final at the same time?

A class cannot be both abstract and final at the same time because it would defeat the purpose of polymorphism. An abstract class is one that is meant to be extended by other classes, so that they can inherit its methods and add their own implementation. A final class, on the other hand, is one that cannot be extended. Therefore, if a class were both abstract and final, it would be impossible to inherit its methods and add your own implementation, which is the whole point of polymorphism.

21. Can you give me some examples of when runtime polymorphism may not work as expected?

One example of when runtime polymorphism may not work as expected is when you are trying to override a method that is declared as final in the parent class. Another example is when you are trying to override a method that is not declared as virtual in the parent class.