Crack Java Interview

Q1. What are the 4 pillars of OOPS?

Answer: 4 pillars of OOPS are:

1. Abstraction

2. Encapsulation

3. Inheritance

4. Polymorphism

1. Abstraction: Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Real world examples:

TV remote: To start the TV, you have to press the power button, you don't have to know about the internal circuit operations like how infrared waves are passing.

Car gears: We know what happens when we change the gear. But we don't know how changing gear works under the hood, that information is irrelevant to us, so it is abstracted.

In java, Abstraction can be achieved in two ways:

1. Abstract classes
2. Interfaces

2. Encapsulation: Encapsulation is a process of Binding data and methods within a class. Think of it like showing the essential details of a class by using the access control modifiers (public, private,

protected). So, we can say that Encapsulation leads to the desired level of Abstraction.

Example:

Java Bean, where all data members are made private and you define certain public methods to the outside world to access them.

1. Inheritance: Using inheritance means defining a parent-child relationship between classes, by doing so, you can reuse the code that is already defined in the parent class. Code reusability is the biggest advantage of Inheritance.

Java does not allow multiple inheritance through classes but it allows it through interfaces.

3. Polymorphism: Poly means many and Morph means forms. Polymorphism is the process in which an object or function takes different forms. There are 2 types of Polymorphism:

1. Compile Time Polymorphism (Method Overloading)
2. Run Time Polymorphism (Method Overriding)

In Method overloading, two or more methods in one class have the same method name but different arguments. It is called as Compile time polymorphism because it is decided at compile time which overloaded method will be called.

Overriding means when we have two methods with same name and same parameters in parent and child class. Through overriding, child class can provide specific implementation for the method which is already defined in the parent class.

Question 2: What is an abstract class?

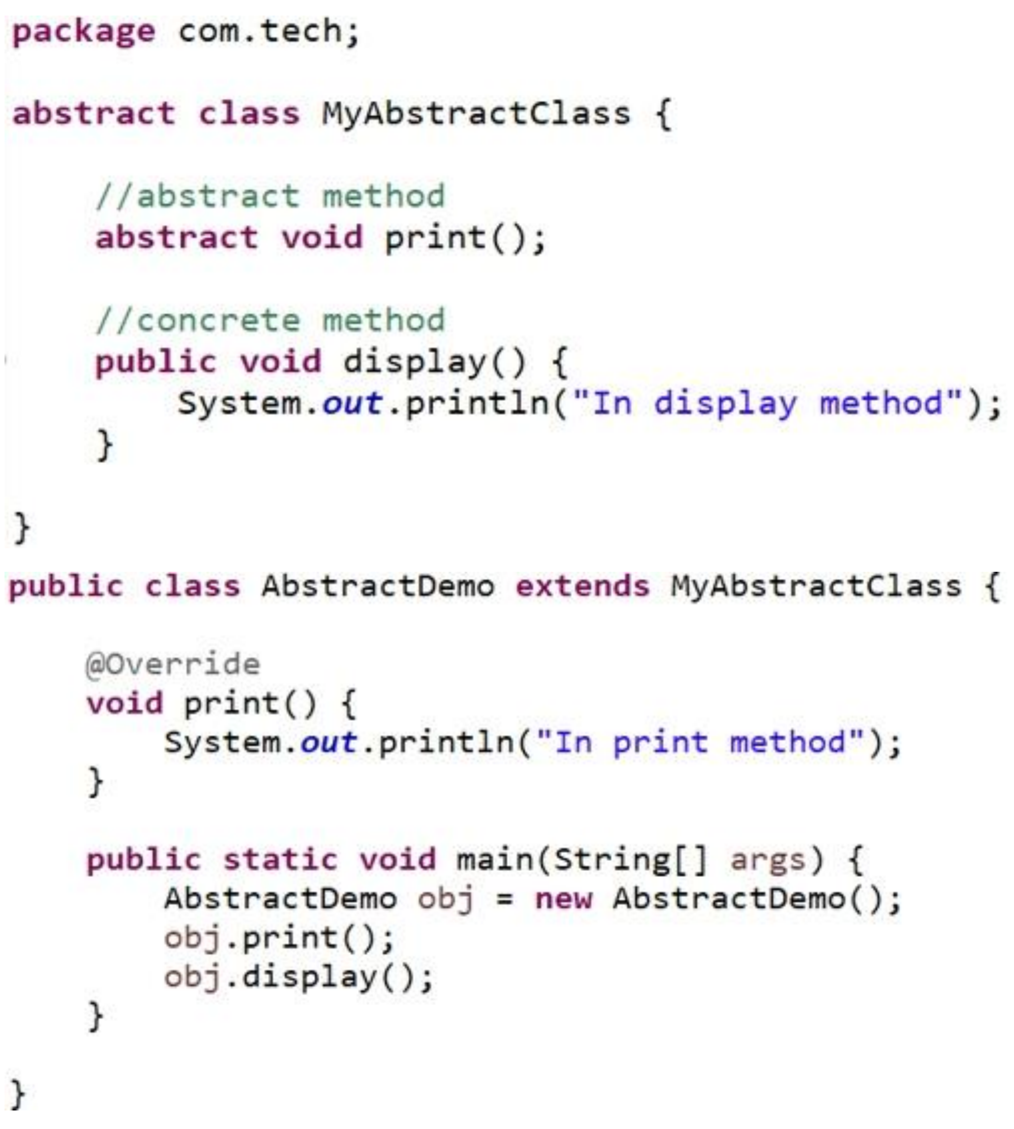
Answer: A class that is declared using “abstract” keyword is known as abstract class. It can have abstract methods (methods without body) as well as concrete methods (methods with body).

Some points to remember:

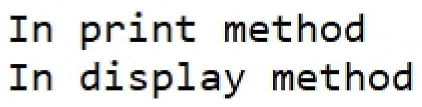
An abstract class cannot be instantiated, which means you are not allowed to create an object of the abstract class. This also means, an abstract class has no use unless it is extended by some other class

If there is any abstract method in a class then that class must be declared abstract

The first non-abstract class which is extending from an abstract class will have to give implementation of the abstract methods defined in abstract class



```
package com.tech;

abstract class MyAbstractClass {

    //abstract method
    abstract void print();

    //concrete method
    public void display() {
        System.out.println("In display method");
    }

}
public class AbstractDemo extends MyAbstractClass {

    @Override
    void print() {
        System.out.println("In print method");
    }

    public static void main(String[] args) {
        AbstractDemo obj = new AbstractDemo();
        obj.print();
        obj.display();
    }

}
```



```
In print method
In display method
```

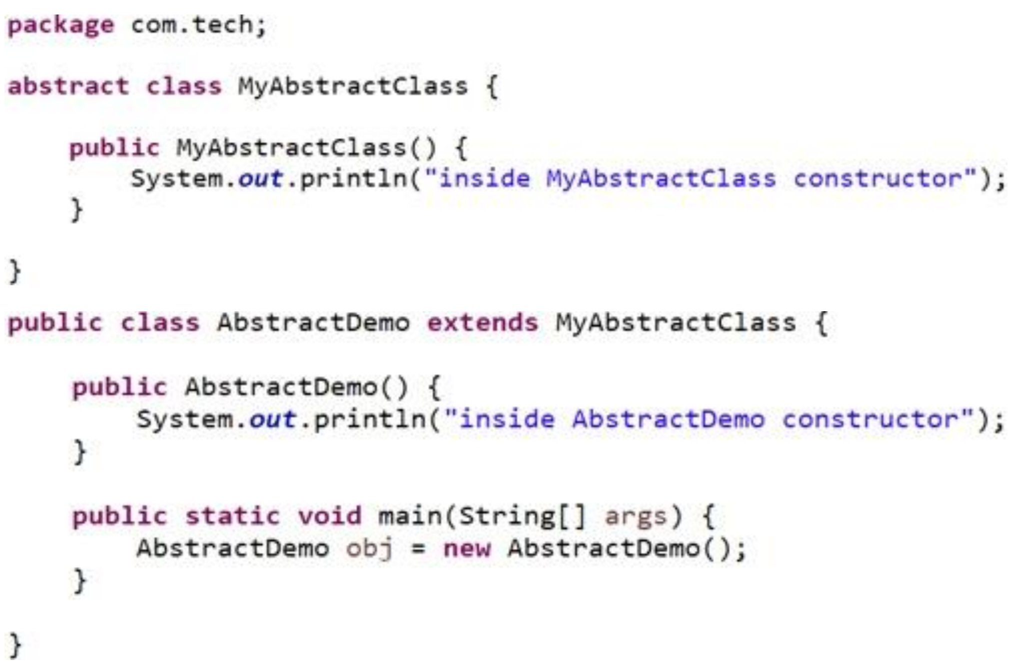
Question 3: Does Abstract class have constructor?

Answer: This is a famous interview question and the answer is: Yes, abstract classes have constructor. Either you can provide it or the default one will be provided by Java. Now, you must be wondering if you cannot create an object of abstract class then what is the need of a constructor.

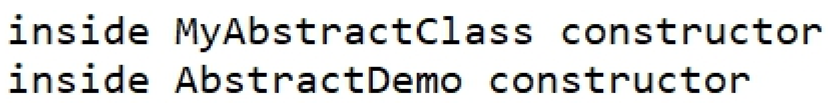
One thing you must know is that the constructors are used when you are

creating an object of a class, to initialize the data members of that class and your abstract class can have data members.

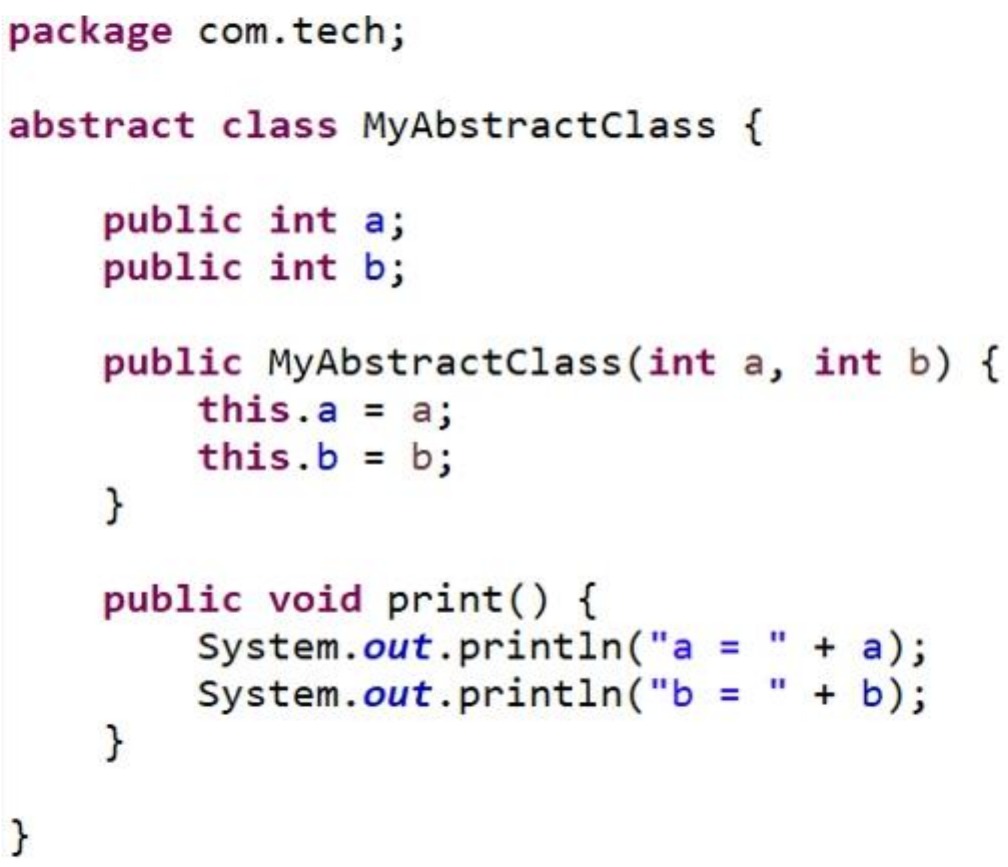
Now, when your class extends abstract class then the same abstract class will become super class for your extending class and remember when you have constructor of your class then first line of your constructor is always a call to super class constructor and this is the time when your abstract class constructor will get called.



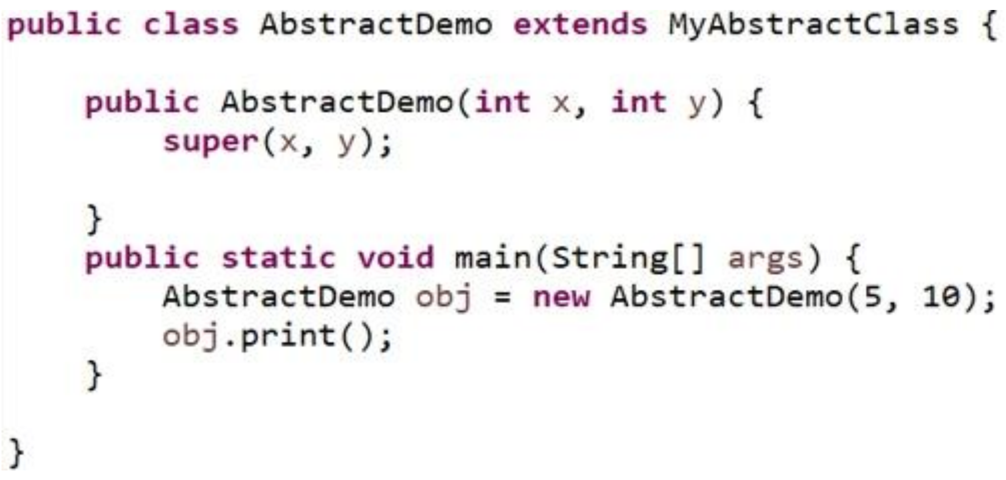
```java
package com.tech;

abstract class MyAbstractClass {

    public MyAbstractClass() {
        System.out.println("inside MyAbstractClass constructor");
    }

}
public class AbstractDemo extends MyAbstractClass {

    public AbstractDemo() {
        System.out.println("inside AbstractDemo constructor");
    }

    public static void main(String[] args) {
        AbstractDemo obj = new AbstractDemo();
    }

}
```



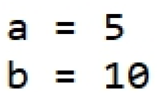
```
inside MyAbstractClass constructor
inside AbstractDemo constructor
```



```java
package com.tech;

abstract class MyAbstractClass {

    public int a;
    public int b;

    public MyAbstractClass(int a, int b) {
        this.a = a;
        this.b = b;
    }

    public void print() {
        System.out.println("a = " + a);
        System.out.println("b = " + b);
    }

}
```



```java
public class AbstractDemo extends MyAbstractClass {

    public AbstractDemo(int x, int y) {
        super(x, y);

    }
    public static void main(String[] args) {
        AbstractDemo obj = new AbstractDemo(5, 10);
        obj.print();
    }

}
```



```
a = 5
b = 10
```

Question 4: What is an Interface?

Answer: An interface in Java is a blueprint of a class. It has static constants and abstract methods.

Interface specify what a class must do but not how to do

An interface is like defining a contract that is fulfilled by implementing classes

An interface is used to achieve full abstraction.

All methods in an interface are public and abstract by default and

all variables declared in an interface are constants i.e. public, static

and final

A class which implements an interface will have to provide implementation of all the methods that are defined in the interface

A class can implement more than one interface, this is how Java allows multiple inheritance.

Since Java 8, we can have default and static methods in an interface

Question 5: Difference between abstract class and interface

Answer: The differences are:

Abstract class can have both abstract and concrete methods but interface can only have abstract methods (Java 8 onwards, it can have default and static methods as well)

Abstract class methods can have access modifiers other than public but interface methods are implicitly public and abstract

Abstract class can have final, non-final, static and non-static variables but interface variables are only static and final

A subclass can extend only one abstract class but it can implement multiple interfaces

An Abstract class can extend one other class and can implement multiple interfaces but an interface can only extend other interfaces

In this question, the interviewer may try to confuse you by saying that from Java 8 onwards, you can have static and default methods in an Interface so now what is the difference between abstract class and interface and the answer you should tell is – We can still extend only one class but can implement multiple interfaces.

Question 6: What to choose – interface or abstract class

Answer: Consider these points while choosing between the two:

When you want to provide default implementation to some of the common methods that can be used directly by the sub-classes then you can use abstract class because it can have concrete methods also, this is not the case with Interface because the child classes that are implementing this interface will have to provide implementation for all the methods that are declared in the interface

If your contract keeps on changing then Interface will create problems because then you will have to provide implementation of those new methods in all the implementing classes, whereas with abstract class you can provide one default implementation to the new methods and only change those implementing classes that are actually going to use these new methods

Most of the times, interfaces are a good choice. It is also one of the best practices, when you code in terms of interfaces.

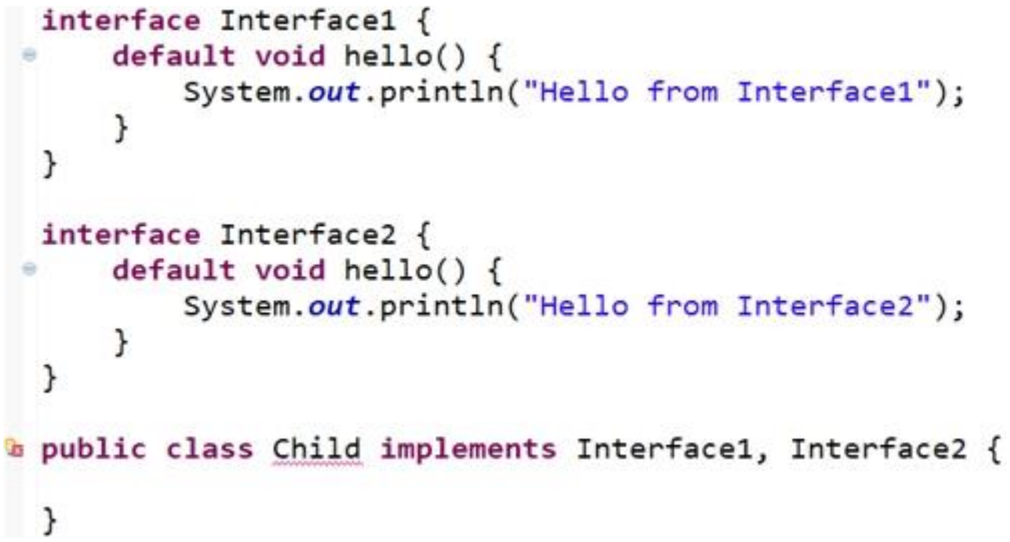
Question 7: Why Java 8 has introduced default methods?

Answer: To extend the capability of an already existing interface, default methods are introduced in Java 8.

Let’s understand this by one example:

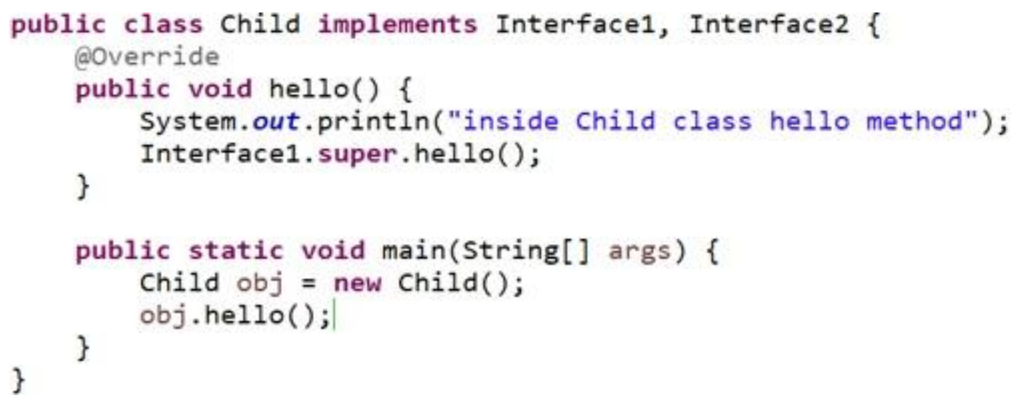
Consider there are 100 classes that are implementing one interface. Now you want to define one new method inside your interface. In this case you will have to change all the implementation classes to fulfill the interface contract. So, Java introduced default methods, here you can provide default implementation of that new method inside your interface and as it is not mandatory to provide implementation of default methods by the implementing classes, all the 100 classes can use the default implementation or if they want they can provide their own implementation by overriding the default method.

Now consider one interesting scenario: You have two interfaces, Interface1 and Interface2 both having default method hello () and one class is implementing these 2 interfaces without giving implementation to this default method. You see the problem here? Yes, it is the famous Diamond Problem.



```
interface Interface1 {
    default void hello() {
        System.out.println("Hello from Interface1");
    }
}

interface Interface2 {
    default void hello() {
        System.out.println("Hello from Interface2");
    }
}

public class Child implements Interface1, Interface2 {

}
```

So, to avoid this error, it is mandatory to provide implementation for common default methods of interfaces



```
public class Child implements Interface1, Interface2 {
    @Override
    public void hello() {
        System.out.println("inside Child class hello method");
        Interface1.super.hello();
    }

    public static void main(String[] args) {
        Child obj = new Child();
        obj.hello();
    }
}
```

```
inside Child class hello method
Hello from Interface1
```

Question 8: Why Java 8 has introduced static methods?

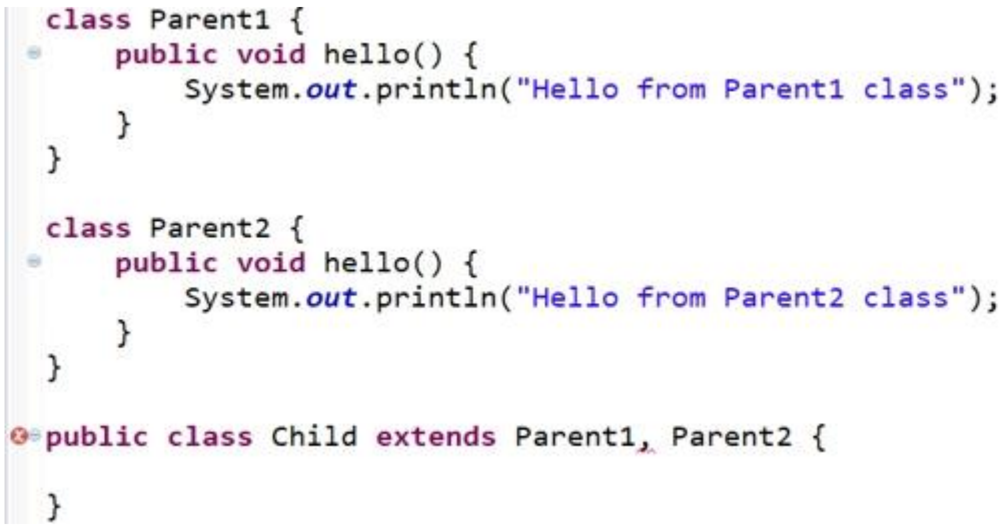
Answer: Consider an example where you want to define a utility class, what you usually do is you define a class which contains static methods and then you call these methods using class name. Now, Java 8 onwards you can do the same thing using an Interface by giving only static methods inside your interface. This way of using Interface for defining utility classes is better as it helps in performance also, because using a class is more expensive operation than using an interface.

Question 9: Why Java does not allow multiple inheritance?

Answer: Multiple inheritance occurs when a class has more than one parent classes.

Why Java does not allow this: let us consider there are 2 parent classes having a method named hello () with same signature and one child class is extending these 2 classes, if you call this hello() method which is same in both parents, which parent class method will get executed – it results into an ambiguous situation, this is also called Diamond Problem .

You will get a compile time error if you try to extend more than one class.



```
class Parent1 {
    public void hello() {
        System.out.println("Hello from Parent1 class");
    }
}

class Parent2 {
    public void hello() {
        System.out.println("Hello from Parent2 class");
    }
}

public class Child extends Parent1, Parent2 {

}
```

Question 10: Can we override final methods?

Answer: No, final methods cannot be overridden.