ABSTRACTION IN JAVA

ABSTRACTION IN JAVA

https://javabypatel.blogspot.com/2017/07/real-time-example-of-abstract-class-and-interface-in-java.html

Data Abstraction is the property by virtue of which only the essential details are displayed to the user.The trivial or the non-essentials units are not displayed to the user.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

Advantages:

1. Only show essential details to end user.
2. Hide complexity.
3. Avoids code duplication and increases reusability.

In java, abstraction is achieved by **interfaces** and **abstract** classes. We can achieve 100% abstraction using interfaces.

	Interface	**Abstract class**
	Interface helps in achieving **pure abstraction** in java.	Abstract class aren't **purely abstraction** in java

	All Interface are **abstract** by default.
	So, it's not mandatory to write abstract keyword with interface.
	
	Example-
	interface MyInterface {}
	// complier will add abstract
	Because of default additions done by compiler, above code will be same as writing below code-
	abstract interface MyInterface {
	}
		It's mandatory to write abstract keyword to make class abstract.
		
		Example-
		abstract class MyAbstractClass{
		abstract void m();
		}

	Interface does not have constructors.	Abstract class have constructors. Constructors are called by constructors of sub-class when object of subclass is created.
Inheritance vs Abstraction	Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
Implementation	Interface doesn't extend classes.	Classes implement Interface [provide the implementation of interfaces] Classes can implement more than one interface.
Type of methods	All the methods in Interface are **public and abstract** by default.	Abstract class can have private, final, abstract, static and instance methods. **Note** : Method cannot be private and abstract. Method cannot be final and abstract.

Encapsulation vs Data Abstraction

Encapsulation	Abstraction
1. Encapsulation is a concept for wrapping of data and code that manipulates the data into a single unit. 2. Encapsulation is a way of data hiding. 3. Encapsulation is achieved by access modifiers and classes.	1. Abstraction is a way to show only essential details to user. 2. Abstraction is way of hiding complexity. 3. Abstraction is achieved by abstract class and interface.

Abstract classes and Abstract methods :

1. An abstract class is a class that is declared with **abstract keyword.**
2. An abstract class may or may not have all abstract methods. Some of them can be concrete methods.
3. Abstract class can have private, final, abstract, static and **instance** methods.
   Note : Method cannot be both private and abstract (at the same time). Method cannot be both final and abstract (at the same time).

   Abstract classes can also have final methods (methods that cannot be overridden). For example, the following program compiles and runs fine.

```
// An abstract class with a final method
abstract class Base {
  final void fun() {
    System.out.println("Derived fun() called");
  }
}

class Derived extends Base {
}

class Main {
        public static void main(String args[]) {
          Base b = new Derived();
          b.fun();
        }
}
```

4. An abstract method is a method that is declared without an implementation.
5. A method defined abstract must always be redefined in the subclass, thus making **overriding** compulsory OR either make subclass itself abstract.
6. Any class that contains one or more abstract methods must also be declared with abstract keyword.

7. There can be no object of an abstract class.That is, an abstract class can not be directly instantiated with the **new** operator.
   [In Java, an instance of an abstract class cannot be created, we can have references of abstract class type though].

```
abstract class Base {
        abstract void fun();
}

class Derived extends Base {
  void fun() {
        System.out.println("Derived fun() called");
  }
}

class Main {
  public static void main(String args[]) {

        // Uncommenting the following line will cause compiler error as
the
        // line tries to create an instance of abstract class.
        // Base b = new Base();

        // We can have references of Base type.
        // b is a pointing[reference] to the Derived object created, but
is of type Base
        Base b = new Derived();
        b.fun();
      }
  }
```

8. An abstract class can have parametrized constructors and default constructor is always present in an abstract class.
   And a constructor of abstract class is called when an instance of a inherited class is created. For example, the following is a valid Java program.

```
// An abstract class with constructor
 abstract class Base {
        Base() {
          System.out.println("Base Constructor Called");
        }

        abstract void fun();
 }

 class Derived extends Base {
```

```
        Derived() {
          System.out.println("Derived Constructor Called");
        }

        void fun() {
          System.out.println("Derived fun() called");
        }
    }

    class Main {
        public static void main(String args[]) {
          Derived d = new Derived();
    }
```

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Interfaces:

1. Interfaces helps in achieving **pure abstraction** in java. Interface are purely abstract in java. Interfaces can only have abstract methods [All Interface are **abstract by default.** So, it's **not mandatory** to write abstract keyword with interface.]
2. Interface does **not have constructors.**
3. Interface doesn't extend classes.
    1. **Multiple inheritance** - Interface allows us to achieve multiple inheritance as well.
       — Interface always **extends another interface**.
       — Interface **can extend more than one interface**.

4. methods in interface are **public and abstract** by default
5.

Why an interface cannot implement another interface?
—> implements means implementation, when interface is meant to declare just to provide interface not for implementation.

WHEN TO USE INTERFACES AND ABSTRACT CLASSES PRACTICALLY?
A] WHEN USE ABSTRACT CLASSES -
In java application, there are some related classes that need to share some lines of code then you can put these lines of code(basically in a method) within abstract class and this abstract class should be extended by all these related

classes.

Let's say we have to choose between **class or interface** for **TerrestrialAnimals**, then one thing will be for sure that **habitat of all Terrestrial animals must be land**. That means we can have **same implementation of habitat method for all Terrestrial animals**. And **food of all Terrestrial animals might be different**.

*So, we will **create abstract class** with -*
instance method = habitat() *[because all Terrestrial animals live on land]*
abstract method = food() *[because food of all Terrestrial animals might be different]*

```
/*
 * abstract class
 */
abstract class TerrestrialAnimals {
	void habitat() {
		System.out.println("Habitat of Terrestrial animal is land");
	}
	abstract void food();
}

/*
 * concrete class - Lion
 */
class Lion extends TerrestrialAnimals {
	@Override
	void food() {
		System.out.println("Lion eat - flesh");
	}
}

/*
 * concrete class - Lion
 */
class Goat extends TerrestrialAnimals {
	@Override
	void food() {
		System.out.println("Goat eat - grass");
	}
}

public class MyClass {
	public static void main(String[] args) {
```

```
        TerrestrialAnimals lion = new Lion();
        lion.habitat();
        lion.food();

        System.out.println();

        TerrestrialAnimals goat = new Goat();
        goat.habitat();
        goat.food();
    }
}
```

Output

Habitat of Terrestrial animal is land Lion eat - flesh
Habitat of Terrestrial animal is land Goat eat - grass

B] WHEN USE INTERFACES -

Let's say we have to choose between **class or interface** for **Animals**, then **habitat of animals might be land or water**.
And food of all animals might be different.

*So, we will **create interface** with -*
abstract method = habitat() *[because animals might be living on land or water]*
abstract method = food() *[because food of all animals might be different]*

```
/*
 * interface
 */
interface Animals {
   abstract void habitat();
   abstract void food();
}

/*
 * concrete class - Lion
 */
class Lion implements Animals {
   @Override
   public void habitat(){
        System.out.println("Habitat of Lion is land");
   }
```

```
    @Override
    public void food(){
        System.out.println("Lion eat - flesh");
    }
}


/*
 * concrete class - Lion
 */
class Whale implements Animals {
    @Override
    public void habitat(){
        System.out.println("Habitat of Whale is water");
    }

    @Override
    public void food(){
        System.out.println("Whale eat - aquatic animals");
    }
}


public class MyClass{
    public static void main(String[] args) {
        Animals lion=new Lion();
        lion.habitat();
        lion.food();

        System.out.println();

        Animals whale=new Whale();
        whale.habitat();
        whale.food();
    }
}
```

Output

```
Habitat of Lion is land
Lion eat - flesh

Habitat of Whale is water
Whale eat - aquatic animals
```

‘ABSTRACT’ KEYWORD IN JAVA

abstract is a non-access modifier in java applicable for classes, methods but **not** variables. It is used to achieve abstraction which is one of the pillar of Object Oriented Programming(OOP). Following are different contexts where *abstract* can be used in Java.

1. ***Abstract classes :*** The class which is having partial implementation(i.e. not all methods present in the class have method definition).
   abstract class class-name{ //body of class }
   Some of the predefined classes in java are abstract. For example, java.lang.Number is an abstract class.
2. ***Abstract methods :*** Sometimes, we require just method declaration in super-classes.This can be achieve by specifying the **abstract** type modifier.
   abstract type method-name(parameter-list);

A class cannot be both abstract and final

In java, you will never see a class or method declared with both **final** and **abstract** keywords. For classes, **final** is used to **prevent inheritance** whereas **abstract** classes **depends upon their child classes for complete implementation**. In cases of methods, **final** is used to prevent **overriding** whereas **abstract methods** needs to be **overridden** in sub-classes.

Interview Questions on Abstraction in java

https://javarevisited.blogspot.com/2013/04/10-abstract-class-and-interface-interview-question-java-answers.html

https://javaconceptoftheday.com/java-interview-questions-on-abstract-class/

https://www.wisdomjobs.com/e-university/java-abstraction-interview-questions.html

http://www.codespaghetti.com/abstract-interview-questions/

https://www.javatpoint.com/corejava-interview-questions-2

1. Can abstract class implements interface in Java? does they require to implement all methods?
   Ans: Yes, abstract class can implement interface by using implements keyword. Since they are abstract, they don't need to implement all methods.

2. Is it compulsory for a class which is declared as abstract to have at least one abstract method?

Ans: Not necessarily. Abstract class may or may not have abstract methods.

3. Why final and abstract cannot be used at a time?
   Ans: Because, final and abstract are totally opposite in nature. A final class or method can not be modified further where as abstract class or method must be modified further. “final” keyword is used to denote that a class or method does not need further improvements. “abstract” keyword is used to denote that a class or method needs further improvements.

4. Can a class contain an abstract class as a member?
   Ans: Yes, a class can have abstract class as it’s member.

5. Abstract classes can be nested. True or false?
   Ans: True. Abstract classes can be nested i.e an abstract class can have another abstract class as it’s member.

6. Can we declare abstract methods as synchronized?
   Ans: No, abstract methods cannot be declared as synchronized. But methods which override abstract methods can be declared as synchronized.

7. Can we declare local inner class as abstract?
   Ans: Yes. Local inner class can be abstract.

8. Can abstract method declaration include throws clause?
   Ans:
   Yes. Abstract methods can be declared with throws clause.
   ```
   abstract class A {
         public abstract void show() throws Exception;
    }
   ```

9. What Is Abstraction In Java?
   Ans:-
   Data Abstraction is the property by virtue of which only the essential details are displayed to the user.The trivial or the non-essentials units are not displayed to the user.
   Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.
   Advantages: [1] Only show essential details to end user
   [2] Hide complexity.
   [3] Avoids code duplication and increases reusability.

10. How To Achieve Abstraction In Java?

    Ans:-

    There are two ways to achieve abstraction in java - (1) Abstract class (0 to 100%). (2) Interface (100%)

11. What Is Abstract Class In Java? [AMOC]

    Ans:-

    [1] An abstract class is a class that is declared with **abstract keyword.**
    [2] An abstract class may or may not have all abstract methods. Some of them can be concrete methods.
    [3] Abstract methods can be only public or protected.
    Abstract class cannot be private, final and static abstract methods.

    But the concrete methods in Abstract class can be private, static or final.

```
abstract class Base {
    Base() {
        System.out.println("dew");
    }

    private void Base1() {
        System.out.println("dew");
    }

    final void Base2() {
        System.out.println("dew");
    }

    static void Base3() {
        System.out.println("dew");
    }

    abstract void fun();
}
```

[4] There can be no object of an abstract class.That is, an abstract class cannot be directly instantiated with the **new** operator.

[5] An abstract class can have parametrized constructors and default constructor is always present in an abstract class.

And a constructor of abstract class is called when an instance of an inherited class is created. For example, the following is a valid Java program.

```
//An abstract class with constructor
    abstract class Base {
        Base() {
            S.o.p("Base Constructor Called");
        }
```

```
        abstract void fun();
}

class Derived extends Base {
        Derived() {
                S.o.p”Derived Constructor Called");
        }

        void fun() {
                S.o.p(“Derived fun() called");
        }
}

class Main {
        public static void main(String args[]) {
                Derived d = new Derived();
        }
}
```

11. What is abstract method in Java?
    Ans:-
    Methods which do not have a body are basically abstract.
    A method prefixed with the abstract keyword in abstract class is a called as an Abstract().
    Abstract methods cannot be private, static or final. Their visibility can be only public or protected.
    Abstract methods are basically enhanced in the classes that extend the Abstract class.

12. Can we declare abstract methods as private? Justify your answer?
    Ans:-
    No, Abstract methods cannot be private. If abstract methods are allowed to be private, then they will not be inherited to sub class and will not get enhanced.

13. Abstract class must have only abstract methods. True or false?
14. Is it compulsory for a class which is declared as abstract to have at least one abstract method?
15. Is it necessary for abstract class to have abstract method?
    Ans:-
    Not, necessarily. Abstract class may or may not have abstract methods.
    We can define concrete as well as non-concrete method[abstract] in

abstract class.

16. What will happen if we do not override all the abstract methods in sub-class?
    Ans:-
    It will throw compile-time error[Add unimplemented methods]. We have to override all the abstract method in sub-class. If you do not want to override all the abstract method in sub-class then you have to make your sub-class as abstract class.
    Eg:-

```
abstract class Base {
        abstract void detail();
}

class Derived extends Base {}       // It will tell that we need to implement the Base.detail method
```

12. Can we declare abstract method in non-abstract class?
    Ans:-
    No, We can't declare abstract method in non-abstract class. It gives compilation error, saying abstract methods can only be declared in abstract classes.
    For example:

```
class Demo {   //Error - The type Demo must be an abstract class to define abstract methods
        abstract void show();      //Error - The abstract method show in type Demo can only be defined by an abstract class
}
```

    Above example will throw compile time error.

15. Difference between abstract class and interface in Java?
    Ans:-

Abstract class	**Interface**
An abstract class can have abstract and non-abstract methods[methods having a body].	The interface has only abstract methods.
An abstract class can have the constructor.	The interface cannot have the constructor.
An abstract class can have static methods.	The interface cannot have static methods.
An abstract class can have main method	The interface cannot have main method.
You can extend one abstract class.	You can implement multiple interfaces.

The abstract class **can provide the implementation of the interface**.	The Interface **can't provide the implementation of the abstract class**.
The **abstract keyword** is used to declare an abstract class.	The **interface keyword** is used to declare an interface.
An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
An **abstract class** can be extended using keyword **extends**	An **interface class** can be implemented using keyword **implements**
A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Similarities:- Neither abstract classes nor interfaces can be instantiated.

16. Can abstract class implements interface in Java? Do they require to implement all methods?
    Ans:-
    Yes, abstract class can implement interface by
    using implements keyword. Since they are abstract, they don't need to implement all methods

17. When do you favour abstract class over interface?
18. If interface & abstract class have same methods and those methods contain no implementation, which one would you prefer?
    Ans:-
    One should ideally go for an interface, as we can only extend one class. Implementing an interface for a class is very much effective rather than extending an abstract class because we can extend some other useful class for this subclass.

19. Why we use interface in java?
    Ans:-
    It is used to achieve fully abstraction.
    By interface, we can support the functionality of multiple inheritance.
    It can be used to achieve loose coupling.

20. What is meant by "Abstract Interface"?
    Ans:-
    An interface is abstract

21. Methods in interface are by default public and abstract. true or false? - True

22. Data member in interface are by default public, static, and final. true of false? - True

23. Can you create instance of abstract class?
24. Can we instantiate a class which does not have even a single abstract methods but declared as abstract?
    Ans:-
    No, We can't instantiate a class once it is declared as abstract even though it does not have abstract methods.

25. Can abstract class have constructors in Java?
    We can't instantiate an abstract class. Then why constructors are allowed in abstract class?
    Ans:-
    A constructor in Java doesn't actually "build" the object, it is used to initialize fields.

    Imagine that your abstract class has fields x and y, and that you always want them to be initialized in a certain way, no matter what actual concrete subclass is eventually created.
    So you create a constructor and initialize these fields.

    Now, if you have two different subclasses of your abstract class, when you instantiate them their constructors will be called, and then the parent constructor i.e the constructor of the abstract class will be called and the fields will be initialized.
    If you don't do anything, the default constructor of the parent will be called

26. Can we use "abstract" keyword with constructor, Instance Initialization Block and Static Initialization Block? - No

27. Can abstract class be final in Java?
28. Why final and abstract cannot be used at a time?
29. Can you use abstract and final both with a method?
    Ans:-
    No, because we need to override the abstract method to provide its implementation, whereas we can't override the final method.

30. Can abstract class have static methods in Java?
    Ans:-

Suppose when you have a concrete method in an abstract class then that method can be static.

```
abstract class AbstractTest {
        static void disp() {
                System.out.println("disp of static method");
        }
}

public class OverloadExample extends AbstractTest {
        public static void main(String[] args) {
                AbstractTest.disp();   —> disp of static method
        }
}
```

Static methods belong to class.

making an abstract method static would make it possible to call an undefined method which is of no use, hence it is not allowed.

```
abstract class AbstractTest {
        abstract static void disp(); —> Compilation Error
}
```

32. Can abstract class contains main method in Java ?

    Ans:-

    Yes, the abstract class can contain the main method.

33. Can we use public, protected and default modifiers with abstract method?

    Ans:-

    Yes, We can use public, protected and default modifiers with abstract method.

34. Which of the following is FALSE about abstract classes in Java

    (a) If we derive an abstract class and do not implement all the abstract methods, then the derived class should also be marked as abstract using 'abstract' keyword

    (b) Abstract classes can have constructors

    (c) A class can be made abstract without any abstract method

    (d) A class can inherit from multiple abstract classes.

    Ans:- d)

41. What is the difference between abstraction and encapsulation?

    Ans:-

    Abstraction hides the implementation details whereas encapsulation wraps code and data into a single unit.

42. Is the following program written correctly? If yes then what will be the

output of the program?

```
abstract class Calculate {
    abstract int multiply(int a, int b);
}

public class Main {
    public static void main(String[] args) {
        int result = new Calculate() {   // one in bold is anonymous class object
            @Override
            int multiply(int a, int b) {
                return a * b;
            }
        }.multiply(12, 32);
        System.out.println("result = " + result);
    }
}
```

Ans:-

Yes, the program is written correctly. The Main class provides the definition of abstract method multiply declared in abstract class Calculation.

The output of the program will be:- **384**

https://www.javabykiran.com/interview/corejava/abstraction-interview-questions

43. Can an interface be extended by another interface in Java ?
44. In which kind of situation would an interface be extended by another interface?

    Ans:-

    Remember that any class that implements an interface must implement the method headings that are declared in that interface. If that particular interface extends from other interfaces, then the implementing class must also implement the methods in the interfaces that are being extended or derived from.

45. How can we define an interface?

    Ans:-

    In Java an interface just defines the methods and not implement them. Interface can include constants.

A class that implements the interfaces is bound to implement all the methods defined in an interface.

Example of Interface :

```
public interface sampleInterface {
    public void functionOne();
    public long CONSTANT_ONE = 1000;
```

```
}
```