Core java Interview questions on Coding Standards

1. **Explain Java Coding Standards for classes or Java coding conventions for classes?**
    - In Java, the name of the class should be purely a noun. A class basically represents an entity in a software application. For instance, in a car racing computer game, a Car can be Class. Most appropriately, a class name should not consist of more than one word. For instance, it is always better to name a class Car rather than DrivingCar.
    - Another extremely important issue to note is that a class name in Java should be written in Pascal case. In Pascal case, first letter of every word should be a capital. Therefore, for the previous car race game example, the class name should be Car and DrivingCar rather than car and drivingcar respectively.

2. **Explain Java Coding standards for interfaces?**
    - Interface names should ideally be adjectives and should end in the prefix “able”.
    - For example, if in a car race game, a car can be converted into some other vehicle type such as a motor bike or truck. The Car class should implement an interface named Convertible, which would contain methods that convert the Car class instance to other vehicle type instances. Convertible is an adjective in this case which refers to the ability of the Car class to be converted into some other vehicle type.
    - By convention, the first letter of every interface should be capital.

3. **Explain Java Coding Standards for Constants?**
    - Every class often contains some constant values that are not changed throughout the life-cycle of the instance of the class. Such values are called constants. For instance, the Car class can contain an integer type constant value of maximum speed which can be any constant. Conventionally, these constant variables should be named as:

      public final static int Max_Speed = 1000;

    - Constant values can be declared public unlike variables since they are required to be accessed in multiple instances of the class. Declaring a variable final in java means that the value of that variable cannot be changed.

- The first letter of every word in a constant variable's name should be capitalized. If a constant variable contains multiple names, they should be separated by underscores.

4. What is 'IS-A ' relationship in java?

- A relationship in Java means different relations between two or more classes. For example, if a class Bulb inherits another class Device, then we can say that Bulb is having is-a relationship with Device, which implies Bulb is a device.
- IS-A Relationship is wholly related to Inheritance. For example – a kiwi is a fruit; a bulb is a device.
    - IS-A relationship can simply be achieved by using extends Keyword.
    - IS-A relationship is additionally used for code reusability in Java and to avoid code redundancy.
    - IS-A relationship is unidirectional, which means we can say that a bulb is a device, but vice versa; a device is a bulb is not possible since all the devices are not bulbs.
    - IS-A relationship is tightly coupled, which means changing one entity will affect another entity.

Advantage of IS-A relationship

- Code Reusability.
- Reduce redundancy.

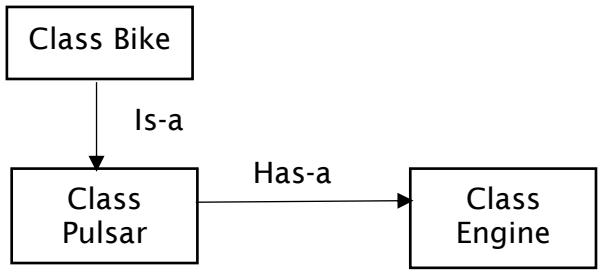
How to achieve IS-A relationship

- IS-A relationship can simply be achieved by extending an interface or class by using extend keyword.

5. What is 'HAS A'' relationship in java?

- Association is the relation between two separate classes which establishes through their Objects.
- Composition and Aggregation are the two forms of association. In Java, a Has-A relationship is otherwise called composition. It is additionally utilized for code reusability in Java.
- In Java, a Has-A relationship essentially implies that an example of one class has a reference to an occasion of another class or another occurrence of a similar class.

- For instance, a vehicle has a motor, a canine has a tail, etc. In Java, there is no such watchword that executes a Has-A relationship. Yet, we generally utilize new catchphrases to actualize a Has-A relationship in Java.



Has-a is a special form of Association where:

- It represents the Has-A relationship.
- It is a unidirectional association i.e. a one-way relationship. For example, here above as shown pulsar motorcycle has an engine but vice-versa is not possible and thus unidirectional in nature.
- In Aggregation, both the entries can survive individually which means ending one entity will not affect the other entity.

6. Difference between 'IS-A' and 'HAS-A' relationship in java?

IS-A	HAS-A
It is a concept of Inheritance	It is a concept of Composition
A class cannot extend more than one class.	A class can have HAS-A relationship with multiple other classes
A final class cannot be extended in Inheritance	We can reuse final classes in Composition
Inheritance is static binding and cannot be changed at runtime	Composition is dynamic binding and is flexible for changes

7. Explain about instanceof operator in java

- The java instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface).
- The instanceof in java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

Example:

```
class Simple1{
 public static void main(String args[]){
 Simple1 s=new Simple1();
 System.out.println(s instanceof Simple1);//true
 }
}
```

8. What does null mean in java?

- In Java, null is a reserved word for literal values. It seems like a keyword, but actually, it is a literal similar to true and false.
- It is case-sensitive.
- It is a value of the reference variable.
- The access to a null reference generates a NullPointerException.
- It is not allowed to pass null as a value to call the methods that contain any primitive data type.

Example 1:

```
public class NullExample1 {
   static NullExample1 obj;
   public static void main(String[] args) {

      System.out.println(obj);
   }

}        //output: null
```

Example 2:

```
public class NullExample2 {
```

```
    public static void main(String[] args) {
      NullExample2 obj = null;
      System.out.println(obj);
    }

}
```

9. Can we have multiple classes in single file ?

- No, you cannot have multiple classes in the same Java file. Each Java file can contain only one public class. If you try to declare more than one public class in a single Java file, you will get a compiler error

10. What all access modifiers are allowed for top class ?

- For top level class only two access modifiers are allowed.
- public and default. If a class is declared as public it is visible everywhere.
- If a class is declared default it is visible only in same package.
- If we try to give private and protected as access modifier to class ,we get the below compilation error.
  "Illegal Modifier for the class; only public,abstract and final are permitted."

11. What are packages in java?

- A java package is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form, built-in package and user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Advantage of Java Package

a. Java package is used to categorize the classes and interfaces so that they can be easily maintained.
b. Java package provides access protection.
c. Java package removes naming collision.

- The package keyword is used to create a package in java.

12. Can we have more than one package statement in source file ?

- No we can't.
- Consider I have created a package com.javabasics and inside that I have a Java class file (source file) as TestPackage

```
package com.shreyank;

class TestPackage {

}
```

- Now, if you try to define another package

```
package com.shreyank;
package com;
```

- Then compiler won't understand what are you trying to do as your source file is inside com.shreyank package.

13. Can we define package statement after import statement in java?

- No, we cannot define a package after the import statement in Java. The compiler will throw an error if we are trying to insert a package after the import statement. A package is a group of similar types of classes, interfaces, and sub-packages. To create a class inside a package, declare the package name in the first statement in our program.

  Example:

```
import java.lang.*;
package test;
public class PackageAfterImportTest {
  public static void main(String args[]) {
    System.out.println("Welcome!!!");
  }
}
```

  Output:

```
PackageAfterImportTest.java:3: error: class, interface, or enum expected
```

```
package test;
^
1 error
```

14. What are identifiers in java?

- Identifiers in Java are symbolic names used for identification. They can be a class name, variable name, method name, package name, constant name, and more. However, In Java, There are some reserved words that can not be used as an identifier.
- For every identifier there are some conventions that should be used before declaring them.

15. What are access modifiers in java?

- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

  There are four types of Java access modifiers:

  1. Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
  2. Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
  3. Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
  4. Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

16. What is the difference between access specifiers and access modifiers in java?

- The meaning of both the access specifiers and the access modifiers is the same. There are no differences between the specifiers and modifiers, and the use of

both is the same. The access modifier is an official term and the new term that we use instead of modifier is specifier.

- So, default, public, protected, and private access modifiers can also be referred to as default, public, protected, and private access specifiers.

17. What access modifiers can be used for class ?

- The classes and interfaces themselves can have only two access modifiers when declared outside any other class.
    a. public
    b. default (when no access modifier is specified)
- Nested interfaces and classes can have all access modifiers.
- We cannot declare class/interface with private or protected access modifiers.

18. What access modifiers can be used for methods?

- public, private, protected, default modifiers can be used for methods in java.

19. What access modifiers can be used for variables?

- public, private, protected, default modifiers can be used for variables in java.

20. What is final access modifier in java?

- It is a modifier applicable to classes, methods, and variables. If we declare a parent class method as final then we can't override that method in the child class because its implementation is final and if a class is declared as final we can't extend the functionality of that class i.e we can't create a child class for that class i.e inheritance is not possible for final classes.
- Every method present inside the final class is always final by default but every variable present inside the final class need not be final.
- The main advantage of the final keyword is we can achieve security and we can provide a unique implementation.
- But the main disadvantage of the final keyword is we are missing key benefits of OOPs like Inheritance(Because of the final class), Polymorphism(Because of the

final method) hence if there are no specific requirements then it is not recommended to use the final keyword.

Example:

```
// Java program to illustrate Final keyword


// import required packages
import java.io.*;
import java.util.*;
// Declaring parent class P
class P {
   // Declaring a first name
   // method
   public void firstName()
   {
      // Display firstname
      System.out.println("Rahul ");
   }
   /// Declaring a final surName
   // method
   public final void surName()
   {
      // Display surname
      System.out.println("Trivedi");
   }
}
// Creating a child class
// of above parent class
class C extends P {
   // overriding the surName
```

```
    // method
    public void surName()
    {
      // Display surname
      System.out.println("Sharma");
    }
    // Main method
    public static void main(String[] args)
    {
      // Display message
      System.out.println("Hi");
    }
}
```

Output:

```
Error: surname() in C cannot override surname() in P
        overridden method is final
```

21. Explain about abstract classes in java?

- Abstraction is a process of hiding the implementation details and showing only functionality to the user.
- Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.
- A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Example:

```
abstract class Bank{
abstract int getRateOfInterest();
}
```

```
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

22. Can we create constructor in abstract class ?

- Constructor is always called by its class name in a class itself. A constructor is used to initialize an object not to build the object. As we all know abstract classes also do have a constructor.
- So if we do not define any constructor inside the abstract class then JVM (Java Virtual Machine) will give a default constructor to the abstract class.
- If you define your own constructor without arguments inside an abstract class but forget to call your own constructor inside its derived class constructor then JVM will call the constructor by default.
- So if you define your single or multi-argument constructor inside the abstract class then make sure to call the constructor inside the derived class constructor with the super keyword.

Example:

```
abstract class Content {
```

```
    int a;

    // Constructor of abstract class
    public Content(int a)
    {

        // This keyword refers to current instance itself
        this.a = a;
    }

    // Abstract method of abstract class
    abstract int multiply(int val);
}

// Class 2
// Helper class extending above Class1
// Child class of Abstract class
class A extends Content {

    // Constructor of Child class A
    A()
    {

        // Super keyword refers to parent class
        super(2);
    }

    // Defining method the abstract method
    public int multiply(int val)
    {

        // Returning value of same instance
        return this.a * val;
    }
}
```

```
// Class 3
// Main class
public class A {

  // Main driver method
  public static void main(String args[])
  {

    // Creating reference object of abstract class
    // using it child class
    Content c = new A();

    // Calling abstract method of abstract class
    System.out.println(c.multiply(3));
  }
}
```

23. What are abstract methods in java?

- A method declared using the abstract keyword within an abstract class and does not have a definition (implementation) is called an abstract method.
- When we need just the method declaration in a super class, it can be achieved by declaring the methods as abstracts.
- Abstract method is also called subclass responsibility as it doesn't have the implementation in the super class. Therefore a subclass must override it to provide the method definition.

Example:

```
abstract class Multiply {

  // abstract methods
  // sub class must implement these methods
  public abstract int MultiplyTwo (int n1, int n2);
  public abstract int MultiplyThree (int n1, int n2, int n3);
```

```
  // regular method with body
  public void show() {
   System.out.println ("Method of abstract class Multiply");
  }
}
// Regular class extends abstract class
class AbstractMethodEx1 extends Multiply {

 // if the abstract methods are not implemented, compiler will give an
error
  public int MultiplyTwo (int num1, int num2) {
   return num1 * num2;
  }
  public int MultiplyThree (int num1, int num2, int num3) {
   return num1 * num2 * num3;
  }

  // main method
  public static void main (String args[]) {
   Multiply obj = new AbstractMethodEx1();
   System.out.println ("Multiplication of 2 numbers: " + obj.MultiplyTwo
(10, 50));
   System.out.println ("Multiplication of 3 numbers: " + obj.MultiplyThree
(5, 8, 10));
   obj.show();
  }
}
```