Types of classes in Java

What are classes in Java

- A class in Java is a template that is used to create and define objects, object data types, and methods.

- Classes as a whole are categories and objects are items within each category.

- A class declaration constitutes of the following parts:
  a. Modifiers
  b. Class name
  c. Keywords
  d. The class body within curly brackets {}

Types of classes in Java

- POJO Class :-Plain Old Java Object.
    - A class which contains only private variables and setter and getter methods to use those variables is called POJO class.
    - It is a pure data structure that has fields and may override some methods from Object (e.g. equals) or some other interface like serializable but does not have the behavior of its own.
- Properties of POJO class –
    - Public setter and getter methods are a must while writing a POJO class.
    - All instance variables should be private.
    - It should not extend pre-specified classes.
    - It should not implement pre-specified interfaces.
    - Should not contain pre-specified annotations.
    - It may not have a no-argument constructor.

```
class POJO {
  private int value=365;
  public int getValue() {
      return value;
   }
   public void setValue(int value) {
      this.value = value;
   }
}
public class Test {
   public static void main(String args[]){
      POJO p = new POJO();
      System.out.println(p.getValue());
   }
}
```

Types of classes in Java

- Static Class :-
    - static is a keyword used to describe how objects are managed within the memory.
    - A static object belongs specifically to the class, instead of instances of that class.
- The sole purpose of the class is to provide blueprints of its inherited classes.
- A static class can contain static members only.
- You cannot create an object for a static class.

```
public static class SBISavings
{
  public void displayOutput(){
System.out.println(" SBISaving is: " + note);
  }
}
```

```
public class Main{
public static void main(String[] args)
{
//calling the method
SBISavings.displayOutput();
}
}
```

Types of classes in Java

- Concrete Class :-
    - Any normal class which does not have any abstract method or a class having an implementation for all of its methods is basically a concrete class.
    - They cannot have any unimplemented methods.
    - A concrete class can extend its parent class, an abstract class or implement an interface if it implements all their methods.
    - It is a complete class that can be instantiated.

```
public class Concrete { // Concrete Class
    static int sum(int x, int y) {
        return a + b;
    }
    public static void main(String args[]) {
        int p = sum(6, 8);
        System.out.println("Sum: " + p);
    }
}
```

Types of classes in Java

- Abstract Class :-
    - An abstract class is declared with an abstract keyword and have zero or more abstract methods.
    - These classes are incomplete classes, therefore, to use an abstract class we strictly need to extend the abstract classes to a concrete class.
    - It can have constructors and static methods as well. It can have final methods which will force the subclass to keep the body of the method unhung.

```
// Java program to illustrate concrete class
//This is an interface
interface X{
int product(int x, int y);
}
// This is an abstract class
abstract class Product implements X{
// this method calculates
// product of two numbers
public int product(int x, int y){
return x * y;
}
}
// This is a concrete class that implements
class Main extends Product{
// main method
public static void main(String args[]){
Main ob = new Main();
int p = ob.product(20, 10);
// print product
System.out.println("Product: " + p);
}
}
```

Types of classes in Java

- Final Class :-
    - Once a variable, method or a class is declared as final, it’s value remains the same throughout.
    - The final keyword in a method declaration indicates that the method cannot be overridden by any subclasses i.e., a class that has been declared final cannot be subclassed.
    - This helps a lot while creating an immutable class like the String class.
    - A class cannot make a class immutable without making it final.

```
final class BaseClass {
   void Display() {
    System.out.print("This is the Display() method of BaseClass.");
   }
}
class DerivedClass extends BaseClass { //Compile-time error - can't
inherit final class
   void Display() {
      System.out.print("This is Display() method of DerivedClass.");
   }
}
public class FinalClassDemo {
   public static void main(String[] arg) {
      DerivedClass d = new DerivedClass();
      d.Display();
   }
}
```

Types of classes in Java

- Inner class :-
    - Inner class means the class which is a member of another class. There are four types of inner classes in java
        - Nested Inner class
        - Method Local inner classes
        - Anonymous inner classes
        - Static nested classes

Nested Inner class

- It can access any private instance variable of an outer class.
- We can have access modifiers private, protected, public and default modifier.

```
class Outer {
   // Simple nested inner class
   class Inner {
      public void show() {
           System.out.println("This is inside a nested class method ");
      }
   }
}
class Main {
   public static void main(String[] args) {
       Outer.Inner in = new Outer().new Inner();
       in.show();
   }
}
```

Method Local inner classes

- An inner class can be declared within a method of an outer class.

```
class Outer {
    void outerMethod() {
        System.out.println("This is outerMethod");
        // Inner class is local to outerMethod()
        class Inner {
            void innerMethod() {
                System.out.println("This is innerMethod");
            }
        }
        Inner y = new Inner();
        y.innerMethod();
    }
}
class MethodDemo {
    public static void main(String[] args) {
        Outer x = new Outer();
        x.outerMethod();
    }
}
```

Anonymous inner classes

- Anonymous inner classes are declared without any name. They can be created in two ways.

```
class Demo {
   void show() {
      System.out.println("This is show method of super class");
   }
}
class FlagDemo {

    //  An anonymous class with Demo as base class
    static Demo d = new Demo() {
        void show() {
            super.show();
            System.out.println("This is Flag1Demo class");
    }
    };
    public static void main(String[] args){
        d.show();
    }
```

-)

Anonymous inner classes

- Anonymous inner classes are declared without any name. They can be created in two ways.

```
interface Hello {
    void show();
}
class Flag2Demo {

    // An anonymous class that implements Hello interface
    static Hello h = new Hello() {
        public void show() {
            System.out.println("This is an anonymous class");
        }
    };

    public static void main(String[] args) {
        h.show();
    }
}
```

-

Static nested classes

- class Outer { s are like a static member of the outer class.

```
    private static void outerMethod() {
      System.out.println("inside outerMethod");
    }

    // A static inner class
    static class Inner {
              public void test(){
                 //logic
              }
          }
  }
  class A{
      public static void main(String[] args) {
          Outer o=new Outer();
          o.Inner.test();
      }
```

Static nested classes

- Static nested classes are like a static member of the outer class.

```
class Outer {
    private static void outerMethod() {
      System.out.println("inside outerMethod");
    }

    // A static inner class
    static class Inner {
      public static void main(String[] args) {
         System.out.println("inside inner class Method");
         outerMethod();
      }
    }

}
```

-