Topics to cover:

1. Different types of classes

1. Different types of classes

A class is a blueprint in the Java programming language from which an individual object can be built. In Java, we may declare a class by using the class keyword. Class members and functions are declared simply within the class. Classes are required for the creation of Java programs. The object-oriented paradigm (OOP) allows users to describe real-world objects. Also, a class is sometimes known as a user-defined data type.

1.1 Types of Classes

There are several types of classes such as the following.

1.1.1 Final Class

When a variable, function, or class is declared final, its value persists throughout the program. Declaring a method with the final keyword indicates that the method cannot be overridden by subclasses. That is a class that is marked final cannot be subclasses. This is very useful when creating immutable classes such as String classes. A class cannot be mutated unless it is declared final.

Example:

```
final class Base {

    void Display()
    {
        System.out.print("Method for Base class.");
    }
}

class Extended extends Base {

    void Display()
    {
        System.out.print("Method of Extended class.");
    }
}

class MainClass {

    public static void main(String[] arg)
    {
        Extended d = new Extended();
        d.Display();
    }
}
```

Output:

```
ERROR!
javac /tmp/ZZBjgU1AvH/MainClass.java
/tmp/ZZBjgU1AvH/MainClass.java:14: error: cannot inherit from final Base
class Extended extends Base {
                       ^
1 error
```

In the above example program, Base class is a final class and so it cannot be inherited.

1.1.2 Static Class

Static is a Java word that explains how objects are kept in memory. A static object belongs to that class rather than instances of that class. The primary function of the class is to provide blueprints for the inherited classes. A static class has only static members. An object cannot be created for a static class.

```
import java.util.*;

class staticclasses {
        static int s;
        static void met(int x, int y)
        {
                System.out.println(
                    "static method to calculate sum");
                s = x + y;
                System.out.println(x + "+" + y);
        }
        static class MyNestedClass {
                static
                {
                    System.out.println(
                        "static block inside a static class");
                }
                public void disp()
                {
                    int x1;
                    int y1;
                    Scanner sc = new Scanner(System.in);
                    System.out.println("Enter two numbers");
                    x1 = sc.nextInt();
                    y1 = sc.nextInt();
                    met(x1, y1);
                    System.out.println("Sum of the 2 numbers-" + s);
                }
        }
}
public class MainClass {
        public static void main(String args[])
        {
                staticclasses.MyNestedClass nestedclass
                    = new staticclasses.MyNestedClass();
                nestedclass.disp();
        }
}
```

Output:

```
static block inside a static class
Enter two numbers
2
3
static method to calculate sum
2+3
Sum of the 2 numbers-5
```

1.1.3 Abstract Class

A class that has zero or more abstract methods and is specified with the abstract keyword is called an abstract class. We must rigorously extend the abstract classes to a concrete class in order to use them because they are incomplete classes. Constructors and static methods can also be included. It can have final methods, which force the subclass to keep the body of the method unhung.

```
interface X {
        int product(int x, int y);
}
abstract class Product implements X {

        public int product(int x, int y) { return x * y; }
}
public class MainClass extends Product {
        public static void main(String args[])
        {
                MainClass ob = new MainClass();
                int p = ob.product(20, 10);
                System.out.println("Product: " + p);
        }
}
```

Output:

```
Product: 200
```

1.1.4 Concrete Class

A normal class with an implementation for all of its methods and no abstract methods is called a concrete class. They are not permitted to have any processes that are not in use at the time. If it implements all of its methods, a concrete class can extend its parent, an abstract class, or an interface. It’s a fully working, instantiable class.

```
public class MainClas {
	static int sum(int x, int y) { return x + y; }

	public static void main(String args[])
	{
		int p = sum(10, 8);
		System.out.println("Sum: " + p);
	}
}
```

Output:

```
Sum: 18
```

1.1.5 POJO Class

“Plain Old Java Object” is an abbreviation for “Plain Old Java Object.” A POJO class has only private variables with setter and getter methods to access them. It’s a pure data structure with fields that can override some Object methods (e.g. equals) or other interfaces (e.g. serializable), but it has no behavior of its own.

POJO class properties:

- When writing a POJO class, public setter and getter methods are required.
- Private variables should be used for all instance variables.
- It should not extend classes that have already been defined.
- It should not implement interfaces that have been pre-defined.
- There should be no pre-defined annotations.
- It might not have a function Object() { [native code] } that takes no arguments.

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

Output:

```
365
```

1.1.6 Singleton Class

A singleton class is one that has just one object at any one moment. Even yet, if we try to create an instance again, the newly created instance refers to the previous one. Any modification we make to the class through any instance impacts the variables in that specific instance as well. It’s commonly used to manage access while working with database connections and socket programming.

The following is used to make Singleton Class:

Make a function Object() { [native code] } that is only available to you.
Create a static function that returns the singleton class’s object (using lazy initialization).

```
class Singleton {

    private static Singleton single_instance = null;

    public String s;

    private Singleton()
    {
        s = "This is a string part of Singleton class";
    }
    // here a private constructor is used

    // Method
    public static Singleton Singleton()
    {
        if (single_instance == null) {
            single_instance = new Singleton();
```

```
                }
                return single_instance;
        }
}

// Main class
public class MainClass {
        public static void main(String args[])
        {
                Singleton x = Singleton.Singleton();
                Singleton y = Singleton.Singleton();

                // change var of x
                x.s = (x.s).toUpperCase();

                System.out.println("String from x is -->" + x.s);
                System.out.println("String from y is -->" + y.s);
                System.out.println("\n");

                y.s = (y.s).toLowerCase();

                System.out.println("String from x is -->" + x.s);
                System.out.println("String from y is -->" + y.s);
        }
}
```

Output:

```
String from x is -->THIS IS A STRING PART OF SINGLETON CLASS
String from y is -->THIS IS A STRING PART OF SINGLETON CLASS


String from x is -->this is a string part of singleton class
String from y is -->this is a string part of singleton class
```

1.1.7 Inner Class

We can define a class within a class in Java, and these classes are referred to as nested classes. It's used to logically arrange classes and achieve encapsulation. The outer class members (including private) can be accessed by the inner class.

Syntax:

```
import java.io.*;

class OuterClass {
        // Write the code
        class NestedClass {
                // Write the code
        }
}
```

There are 4 types of inner classes:

- Nested Inner class
- Anonymous inner classes
- Static nested classes
- Method Local inner classes

A. Nested Inner Class:

It has access to an outer class's private instance variables. The access modifiers private, protected, public, and default can be applied to any instance variable.

```
class Outer {
        class Inner {
                public void show()
                {
                        System.out.println("Inside a nested class");
                }
        }
}

public class MainClass {
        public static void main(String[] args)
        {
                Outer.Inner in = new Outer().new Inner();
                in.show();
        }
}
```

Output:

```
Inside a nested class
```

B. Anonymous Inner Class:

Basically, these classes are declared without any name.

Example 1: Using Subclass

```
class Outer {
        void show()
        {
                System.out.println("Show method of super class");
        }
}

public class MainClass{

        static Outer o = new Outer() {
                void show()
                {
                        super.show();
                        System.out.println("Demo class");
                }
        };

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

Output:

```
Show method of super class
Demo class
```

Example 2: Using Interface

```
interface INF{
    void show();
}

class SomeClass implements INF{
    public void show(){
        System.out.println("I am show method in SomeClass");
    }
}

public class Main{
    public static void main(String[] arg){
        System.out.println("Hello");
        INF d = new INF(){
            public void show(){
                System.out.println("I am anonymous inner class in Main class");
            }
        };

        d.show();

        SomeClass sc = new SomeClass();
        sc.show();

        System.out.println("Object d: "+d);
        System.out.println("Object sc: "+sc);
    }
}
```

Output:

```
Hello
I am anonymous inner class in Main class
I am show method in SomeClass
Object d: Main$1@251a69d7
Object sc: SomeClass@6b95977
```

C. Static Nested Class:

These classes are like static members of the outer class.

```
public class MainClass {
	static int data = 100;
	static class Inner {

		void msg()
		{
			System.out.println("data is " + data);
		}
	}

	public static void main(String args[])
	{
		GFG.Inner obj = new GFG.Inner();
		obj.msg();
	}
}
```

Output:

```
data is 100
```

D. Method Local inner Class:

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

```
class Outer {
	void outerMethod()
	{
		System.out.println("Outer Method");
		class Inner {
			void innerMethod()
			{
				System.out.println("Inner Method");
			}
		}

		Inner y = new Inner();
		y.innerMethod();
	}
}
public class MainClass {
        public static void main(String[] args)
        {
                Outer x = new Outer();
                x.outerMethod();
        }
}
```

Output:

```
Outer Method
Inner Method
```