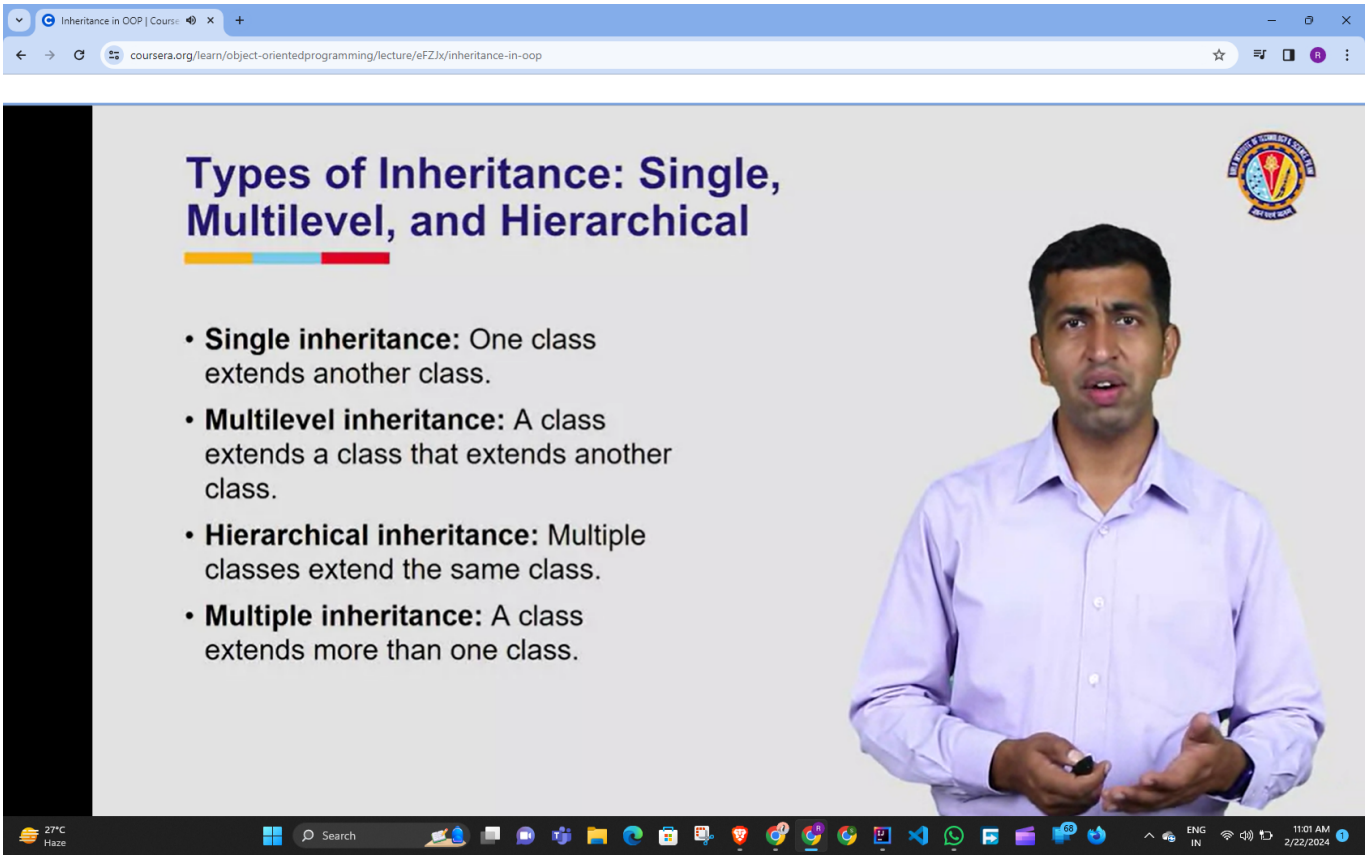
Inheritance

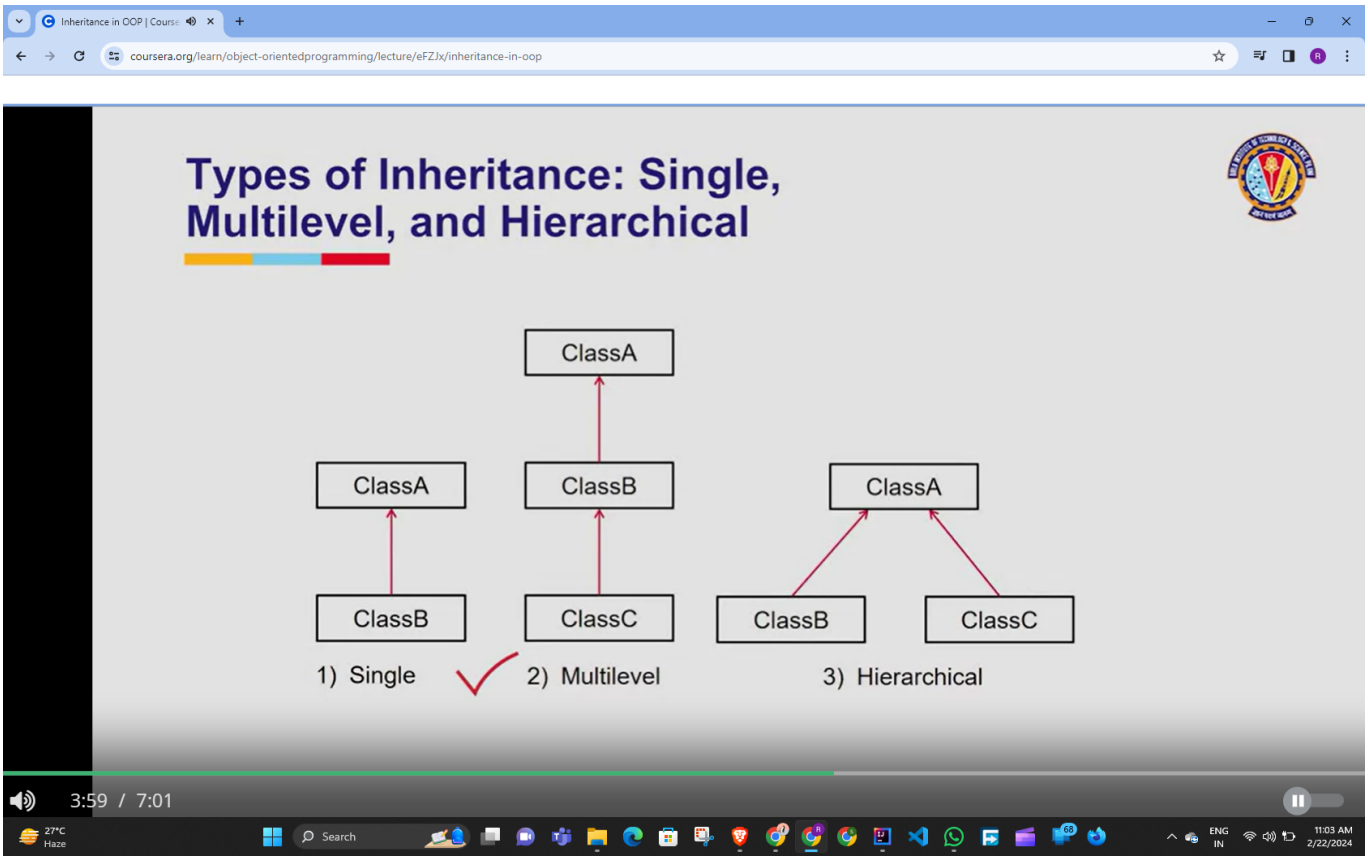
In inhtnce one class inherits methods and feautres from another class

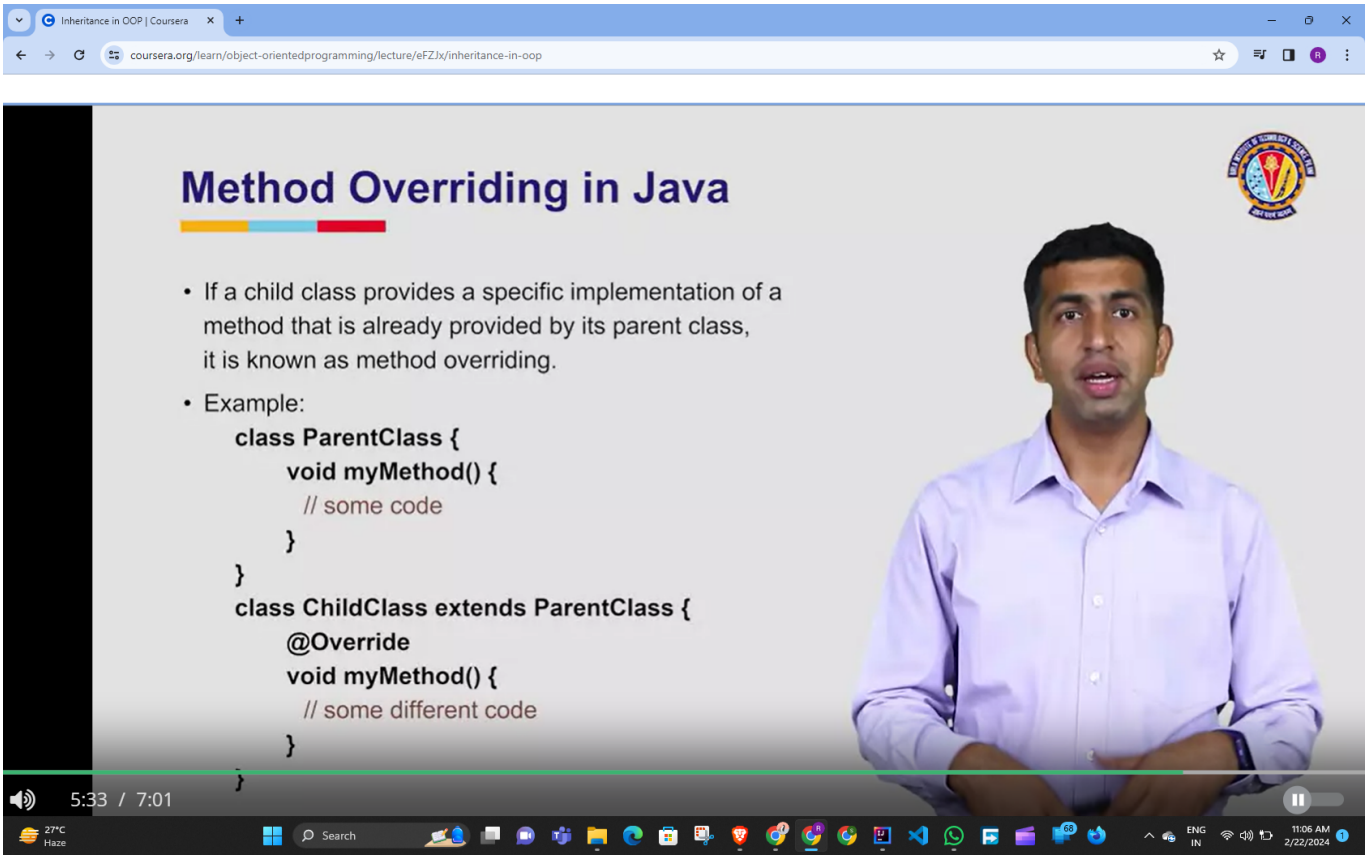
like child inherits feures from parent

in java "extends" is used to denote inheritance

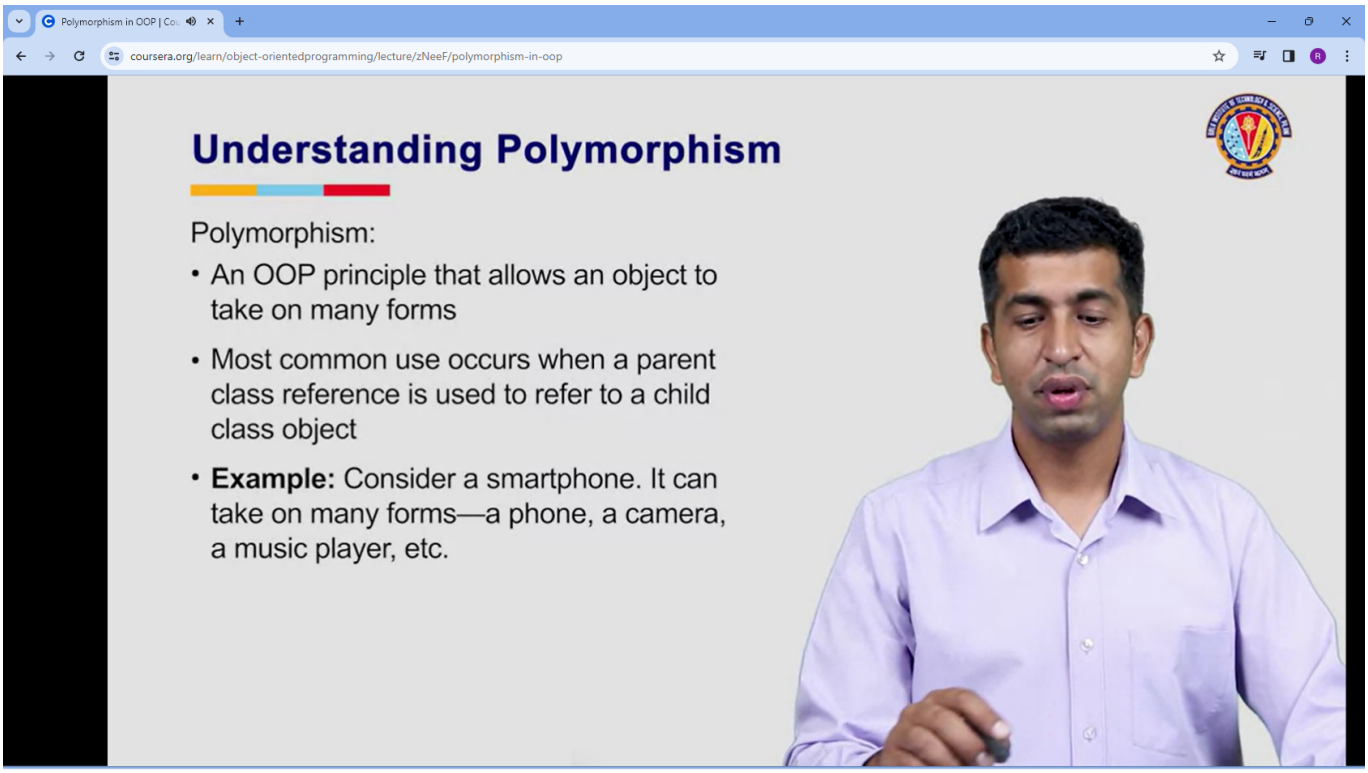
```
class parentclass{
//
}
class childclass extends parentclass{
//
}
```



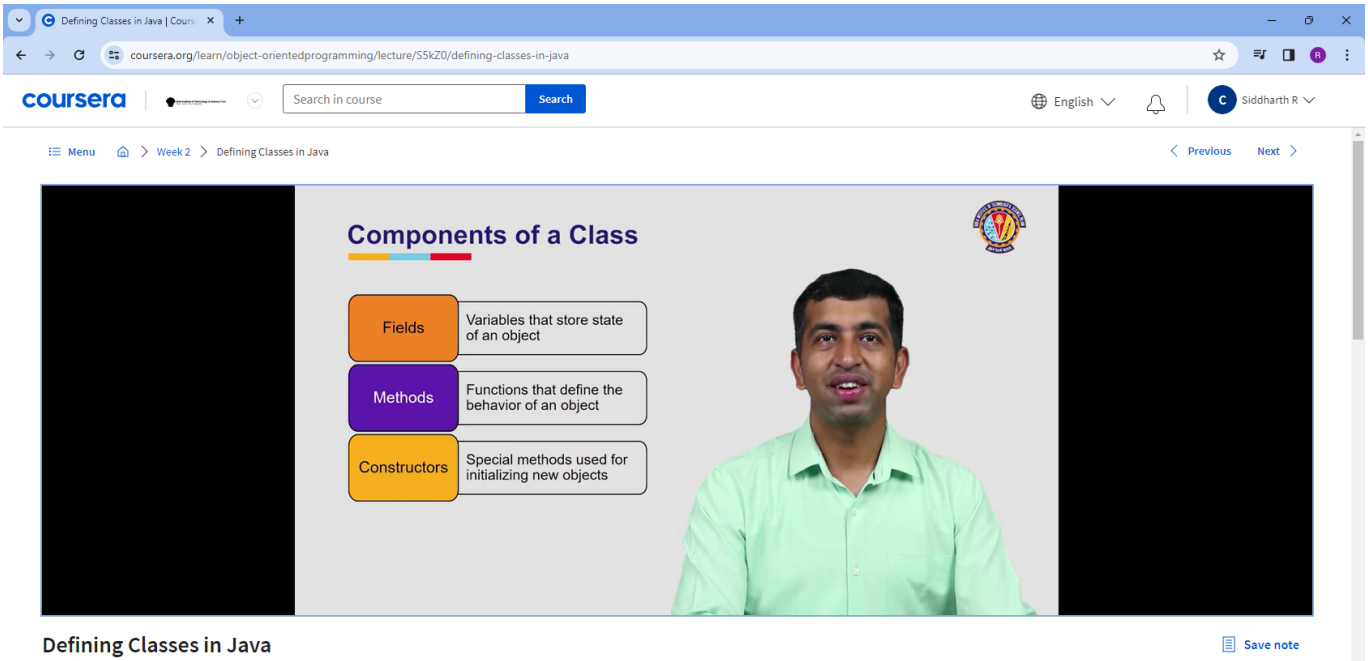




Polymorphism



lymorphism in OOP



Defining Classes in Java

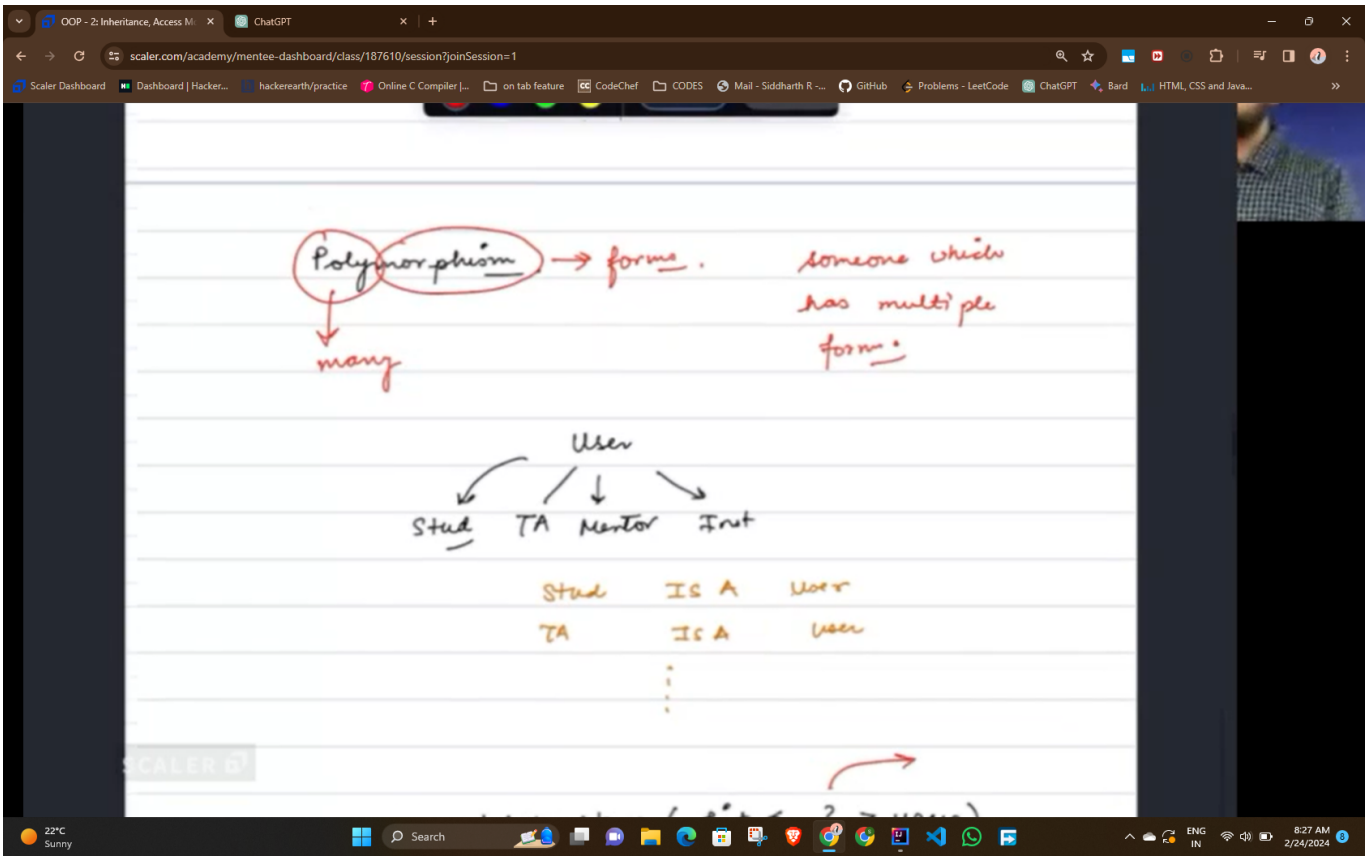
Save note

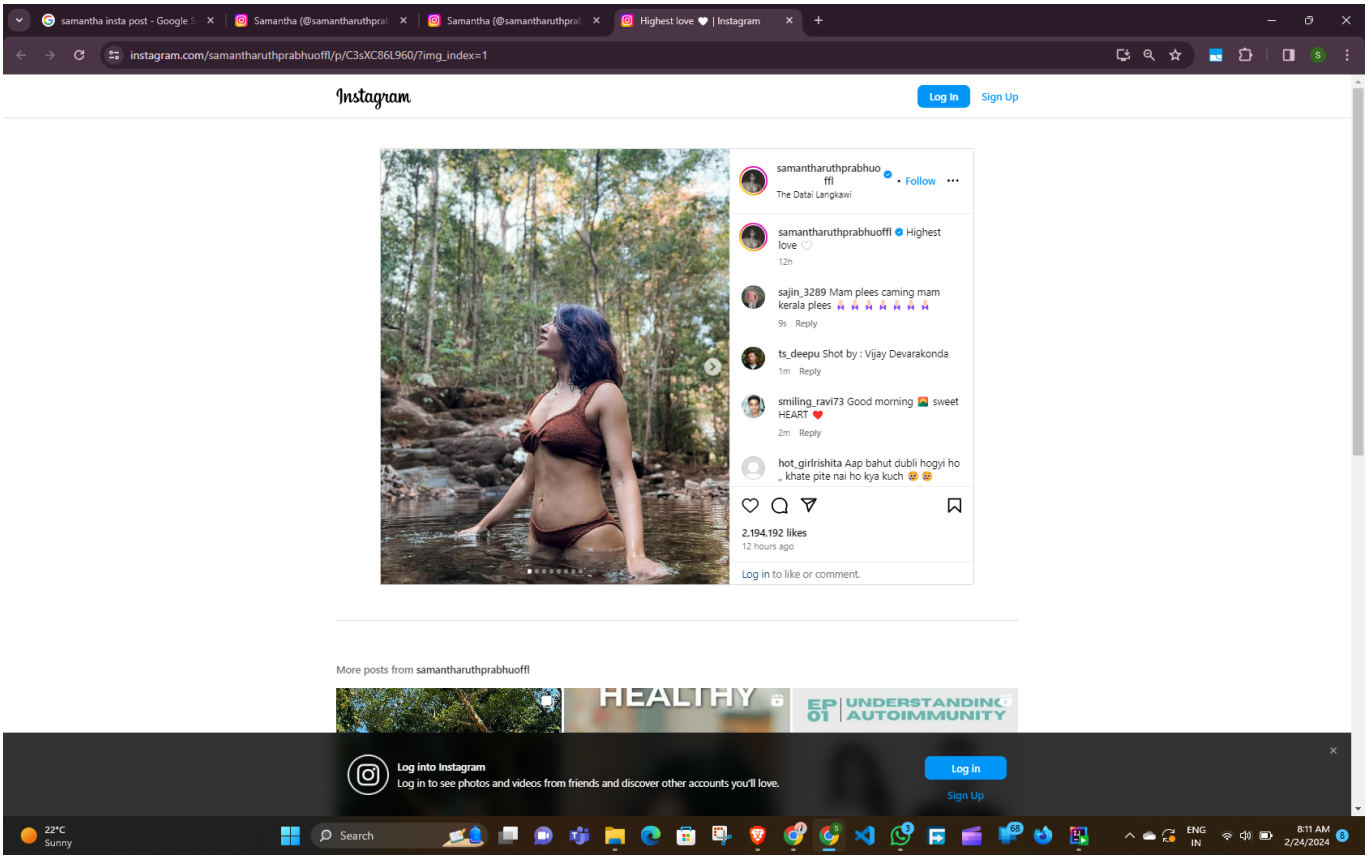
Transcript Notes Downloads Discuss

0:00 [MUSIC] Hello everyone, in today's video, we will discuss about what is a class, why do we need a class, what are the things that are part of a class and how do we create and use a class? Okay, as we all know that classes are very important components of object-related programming. Primarily, class is a blueprint that is used to create an object. Basically, objects are a data

English

Help Us Translate





Copy constraints have different ways

Deep copy & shallow copy

If u r directly copying references , it is shallow copy

If ur creating an object then copying its primitives its deep copy

Copying References in Java:

```
public class CopyReferencesExample {
    public static void main(String[] args) {
        int[] array1 = {1, 2, 3};
        int[] array2 = array1;  // copying reference

        array1[0] = 99;
        System.out.println(Arrays.toString(array2));  // Output: [99, 2, 3]
    }
}


        In this example, array2 is a reference to the same array as array1. So,
modifying array1 also affects array2.
```

2. Copying Primitives in Java:

Java's primitive types (e.g., int, float, char) are copied by value. When you assign the value of one primitive variable to another, changes to one variable do not affect the other.

```
public class CopyPrimitivesExample {
    public static void main(String[] args) {
        int x = 5;
        int y = x;  // copying value

        x = 10;
        System.out.println(y);  // Output: 5
    }
}
```

In this case, changing the value of x does not affect the value of y.

this:

this is a reference variable in Java that refers to the current object. It is often used to differentiate instance variables from local variables when they have the same name. this is also used to invoke the current object's method. It can be used in constructors to call another constructor from the same class (using this()).

```
public class MyClass {
    private int value;

    public MyClass(int value) {
        this.value = value; // 'this' refers to the current object
    }
}
```

super:

super is a keyword in Java that is used to refer to the immediate parent class object. It is often used to invoke the parent class methods, access parent class fields, or invoke the parent class constructor. super() is used to call the constructor of the parent class. This is typically done in the constructor of the child class.

```
public class ChildClass extends ParentClass {
    public ChildClass() {
        super(); // calls the constructor of the parent class
    }
}

        public class ParentClass {
        public void display() {
        System.out.println("This is the parent class.");
        }
        }

        public class ChildClass extends ParentClass {
        public void display() {
        super.display(); // invokes the 'display' method of the parent class
        System.out.println("This is the child class.");
        }
        }
```

In summary, this is used to refer to the current object or to differentiate between instance and local variables, while super is used to refer to the immediate parent class and is often used to access or invoke members of the parent class.

Definition:

Class: A class in Java is a blueprint for creating objects. It defines the properties (fields) and behaviors (methods) that objects instantiated from the class will have. Classes support the concept of encapsulation, inheritance, and polymorphism.

```
public class MyClass {
    private int myField;

    public void setMyField(int value) {
        myField = value;
    }

    public int getMyField() {
        return myField;
    }
}
```

Interface: An interface is a collection of abstract methods (methods without a body) and constant declarations. It defines a contract for classes that implement it, specifying the methods that must be implemented by those classes. Interfaces support multiple inheritance and are used to achieve abstraction and provide a common set of methods for unrelated classes.

```
public interface MyInterface {
    void setMyField(int value);
    int getMyField();
}
```