12. Java Enum, POJO and Final Classes Explained with Example.

POJO Class:

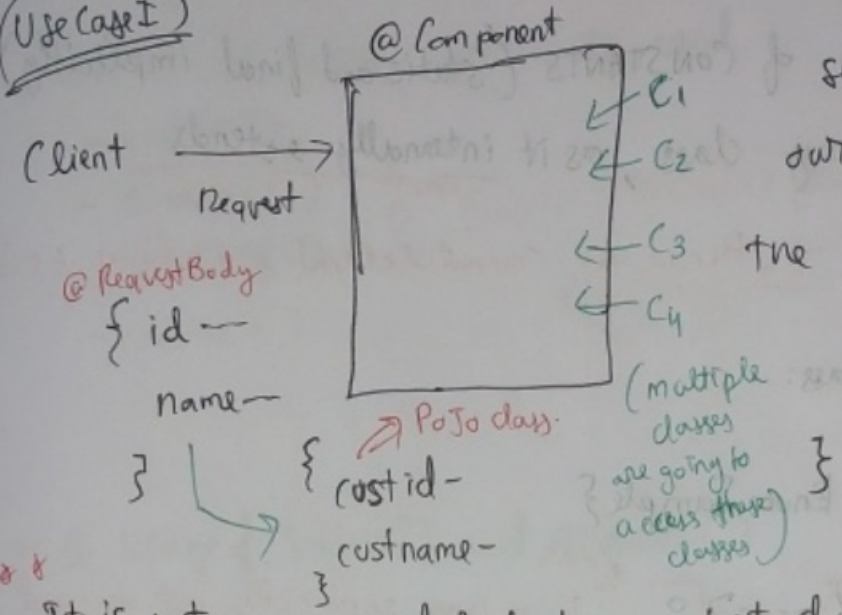
- Stands for "Plain Old Java Object"
- Contains variables and its getter and setter methods
- Class should be public
- Public default constructor
- No annotations should be used like @Table, @Entity, @Id etc
- It should not extend any class or implement any interface.

Example:

```
public class Student {

    int name;
    private int rollNumber;
    protected String address;

    public int getName(){
        return name;
    }

    public void setName(int name){
        this.name = name;
    }

    public int getRoll(){
        return rollNumber;
    }
    public void setRoll(int roll){
        rollNumber = roll;
    }
    public String getAddress(){
        return address;
    }
    public void setAddress(String addr){
        address = addr;
    }
}
```

where do we use POJOs?

(Use Case I)



Lets say Client sends a request to our Api.

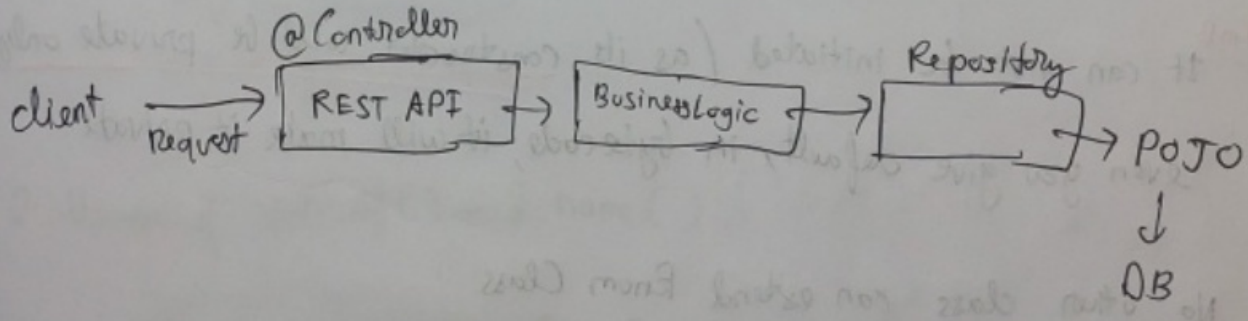
the RequestBody is{

id --

name --

It is not recommended that we start doing changes on the request body itself. Since multiple classes like C_1, C_2, C_3 are going to access the request details, Hence we use a 1-to-1 mapping from RequestBody → POJO class.

(Use Case 2)



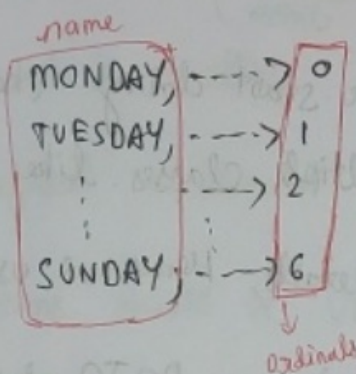
P.T.O.

ENUM class:

- It has a collection of CONSTANTS (static and final implicitly)
- It cannot extend any class, as it internally extends java.lang.Enum → Hence we cannot extend more than 1 class

Normal Enum class:

```
public enum EnumSample{
    MONDAY,
    TUESDAY,
    .
    .
    .
    SUNDAY;
}
```



If we have explicitly not assigned any values to them, then internally they are assigned values [0, 1, 2 ...] - Like Array Indices

- It can implement interfaces, No Restriction on that.
- It can have variables, constructors, methods

- Imp: It can not be initiated (as its constructor will be private only, even you give default, in bytecode, it will make it private
- No other class can extend Enum Class

- It can have abstract method, and all constant should implement that abstract method.

Usage of Enum.

Common Functions which is used
- values()
- ordinal()
- valueOf()
- name()

Static methods present in (java.lang.Enum)

1. Usage of Values() and ordinal()

EnumSample[] return type

```
for (EnumSample sample: EnumSample.values()){
    sout (sample.ordinal());
}
```

static method present inside java.lang.Enum

output
0
1
2
3
4
5
6

2. Usage of ValueOf() and name()

```
EnumSample enumVariable = EnumSample.valueOf("FRIDAY");
sout(enumVariable.name());
```

matches all the constant by converting them into String with this.

Output → FRIDAY

Enum with custom values:

*** V-Imp and Very Useful

```
public enum EnumSample {

    MONDAY(100, "1st day"),
    TUESDAY(101, "2nd day"),
    WEDNESDAY(102, "3rd day"),
        :
    SUNDAY(106, "7th day");

    private int val;
    private String comment;

    EnumSample(int val, String comment){
        this.val = val;
        this.comment = comment;
    }

    public int getVal(){
        return val;
    }

    public String getComment(){
        return comment;
    }

    public static String getCommentForEnum(int val){
        for(EnumSample sample : EnumSample.values()){
            if(sample.getVal() == val) return sample.getComment();
        }
        return "";
    }
```

This is how the constructor is getting used.
It is setting val and comment for each day.

V-Imp Use of the Constructor.

Parameterised Constructor
But this is a private constructor.
What use of it how can we initialize.

(***) Methods that we have defined here are for each constant.

Method Overriding in Enums.

```
public enum EnumSample{

    MONDAY(
        @Override
        public void dummyMethod(){
            Sout("this is Monday method");
        }
    ),
    TUESDAY,
    .
    .
    .
    SUNDAY;

    public void dummyMethod(){
        Sout("this is default method");
    }
}.
```

```
EnumSample sample1 = EnumSample.FRIDAY;
EnumSample sample2 = EnumSample.MONDAY;
~~Sample1.default~~
Sample2.dummyMethod()
sample1.dummyMethod()
```

Abstract Methods in Enums.

[If can have abstract method but all Constants should implement it]

```
public enum EnumSample{

    MONDAY(
        @Override
        public void print(){
            Sout("Monday");
        }
    ),
    FRIDAY(
        @Override
        public void print(){
            Sout("FRIDAY");
        }
    );
    public abstract void print();
}.
```

All child, basically All constants have to implement the abstract method.

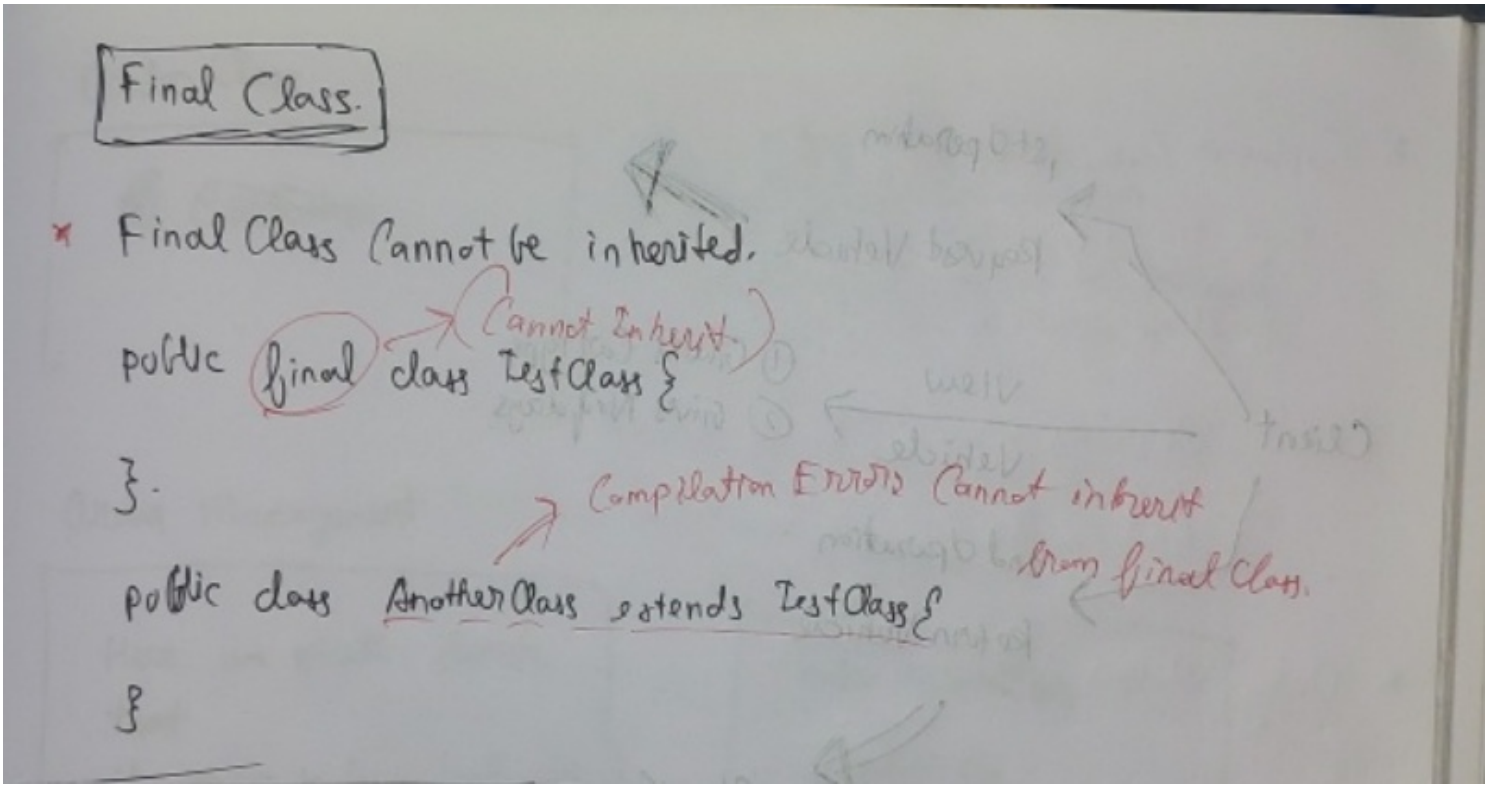
Interface Implementation in Enum

```
interface MyInterface {
    public String addSuffix();
}

public enum EnumSample implements MyInterface {
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    :
    SUNDAY;

    @Override
    public String addSuffix() {
        return this.name + "suffix";
    }
}
```

Applies to all.

Usage.

```
EnumSample mondaySample = EnumSample.MONDAY;
Sout( mondaySample.addSuffix());
```


Final Class.
Final Class Cannot be inherited.
public final class TestClass {
(Cannot Inherit)
}.
public class AnotherClass extends TestClass {
Compilation Error: Cannot inherit from final class.
}