6__17-07-2022

Sunday, 17 July 2022 8:01 PM

Topics to cover -

→ Difference between method overloading and method overriding

→ Static keyword

→ final keyword

→ Design principles

Method overloading	Method overriding
→ It is a compile time polymorphism.	→ It is a run-time polymorphism.
→ It occurs within the class.	→ It is performed in two classes with the help of inheritance.
→ It helps to increase readability of the program.	→ It is used to provide specific implementation to method which already has implemen-

	tation in the parent class.
→ Methods must have same name and different signatures.	→ Here methods must have same name and same signature.
→ Return type can or cannot change.	→ Return type must be same.
→ Private and final methods can be overloaded.	→ Private and final methods cannot be overridden.
→ Argument list should be different.	→ Argument list should be same.

Static Keyword

- → static variable
- → static method
- → static constructor
- → static class

```
public class A
{
    public class B
    {
        =
    }
    public class C
    {
        =
    }
}
```

Nested classes
↓
one class within another class.

Static variable

↳ when a variable is declared as static then a single copy of the variable is created and shared among all objects at the class level.

↳ If you want to access them then do it with the help of class name, they do not require objects for access.

Eg For a non static class (Regular class)

```
Program obj = new Program();
SOP (obj. name) ✓
```

For a static variable

```
SOP (Program. name);
```

Static methods

↳ Accessed with the name of the class.

↳ Can access static and non-static fields.

↳ Static modifier ensures implementation is the same accross all class instances.

Syntax– for a variable

```
public static int a = 5;
```

For a static method.

```
public static void print()
{
    ==
}
```

Static constructor

Constructors ┬ Default constructor

- Parameterized constructor
- Copy constructor
- Private constructor
- Static constructor

```
public static class A
{
    public A()
    {
        =
    }

    static A()
    {
        =
    }
}
```

Note –

If your class has both – default and static constructor then static will get

that default

called first and after [illegible]

will get executed.

But a static constructor is not allowed in java.

Static class

↳ It is not allowed to create object of static class.

```
outer class
{
    static class
    Inner class
}
```

```
Outer Class . Static Class obj = new
        outer class. static Class ();
```

Note –

A static class may be instantiated

without instantiating its outer class.

Inner classes can access both static and non-static members of the outer class.

+

public static void main (String [] args)

public – Access modifier which allows the main method to be accessible everywhere.

static – static Keyword enables us to call this method directly using class name without creating an object of it.

void – No return type.

main – Method name. Entry point for your program.

```
public class OuterClass {
   private static String message = "hello";

   //static nested class
   public static class StaticClass
   {
      public void printStaticClass()
      {
         System.out.println("Static class");
         System.out.println(message);
      }

      public static void printMessage()
      {
         System.out.println("Printing message");
      }
   }

   //non static class
   public class InnerClass
   {
      public void printInnerClass()
      {
         System.out.println("Inner Class");
         System.out.println(message);
      }
   }
}
```

```
public class Program {
   public static void main(String[] args) {
      //creating instance of static class
      OuterClass.StaticClass staticClass = new OuterClass.StaticClass();
      staticClass.printStaticClass();
      OuterClass.StaticClass.printMessage();

      //for a non static class

      //another approach for these two statements
```

```
        /**
         * OuterClass outer = new OuterClass();
         *        OuterClass.InnerClass inner = outer.new InnerClass();
         */

        OuterClass.InnerClass inner = new OuterClass().new InnerClass();

        inner.printInnerClass();
    }
}
```

Output:

```
Static class
hello
Printing message
Inner Class
hello
```

Final Keyword

- → final variable
  - ↳ To create a constant variable
- → final methods
  - ↳ To prevent method overriding
- → Final class
  - ↳ To prevent inheritance.

Design principles

→ Easy

→ Scalable

→ Better customer experience

→ customizable

Different types of design principles—

1. DRY

↳ Do not repeat yourself

↳ This principle states that each small piece of code may only occur exactly once in the entire implementation.

↳ This helps us to provide scalable, maintainable and reusable code.

2. KISS

↳ keep it simple stupid
OR
keep it simple short.

↳ This principle states that try to keep each small piece of code simple and avoid unnecessary complexities.

↳ Always write your code which is easy to debug.

↳ This helps us to write easy maintainable code.

3. YAGNI

↳ You ain't gonna need it

↳ This principle says that always

implement things when you actually need them and never implement things before you need them.

SOLID

S - Single Responsibility Principle (SRP)

O - Open/close Principle (OCP)

L - Liscov Substitution Principle (LSP)

I - Interface Segregation Principle (ISP)

D - Dependency Inversion Principle (DIP)

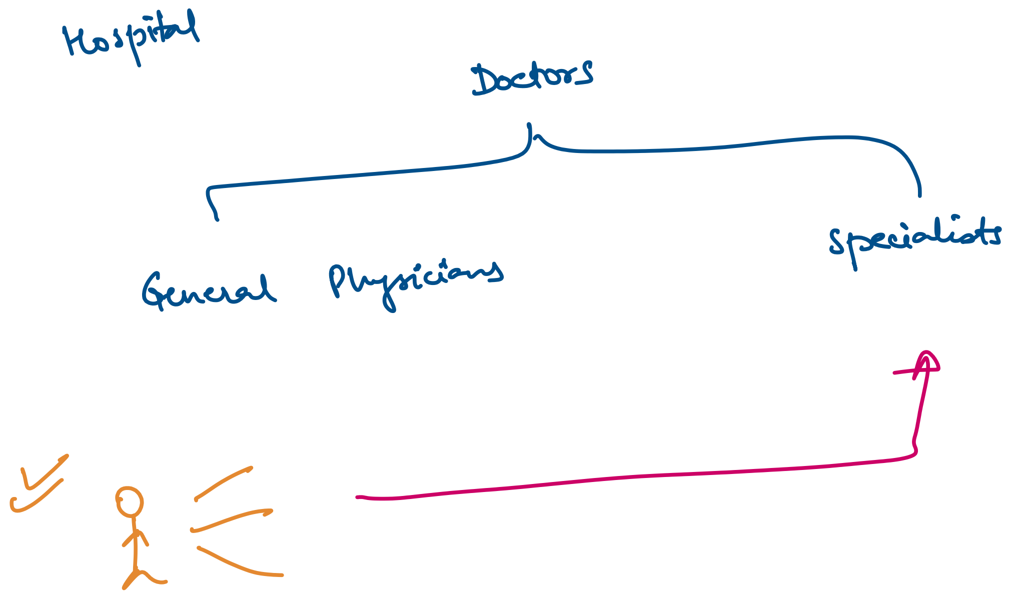
single Responsibility Principle

1 class should do only one thing.

→ To Reduce confusion

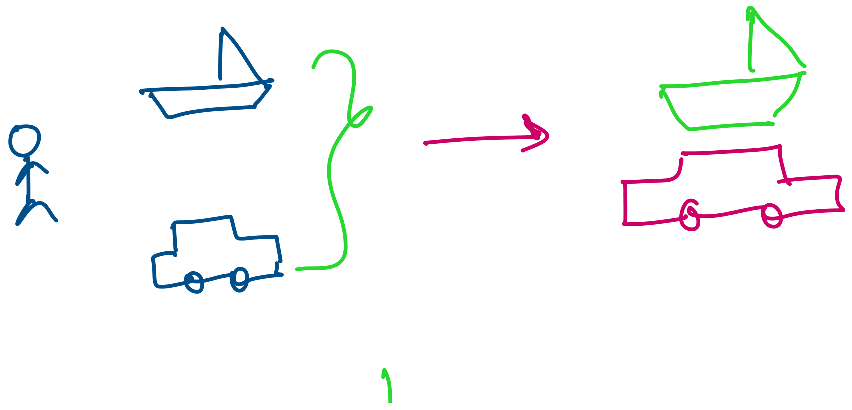
→ To make it loosely coupled

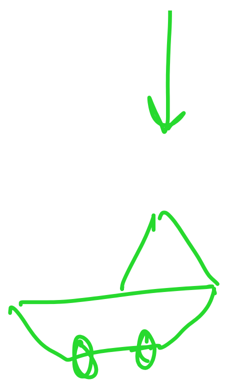
→ To make it more specialized.



S → SRP
↳ one class should be doing only one thing.

Eg -

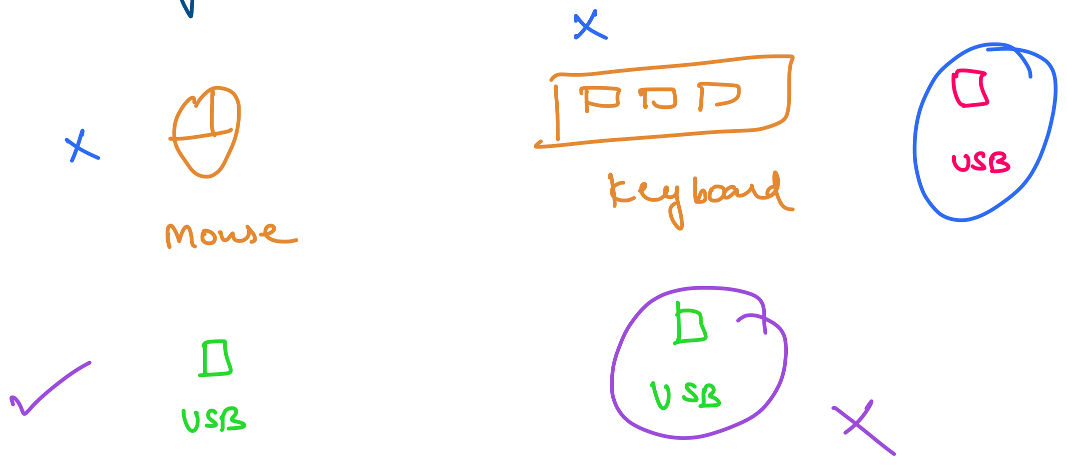




Boat ✗ → Car ✗

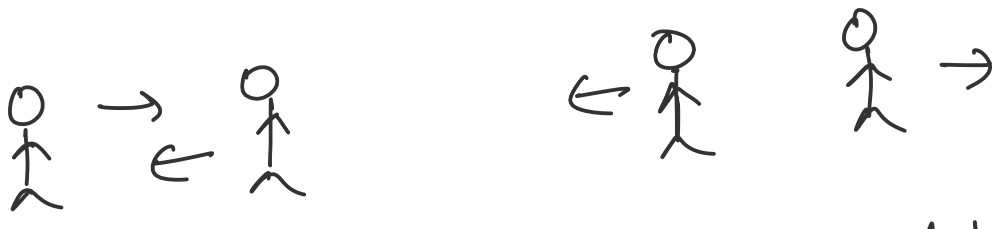
Car ✗ → Boat ✗

Another eg -



Coupling

↳ Relationships



Tightly coupled

loosely coupled

$$\begin{tikzcd}
 & \text{Relationships} \arrow[dl, dash] \arrow[dr, dash] & \\
\text{Good} & & \text{Bad}
\end{tikzcd}$$

$$\underbrace{\text{Good} \qquad\qquad \text{Bad}}_{\text{But it is important}}$$

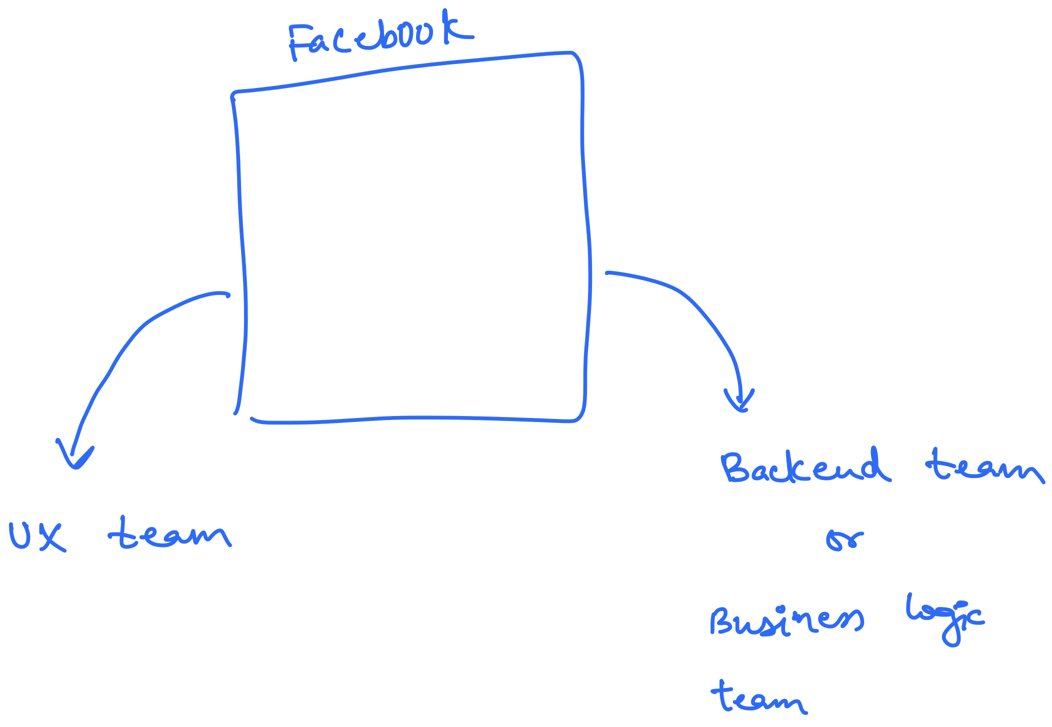
Relationship between the objects and that is also going to be important.

Relationship between objects is called as coupling.

$$\begin{tikzcd}
 & \text{Coupling} \arrow[dl, dash] \arrow[dr, dash] & \\
\text{Tight coupling} & & \text{Loose coupling}
\end{tikzcd}$$

```
class A
{
    B.sleep();
}

class B
{
    A.eat();
}
```


Facebook
UX team
Backend team
or
Business logic
team