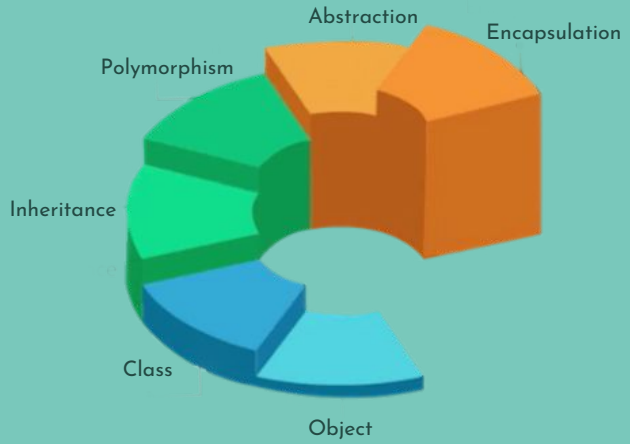
Soft Bodies Group

Design Pattern

Present by Amir Arsalan Yavari

Repeated challenges



- How to make sure we have only one instance of an object?
- How can the rest of the objects be informed by a change in one object?
- How to dynamically execute one of several algorithms?
- Etc ...

CONTENTS OF THIS PRESENTATION

- What is Design Patterns?
- Some types of Design Patterns
  - Bluh
    - Definition
    - Example
  - Bluh
  - Bluh
- Conclusion
- Resources

01

Creational Patterns

design patterns that deal with object creation mechanisms and are used in situations when basic form of object creation could result in design problems or increase complexity of a code base.

02

Structural Patterns

design patterns that ease the design by identifying a simple way to realise relationships between entities or defines a manner for creating relationships between objects.

03

Behavioral Patterns

design patterns that ease the design by identifying a simple way to realise relationships between entities or defines a manner for creating relationships between objects.

Classification of Patterns

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adaptor	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adaptor bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

These are some patterns. Not all of them

Some example

- Singleton
- Strategy
- Observer
- Decorator

Singleton Pattern

It is Creational and related to Objects...

Example: network manager

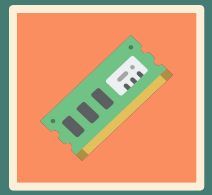
challenge!

We want to make sure we have only one instance of an object?

Using static?

Losing referencing the object

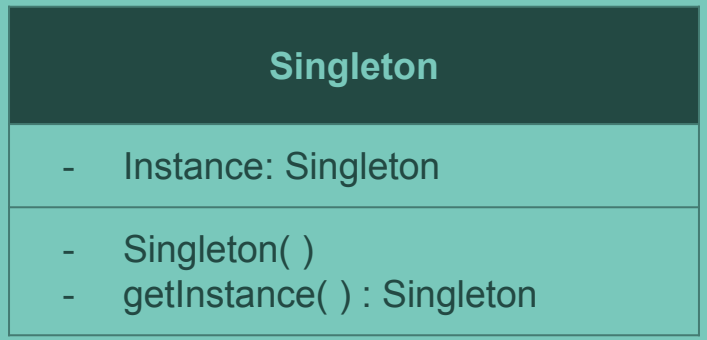
Losing inheritance



Stored in stack instead of heap

Therefore, it is recommended to store object inside of the class itself

Class Diagram



Singleton Patterns

```
public class Singleton
{
  private static Singleton instance;

  private Singleton()
  {
    // private constructor
  }

  synchronized public static Singleton getInstance()
  {
    if (instance == null)
    {
        instance = new Singleton();
    }
    return instance;
  }
}
```

Some example

- Singleton
- Strategy
- Observer
- Decorator

Strategy Pattern

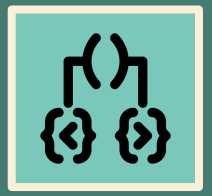
It is Behavioral and
related to Objects...

Example: Layoutmanager
in java
Multiplatform Application

challenge!

How to to change object behavior in run-time?

Using Consecutive conditions?



Losing OOP principles

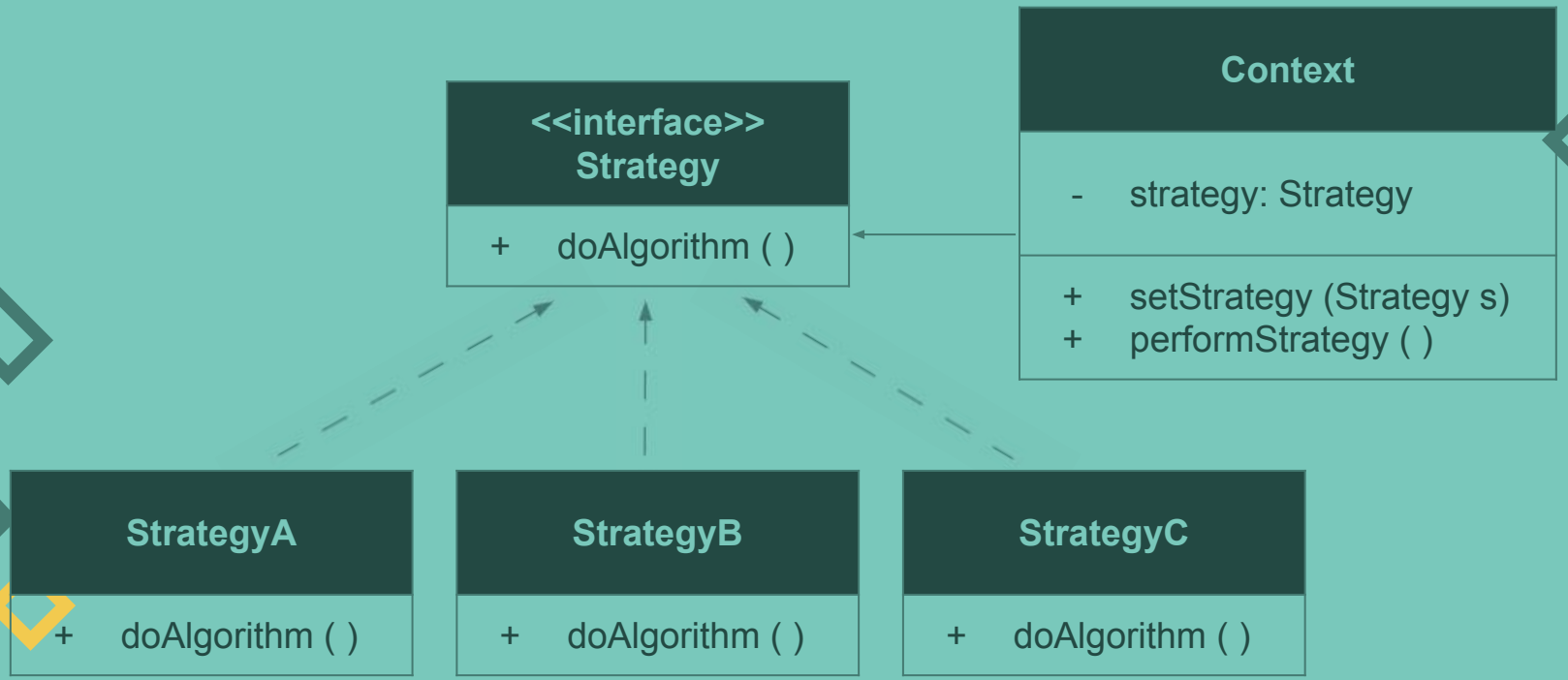
Using Inheritance or Interface?

Can not changing in run-time

Therefore, it is recommended to HAS-A relation instead of IS-A

(Exactly using Composition instead of Interface)

Class Diagram



Strategy Patterns

```java
public interface Strategy {
    public void doAlgorithm();
}

public class StrategyA implements Strategy {
    public void doAlgorithm() {
        // A strategy impelementation
    }
}

public class StrategyB implements Strategy {
    public void doAlgorithm() {
        // B strategy impelementation
    }
}

public class StrategyC implements Strategy {
    public void doAlgorithm() {
        // C strategy impelementation
    }
}

public abstract class Object {
    Strategy strategy;

    public Object() {}

    public void setStrategy(Strategy behavior) {
        strategy = behavior;
    }

    public void performStrategy() {
        strategy.doAlgorithm();
    }
}
```

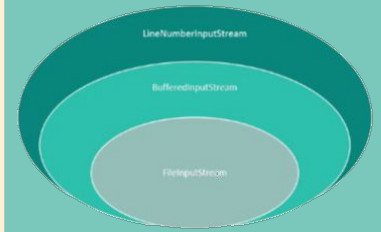
Some example

- Singleton
- Strategy
- Observer
- Decorator

Observer Pattern

It is Behavioral and
related to Objects...

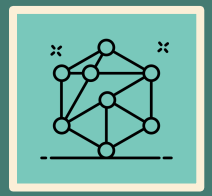
Example: using in java I/O
Expanding elements in UI



challenge!

By changing one of the objects, other objects be informed of that change

Storing Reference of an Object

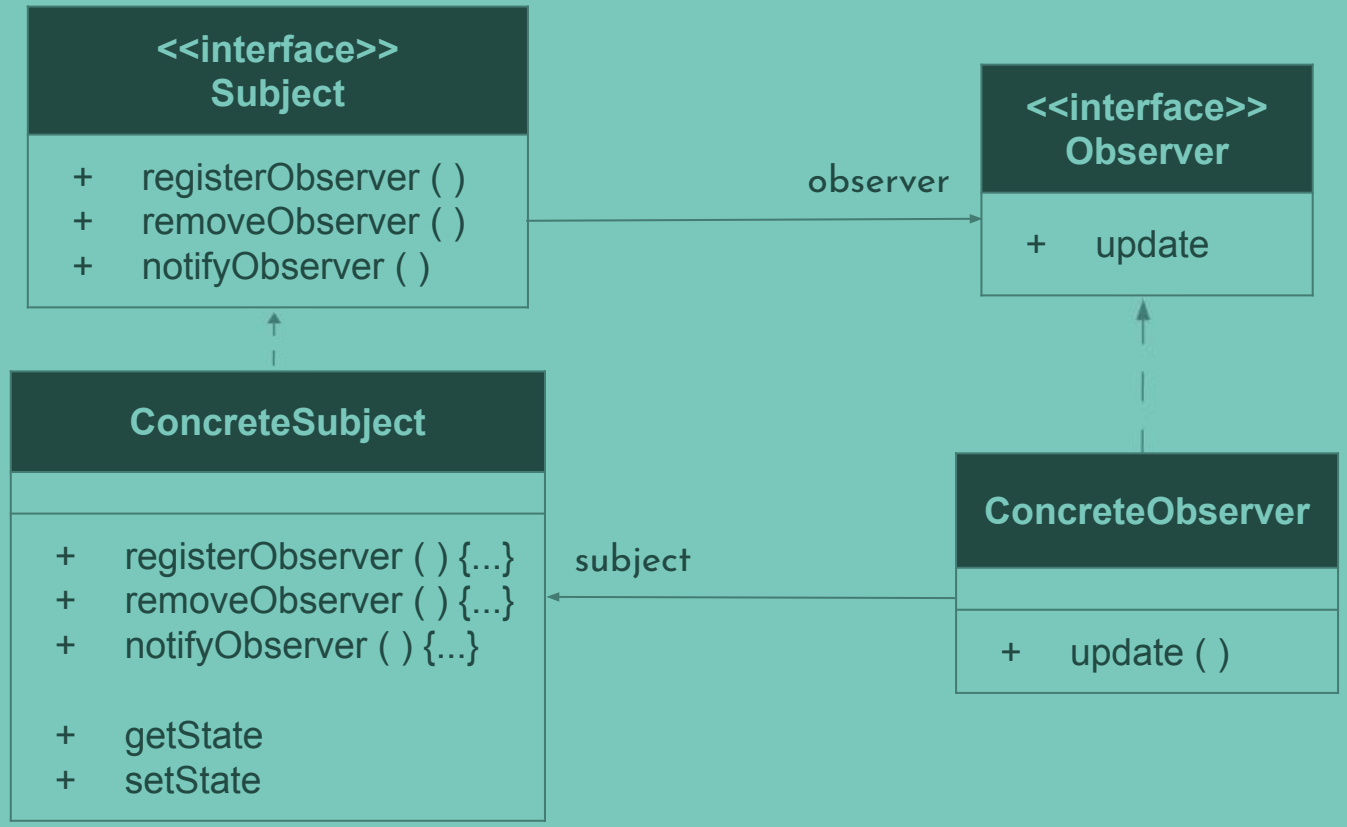


Make objects tightly coupled

NOTE: Tightly coupled mean There is high convectivity and dependency between objects

Therefore, it is recommended to registering if an event is important for object...

Class Diagram



Observer Patterns

```java
public abstract class Observer {
   protected Subject subject;
   public abstract void update();
}
```

```java
import java.util.ArrayList;
import java.util.List;

public class Subject {

   private List<Observer> observers = new
ArrayList<Observer>();
   private int args;

    public void registerObserver(Observer o) {
        observers.add(o);
    }

    public void removeObserver(Observver o) {
        int i = observers.indexOf(o);
        if (i >= 0)
            observers.remove(i);
    }

    public void notifyObservers() {
        for(Observer observer : observers) {
            observers.update(arg);
        }
    }

    public void setParams(int arg) {
        this.arg = arg;
        this.notifyObservers();
    }
}
```

Some example

- Singleton
- Strategy
- Observer
- Decorator

Decorator Pattern

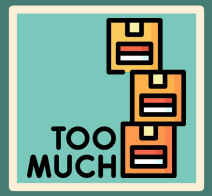
It is Structural and related
to Objects...

Example: data binding in
UI like reactjs & vuejs
Events in C# ...

challenge!

Dynamically adding more features to the objects

Is the Inheritance enough?!



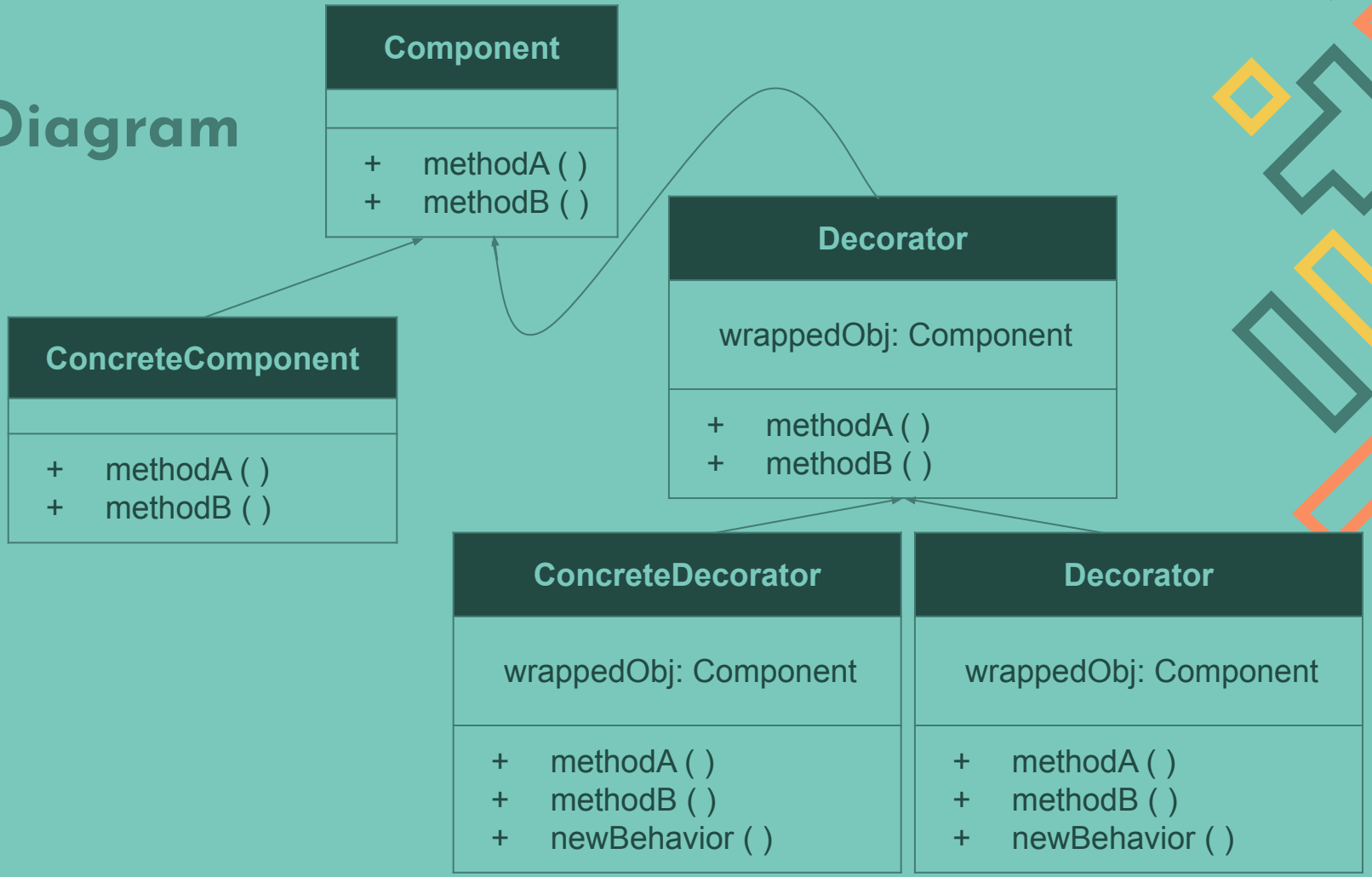
Many classes that confuse us

Using attribute to save states

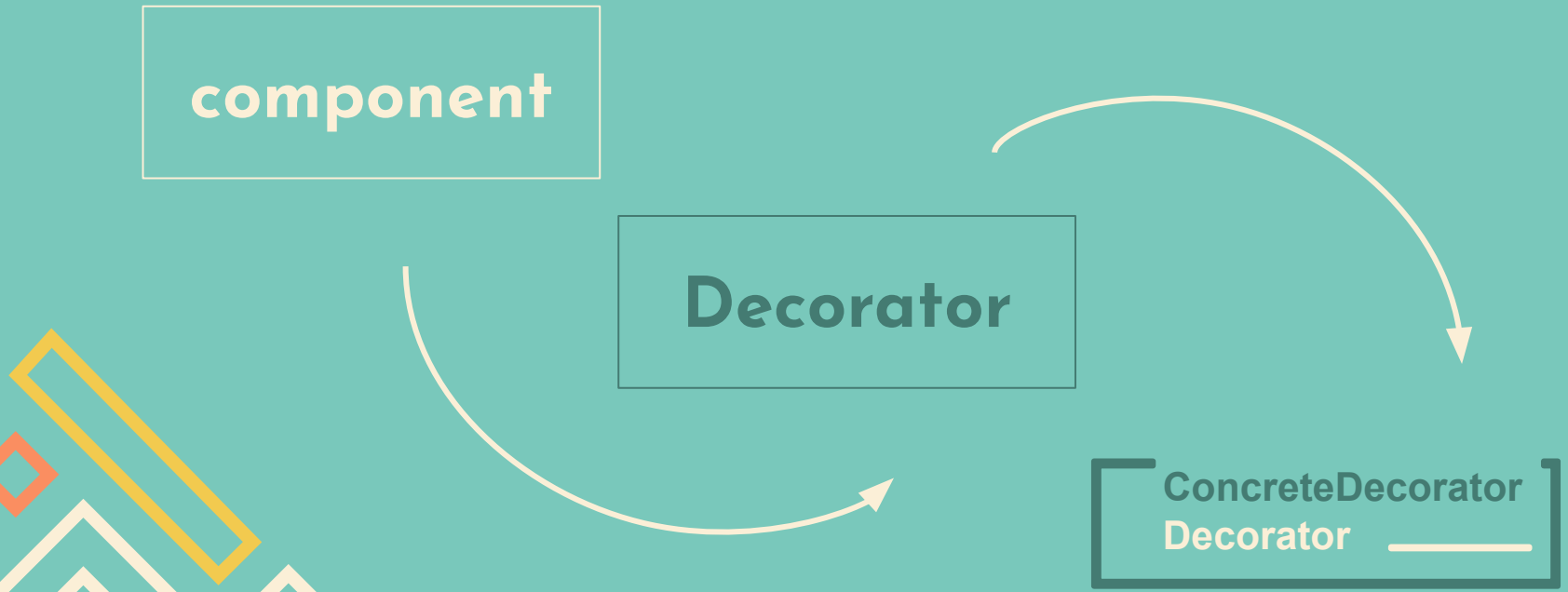
Have no control for preventing acquiring state if another one happened

Therefore, it is recommended to use Decorator. Using composition and HAS-A

Class Diagram



OUR CREATIVE PROCESS



Decorator Patterns

```java
public interface Component {
    void methodA();
    void methodB();
}

public class ConcreteComponent implements Component {

    @Override public void methodA() {
        // put your function body :D
    }
    @Override public void methodB() {
        // put your function body :D
    }
}

public abstract class Decorator implements Component {
    protected Component component;

    public component(Component c) {
        this.component = c;
    }

    public void methodA() { component.methodA(); }
    public void newBehavior() {
        // blob blob blob
    }
}
```

THANKS!

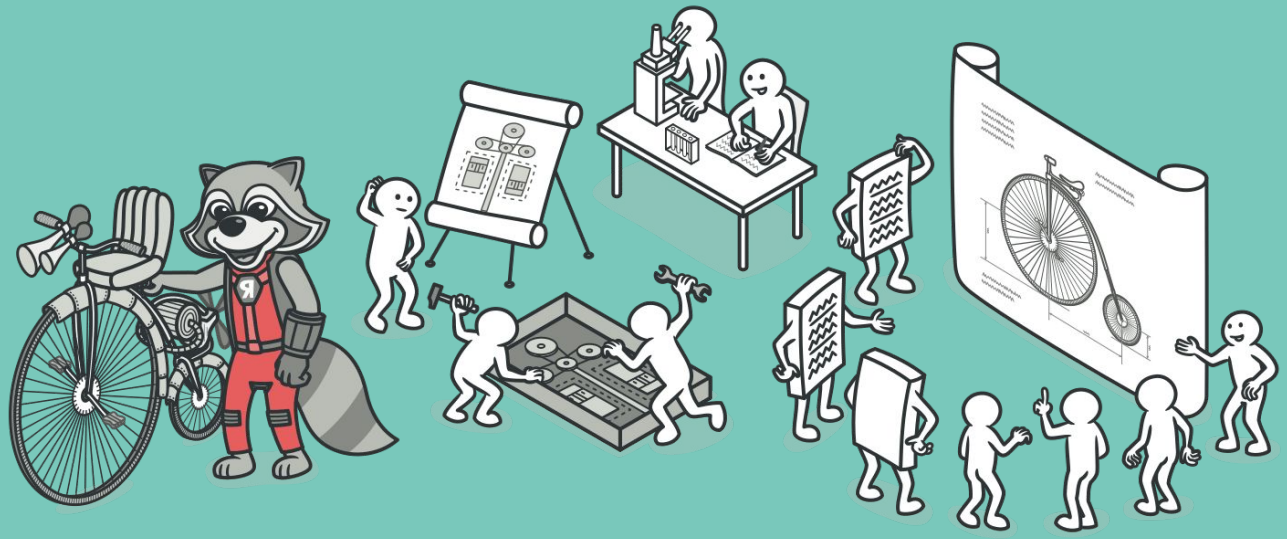
Do you have any questions?

amirarsalan.yavari@ec.iut.ac.ir
+98 914 0127 683

Please keep this slide for attribution.

Learn More: you can learn more in the https://refactoring.guru/design-patterns

Also design patterns handbook description and implementation of the most of them exist in https://www.geeksforgeeks.org/software-design-patterns/



ALTERNATIVE RESOURCES

- https://refactoring.guru/design-patterns
- https://www.digitalocean.com/community/tutorials/gangs-of-four-gof-design-patterns
-