



JÖNKÖPING UNIVERSITY

School of Engineering

ANDROID DESIGN PATTERNS

Peter Larsson-Green

Jönköping University

Spring 2020

DESIGN PATTERNS

What is a design pattern?

- A solution to a general design problem.
- Compare it with computational problems:
 - A computational problem has well defined input.
 - A computational problem has well defined output.
 - The algorithm solves the problem (maps input to output).
- Example:
 - How to support multiple languages?



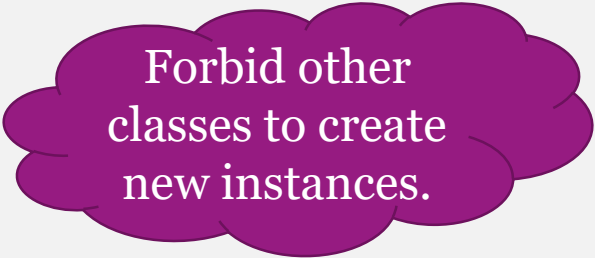
E.g. `sum (3)` → 6.

THE SINGLETON PATTERN

We only want to allow one instance of a class to be created.

- The instance should be easy to retrieve.
- The instance should be created the first time we retrieve it.

```
public class MySingletonClass{  
    private static MySingletonClass instance;  
    public static MySingletonClass getInstance() {  
        if(instance == null){ instance = new MySingletonClass(); }  
        return instance;  
    }  
    private MySingletonClass() {}  
}
```



Forbid other
classes to create
new instances.

THE ADAPTER PATTERN

We want a solution that works with different data structures.

- Bad example:

```
public class Summerizer{  
    public static int getSum(int[] numbers) {  
        int sum = 0;  
        for(int i=0; i<numbers.length; i++) {  
            sum += numbers[i];  
        }  
        return sum;  
    }  
}
```

List with numbers?

- Create temporary array?
 - Duplicates the data and takes time 😞
- Create similar method for lists?
 - Duplicates the code 😞

THE ADAPTER PATTERN

```
public class Summerizer{  
    public static int getSum(Adapter numbers){  
        int sum = 0;  
        for(int i=0; i<numbers.getLength(); i++){  
            sum += numbers.getNumber(i);  
        }  
        return sum;  
    }  
}
```

```
public interface Adapter{  
    int getLength();  
    int getNumber(int index);  
}
```

THE ADAPTER PATTERN

```
public class ArrayAdapter implements Adapter{  
    private int[] numbers;  
    public ArrayAdapter(int[] numbers) {  
        this.numbers = numbers;  
    }  
    public int getLength() {  
        return numbers.length;  
    }  
    public int getNumber(int index) {  
        return numbers[index];  
    }  
}
```

```
public interface Adapter{  
    int getLength();  
    int getNumber(int index);  
}
```

```
int[] numbers = {1, 2, 3};  
int sum = Summerizer.getSum(  
    new ArrayAdapter(numbers)  
);
```

THE ADAPTER PATTERN

```
public class ListAdapter implements Adapter{
    private List<Integer> numbers;
    public ListAdapter(List<Integer> numbers) {
        this.numbers = numbers;
    }
    public int getLength() {
        return numbers.size();
    }
    public int getNumber(int index) {
        return numbers.get(index);
    }
}
```

```
public interface Adapter{
    int getLength();
    int getNumber(int index);
}

List<Integer> numbers =
    new ArrayList<>();
numbers.add(1);
numbers.add(2);
numbers.add(3);
int sum = Summerizer.getSum(
    new ListAdapter(numbers)
);
```


THE FLUENT INTERFACE PATTERN

Simplify calling multiple methods on the same object.

- Bad example:

```
public class Dog{  
    private int age;  
    private String name;  
    public void setAge(int age) { this.age=age; }  
    public void setName(String name) { this.name=name; }  
}
```

```
Dog dog = DogManager.getDogById(1);  
dog.setAge(10);  
dog.setName("Doggy");
```

THE FLUENT INTERFACE PATTERN

Simplify calling multiple methods on the same object.

- Good example:

```
public class Dog{  
    private int age;  
    private String name;  
    public Dog setAge(int age) { this.age=age; return this; }  
    public Dog setName(String name) { this.name=name; return this; }  
}
```

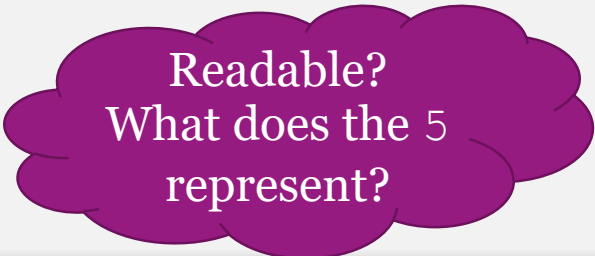
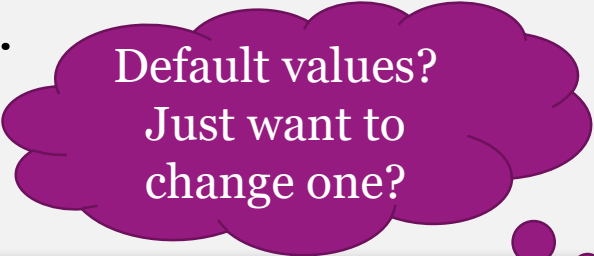
```
DogManager.getDogById(1)  
    .setAge(10)  
    .setName("Doggy");
```

THE BUILDER PATTERN

Simplify creation of classes with a lot of customization.

- Bad example:

```
public class Game{  
    public Game(int numberOfBirdEnemies, int numberOfCatEnemies,  
                int numberOfLives, ...) {  
        this.numberOfBirdEnemies = numberOfBirdEnemies;  
        this.numberOfCatEnemies = numberOfCatEnemies;  
        this.numberOfLives = numberOfLives;  
        ...  
    }  
}
```



```
Game game = new Game(5, 2, 3, ...);
```

THE BUILDER PATTERN

```
public class Game{  
    public Game(int numberOfBirdEnemies, int numberOfCatEnemies,  
                int numberOfLives, ...) { ... }  
  
    public static class Builder{  
        private int birds = 5;  
        private int cats = 2;  
        private int lives = 3;  
        public Builder setNumberOfLives(int n) { lives=n; return this; }  
        public Game build() { return new Game(birds, cats, lives); }  
    }  
}
```

```
Game game = new Game.Builder().setNumberOfLives(5).build();
```