STATE PATTERN

By the Group of Four

State in a Day Life

- Think of it a little bit, many things in our life have states.
  - Elevator : even an elevator has states
    - Push up or down button
    - Waiting for the elevator to arrive
    - Door open
    - Etc.

JAVA in real-life

- Many machines or scenarios in your life can also be implement with Programming too.
- Such as
  - The Elevator
  - Bank Account
  - Etc.

JAVA in real-life (cont.)

- And many those stuff also have states
- How can we implement the state of thing in Object Oriented Design with quality?
- There are many ways to do it
- But the best approch is the State Pattern!!!

Introduction

- State Pattern is one of the 23 Patterns of Design Patterns by the Gang of Four
- State Pattern is a Behavioral Pattern(we will tell you later why).
- We, the Group of Four are the subclass of them and going to be their leaf to explain and illustrate the State Patterns to you.

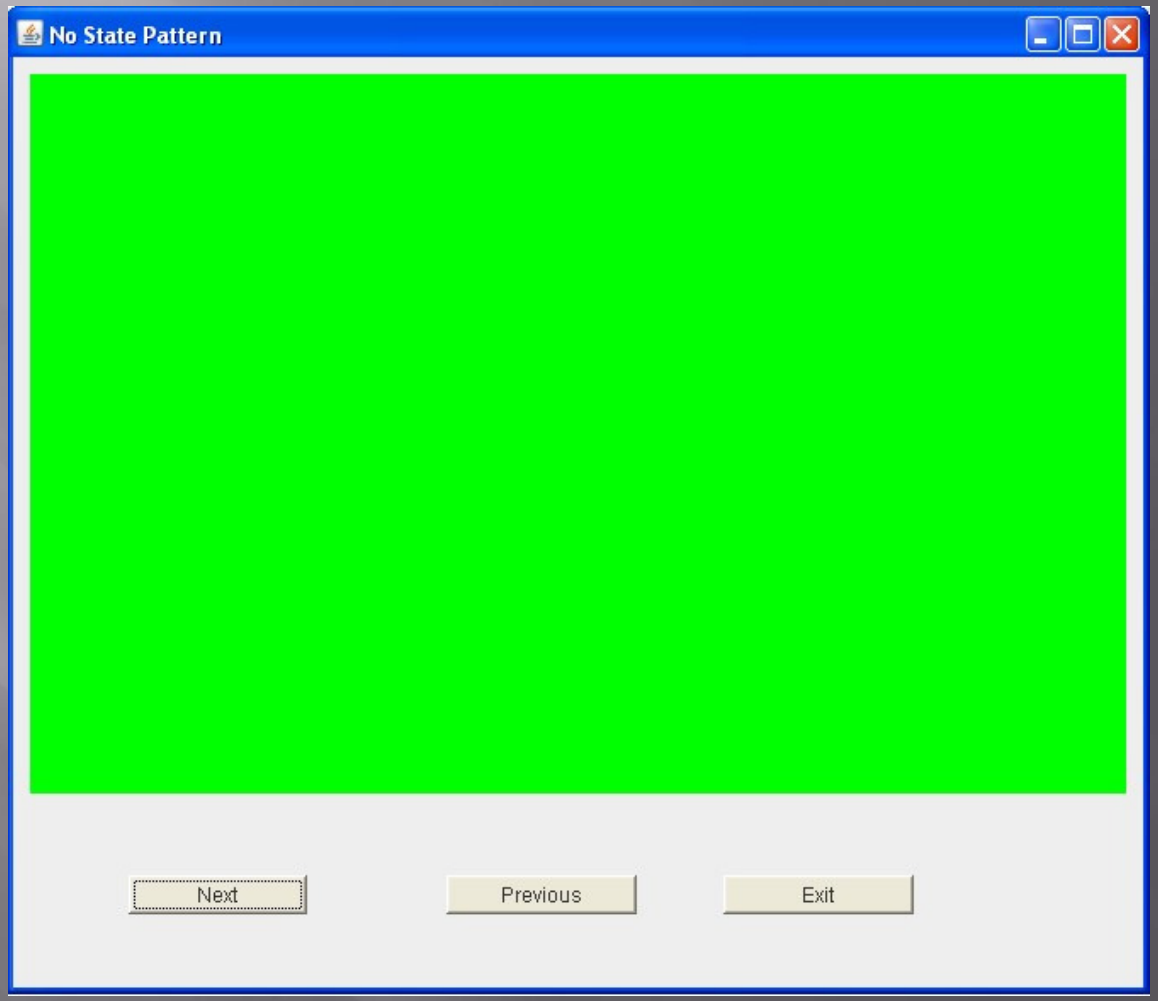
Let's try some example

- As we told you before that object in real-life also has state but to illustrate we would like to show you an easy test with the Graphic user interface.
- Why? To give you the idea of how the state change in the GUI.

Example

- Consider a class with next() and previous() whose behavior changes depend on the state of the object.
- The next() operation will change the state of the canvas object to the next color.
- The previous() operation will change the state of the canvas object to the previous color

This is what the UI look like



Example(cont.)

- Here is how the color state cycle.

- Now you can implement this class.

Try it without State Pattern

- To implement this without a state pattern is very easy.
- You are giving part of a dummy context with part of a GUI.
- In the GUI you only need to implement only the action call nextState() and previousState.

Here is one of the solution

```
public void next() {
    if (state == Color.red) {
        state = Color.blue;
    } else if (state == Color.blue) {
        state = Color.black;
    } else if (state == Color.black) {
        state = Color.green;
    } else if (state == Color.green) {
        state = Color.red;
    }
}
```

```
public void previous() {
    if (state == Color.red) {
        state = Color.green;
    } else if (state == Color.green) {
        state = Color.black;
    } else if (state == Color.black) {
        state = Color.blue;
    } else if (state == Color.blue) {
        state = Color.red;
    }
}
```

```
    private void nextState(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:

        context.next();
        canvas1.setBackground(context.getState());

}

    private void previousState(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:

        context.previous();
        canvas1.setBackground(context.getState());

}
```

What is the problem?

- There is no problem with it if it only has 4 state and a tiny size of code.
- What if
  - The code to implement one state are huge
  - Let just say that some may have 20 state
  - Man!! You ‘re probably die from coding and managing the change request.

Here is where State pattern come in

- Now I would like to introduce you to state pattern.
- Using this pattern will solve this problem for you.

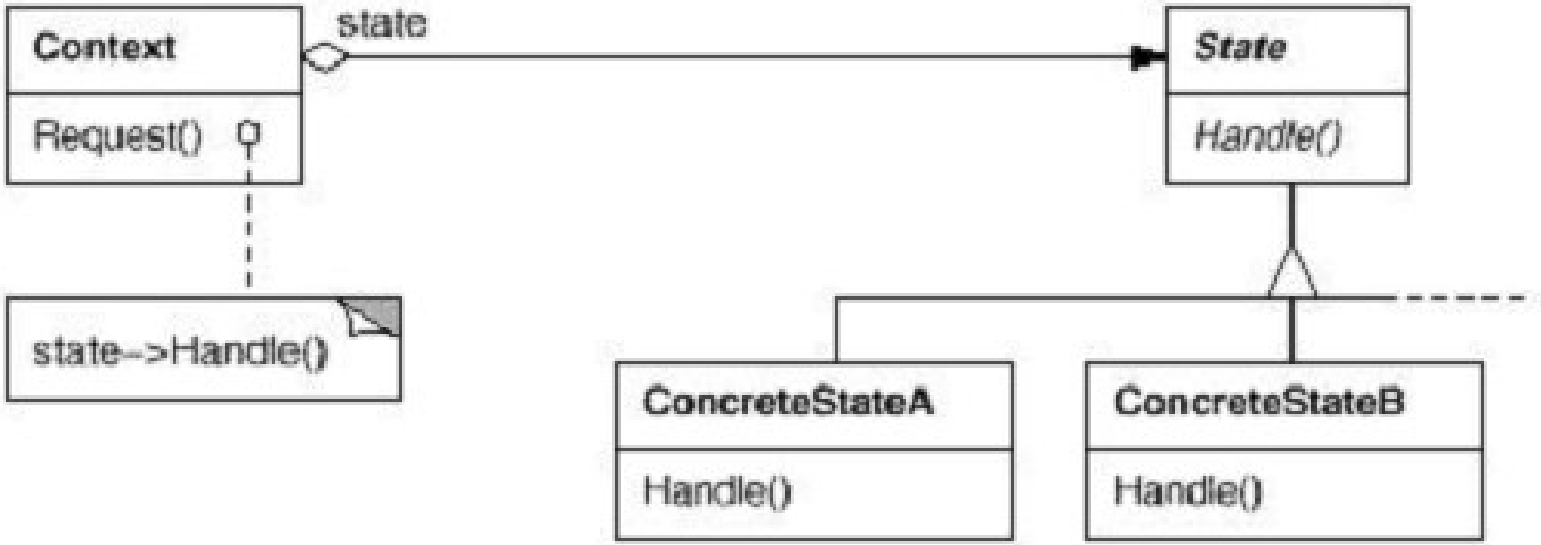
State pattern

- Intent
  - Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
- That ‘s why it’s a behavioral pattern because it allow object to alter its behavior
- Let’s see how.

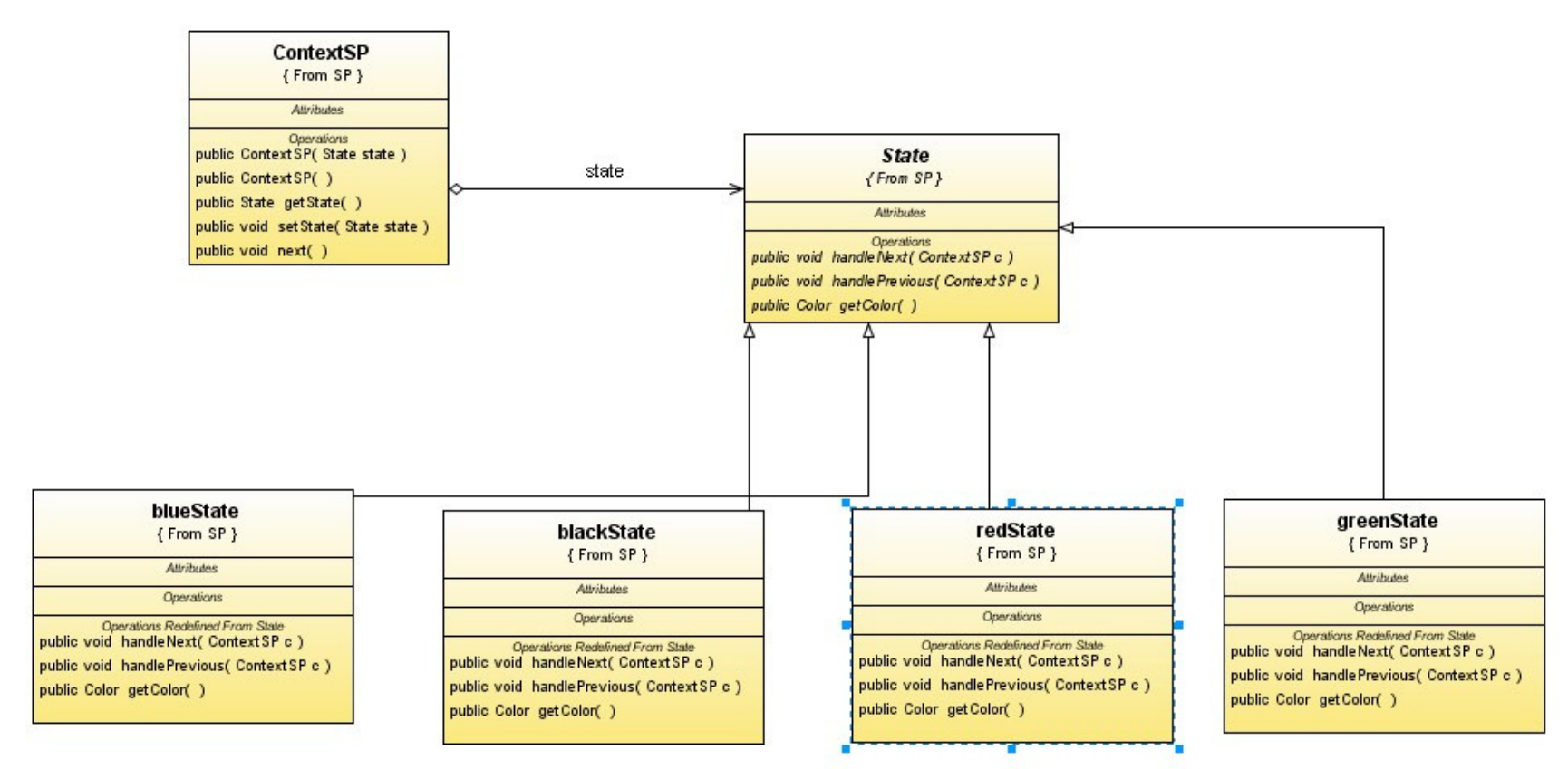
Structure

- First please try to create the UML class diagram for the previous example with state pattern.
- Try it your way then your can come up and show it to the entire class.
- The structure is given in the next slide.

The Structure of state pattern



Class Diagram



Implemantation

- Here come the exercise2
  - You are giving the part of Context , GUI , concreteState class.
  - The instruction is on the top
  - Go implement it.

Solution

```
public class ContextSP {
  // The contained state.
  private State state = null;  // State attribute
  // Creates a new Context with the specified state.
  public ContextSP(State state) {this.state = state;}

  // Creates a new Context with the default state.
   public ContextSP() {this(new redState());}
   // Returns the state.
   public State getState() {return state;}
   // Sets the state.
   public void setState(State state) {this.state = state;}

   /**
    * The push() method performs different actions depending
    *   on the state of the object.  Using the State pattern,
    *   we delegate this behavior to our contained state object.
    */
   public void next() {state.handleNext(this);}
   /**
    * The pull() method performs different actions depending
    *  on the state of the object.  Using the State pattern,
    *  we delegate this behavior to our contained state object.
    */
   public void previous() {state.handlePrevious(this);}
}
```

```
    private void nextState(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:

        context.next();
        this.canvas1.setBackground(context.getState().getColor());

    }

    private void previousState(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:

        context.previous();
        this.canvas1.setBackground(context.getState().getColor());

    }
```

```
public class blackState extends State {
      // Next state for the Black state:
      //    On a push(), go to "red"
      //    On a pull(), go to "green"
      public void handleNext(ContextSP c) {
        c.setState(new greenState());
      }
      public void handlePrevious(ContextSP c) {
        c.setState(new blueState());
      }
      public Color getColor() {return (Color.black);}
   }
```

When to use State Pattern

- An object's behavior depends on its state, and it must change its behavior at run-time depending on that state.
- Operations have large, multipart conditional statements that depend on the object's state. The State pattern puts each branch of the conditional in a separate class.

Advantages of State pattern

- Localized the behavior of each state into its own class.
- Remove all duplicate if statements.
- Flexible
- Potential extensibility
- In some application it increase testability.

Disadvantage

- The only disadvantage of this pattern is that is you have more state say 100 state you will have 100 ConcreteState class too.

Thank you

- Now you know the State pattern
- I hope this presentation help you in some way
- If there any mistakes I would like to apologise for those.