State Pattern

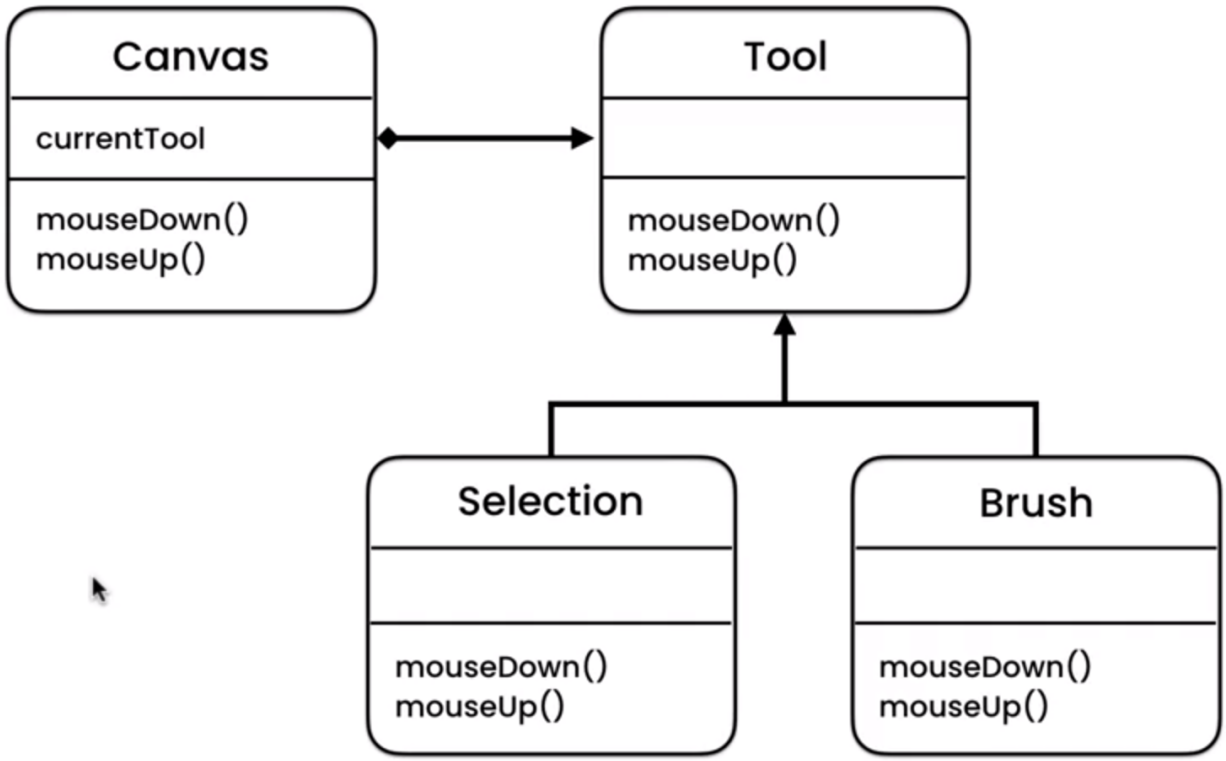
Example Problem : Building a drawing application like photoshop - you have a palette of tools and the canvas behaves differently depending on the tools you select. I.e. the canvas responds to mouses events such as mouse up and mouse own, but what it does changes deppending on the current tool selected. Using an Enum to represent the ToolType you could implement the following :

```
public class Canvas {
  private ToolType currentTool;

  public void mouseDown() {
    if (currentTool == ToolType.SELECTION)
      System.out.println("Selection icon");
    else if (currentTool == ToolType.BRUSH)
      System.out.println("Brush icon");
    else if (currentTool == ToolType.ERASER)
      System.out.println("Eraser icon");
  }

  public void mouseUp() {
```

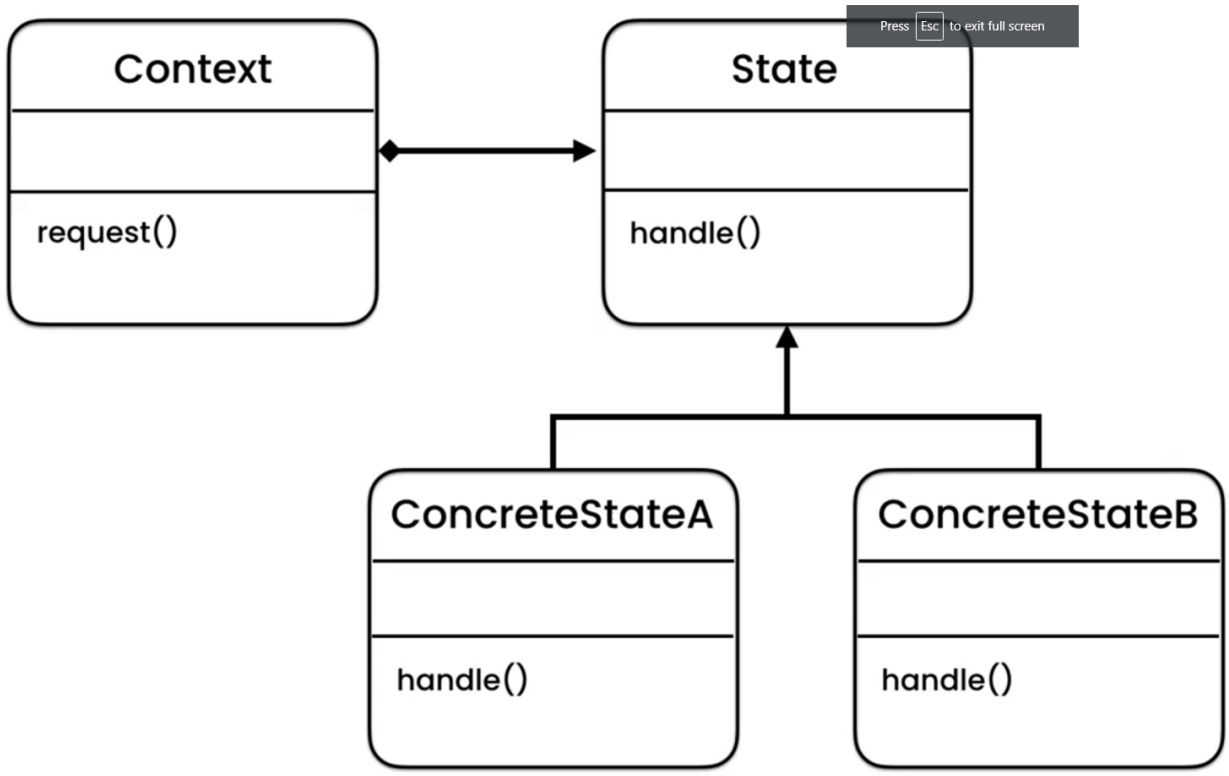
There are a few issues with this implementation - it will lead to repeated if elese logic in all of the event / action methods and as a new action is added the Canvas class will have to be updated violating the OCP. this will make the code difficult to maintain

Solution

What we need is for the canvas to behave differently based on the currently selected tool. We can apply polymorphism to achieve this.



We can create an abstratct tool class with two abstract methods - mouseDown() and mouseUp() which are implemented by child Tool classes and depending on which child is passes to the Canvas class the behavior will vary. As per the GOF book the Canvas class is known as the Context class whiich handlles a request depending on the abstract State class and the ConcreteState child class implement hohw the requests are handled :



In the implementation we replace the abstract class with an interface as we are not roviding any common code to the children :

```
public class Canvas {
  private Tool currentTool;

 public void mouseDown() {
    currentTool.mouseDown();
 }

  public void mouseUp() {
    currentTool.mouseUp();
 }

  public Tool getCurrentTool() {
    return currentTool;
 }

  public void setCurrentTool(Tool
currentTool) {
    this.currentTool = currentTool;

 }
}
```

```
public interface
Tool {
  void mouseDown()
;
 void mouseUp();
}
```

```
public enum
ToolType {
 SELECTION,
 BRUSH,
 ERASER
}
```

```
public class BrushTool
implements Tool {
  @Override
  public void mouseDown
() {
    System.out.println(
"Brush icon");
 }

  @Override
  public void mouseUp()
{
    System.out.println(
"Draw a line");
 }
}
```

```
public class EraserTool implements Tool {
  @Override
  public void mouseDown() {
    System.out.println("Eraser icon");
 }

  @Override
  public void mouseUp() {
    System.out.println("Erase something");
 }
}
```

```
public class SelectionTool implements Tool {
  @Override
  public void mouseDown() {
    System.out.println("Selection icon");
 }

  @Override
  public void mouseUp() {
    System.out.println("Draw a dashed rectangle");
 }
}
```

```
public class Main {

 public static void main(String[] args) {

var canvas = new Canvas()

canvas.setCurrentTool(new SelectionTool());

canvas.mouseDown();

canvas.mouseUp();

//outputs "Selection Icon" then "Draw a dashed rectangle"

}

}
```