Rahin Jain
CS 342 - Software Design
HW 4

Summary of the Template Method Design Pattern:

The Template Method pattern defines the skeleton of an algorithm in a superclass but delegates some steps to subclasses. Subclasses can override specific steps of the algorithm without changing its overall structure. This pattern promotes code reuse and enables the extension of algorithms while maintaining a common structure.

Benefits of the Template Method Pattern:

The Template Method pattern provides a clear separation of concerns between the invariant parts of an algorithm and the variant parts. It promotes code reuse by allowing subclasses to override specific steps while inheriting the common structure from the superclass. For example, in a GUI application like JavaFX, the Template Method pattern can be useful for creating reusable UI components with a consistent layout and behavior while allowing customization of specific actions or events.

Drawbacks of the Template Method Pattern:

One drawback of the Template Method pattern is that it can lead to a rigid structure, making it difficult to modify or extend the algorithm beyond what the template allows. Overuse of this pattern might also result in a complex hierarchy of classes, making the codebase harder to understand and maintain. For instance, in a large-scale application, if multiple subclasses need significant variations in the algorithm, the Template Method pattern might become cumbersome and inflexible.

Screenshot of Code Implementation:

Screenshot of Code Implementation:

```
2 usages  1 inheritor
abstract class TemplateMethod {
    9 usages
    protected Button button1;
    5 usages
    protected Button button2;
    5 usages
    protected TextField textField1;
    6 usages
    protected TextField textField2;
    7 usages
    protected BorderPane root;

    1 usage
    public TemplateMethod(Button button1, Button button2, TextField textField1, TextField textField2) {
        this.button1 = button1;
        this.button2 = button2;
        this.textField1 = textField1;
        this.textField2 = textField2;
        this.root = new BorderPane();
    }

    1 usage
    public void setupUI(Stage primaryStage) {
        setTitle(primaryStage);
        setButtons();
        setTextFields();
        setActions();
        setRootStyle();
    }
```

```
    1 usage   1 implementation
    protected abstract void setTitle(Stage primaryStage);

    1 usage   1 implementation
    protected abstract void setButtons();

    1 usage   1 implementation
    protected abstract void setTextFields();

    1 usage   1 implementation
    protected abstract void setActions();

    1 usage
    private void setRootStyle() {
        root.setPadding(new Insets( v: 50,  v1: 10,  v2: 20,  v3: 20));
        root.setStyle("-fx-background-color: #292e2b;");
    }

    1 usage
    public BorderPane getRoot() {
        return root;
    }
}
```

```
class ConcreteTemplateMethod extends TemplateMethod {

    1 usage
    public ConcreteTemplateMethod(Button button1, Button button2, TextField textField1, TextField textFi
        super(button1, button2, textField1, textField2);
    }

    1 usage
    @Override
    protected void setTitle(Stage primaryStage) {
        primaryStage.setTitle("Rahin Jain Homework 3");
    }

    1 usage
    @Override
    protected void setButtons() {
        button1.setShape(new Circle( v: 1));
        button2.setShape(new Circle( v: 1));
        button1.setStyle("-fx-background-color: #399952;");
        button2.setStyle("-fx-background-color: #399952;");
        VBox buttons = new VBox( v: 20, button1, button2);
        root.setLeft(buttons);
    }

    1 usage
    @Override
    protected void setTextFields() {
        textField1 = new TextField( s: "enter text here then press button 1");
        textField2 = new TextField();
        textField2.setEditable(false);
        root.setCenter(textField1);
        root.setRight(textField2);
    }
```

```
    1 usage
    @Override
    protected void setActions() {
        button1.setOnAction(e -> {
            String newText = textField1.getText() + " : from the center text field!";
            textField2.setText(newText);
            button1.setDisable(true);
            button1.setText("pressed");
        });

        button2.setOnAction(e -> {
            textField1.clear();
            textField2.setText("final string goes here");
            button1.setDisable(false);
            button1.setText("button one");
        });
    }
}
```

GUI Description and Expected Behavior:

The GUI created using JavaFX consists of two buttons ("button 1" and "button 2") placed in a vertical layout on the left side, a text field ("enter text here then press button 1") at the center, and another text field on the right side. Clicking "button 1" appends text from the center text field to the right text field and disables the button. Clicking "button 2" clears the center text field and sets a predefined text to the right text field.

Ways to Test the Design Pattern:

1. Empty Test: Ensure that the superclass algorithm works correctly without any overridden methods.
2. Single Override Test: Test a subclass where only one method of the template is overridden to ensure it behaves as expected.
3. Many Overrides Test: Test a subclass where multiple methods of the template are overridden to validate complex customization.
4. Integration Test: Test the interaction between the superclass and its subclasses to ensure proper collaboration.
5. Boundary Test: Test edge cases where subclasses provide extreme variations to the algorithm to verify the robustness of the pattern.

Screenshot of JUnit Test:

```java
@Before
public void setUp() {
    // Initialize JavaFX Toolkit
    new JFXPanel();

    // Initialize UI components
    button1 = new Button( s: "button 1");
    button2 = new Button( s: "button 2");
    textField1 = new TextField( s: "Test");
    textField2 = new TextField();
}

@Test
public void testButton1Action() {
    Platform.runLater(() -> {
        // Simulate button click
        button1.fire();

        // Verify text update in textField2
        assertEquals("Test : from the center text field!", textField2.getText());

        // Verify button disable state
        assertTrue(button1.isDisabled());
    });
}
```