Bridge Design Pattern

> Designed by ZHU Yueming in 2024 Dec. 20th

Experimental Objective

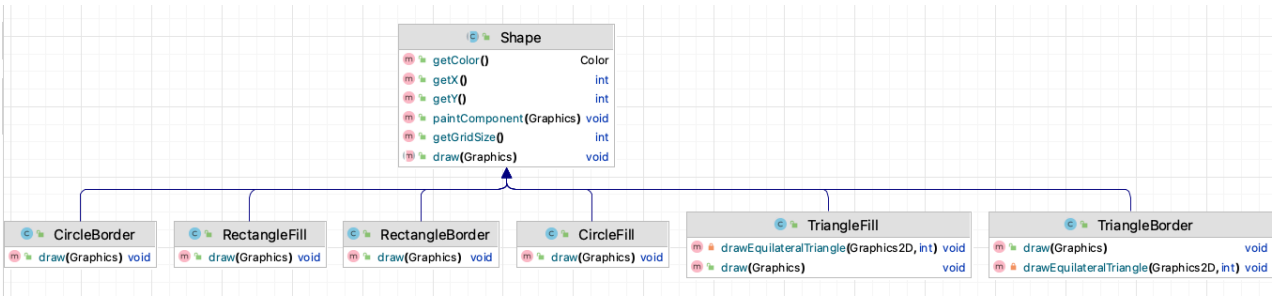
Understand the bridge design pattern and its usage scenarios

Introduction to Source Code

The original source code is built using Swing and includes six separate classes to handle two operations (fill and border drawing) for three different shapes (Circle, Triangle, Rectangle). These classes are:

1. `CircleFill`
2. `CircleBorder`
3. `TriangleFill`
4. `TriangleBorder`
5. `RectangleFill`
6. `RectangleBorder`

The code diagram of the source code is:



Disadvantages of the Original Design

1. **Class Explosion**: Each new shape or drawing operation requires additional classes. For example, adding another shape like `Square` would require two more classes (`SquareFill` and `SquareBorder`).
2. **Low Maintainability**: The tight coupling between shapes and operations makes it difficult to modify or extend the functionality without affecting existing code.
3. **Code Duplication**: Similar drawing logic might be repeated across multiple classes, leading to redundant code.
4. **Scalability Issues**: As the number of shapes and operations increases, the design becomes increasingly unwieldy and difficult to manage.

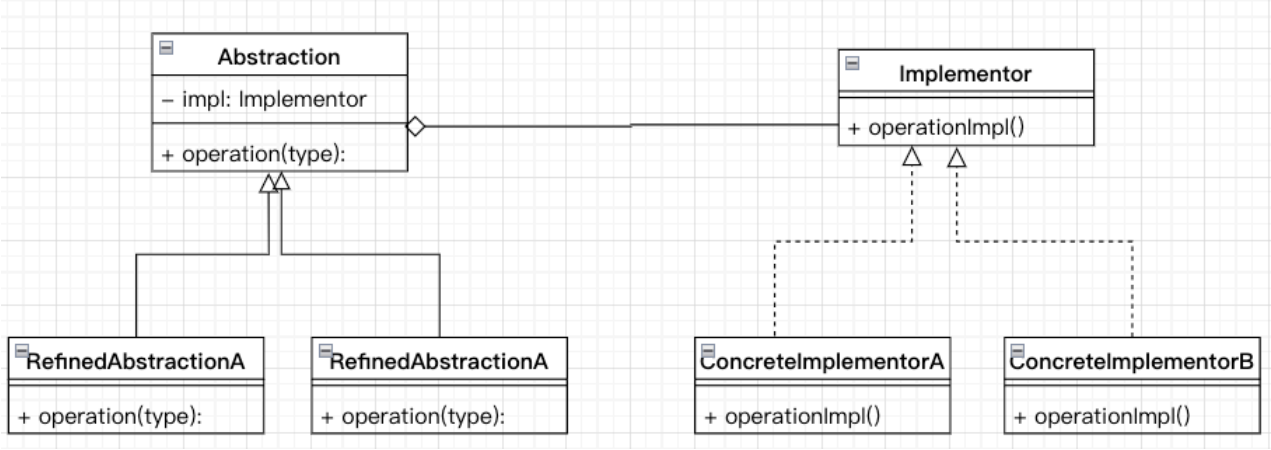
Advantages of Using the Bridge Pattern

The Bridge Design Pattern decouples an abstraction from its implementation, allowing both to vary independently. Applying this pattern in the context of the source code provides the following benefits:

1. **Reduces Class Explosion**: Instead of having a separate class for every combination of shape and operation, the Bridge Pattern separates the shape hierarchy from the drawing operation hierarchy.
2. **Enhances Flexibility**: Adding new shapes or new drawing operations requires minimal changes to the existing codebase.
3. **Improves Code Reusability**: Shared logic can be centralized in one place, avoiding duplication.
4. **Increases Maintainability**: Changes to one part of the system (e.g., adding a new operation) do not affect other parts, making the code easier to maintain and extend.

Bridge Design Pattern Implementation

The UML diagram of bridge design pattern:



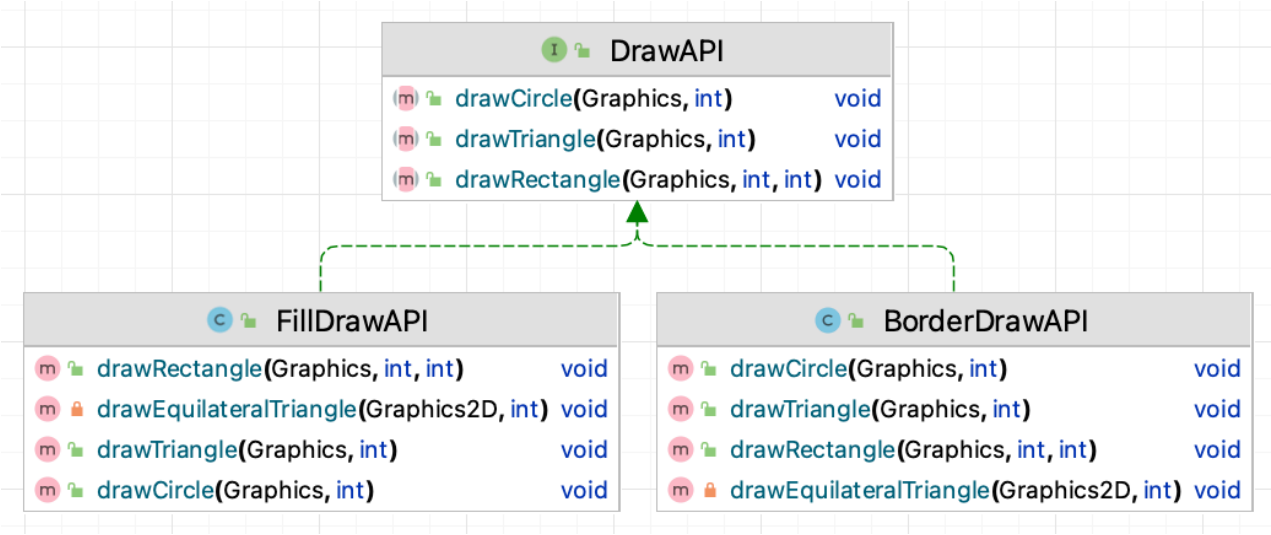
Step 1: Define the `DrawAPI` Interface

```
public interface DrawAPI {
    void drawCircle(Graphics g, int gridSize);
    void drawRectangle(Graphics g, int width, int height);
    void drawTriangle(Graphics g, int gridSize);
}
```

Step 2: Implement the `DrawAPI`

Create two implementations for the `DrawAPI`: one for filled shapes and another for shapes with borders.

```
FillDrawAPI
BorderDrawAPI
```



Step 3: Modified Shapes

1. Add one field drawAPI into shape class.

```
protected DrawAPI drawAPI;
```

2. Change signature of constructor and add a parameter about api.

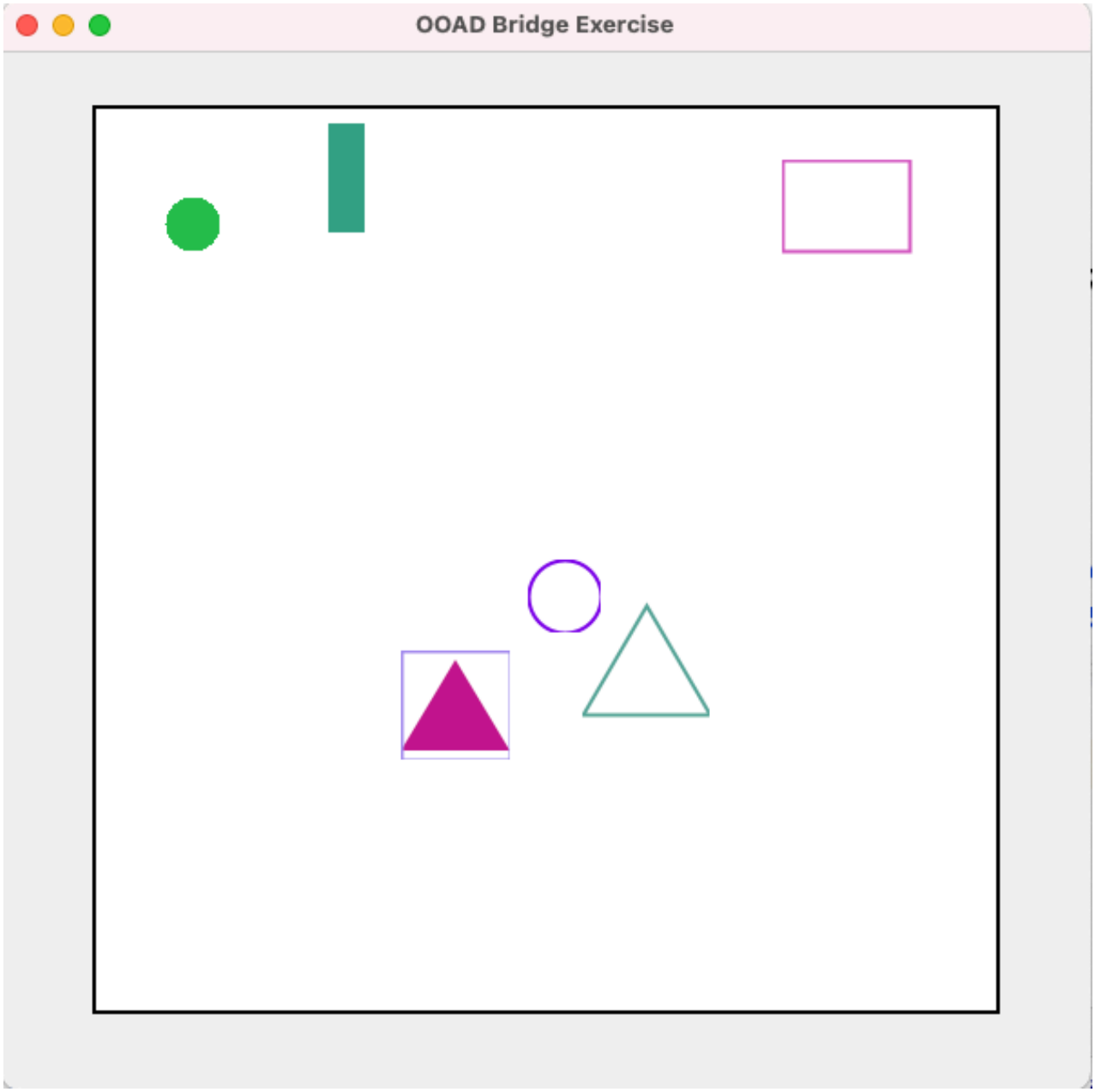
```
public Shape(int x, int y, int gridSize, DrawAPI drawAPI)
```

3. Modify the implement classes of shapes:

   Remove 6 subclasses of shapes, and then create `Circle`, `Rectangle` and `Triangle` instead.

Step 4: Modified Main class

Rewrite the main method to achieve the same result as the original code by using the bridge mode.


OOAD Bridge Exercise