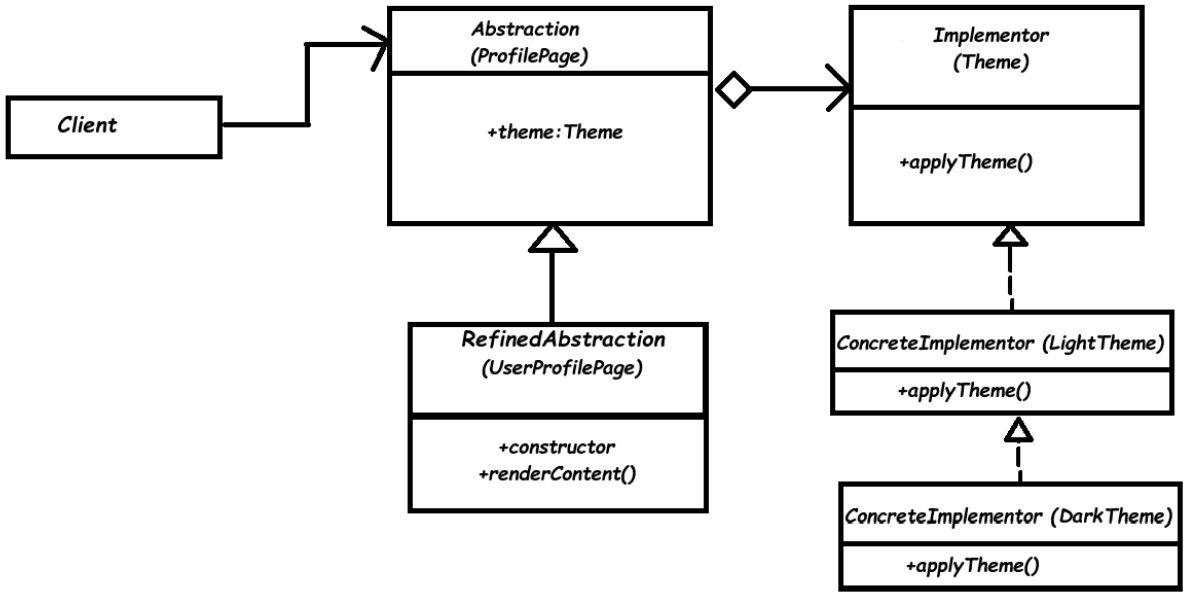
Description:

The Bridge Pattern divides and organizes a single class that has multiple variants of some functionality into two hierarchies: abstraction and implementations. The Bridge Pattern applies the Single Responsibility and the open-closed Principles and independently introduces new abstraction and implementation, making it very easy to switch between them at runtime.

Example:

The UML diagram and code is given below:



Code:

```
// Implementor: Theme
interface Theme {
    void applyTheme();
}

// Concrete Implementor: LightTheme
class LightTheme implements Theme {
    @Override
    public void applyTheme() {
        System.out.println("Applying Light Theme");
    }
}

// Concrete Implementor: DarkTheme
class DarkTheme implements Theme {
    @Override
    public void applyTheme() {
        System.out.println("Applying Dark Theme");
    }
}
```

```
// Abstraction: ProfilePage
abstract class ProfilePage {
    protected Theme theme;

    public ProfilePage(Theme theme) {
        this.theme = theme;
    }

    public abstract void renderContent();
}

// Refined Abstraction: UserProfilePage
class UserProfilePage extends ProfilePage {
    private String username;

    public UserProfilePage(String username, Theme theme) {
        super(theme);
        this.username = username;
    }

    @Override
    public void renderContent() {
        theme.applyTheme();
        System.out.println("Rendering user profile page for: " + username);
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        //Here creating two theme objects:lightTheme and darkTheme.
        Theme lightTheme = new LightTheme();
        Theme darkTheme = new DarkTheme();



        //Here creating two userProfilePage objects: userProfileLight and userProfileDark
        ProfilePage userProfileLight = new UserProfilePage("john_doe", lightTheme);
        ProfilePage userProfileDark = new UserProfilePage("jane_smith", darkTheme);


        userProfileLight.renderContent();
        userProfileDark.renderContent();
    }
}
```

Summary:

Here, the interface is Theme, which defines the contract for applying themes through the *applyTheme* method. In this example, concrete Implementations are *LightTheme* and *DarkTheme* of the Theme interface. *ProfilePage* is the abstraction of this example, which represents the high-level structure of a user profile page. *UserProfilePage* is the refined abstraction extending *ProfilePage*. It represents a specific type of user profile page and contains user-specific content. The code follows the Open-Closed Principle, as it is open for extension but closed for modification. It allows you to add new themes (implementations of Theme) without modifying existing code (*ProfilePage* and *UserProfilePage*). You can easily extend the system by creating new themes or user profile page types. The Bridge in this pattern separates the *ProfilePage* abstraction and the Theme implementation. The Bridge allows us to vary both independently. In the code, *ProfilePage* and its subclasses represent the abstraction, and Theme and its implementations (*LightTheme* and *DarkTheme*) represent the implementation.