**The Scenario**:

Imagine you're building a system where you need to display different types of website pages (like code pages and output pages) with various themes (like dark mode and light mode). You want to make sure that the pages can change their themes without affecting their core functionality.

**Bridge Design Pattern Explained**:

* **Separating Abstraction and Implementation**: The Bridge pattern is all about separating the main features (abstraction) of something from the way it's implemented (implementation). In this case, you want to separate the website pages (abstraction) from the themes (implementation).
* **Abstraction Interface:** The IWebsite interface represents the common features that website pages should have. In this code, it's the GetContent() method that returns the content of a page.
* **Concrete Abstractions**: The CodePage and OutputPage classes are concrete implementations of the IWebsite interface. They represent different types of pages and have their specific content.
* **Implementation Interface**: The ITheme interface defines the methods that themes should have. In this code, it's the GetColor() method that returns the color or theme style.
* **Concrete Implementations**: The DarkTheme and LightTheme classes are concrete implementations of the ITheme interface. They represent different themes and return their respective color styles.
* **Bridge Connection**: In the concrete page classes (CodePage and OutputPage), there's a "bridge" connection with the theme classes (DarkTheme and LightTheme). This connection helps the pages use a theme's color style without worrying about the specifics of how themes work.
* **Creating Instances**: In the Main method, you create instances of different themes (DarkTheme and LightTheme) and use them to create different types of pages (CodePage and OutputPage).
* **Generating Page Content**: When you call the GetContent() method on a page, it uses the theme's color style to generate content. This allows you to have pages with different content, but with colors that match the chosen theme.

**In Simple Words**: The Bridge pattern is like building a connection between two parts of a system so they can work together while being flexible to change. In this code, you connect website pages with themes, allowing you to have different pages and themes that can be swapped in and out without messing up how they work together. This makes it easier to customize and update your system's appearance and functionality separately.

**Step 1: Define Interfaces and Classes**

In this step, two sets of interfaces and classes are defined: one for the website pages (IWebsite, CodePage, OutputPage) and the other for the themes (ITheme, DarkTheme, LightTheme).

internal class Program

{

**// Interface for website pages**

interface IWebsite

{

string GetContent();

}

**// Concrete CodePage class implementing IWebsite**

class CodePage : IWebsite

{

protected ITheme theme;

public CodePage(ITheme theme)

{

this.theme = theme;

}

public string GetContent()

{

return $"Code page in {theme.GetColor()}";

}

}

**// Concrete OutputPage class implementing IWebsite**

class OutputPage : IWebsite

{

protected ITheme theme;

public OutputPage(ITheme theme)

{

this.theme = theme;

}

public string GetContent()

{

return $"Output page in {theme.GetColor()}";

}

}

**// Interface for theme**

interface ITheme

{

string GetColor();

}

**// Concrete DarkTheme class implementing ITheme**

class DarkTheme : ITheme

{

public string GetColor()

{

return "Dark Mode";

}

}

**// Concrete LightTheme class implementing ITheme**

class LightTheme : ITheme

{

public string GetColor()

{

return "Light Mode";

}

}

**// ...**

}

**Step 2: Using the Bridge Pattern**

In this step, the Bridge pattern is used to connect the website pages and the themes. The CodePage and OutputPage classes take a reference to an ITheme instance, and they use this theme's color when generating their content.

static void Main(string[] args)

{

var darkTheme = new DarkTheme();

var lightTheme = new LightTheme();

var codepage = new CodePage(lightTheme);

var outputpage = new OutputPage(darkTheme);

Console.WriteLine(codepage.GetContent());

Console.WriteLine(outputpage.GetContent());

}

**Explanation of Bridge Pattern:**

The Bridge pattern is used to separate an abstraction (abstraction here is the IWebsite interface) from its implementation (implementation here is the ITheme interface). This separation allows both the abstraction and the implementation to change independently without affecting each other.

In this example,

* IWebsite interface represents the abstraction, and the IWebsite implementations (CodePage and OutputPage) represent the refined abstractions. The ITheme interface represents the implementation.
* By passing an instance of a theme (ITheme) to the website page classes (CodePage and OutputPage), the Bridge pattern allows the content generation in the pages to use the theme's color without being tightly coupled to a specific theme.
* In the Main method, different instances of themes (DarkTheme and LightTheme) are created. Then, concrete website page instances (CodePage and OutputPage) are created, each linked to a specific theme. When the content of the pages is generated, the theme's color is incorporated, and the result is printed.

In this code example, the Bridge pattern allows the decoupling of website pages and themes, enabling changes in themes and pages independently. It promotes flexibility and maintainability in cases where you want to create different combinations of content and themes.