**Exercise 5: Implementing the Decorator Pattern**

**Scenario:**

You are developing a notification system where notifications can be sent via multiple channels (e.g., Email, SMS). Use the Decorator Pattern to add functionalities dynamically.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **DecoratorPatternExample**.
2. **Define Component Interface:**
   * Create an interface **Notifier** with a method **send()**.
3. **Implement Concrete Component:**
   * Create a class **EmailNotifier** that implements Notifier.
4. **Implement Decorator Classes:**
   * Create abstract decorator class **NotifierDecorator** that implements **Notifier** and holds a reference to a **Notifier** object.
   * Create concrete decorator classes like **SMSNotifierDecorator**, **SlackNotifierDecorator** that extend **NotifierDecorator**.
5. **Test the Decorator Implementation:**
   * Create a test class to demonstrate sending notifications via multiple channels using decorators.

**CODE:**

interface INotifier

{

void Send(string message);

}

class EmailNotifier : INotifier

{

public void Send(string message) => Console.WriteLine($"Email: {message}");

}

abstract class NotifierDecorator : INotifier

{

protected INotifier notifier;

public NotifierDecorator(INotifier notifier) => this.notifier = notifier;

public virtual void Send(string message) => notifier.Send(message);

}

class SMSDecorator : NotifierDecorator

{

public SMSDecorator(INotifier notifier) : base(notifier) { }

public override void Send(string message)

{

base.Send(message);

Console.WriteLine($"SMS: {message}");

}

}

class Program

{

static void Main()

{

INotifier notifier = new SMSDecorator(new EmailNotifier());

notifier.Send("Hello!");

}

}

**OUTPUT:**

