**Shape Drawing Application**   
This application uses the Factory Method pattern to dynamically create and draw shapes based on user input (Circle or Square).

1. **Shape:**  
   An interface that defines the draw() method.
2. **Concrete Shapes:**
   * Circle: Implements the Shape interface to draw a circle.
   * Square: Implements the Shape interface to draw a square.
3. **ShapeFactory:**  
   Abstract class that defines the createShape() method to produce shape objects.
4. **Concrete Factories:**
   * CircleFactory: Creates a Circle object.
   * SquareFactory: Creates a Square object.
5. **ShapeDrawingMain:**  
   Client class that takes user input to create and draw the selected shape.

**GUI Framework**

The GUI Framework application supports two themes: Light and Dark. Using the Abstract Factory pattern, it dynamically generates different UI components (Buttons and TextBoxes) depending on the selected theme.

1. **GUIFactory**:  
   An interface that defines methods for creating Button and TextBox objects.
2. **Concrete Factories:**
   * LightThemeFactory: Produces light-themed Button and TextBox.
   * DarkThemeFactory: Produces dark-themed Button and TextBox.
3. **Product Interfaces:**
   * Button: Defines the render() method for buttons.
   * TextBox: Defines the render() method for text boxes.
4. **Concrete Products:**
   * LightButton and LightTextBox: Light-themed components.
   * DarkButton and DarkTextBox: Dark-themed components.
5. **GUIApplication:**  
   Client class that prompts the user to select a theme and renders the corresponding components dynamically.

**Database Integration System**

The Database Integration System uses the Adapter Pattern to adapt different database APIs to a standard interface, ensuring compatibility with a modern system.

* **DatabaseClient**: The client that interacts with the database, unaware of underlying implementation differences.
* **DatabaseAdapter**: Interface that standardizes database operations like connect, query, and disconnect.
* **LegacyDatabaseAdapter and ModernDatabaseAdapter:** Concrete implementations of the adapter, wrapping legacy and modern database APIs to conform to the standard interface.
* **DatabaseApp**: Main class that allows integration of different databases dynamically by using appropriate adapters.

**Notification System**

The Notification System uses the Bridge Pattern to decouple the abstraction (notification types) from their implementation (sending mechanism). This allows for flexible configuration of different notification channels (Email, SMS) and sender methods at runtime.

* **Notification**: Abstract class that defines a notification type and holds a reference to NotificationSender.
* **EmailNotification and SMSNotification:** Concrete implementations of Notification, specifying how each type sends messages.
* **NotificationSender**: Interface for different notification sending methods (e.g., email, SMS).
* **EmailSender and SMSSender:** Concrete implementations of NotificationSender that define how notifications are delivered.
* **NotificationApp**: Main class that dynamically selects the notification type and sender based on user input.

**Weather Monitoring System**

This system uses the Observer Design Pattern to implement a weather monitoring application. Multiple display devices (observers) receive real-time weather updates from a central weather station (subject).

1. **WeatherStation** (Subject):
   * Centralized class that maintains a list of observers (DisplayDevices).
   * Provides methods to register, remove, and notify observers whenever weather data changes.
   * Weather data like temperature, humidity, etc., can be updated, which triggers notifications to all registered observers.
2. **DisplayDevice** (Observer):
   * Interface that defines the update(float temperature, float humidity) method.
   * Concrete display classes (e.g., PhoneDisplay, WebDisplay) implement this interface to receive weather updates and display the data in different formats.
3. **Concrete Observers:**
   * PhoneDisplay and WebDisplay: These classes implement the DisplayDevice interface and display weather information according to their display style.
   * Each observer receives updates automatically whenever the weather station's data changes.
4. **Observer Pattern:**
   * The weather station (subject) notifies all registered observers when weather data changes, ensuring each display device is always in sync with the latest weather conditions.

**Payment Processing System**

This system demonstrates the Strategy Design Pattern with a Singleton for managing different payment methods in an e-commerce application. It supports dynamic selection of payment strategies (Credit Card, PayPal, Bitcoin) at runtime.

1. **PaymentStrategy**: Interface that defines a common method pay(double amount) for all payment methods.
2. **CreditCardPayment**, PayPalPayment, BitcoinPayment: Concrete classes implementing the PaymentStrategy interface. Each defines a specific payment behavior.
3. **PaymentProcessor**: A singleton class responsible for holding the selected PaymentStrategy. It provides methods for setting the strategy and processing payments.
   * Ensures only one instance of PaymentProcessor exists using the Singleton pattern.
   * Dynamically executes the selected strategy’s pay method.
4. **PaymentApp**: The main class where users can choose a payment method via console input, set the appropriate strategy, and process a payment.

**Smart Office Room Booking System**

This Smart Office Management system allows users to configure multiple meeting rooms, book or cancel bookings, manage room occupancy, and track usage statistics. The **OfficeConfiguration** class implements the Singleton pattern to manage the list of rooms. Each **Room** tracks its booking status, occupancy, and usage statistics like total occupancy time and the number of times used. The Command pattern is used for room booking and cancellation, with **RoomBookingCommand** and **RoomCancellationCommand** classes executing these actions. Observers like Light and **AirConditioner** automatically update room statuses, turning on or off based on occupancy changes. The system also provides a **MainMenu** to interact with users for booking, occupancy management, and usage statistics display.