**Factory Pattern: Notification System**

**Description:**

The **Factory Pattern** is a creational design pattern that provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created. This pattern is used to encapsulate object creation logic and provide a way to delegate the instantiation process to subclasses or factory methods.

**Components:**

1. **Product Interface (Notification):**  
   Defines the interface for the objects the factory method will create. Each concrete class will implement this interface.
2. **Concrete Products (EmailNotification, SMSNotification, PushNotification):**  
   Implement the product interface and provide specific implementations for different types of notifications.
3. **Factory (NotificationFactory):**  
   Contains a method for creating objects. It decides which concrete class to instantiate based on input parameters, encapsulating the instantiation logic.

**How It Works:**

1. **Request Creation:** The client requests an instance of a notification through the factory, specifying the type of notification it needs (e.g., Email, SMS, Push).
2. **Object Creation:** The factory method (createNotification) determines which concrete class to instantiate based on the type provided.
3. **Use of Object:** The client uses the created object through the common interface, without needing to know the specific class of the object.

**Real-Life Example: Notification System**

In a notification system:

* **Notification (Product Interface):** Defines the common method notifyUser for sending notifications.
* **EmailNotification, SMSNotification, PushNotification (Concrete Products):** Each class implements the Notification interface to provide specific ways of notifying users.
* **NotificationFactory (Factory):** Contains the createNotification method that decides which concrete notification class to instantiate based on the provided type.

**Scenario:**

1. **Client Request:** The client needs to send a notification. It requests an EmailNotification from the NotificationFactory.
2. **Factory Instantiation:** The NotificationFactory creates an instance of EmailNotification and returns it to the client.
3. **Notification Execution:** The client calls notifyUser on the EmailNotification instance, resulting in "Sending Email Notification" being printed.
4. **Different Notification:** Similarly, if the client requests an SMSNotification, the factory creates an SMSNotification and handles it in the same way.

**Benefits:**

1. **Encapsulation:** The Factory Pattern encapsulates the object creation logic, hiding the specific classes from the client.
2. **Flexibility:** New types of notifications can be added without changing the client code, only by extending the factory method.
3. **Code Simplification:** Clients are simplified as they interact with a common interface and do not need to handle the complexities of object creation.