**Assignment 2: Implementing the Observer Pattern**

### **Background**

The **Observer Pattern** defines a one-to-many dependency between objects. When one object (the **Subject**) changes state, all its dependents (the **Observers**) are notified and updated automatically.

This pattern is widely used in real systems:

* UI frameworks (event listeners)
* Messaging/notification systems
* Stock price trackers
* Chat applications

### **Problem Statement**

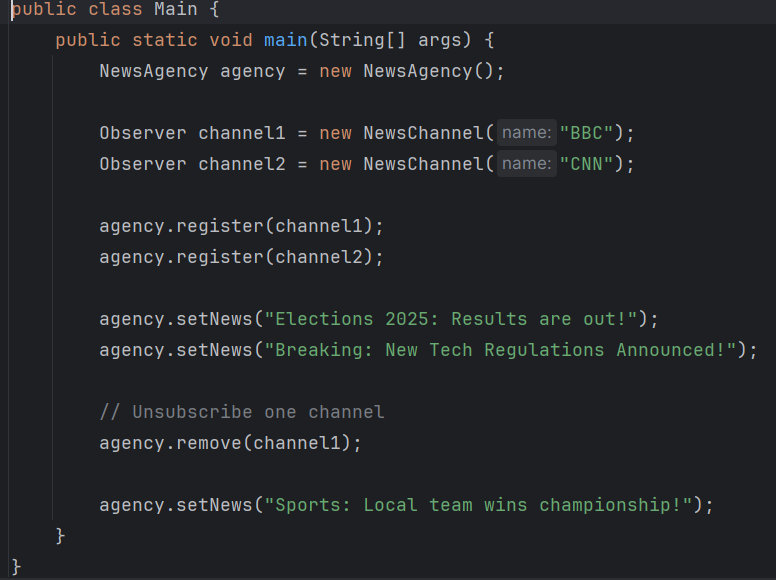
You are tasked with implementing a **Notification System** using the **Observer Pattern**.

* **Subject (Publisher)**: NewsAgency
  + Maintains the current **news headline** (state).
  + Allows Observers to **subscribe/unsubscribe**.
  + Notifies all Observers whenever a new headline is published.
* **Observer (Subscriber)**: NewsChannel
  + Each channel subscribes to the NewsAgency.
  + When notified, it **updates its state** with the latest headline.

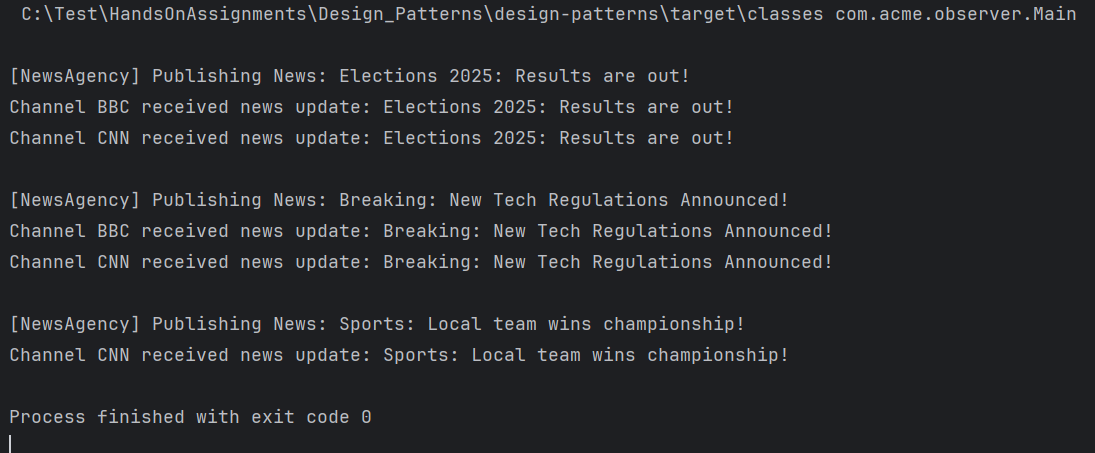
### **Requirements**

1. Create an interface Observer with a method update(String news).
2. Create an interface Subject with methods:
   1. register(Observer o)
   2. remove(Observer o)
   3. notifyObservers()
3. Implement a concrete class NewsAgency (Subject):
   1. Holds a list of Observers.
   2. Has a setNews(String news) method that updates the news and notifies observers.
4. Implement a concrete class NewsChannel (Observer):
   1. Stores the latest news headline received.
   2. Implements update(String news) to print out:  
       "Channel <name> received news update: <news>".

### **Example Run**



**Output:**



### **Tasks**

* Implement Observer and Subject interfaces.
* Implement NewsAgency (subject) and NewsChannel (observer).
* Demonstrate the working in a Main class.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*