A basic example of a simple Java program:

java

Copy code

public class SimpleExample {

public static void main(String[] args) {

Data data = new Data();

data.setValue(42);

System.out.println("Data value is: " + data.getValue());

}

}

class Data {

private int value;

public int getValue() {

return value;

}

public void setValue(int value) {

this.value = value;

}

}

This code sets a value in a Data object and retrieves it. It's straightforward but lacks any interaction between objects.

**Using the Observer Pattern**

The same functionality can be implemented with the Observer pattern to make the system more dynamic and extendable:

java

Copy code

import java.util.ArrayList;

import java.util.List;

// Observer Interface

interface Observer {

void update(int newValue);

}

// Concrete Observer

class Display implements Observer {

@Override

public void update(int newValue) {

System.out.println("Data value updated to: " + newValue);

}

}

// Subject

class Data {

private int value;

private List<Observer> observers = new ArrayList<>();

public void addObserver(Observer observer) {

observers.add(observer);

}

public void removeObserver(Observer observer) {

observers.remove(observer);

}

public int getValue() {

return value;

}

public void setValue(int value) {

this.value = value;

notifyObservers();

}

private void notifyObservers() {

for (Observer observer : observers) {

observer.update(value);

}

}

}

// Main Class

public class ObserverExample {

public static void main(String[] args) {

Data data = new Data();

Display display = new Display();

data.addObserver(display);

data.setValue(42); // Automatically notifies the observer

}

}

**Benefits of Using the Observer Pattern**

1. **Decoupling**: The Observer pattern decouples the Data class from the Display class. Data doesn’t need to know who is observing it; it only knows that it should notify its observers.
2. **Extendability**: Adding more observers (e.g., Logger, GraphUpdater, etc.) requires no changes to the Data class. You simply implement the Observer interface and attach the new observer.
3. **Dynamic Updates**: Changes to the Data object automatically notify all registered observers, ensuring consistency across the system.
4. **Scalability**: The system is better suited for handling complex relationships where multiple components need to react to changes.