Author: Pedro Jahir Hinojosa García

Instructor: Miguel Rugerio

Activity: Observer method

Date: 31/08/2024

The observer pattern notifies all objects observing a subject whenever the subject undergoes a change. To practice this pattern, I designed a character with methods to eat, jump, and fight. Every time the player performs an action, the objects interact with it and execute their respective methods, as shown in Image 1.

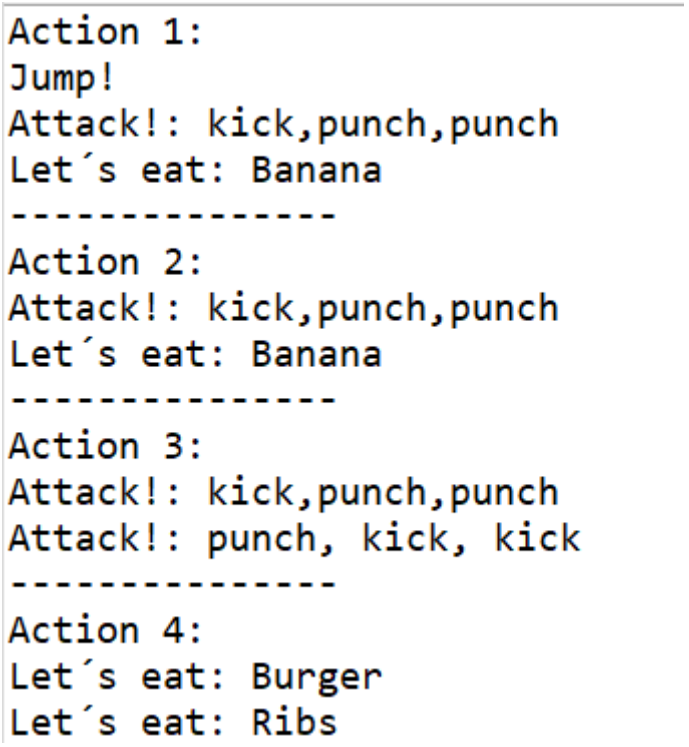

Image 1: Output example.

The implementation consists of creating a list where all changes are registered, with three main methods: add (to add a method), detach (to delete a method), and update (to notify the objects of a change), as shown in Code 1. The Player class extends Subject because it is the object being "observed" by others and only needs to implement a method to notify the observers of changes, as seen in Code 2.

Following this, the Observer class is created to provide a method for updating and constructing a subject, as demonstrated in Code 3. Next, the observers must implement their own methods to interact with the subject and override any methods if needed, as shown in the example in Code 4. At the end, the player receives actions, and the player implements them, as well as deletes or replaces actions as necessary, Code 5.

```java
package oberver;
import java.util.*;

//This class make a list with all the methods that can be implemented to
//the subject class that will have several observers
public abstract class Subject {
    List<Observer> observers = new ArrayList<>();
    void attach(Observer o) {
        observers.add(o);
    }
    void detach(Observer o) {
        observers.removeIf(x -> x.equals(o));
    }
    void notiffy() {
        for(Observer o:observers)
            o.update();
    }
}
```

Code 1: Subject class, with the three main methods.

```java
public class Player extends Subject{

    void action() {
        System.out.print("");
        notiffy();
    }

}
```

Code 2: The main player is the main Subject.

```java
//the observer classes most see the subject class if it changes
public abstract class Observer {
    Subject sub;

    Observer(Subject sub){
        this.sub=sub;
        sub.attach(this);
    }

    abstract void update();

}
```

Code 3: The main class that will be herded by all the observers classes.

```java
public class ComboAtack extends Observer{
    String comAt;
    public ComboAtack( String comAt,Subject sub) {
        super(sub);
        this.comAt = comAt;
    }
    void doCombo() {
        System.out.println("Attack!: "+comAt);
    }
    @Override
    public void update() {
        doCombo();
    }
    @Override
    public int hashCode() {
        return Objects.hash(comAt);
    }
//This override is needed because the equals in string check if are the same object
//not the same string, this basically solve the problem and check if the strings are the same
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        ComboAtack other = (ComboAtack) obj;
        return Objects.equals(comAt, other.comAt);
```

Code 4: Observer class example.

```java
public class Principal {
    public static void main(String[] args) {
        Player player1 = new Player();

        System.out.println("Action 1: ");

        Observer jump = new Jump(player1);

        new ComboAtack("kick,punch,punch", player1);
        new Eat("Banana",player1);

        player1.action();
        System.out.println("---------------");
        System.out.println("Action 2: ");
        player1.detach(jump);
        player1.action();

        System.out.println("---------------");
        System.out.println("Action 3: ");
        player1.detach(new Eat("Banana",player1));
        new ComboAtack("punch, kick, kick", player1);
        player1.action();

        System.out.println("---------------");
        System.out.println("Action 4: ");
        player1.detach(new ComboAtack("punch, kick, kick", player1));
        player1.detach(new ComboAtack("kick,punch,punch", player1));
        new Eat("Burger",player1);
        new Eat("Ribs",player1);
        player1.action();
    }
```

Code 5: Final class where the observers interact with the player.