



Daniel Ivan Anaya Alvarez

Activity Observer and Jtesting

For this activity i'm going to implement the observer pattern also do the Jtesting on it, the business model I use was for a convenience store that would advise about any product if its available or no and if in case it changes its Price it would update and pass it through the observer, the subject has the list of products and basically the product is the implementation, finally we have the client that is the one thats interested of seeing if there is any changes on the products.

Subject.java

```
package Actividad0;

import java.util.ArrayList;
import java.util.List;

public abstract class Subject {

    //Make a list from the observers
    List<Observer> observers = new ArrayList<>();

    //Attach to the list any observer that goes on the code
    void attach(Observer obs) {
        observers.add(obs);
    }
    //detach any observer found in the list
    void detach(Observer obs) {
        observers.removeIf(x -> x.equals(obs));
    }

    //Notify the update of the objects
    void notifyobservers(){
        for(Observer obs:observers)
            obs.update();
    }
}
```

Observer.java

```
package Actividad0;
```

```

public abstract class Observer {

    //HAS A subject
    Subject sub;

    //Constructor de subject
    Observer(Subject sub){
        this.sub = sub;
        sub.attach(this);
    }

    //Metodo abstracto de update
    abstract void update();

}

```

Product.java

```

package Actividad0;
import java.util.*;

public class Product extends Subject {

    private String name;
    private double price;
    private boolean available;

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
        this.available = false; // Product not available by default
    }

    public void setPrice(double newprice) {
        this.price = newprice;
        notifyobservers(); // //Notify the program when the product
changes prices
    }

    public void setAvailability(boolean available) {
        this.available = available;
        notifyobservers(); // Notify to the clients when the product is
available
    }

    public double getPrice() {
        return price;
    }

    public boolean isAvailable() {
        return available;
    }
}

```

```

        public String getName() {
            return name;
        }
    }
}

```

Client.java

```

package Actividad0;

public class Client extends Observer {

    private String name;

    public Client(String name, Subject product) {
        super(product); // Takes the product automatically
        this.name = name;
    }

    @Override
    public void update() {
        Product product = (Product) sub; // We cast the subject
        System.out.println("Hello " + name + ", the product " +
            product.getName() +
                " has new updates. Price: " +
            product.getPrice() +
                ", Availability: " + (product.isAvailable() ?
                "Yes" : "No"));
    }
}

```

Principal.java

```

package Actividad0;

public class Principal {

    public static void main(String[] args) {

        //We make our products
        Product sodacoca = new Product("Cocacola 1lt", 20);
        Product gansito = new Product("Gansito", 21);

        //We create our clients
        Client client1 = new Client("Pedro", sodacoca);
        Client client2 = new Client("Sara", gansito);

        //We see the prints
        client1.update();
        client2.update();
    }
}

```

```

        //Set the availability to true
        sodacoca.setAvailability(true);
        gansito.setAvailability(true);

        //We change the prices and update them
        sodacoca.setPrice(24);
        gansito.setPrice(30);
    }
}

```

Jtesting.java

```

package Actividad0;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class Jtesting {

    private Product product;
    private Client client1;
    private Client client2;

    @BeforeEach
    void setUp() {
        // Inicializar el producto y los clientes antes de cada prueba
        product = new Product("Bread", 1.0);
        client1 = new Client("Alice", product);
        client2 = new Client("Bob", product);
    }

    @Test
    void testAttachObserver() {
        // Check the observers
        assertEquals(2, product.observers.size());
        assertTrue(product.observers.contains(client1));
        assertTrue(product.observers.contains(client2));
    }

    @Test
    void testDetachObserver() {
        // Eliminate a client and see if it worked
        product.detach(client1);
        assertEquals(1, product.observers.size());
        assertFalse(product.observers.contains(client1));
        assertTrue(product.observers.contains(client2));
    }

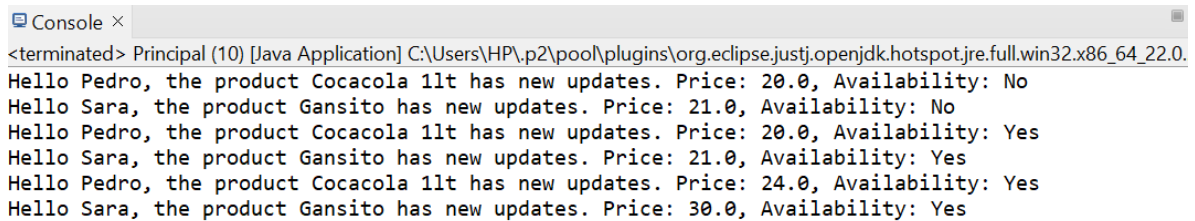
    @Test
    void testNotifyObservers() {
        // Test if clients receive information
        client1.update();
    }
}

```

```
        client2.update();

        product.setPrice(1.2); // changes the prices
    }
}
```

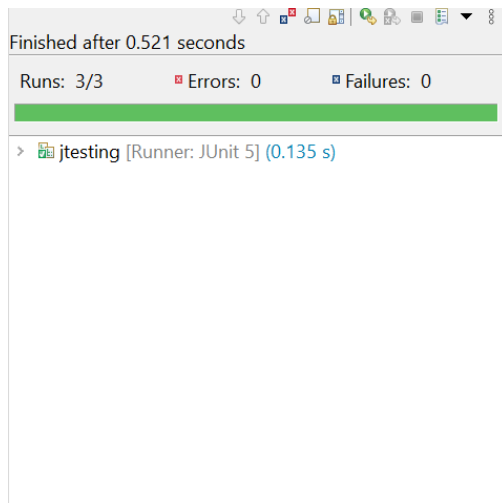
Print



The screenshot shows a Java console window titled "Console x". The output text is as follows:

```
<terminated> Principal (10) [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.  
Hello Pedro, the product Cocacola 1lt has new updates. Price: 20.0, Availability: No  
Hello Sara, the product Gansito has new updates. Price: 21.0, Availability: No  
Hello Pedro, the product Cocacola 1lt has new updates. Price: 20.0, Availability: Yes  
Hello Sara, the product Gansito has new updates. Price: 21.0, Availability: Yes  
Hello Pedro, the product Cocacola 1lt has new updates. Price: 24.0, Availability: Yes  
Hello Sara, the product Gansito has new updates. Price: 30.0, Availability: Yes
```

Coverage Prints



The screenshot shows a JUnit test runner interface. At the top, it says "Finished after 0.521 seconds". Below this, there is a summary bar with "Runs: 3/3", "Errors: 0", and "Failures: 0". A green progress bar is shown below the summary. The main area of the interface shows a tree view with a single entry: "> junitesting [Runner: JUnit 5] (0.135 s)".

Outline Coverage x

Actividad0 (30 ago 2024 2:02:11 p.m.)

Element	Covera...	Covered Ins...	Missed Inst
> ActividadObserver	78.2 %	186	

Coverage Client.java

```

public class Client extends Observer {

    private String name;

    public Client(String name, Subject product) {
        super(product); // Takes the product automatically
        this.name = name;
    }

    @Override
    public void update() {
        Product Product = (Product) sub; // We cast the subject
        System.out.println("Hello " + name + ", the product " + Product.getName() +
            " has new updates. Price: " + Product.getPrice() +
            ", Availability: " + (Product.isAvailable() ? "Yes" : "No"));
    }
}

```

Coverage Observer.java

esting.java Client.java Observer.java × Pri

```
package Actividad0;
```

```
public abstract class Observer {
```

```
    //HAS A subject  
    Subject sub;
```

```
    //Constructor de subject
```

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

```
    //Metodo abstracto de update  
    abstract void update();
```

```
}
```

Coverage Principal.java

```

1 package Actividad0;
2
3 public class Principal {
4
5     public static void main(String[] args) {
6
7         //We make our products
8         Product sodacoca = new Product("Cocacola 1lt", 20);
9         Product gansito = new Product("Gansito", 21);
10
11         //We create our clients
12         Client client1 = new Client("Pedro", sodacoca);
13         Client client2 = new Client("Sara", gansito);
14
15         //We see the prints
16         client1.update();
17         client2.update();
18
19         //Set the availability to true
20         sodacoca.setAvailability(true);
21         gansito.setAvailability(true);
22
23         //We change the prices and update them
24         sodacoca.setPrice(24);
25         gansito.setPrice(30);
26     }
27
28 }
29

```

Coverage Product.java


```

1 package Actividad0;
2 import java.util.*;
3
4 public class Product extends Subject {
5
6     private String name;
7     private double price;
8     private boolean available;
9
10    public Product(String name, double price) {
11        this.name = name;
12        this.price = price;
13        this.available = false; // Product not available by default
14    }
15
16    public void setPrice(double newprice) {
17        this.price = newprice;
18        notifyobservers(); // //Notify the program when the product changes prices
19    }
20
21    public void setAvailability(boolean available) {
22        this.available = available;
23        notifyobservers(); // Notify to the clients when the product is available
24    }
25
26    public double getPrice() {
27        return price;
28    }
29
30    public boolean isAvailable() {
31        return available;
32    }
33

```

Coverage Subject.java

```

1 package Actividad0;
2
3 import java.util.ArrayList;
4
5
6 public abstract class Subject {
7
8
9     //Make a list from the observers
10    List<Observer> observers = new ArrayList<>();
11
12    //Attach to the list any observer that goes on the code
13    void attach(Observer obs) {
14        observers.add(obs);
15    }
16    //detach any observer found in the list
17    void detach(Observer obs) {
18        observers.removeIf(x -> x.equals(obs));
19    }
20
21    //Notify the update of the objects
22    void notifyobservers(){
23        for(Observer obs:observers)
24            obs.update();
25    }
26
27 }
28

```

Conclusion

At the end of this activity we could see that the observer pattern is very important if you need if a parameter changes almost in real time and you desire to see the value of it. Also the Jtesting helps us to see if our code is working as intended without errors, this helps us in case that we push any code to production we can assure that is going to work.