Academia Xideral Java Monterrey

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;
```

```java
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

```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

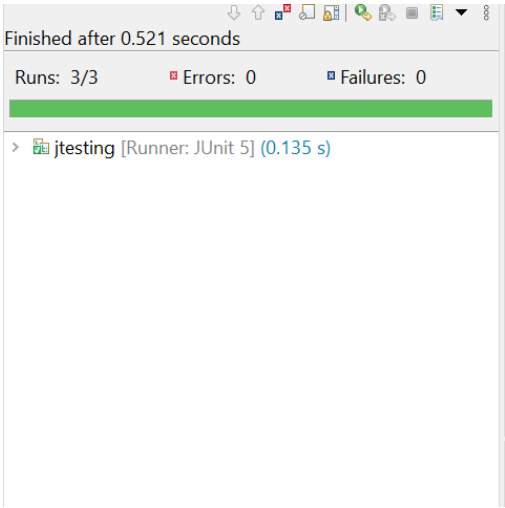
```
package Actividad0;

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

```
Console ×
<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



Outline | Coverage

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

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

Coverage Client.java

```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

```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();

}
```

Coverage Principal.java

```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);
    }

}
```

Coverage Product.java

*jtesting.java | Client.java | Observer.java | Principal.java | Product.java × | Subject.java

```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;
        }
```

Coverage Subject.java

```java
package Actividad0;

import java.util.ArrayList;

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();
    }

}

```

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.