Programa Observer

El programa ejemplifica Observer, este es un patrón de diseño en programación que se utiliza para definir una relación de uno a muchos entre objetos. Este patrón se utiliza para crear una estructura donde un objeto, conocido como el observable, mantiene una lista de sus "observadores" y notifica automáticamente a ellos cualquier cambio en su estado. Esto permite que los observadores reaccionen a los cambios en el sujeto sin necesidad de acoplamiento fuerte entre ellos.

En este caso los observadores serán operaciones de comprobación, al momento de crearse se “suscribirán” automáticamente y en el momento de invocarse el método comprobar() se realizaran estas comprobaciones resultando en un valor true o false.

Entonces tenemos la clase abstracta Observer la cual contiene un objeto del tipo Operaciones, que se definirá mas adelante. Igual contiene un constructor que de igual manera recibe un objeto del tipo Operaciones y se inicializa el objeto propio de la clase. También tenemos un método update() el cual nos servirá mas adelante.

```
public abstract class Observer{
    Operaciones op;

    public Observer(Operaciones op){
        this.op = op;
        op.attach(this);
    }

    abstract void update();
}
```

Luego tenemos la clase abstracta Operaciones antes mencionada, aquí tenemos un arrayList de objetos del tipo Observer. de igual manera se definen los métodos que nos servirán para suscribir a los futuros actores. que son attach(), detach() y notificar().

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

public abstract class Operaciones {
    List<Observer> listObservers = new ArrayList<>();
```

```
    void attach(Observer o){    //agregar nuevo
        listObservers.add(o);
    }

    void detach(Observer o){    //desuscribir
        listObservers.remove(o);
    }

    void notificar(){           //mandar la acción
        for (Observer o : listObservers) {
            o.update();
        }
    }
}
```

Después tenemos la clase Comprobador, que hereda de Operaciones, en esta clase solo definimos el método comprobar(), que manda a llamar al método notificar de la clase Operaciones

```
public class Comprobador extends Operaciones {

    void comprobar(){
        notificar();
    }
}
```

Luego ya tenemos las 3 clases que definen la naturaleza del problema a resolver: si un numero es primo, si un numero es multiplo de otro y si un numero es el factorial de otro

```
//----------------------Numero Primo-------------------------

public class Primo extends Observer {

    int num;

    public Primo(int num, Operaciones op){
        super(op);
        this.num = num;
    }

    boolean comprobarPrimo(){
        for (int i = 2; i < this.num; i++) {
            if (this.num % i == 0) {
                return false;
```

```
            }
        }
        return true;
    }

    @Override
    void update(){
        System.out.println("El numero "+ this.num+ " es primo? : "
        + comprobarPrimo());
    }

}
```

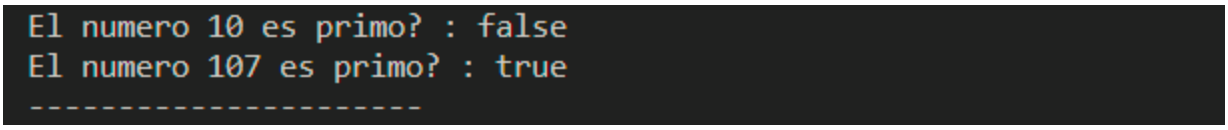
```
//----------------------Numero multiplo de otro-------------------------
public class MultiplodeOtro extends Observer{
    int a;
    int b;

    public MultiplodeOtro(int a, int b, Operaciones op){
        super(op);
        this.a = a;
        this.b = b;
    }

    boolean comprobarMultiplo(){
        if (b%a == 0) {
            return true;
        }
        return false;
    }

    @Override
    void update(){
        System.out.println("El numero " + this.a + " es multiplo de " + this.b
        + "? : " + comprobarMultiplo());
    }
}
```

```
//----------------------Factorial de un numero-------------------------

public class Factorial extends Observer{

    int base;
    int fact;

    public Factorial(int base, int fact, Operaciones op){
        super(op);
```

```
        this.base = base;
        this.fact = fact;
    }

    boolean comprobarFactorial(){
        //si fact es mayor a base se descarta desde un inicio la comprobacion
        if(base > fact)
            return false;

        int aux=1;
        for (int i = 1; i <= base; i++) {
            aux *= i;
        }
        if (aux == fact) {
            return true;
        }else
        return false;

    }

    @Override
    void update(){
        System.out.println("El numero " + this.fact + " es el factorial de "
         + this.base + "? : " + comprobarFactorial());
    }
}
```

Ya en la clase principal se ejemplifica como funciona el programa, primero se crea un Comprobador, y se generan varios objetos del tipo Primo. al momento de crearse un nuevo objeto se agrega automáticamente a la lista de Operaciones y hacemos las comprobaciones con el método comprobar()

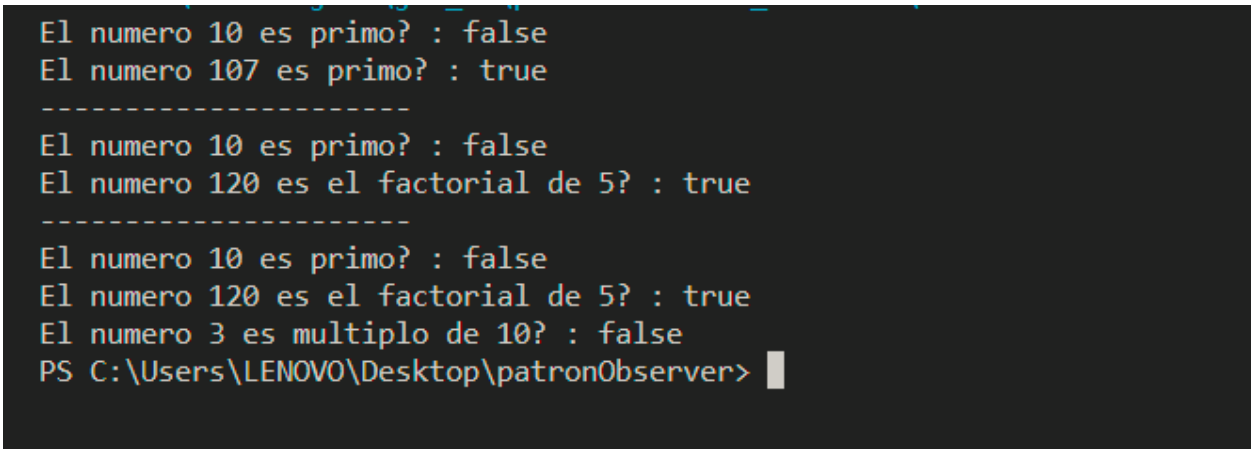
```
public class Main {
    public static void main(String[] args) {
        Comprobador c = new Comprobador();

        Primo p1 = new Primo(10, c);
        Primo p2 = new Primo(107, c);

        c.comprobar()

        ...
    }
}
```



Resultado en la terminal

Luego probamos el método detach(), y agregamos más objetos del tipo Factorial y de MultiplodeOtro

```
c.detach(p2);


        System.out.println("----------");

        Factorial f1 = new Factorial(5, 120, c);

        c.comprobar();

        System.out.println("----------");

        MultiplodeOtro m1 = new MultiplodeOtro(3, 10, c);

        c.comprobar();

    }

}
```



Resultado completo de la terminal