Gregorio Vidoy Fajardo

Ejercicio 2 P1

Tabla de contenidos:

DESCRIPCIÓN DEL PROBLEMA:

Programar, utilizando el patrón de diseño más adecuado, la simulación de un programa simple de monitorización de datos meteorológicos. El programa Simulador se encarga de generar aleatoriamente una temperatura dentro de un rango pre-definido: [t_1, … , t_2] de temperaturas, cada 60 s. aproximadamente.

t_1 y t_2 no se conocen hasta que comienza la ejecución del programa Simulador, ya que dicho rango de temperaturas dependerá de la época del año y de la región. El programa Simulador definirá un método ejecutable (el método público run()) si se decide su implementación como una hebra, tal como la siguiente:

```
Random r= new Random(t2);
int temperatura;
while (true){
  temperatura= r.next(Integer);
  try {sleep(60)}
  catch(java.lang.InterruptedException e){
    e.printStackTrace();
  }
  observablePantalla.notificarObservador();
}
```

Se supondrá que un programa de usuario crearía una única instancia de la clase Pantalla, que a su vez implementa la interfaz Observador. Esta clase define un método público y estático: refrescarPantalla(), que muestra el valor actual de la temperatura al usuario en un formato textual.

La clase Pantalla tiene, por tanto, un carácter de observador y ha de implementar la interfaz correspondiente.

La implementación de la simulación se ha de realizar utilizando creación de hebras de Java. Además, se deben añadir observadores adicionales, tales como: botonCambio, graficaTemperatura, tiempoSatelital.

botonCambio sería una entidad observadora a la que notificaría el programa Simulacion, pero también podría cambiar el estado de la simulación asignando directamente el valor de la temperatura actual.

Suponemos la existencia de una clase interfaz Observador de Java, que declara el método de actualización: manejarEvento(), para que lo implementen las clases observadoras concretas.

El método manejarEvento() ha de contener la lógica necesaria para actualizar la presentación de la información al programa de usuario en grados Celsius o Fahrenheit y también ha de encargarse de “repintar” la pantalla.

Por motivos de reusabilidad del código, podemos asumir dos clases observables: ObservablePantalla (subclase) y Observable de la que hereda, consiguiéndose, de esta forma, un código más claro y transportable. Ambas clases implementarán los métodos públicos: incluirObservador(Observador o), notificarObservador().

A instancia del objeto Simulador se crea el observablePantalla, asignándole memoria. A su vez, el método incluirObservador() asignará memoria a un objeto “o” (puede ser una lista de objetos) que implementa la interfaz Observador.
El método notificarObservador() se puede implementar haciendo una llamada al método de actualización manejarEvento() de los observadores correspondientes.

RELEASE NOTE:

-Compilado con Java 1.8
-Clase Main Cliente.

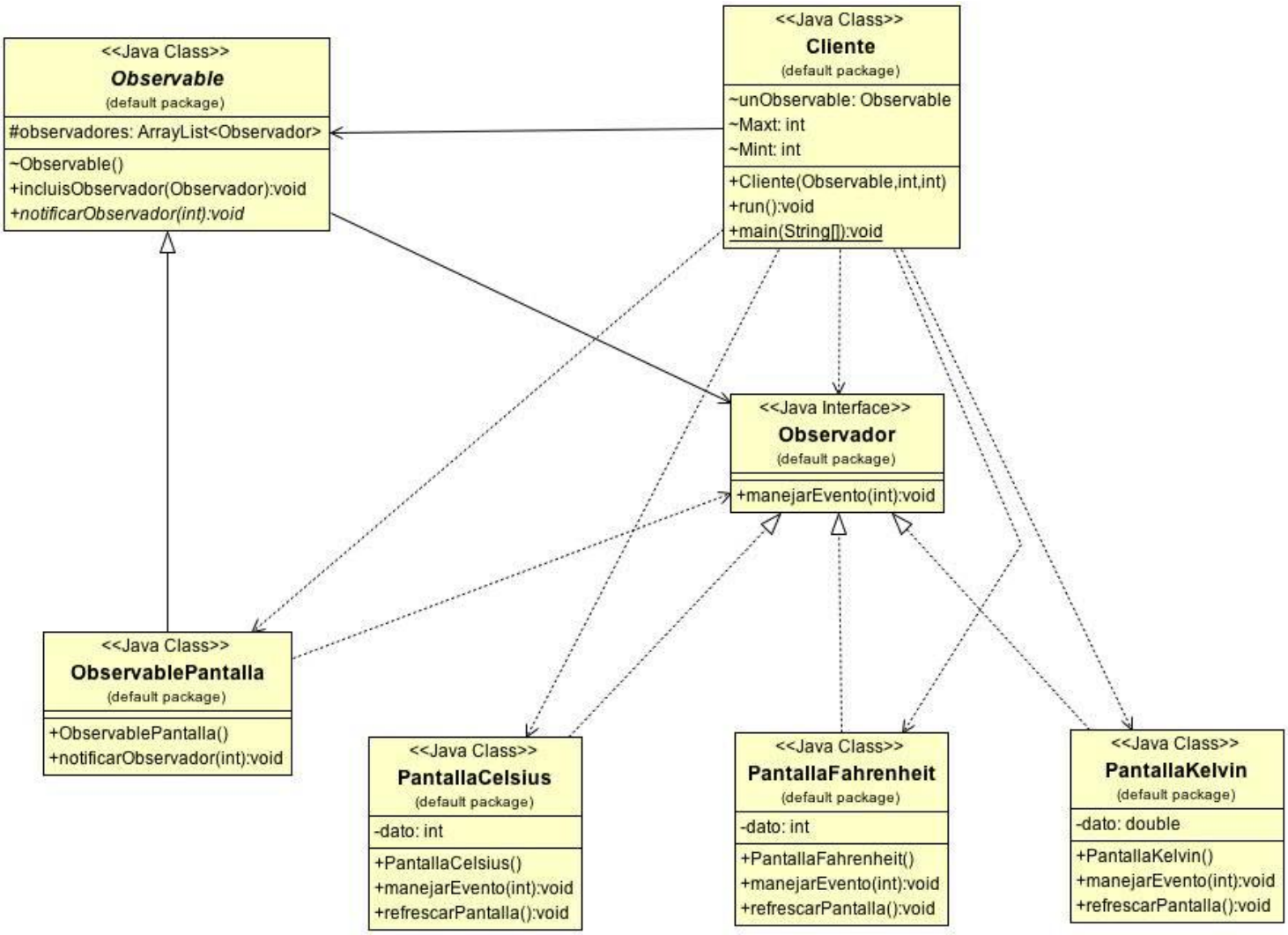
SOLUCIÓN AL PROBLEMA:

-Para la resolución del problema he empleado el patrón de diseño Observable-Observador.
-Para crear el observador se define una Interface, Observador, en la que se implementan los observadores, estos son: PantallaCelsius “Muestra los grados en Celsius”, PantallaFahrenheit “Muestra los grados en Fahrenheit” y PantallaKelvin “Muestra los grados en kelvin”.
-Observable se implementa como una clase abstracta, de la que se especializa ObservablePantallla, el cual tiene un arraylist de observadores y se encargara de notificarles a todos.
-Para poder funcionar, el programa ejecuta con una hebra el simulador que va proporcionando datos al ObservarPantalla con un random entre un máximo y un mínimo.

EXPLICACIÓN DEL PROGRAMA:

El programa comienza e instancia tres observadores, uno con cada pantalla de las diferentes que tengo, fijamos un máximo y un mínimo de 40 , 9.
Instanciamos un Observable de tipo pantalla e incluimos las tres pantallas.
Posteriormente iniciamos el simulador con otra hebra diferente.

UML:


<<Java Class>>
Observable
(default package)
#observadores: ArrayList<Observador>
~Observable()
+incluisObservador(Observador):void
+notificarObservador(int):void
<<Java Class>>
Cliente
(default package)
~unObservable: Observable
~Maxt: int
~Mint: int
+Cliente(Observable,int,int)
+run():void
+main(String[]):void
<<Java Interface>>
Observador
(default package)
+manejarEvento(int):void
<<Java Class>>
ObservablePantalla
(default package)
+ObservablePantalla()
+notificarObservador(int):void
<<Java Class>>
PantallaCelsius
(default package)
-dato: int
+PantallaCelsius()
+manejarEvento(int):void
+refrescarPantalla():void
<<Java Class>>
PantallaFahrenheit
(default package)
-dato: int
+PantallaFahrenheit()
+manejarEvento(int):void
+refrescarPantalla():void
<<Java Class>>
PantallaKelvin
(default package)
-dato: double
+PantallaKelvin()
+manejarEvento(int):void
+refrescarPantalla():void