Ingeniería de Software I

Examen Final - Pauta de Solución

Departamento de Sistemas

Facultad de Ingeniería Electrónica y Telecomunicaciones

Universidad del Cauca

Parte A. Caso de Desarrollo 1 (Calentamiento)

Se desea implementar el RelojFlex definido y modelado en clase usando los modelos presentados en las figuras 1 y 2 (realización del caso de uso ejecutar reloj)

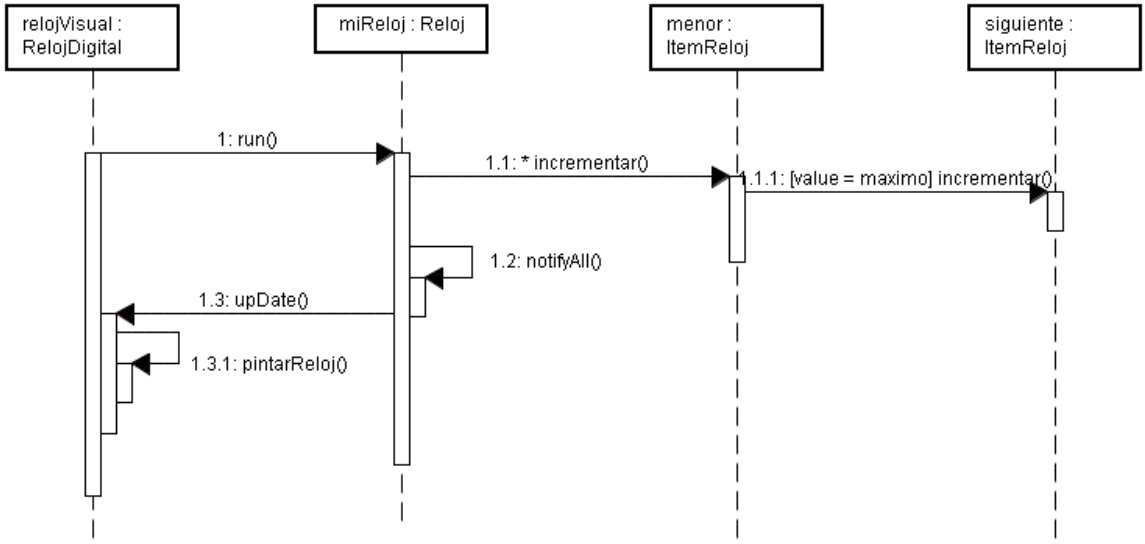


Figura 1. Diagrama de secuencia del caso de uso ejecutar reloj

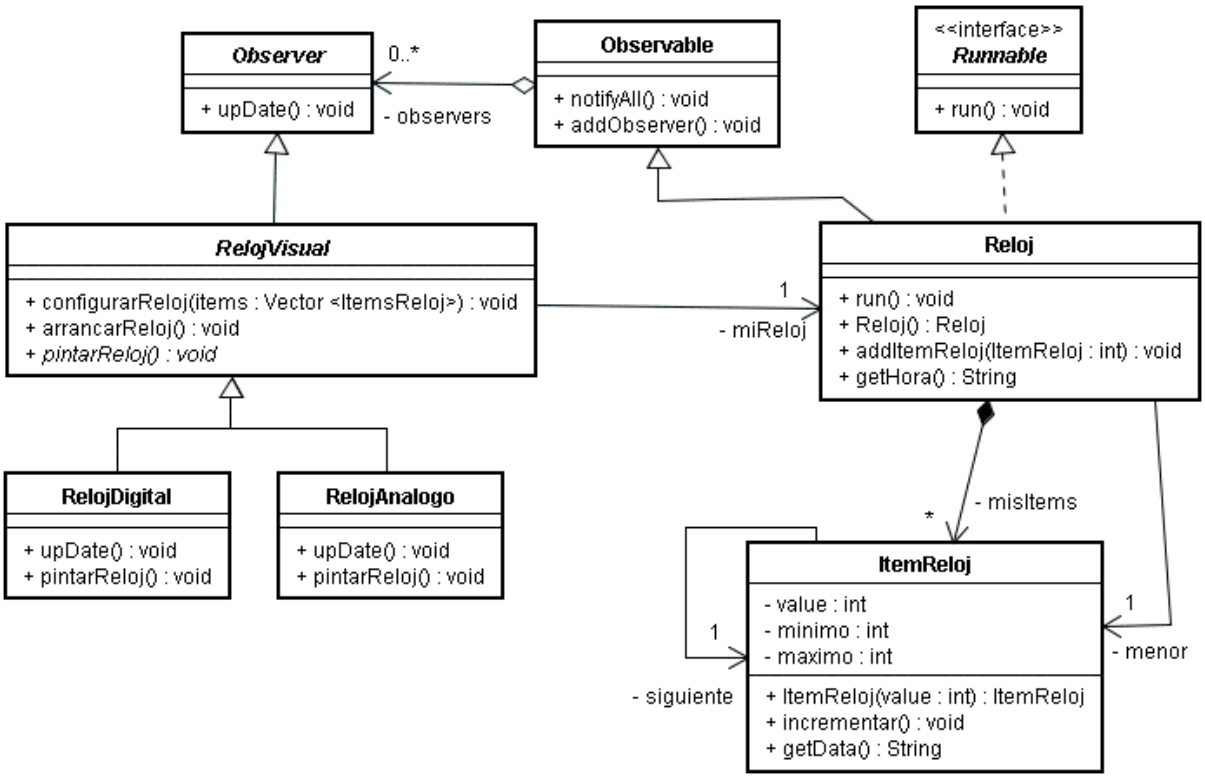


Figura 2. Diagrama parcial de clases a nivel de diseño

1. Respecto a los modelos antes presentados conteste las siguientes preguntas:
    a. (0.5) Explique épocas en pocas palabras cómo se ha aplicado el principio de diseño orientado a objetos "Programe para una interface y no para una implementación" para desacoplar la Clase Reloj de la clase RelojVisual.

Respuesta. El reloj se implementa como un Observable que sólo conoce la interface de los observers, así les notifica el cambio sin saber quienes son sus observers o cómo estén implementados. Así RelojVisual es notificado por parte del Reloj, sin que este último dependa de su implementación.

    b. (0.3) Implemente en su lenguaje orientado a objetos preferido la clase Reloj (atributos, relaciones y métodos) respetando fielmente el diseño. Respecto a los métodos, programe internamente sólo los métodos run () y addItemReloj (), los demás métodos déjelos implementados pero su parte interna déjela en blanco y retornando un valor por defecto (null, 0, etc.) si le corresponde retornar algún valor.

Respuesta:

```
public class Reloj extends Observable implements Runnable {
private Vector <ItemReloj> misItems;
private ItemReloj menor;
```

```
public Reloj(){}
public String getHora(){ return null }
public void addItemReloj(ItemReloj item){}
public run(){
while(true & menor!=null) {
    menor.incrementar();
    notifyAll();
    }
}
}
```

c. (0.4) Responda con justificación ¿Hay un framework en esta solución? ¿Qué clases lo componen? Cómo se implementa el control invertido o principio de Hollywood. Respuestas sin la justificación correcta no serán válidas.

Respuesta. Si hay un framework, el observer, el observable y la interface runnable permiten una ejecución concreta de elementos abstractos que son observados, observan y corren con control propio. El reloj (todas sus clases) es una derivación de ese framework, no tiene el control de ejecución, ni de la notificación de observados a observadores, este control se mantiene en el framework. Así el framework llama al código de la aplicación y no viceversa. Las clases que lo implementan son Observer, Observable y la interface Runnable. El framework ejecuta comportamiento el completo y dentro de éste ejecuta los métodos definidos en la aplicación (control invertido) y no al revés, esto lo realiza gracias a la posibilidad polimórfica de los mensajes en la orientación a objetos.

d. (0.3) Programe en su lenguaje orientado a objetos preferido la clase Observable (atributos, relaciones y métodos) respetando fielmente el diseño. Programe internamente sólo el método notifyAll (), los demás métodos déjelos implementados pero su parte interna déjela en blanco y retornando un valor por defecto (null, 0, etc.) si le corresponde retornar algún valor.

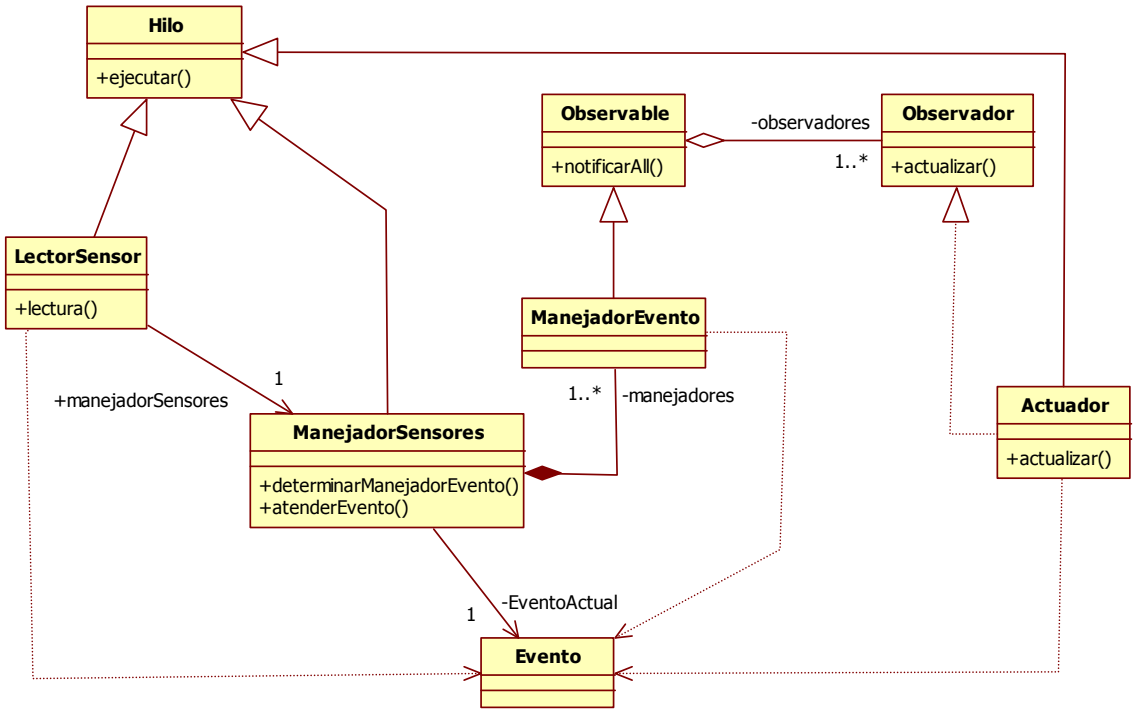
Respuesta, se acepta la implementación con vectores, arreglos o cualquier estructura de datos …

```
public class  Observable{
        private Vector <Observer> observers;

        public void addObserver(){}
    public void notifyAll(){
            for (Enumeration <Observer> e = observers.elements();

    e.hasMoreElements;)
             e.next().upDate();
        }

}
```

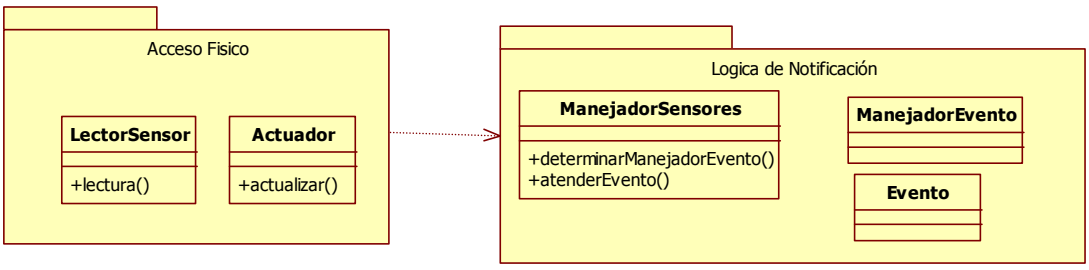
Parte B. Caso de Desarrollo 2

Se está construyendo un sistema para mejorar la seguridad de las viviendas. El sistema usa sensores de humedad, humo, robo, etc, para determinar la presencia de algún evento alarmante. De ocurrir un evento alarmante y dependiendo del tipo de evento, se activan alarmas de robo, incendio, inundación, etc. Además se siguen una serie activaciones que son importantes, por ejemplo si el evento alarmante es de robo, se cierran las puertas, no hay activación de alarmas (para no alarmar al ladrón) y se llama a la policía silenciosamente. Si es incendio se activan las alarmas, se abren las puertas y se llama a la policía. Para el caso de uso "Atender un evento" se ha desarrollados el diagrama de secuencia que se presentado en la figura 3. En general para un conjunto de condiciones de alarma, ocurrirá una serie de activaciones que depende de cada vivienda, sus amenazas y la capacidad económica para hacer una instalación más o menos sofisticada. De acuerdo a esto responda:

1. Diseño del sistema
    a. Proponga un modelo de clases para la secuencia propuesta (1.0). Tenga en cuenta
        i. Desde el objeto lectorSensor viaja con el mensaje un objeto de la clase Evento con la información del evento disparado.
        ii. Las flechas sin puntas rellenas son mensajes asincrónicos, es decir no se espera respuesta ni que se ejecute la operación respuesta de mensaje.
        iii. Las flechas con puntas rellenas son los mensajes tradicionales sincrónicos (como los del reloj antes expuesto) donde se cede el control.
        iv. Los lectores de sensores y los actuadores son autónomos en ejecución, por eso deben heredar de la clase Thread (Hilo) y deben implementar un método run(), es decir ejecutar.
        v. La flexibilidad para agregar y notificar actuadores dependiendo del evento.



b. Proponga una vista de módulos como parte de la arquitectura lógica. Tenga en cuenta como principal atributo de calidad motivador, la modificabilidad del sistema. (0.5)



2. Implementación del Sistema

Programe las clases a las que pertenezcan los objetos gestorDeActuadores, actuador Alarma y actuadorLíneaTelefonica, de acuerdo al diagrama de secuencia, a la información suministrada en el punto 1 y al modelo de clases obtenido en el punto 1. (0.5)

```
public class manejadorEvento extends observable{
    }

public class Actuador extends Hilo{
```

```
  public void Activar(){
                this.run(); // en run deberá estar la lógica de la actuación
            }

}

Hasta aquí la solución a este punto …
-------------------------------------------------------------------------

Para los que implementaron el Manejador de Sensores (parte de la solución
alternativa)

public class ManejadorSensores{
    private Vector misManejadores;
     Evento eventoActual;


   public activarEvento(Evento e){
            eventoActual = e;
            run();
             }
 public void run(){
          atenderEvento(eventoActual);
}
Public void atenderEvento(Evento e){
         ManejadorEvento manejadorEvento := determinarManejadorEvento();
        manejadorEvento.notifyAll();
}
```

3. Transformación de modelos. Dado que un modelo de clases a nivel de diseño tiene la información suficiente para su implementación suena lógico que esto pueda automatizarse mediante una transformación de modelos a texto (Model2Text), por ejemplo "De UML a texto java". De acuerdo a los enfoques presentados en la lectura "Literature Study on Model Transformations", sustente cual sería el tipo de lenguaje que más se ajustaría para la implementación de dichas transformaciones. De un pequeño ejemplo de transformación en alto nivel (mapeando, programando o reemplazando). (0.5)

   Respuesta: transformación basada en plantillas (Templates). Por ejemplo

   En una clase UML extraer Nombre de la Clase, Lista de sus superclase y Lista de las ainterfaces que implementa y reemplazar los <> de acuerdo a las cadenas de caracteres en la siguiente plantilla:

   public class <NombreClase> extends <NombreSuperClase1>, <NombreSuperClase2>, … , implements <NombreInterface1>, <NombreInterface2>, …..{

   //Así para el resto de elementos de la clase, vale el ejemplo si es un atributo o una //operación

   }

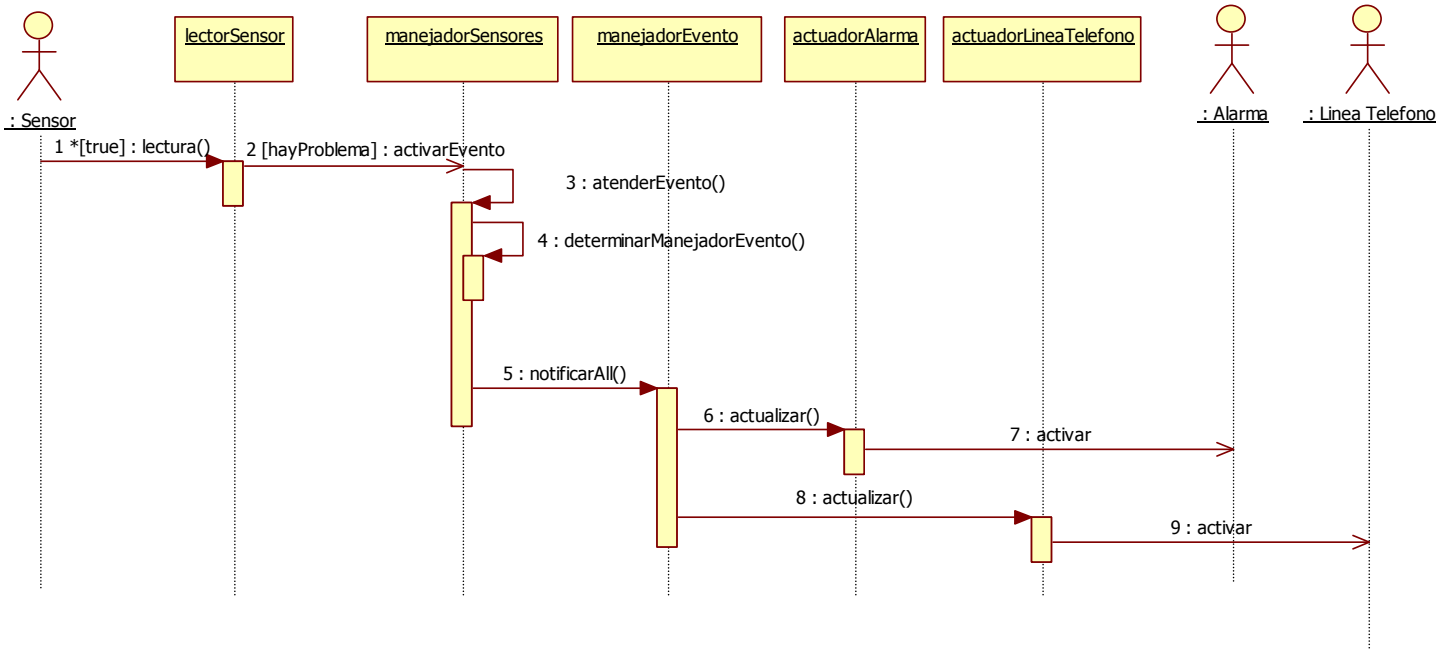


Figura 3. Diagrama de secuencia del caso de uso Atender Evento