Práctica 03 – Gestión de Alarmas

Zdravko Dimitrov Arnaudov

Sara Grela Carrera

1.- Completar diagramas en Magick Draw:

Hemos completado el diagrama de estados que quedaba por terminar y hemos creado el modelo de construcción en el mismo proyecto, cambiando el diagrama de clases según el patrón State. Para ello, hemos creado la clase abstracta "AlarmasState", que tiene un hijo por cada estado en el que puede estar el sistema (en este caso, "desprogramado", "programado" y "sonando"). De esta forma conseguimos que el conjunto de alarmas sea independiente del estado en el que se encuentra el sistema.

2.- Adaptación del diseño a implementación aplicando el patrón State.

Siguiendo el diagrama de clases realizado, hemos creado las clases "Alarma", "Alarmas", "AlarmasState", "Desprogramado", "Programado" y "Sonando".

- Clase "Alarma": contiene la información de una alarma (su nombre y hora a la que sonará) y métodos para su gestión.
- Clase "Alarmas": contiene dos colecciones de alarmas (unas activas y otras desactivadas), métodos relativos a las señales que recibe el sistema ("nueva alarma", "alarma off", etc.) y métodos auxiliares necesarios ("existe alarma", "busca alarma", etc.).
- Clase "AlarmasState": contiene un atributo por cada estado y los métodos relativos a las señales que recibe el sistema.
- Clases "Desprogramado", "Programado" y "Sonando": son las clases hijas que extienden los métodos de la clase padre "AlarmasState", pero solo implementan aquellos métodos a los que responden. Las dos últimas clases necesitan un Timer (junto a una TimerTask) para gestionar los eventos temporizados.

Antes de probar la implementación, haremos una interfaz gráfica sencilla "MainWindow" con JavaSwing.

3.- Aplicación del patrón MVC:

Teníamos creadas las clases pertenecientes al modelo y a la vista, pero no estábamos aplicando este patrón correctamente. Separamos las clases en tres paquetes:

- Paquete "modelo": incluye las clases mencionadas en el anterior apartado exceptuando la interfaz gráfica. Creamos una interfaz "IModelo", de la que la clase "Alarmas" implementará sus métodos (los mencionados anteriormente).
- Paquete "vista": incluye la interfaz gráfica "MainWindow", que implementará los métodos de la interfaz "IVista", y el programa principal "AlarmasMVCMain".
- Paquete "controlador": incluye la clase "AlarmasControlador", que hace de intermediario entre la vista (IVista) y el modelo (IModelo) invocando los métodos necesarios en cada uno para cada señal que se reciba.

4.- Gestión de errores:

1- No se debería poder añadir una nueva alarma programada si su hora es anterior a la hora actual o si ya existe otra alarma programada para esa misma hora:

En la clase controladora, para el método "nueva alarma", controlamos primero si la hora de la alarma a añadir es anterior a la hora actual. Si no es así, controlamos que la hora de la alarma a añadir no se

encuentre ya en alguna de las alarmas existentes (tanto activas como desactivadas). En caso de cumplir todos estos requisitos, podrá añadirse correctamente.

2- Una alarma cuya hora ya ha pasado, si la desactivamos y posteriormente la volvemos a activar, debería activarse para el día siguiente:

Ejemplo: Alarma1 programada para las 12:00 del 01/04/2021. La alarma suena. Desactivamos la alarma. La volvemos a activar. Como ya han pasado las 12:00 del día indicado, debería estar activada para las 12:00 del 02/04/2021.

En la clase controladora, para el método “alarma on”, controlamos que, si esta situación se da, programamos la fecha de la alarma indicada para el día siguiente.