TimeTracker

Diseño del Software

2020-2021

Daniel Cendagorta-Galarza Friend (NIU: 1459480)
David Martí Zaragoza (NIU:1394933)
Joan Mata Pàrraga (NIU: 1533089)
Ismael Pajuelo Berjano (NIU: 1456941)

Índice:

1.- Introducción:

Durante estas últimas semanas hemos estado trabajando en la creación del software del TimeTracker. Uno de los principales inconvenientes que nos encontramos al inicio fue que nos incorporamos a la asignatura en diferentes tiempos y eso retrasó la formación del grupo y complicó la coordinación al principio.

Basándonos en el UML, hemos estructurado nuestro proyecto principalmente mediante cinco clases: *Reloj, Actividad, Proyecto, Tarea e Intervalo*. Estas clases son las que implementan la mayoría de las funcionalidades del programa. Tanto *Proyecto*, como *Tarea*, heredan de la clase *Actividad* y comparten la gran mayoría de variables, funciones y métodos, a excepción de algunas. *Intervalo* existe a partir de *Tarea*, ya que para que exista un intervalo se debe crear primero una tarea.

La clase *Proyecto* contiene una lista de *Tareas*. Ésta, a su vez, contiene una lista de *Intervalos*. La clase *Intervalo* está basada en la clase *Observer* de java. Una tarea está creada para ser comenzada. Desde que la comienzas hasta que la acabas o la pones en pausa, sucede un intervalo de tiempo. Esa es la idea que hemos querido implantar con este método.

Cierto es, que para que una clase sea observadora, necesitamos otra a la que observar. La clase *Reloj*, realiza dicha función ya que está basada en la clase *Observable* de java. Su función principal es decirnos qué hora es la actual cada un cierto periodo de tiempo. El *Intervalo* ve dicha información y la almacena.

2.- Estilo de codificación:

En la codificación del código intentamos que las variables y las funciones tengan nombres claros y relacionados con lo que hacen. Las variables tienen una letra al principio que está relacionada con la clase a la que pertenecen. Por ejemplo, las variables de la clase *Proyecto* siempre comienzan por "p_", las de la clase *Tarea* por "t_", clase *Actividad* por "a_" y la clase *Intervalo* por "i_".

Con el idioma de las variables también tratamos de dar información. Las escritas en castellano son variables "estáticas", es decir, son aquellas variables que pertenecen a una clase y puedes llamar desde todas las funciones internas de la clase. Si se modifica su información, se varía la información de la clase y con ello la información del proyecto completo. En cambio, las variables en inglés son variables "volátiles", es decir, pertenecen únicamente a la función donde se mencionan. Al finalizar dicha función, la variable se pierde.

El main está hecho para poder tener todos los test juntos y no tener que salir del proyecto y volver a compilar para ejecutarlo de nuevo. Joan lo configuró para que lo primero que pregunte el programa al compilar sea que hito de entrega (milestone) desea ejecutar. Como se han estipulado 3 desde la gestión de la asignatura, si pulsa la tecla "1" realizará el primero y así sucesivamente. La tecla "0" es la que utilizamos nosotros los miembros del grupo para realizar las pruebas para verificar que funcionan las clases, funciones y variables tal y como queremos. En caso de pulsar cualquier otra número, te enseña un mensaje de error y vuelve a preguntar si quieres hacer algún test. Para salir del programa hay que escribir "-1" y se cierra.

Dentro del main las variables intentan seguir el mismo estilo que en el resto del proyecto aunque con algunas peculiaridades. En cada apartado, de cada milestone, se definen las variables con un "m1_", "m2_", o "m3_" previo para definir a qué hito corresponden y luego al compilar no hayan problemas. Si es cierto que supone un pequeño uso extra de memoria.

3.- Trabajo realizado:

Lo primero que se realizó por parte de David y Daniel fue la idea principal del UML. En el que se podía apreciar la estructura de clases que queríamos que cogiera nuestro proyecto y algunas de las que pensamos que serían sus funciones. Llevó varias iteraciones conseguir un UML correcto pues al principio nos tuvimos que repasar las distintas relaciones y nomenclaturas de diagramas de clases, pero finalmente alcanzamos un buen punto de salida para empezar a programar.

Daniel estuvo trabajando con la clase *Reloj*, ya que es una de las herramientas principales del programa. En la implementación de esta clase nos dimos cuenta de que todavía no teníamos los conceptos muy claros respecto a los patrones de diseño Observable-Observer y Singleton, ya que tuvimos que modificar varias veces el código para añadir las funcionalidades propias de estos patrones de diseño. Mientras tanto, Joan comenzó a darle forma a las clases con las variables básicas y necesarias para crear dichas clases. El primer problema que nos retrasó fue que no conseguimos unir los diferentes archivos de las clases como un simple proyecto. Después de largas pruebas, intentos y con la ayuda del comando “package” conseguimos generar un proyecto.

Otro error que nos causó un gran revés sucedió a la hora de crear los constructores de la clase *Actividad* y de sus extensiones que por falta de conocimientos en java no sabíamos cómo se codificaban. Posterior a un exhaustivo buceo por las páginas del “Buscador de Google”, por parte de Daniel y Joan, conseguimos dar con la solución al problema y seguir adelante.

Conseguimos tener codificado los esqueletos de las clases y del reloj. Una vez ahí, todos los componentes del grupo, comenzamos a generar las diferentes funciones y con ellas, añadir variables, listas, etc., todo aquello que vemos necesario y útil para nuestro proyecto.

La jerarquía del proyecto fue codificada por Joan junto con el proceso de muestreo por pantalla de los intervalos y duraciones de tiempo de las respectivas variables. Eso supuso el mayor inconveniente del proyecto, ya que conectar el *Reloj*

con el *Intervalo* nos supuso una gran contienda. Una de las soluciones que se tomó fue crear otra clase *Reloj* extendida de la clase *Thread* para, al menos, poder acabar la jerarquía y el muestreo y comprobar que todo funcionaba de manera correcta.

Una vez estuvo hecho se acabaron de pulir algunas codificaciones y funciones, mientras seguíamos “dándole vueltas” a la aplicación del *Reloj* correcto. El *Reloj* auxiliar fue sustituido en cuanto acabó su función de apoyo.

Una vez comprobamos que la jerarquía del proyecto funcionaba correctamente, Daniel codificó el patrón de diseño Observable en la clase *Reloj* y David codificó el patrón de Observer para la clase *Intervalo*. A su vez, entre Daniel y Joan, se creó la implementación de la clase *Runnable* por parte de la clase *Reloj* para que esta pudiera ser ejecutada en un *Thread* aparte. Esto nos llevó a un buen “comedero de cabeza”, debido a la manera en la que se debía declarar el nuevo *Thread* y a su vez mantener la funcionalidad de la clase *Observer* de añadir observadores. El principal problema que tuvimos fue que al crear el objeto *Thread* no podíamos acceder a las funciones de la clase *Reloj* ni *Observable.* Finalmente, nos dimos cuenta de que debíamos acceder a estas funciones mediante la uniqueInstance del reloj y así fue.

4.- Conclusiones:

En general el proyecto nos ha costado, en primer lugar debido a la extraña formación del grupo, lo cual retrasó un poco el comienzo de la codificación. En segundo lugar la falta de compenetración entre los compañeros, ya que no conseguimos entendernos bien y eso nos llevó a perder tiempo codificando funciones y clases mal pensadas. Además, por una falta de organización y comprensión, sobreescribimos funciones de otros compañeros que estaban bien implementadas, lo que nos llevó a perder aún más tiempo.