[Universidad de Burgos]

[Informe sobre aplicación KANBAN]

[DISEÑO Y MANTENIMIENTO DEL SOFTWARE]

Mario Flores Espiga
Sergio López Bueno

TABLA DE CONTENIDO

OBJETIVOS

El proyecto consistirá en la elaboración de una **pequeña aplicación de tablones kanban para scrum**, a realizar en **grupos de dos personas**. El lenguaje de implementación será **Java**.

Se realizarán **dos entregas** a través de un **repositorio online** (GitHub, GitLab...), los días **25 de noviembre** y **23 de diciembre**, con un peso del **15%** y el **25%** respectivamente sobre la nota final de la asignatura. Si los commits están marcados como *primera* y *segunda entrega* (ambos anteriores a sus respectivas fechas límite), se usarán éstos como entregas. Si no hay ninguno etiquetado, se usará el último commit antes de su vencimiento (aunque se trate de commits intermedios de desarrollo). **La memoria debería estar también disponible en el repositorio**. En la primera entrega se valorará la implementación de las funcionalidades esenciales y el proceso de diseño hasta llegar a ellas (10 primeras historias de usuario; ver más abajo). En la segunda entrega se valorarán el diseño e implementación finales (todos los requisitos, además de las correcciones sugeridas en la entrega anterior). En ella se implementará el punto 11 o el 12 (a elegir por cada grupo); de ambos 11 y 12 se documentará el diseño y por qué facilita su futura extensión, y además se dará la implementación de uno de ellos. Se tomará como punto de partida el repositorio https://github.com/Kencho/ubu-gii-dms-po1c, del que cada equipo hará un *fork* cuando lo indique el profesor para poder trabajar sobre su propia copia independiente.

Sobre la lógica de negocio: La aplicación debería permitir un uso simplista de la metodología/orientaciones *Scrum*, en el que se definen **tareas** asociadas a **historias de usuario y/o defectos** a corregir, representadas físicamente en tarjetas. Cada una de estas tareas se encuentra en un estado concreto dentro de un *backlog*. Hay dos tipos de backlogs: **Product backlog**, con las tareas definidas, pero no asignadas a un ciclo concreto (*sprint*), y los **Sprint backlog**, o *backlogs* de cada iteración específica. Las tareas en el *Product backlog* sólo estarán en un estado *pendiente*, mientras que en los *Sprint backlog* pueden estar como *pendientes ("To do")*, *en proceso ("Doing")*, *en validación ("QA/Testing")*, o *completadas ("Completed" o "Finished")*. Al planificar la siguiente iteración, se extraen tareas del *Product backlog* y se colocan en el *Sprint backlog* que se está definiendo, como *pendientes*. Durante el *sprint*, estas tareas se irán moviendo entre estados hasta que se completen. Todos los *sprints* tendrán una **duración equivalente**, y cada uno tendrá asociada una **fecha de inicio**. Además del requisito asociado, cada tarea debe tener al menos los siguientes campos: **Título**, **descripción**, **coste**, **beneficio**, y **miembro asignado**.

Requisitos de la aplicación en forma de historias de usuario:

1. Como **evaluador** quiero **acceso al código y su historial** para **ver la implementación**
2. Como **evaluador** quiero **una memoria** para **evaluar el uso de buenas prácticas de diseño y patrones de diseño**
3. Como **evaluador** quiero **instrucciones** para **poder construir y lanzar la aplicación**
4. Como **cliente** quiero **poder añadir miembros al equipo**
5. Como **cliente** quiero **poder añadir sprint backlogs para las nuevas iteraciones**
6. Como **cliente** quiero **poder añadir tareas al product backlog**
7. Como **cliente** quiero **poder mover tareas del product backlog al sprint backlog**
8. Como **cliente** quiero **poder mover tareas entre las fases del ciclo de vida**
9. Como **cliente** quiero **poder editar las tareas**
10. Como **cliente** quiero **que los backlogs y tareas se mantengan cuando vuelva a abrir la aplicación**
11. Como **cliente** quiero **que la aplicación pueda tener nuevas formas de guardar el modelo en el futuro (implementar una si no se implementa el punto 12)**
12. Como **cliente** quiero **que la aplicación pueda tener nuevas formas de interacción en el futuro (implementar una si no se implementa el punto 11)**

REALIZACIÓN

Se ha seguido el esquema subido en github, con alguna modificación de carácter leve. Se ha utilizado el modelo-vista-controlador mas un componente de persistencia.

Modelo:

Backlog

Clase padre de la que heredan SprintBacklog y ProductBacklog

Defecto

Clase que hereda de Requisito

HistoriaDeUsuario

Clase que hereda de Requisito

MiembroDeEquipo

En esta clase se ha codificado a un usuario de la aplicación KANBAN.

Como atributos tiene su nombre, edad y DNI, que actuará de cara a la persistencia como clave primaria.

Los métodos de la clase son simplemente getters y setters para los distintos atributos.

ProductBacklog

Hereda de Backlog y representa el lugar donde están las tareas antes de ser movidas a un sprint.

Las tareas se almacenan en una lista.

Tiene métodos para devolver la lista completa de las tareas o en su defecto una tarea específica, pasando como parámetro la referencia a la tarea.

Requisito

Clase que representa un requisito asociado a una tarea.

Se compone simplemente de un String que almacena el texto del requisito.

SprintBacklog

Clase similar a ProductBacklog.

Contiene un método para imprimir las distintas tareas almacenadas y su estado, puesto que en SprintBacklog a diferencia de ProductBacklog las tareas pueden estar en diferentes estados (TODO, DOING, TEST, FINISHED)

Tarea

En esta clase se codifican las tareas del tablero KANBAN.

Una tarea tiene los atributos título, descripción, coste, beneficio, asignadoA (Miembro al que esta asignada la tarea) y requisito.

Además, para la implantación del estado de las tareas se usa un tipo enumeración.

Tiene getters y setters para todos los atributos, y un método toString que imprime el título de la tarea a efectos de identificación.

Vista:

Vista

Clase que se encarga de pedir al usuario las diferentes opciones que están disponibles en la aplicación.

Controlador:

Gestor

Clase que se encarga de inicializar todas las clases necesarias para el funcionamiento del programa. Es la clase que contiene el main que inicia el programa.

Persistencia:

Guardado

Clase que almacena el tipo de dato guardado y la estrategia de guardado. Tiene el método SaveToCSV que hará la llamada correspondiente para guardar los datos dependiendo de la estrategia a seguir. Corresponde a la clase “Contexto” del patrón estrategia.

TipoDatoGuardado

Clase abstracta que contiene el método abstracto SaveToCSV que implementarán sus hijos. Corresponde a la clase “Estrategia” del patrón estrategia.

MiembroCSV

Clase que se encarga de implementar el método SaveToCSV que guardará los datos de los miembros. Hereda de TipoDatoGuardado, se corresponde a la clas “EstrategiaConcreataA” del patrón estrategia.

TareReqCSV

Clase que se encarga de implementar el método SaveToCSV que guardará los datos de las Tareas y de los Requisitos. Hereda de TipoDatoGuardado, se corresponde a la clas “EstrategiaConcreataB” del patrón estrategia.

CargadorDeDatos

Clase que se encarga de cargar todos los datos de la aplicación. Contiene los métodos necesarios para cargar cada uno de los objetos del programa.

PERSISTENCIA

Para la persistencia se ha optado por utilizar lectura y escritura en archivos CSV debido a su sencillez tanto para su programación como su uso frente a soluciones más complejas como el acoplamiento de una base de datos.

En concreto se utiliza un archivo para cada tipo de entidad, con una línea para cada entidad concreta, guardando sus atributos separados por un carácter definido dentro del código.

También se podría haber usado una opción similar mediante archivos xml o json, de esta manera nos podríamos haber evitado la creación de tantos archivos al guardar diferentes clases con sus atributos en el mismo documento.

Otras opciones más complejas que conllevarían trabajar con bases de datos podrían ser las siguientes:

La primera forma de persistencia con BD que se podría aplicar a este programa es mediante JDBC guardando "manualmente" mediante sentencias JDBC los datos en una base de datos. Para ello añadiríamos otra clase que heredara de TipoDatoGuardado y crearíamos la nueva forma de guardado en el método SaveToCSV (habría que cambiarle el nombre a otro más genérico).

También se podría pensar hacerlo mediante JPA Hibernate ya que se guardarían los datos de manera mas natural y automática al gestionar Hibernate las relaciones entre clases. Lo negativo de esta nueva forma es que habría que añadir a prácticamente todas las clases cuáles son sus relaciones con las demás clases.

Lamentablemente por falta de tiempo no ha sido posible la realización de la nueva forma de persistencia.

PATRONES DE DISEÑO

Debido a las diferentes formas que hay de guardar los diferentes datos (en diferentes archivos), se ha decidido utilizar el patrón estrategia para que una misma clase realice las diferentes formas de guardar los datos. Esto también nos ayudará en un futuro a la hora de escalar la aplicación hacia nuevas formas de persistencia ya que solo necesitaremos añadir las nuevas estrategias (EstrategiasConcretas) a las clases ya creadas.

INSTRUCCIONES DE USO

Para ejecutar la aplicación basta con importar el proyecto java y ejecutarlo. La clase Gestor.java es la que contiene el main principal del programa.

Una vez ejecutado solo hay que seguir los pasos que indica el programa en modo texto.

Al iniciar el programa te da una serie de opciones para poder introducir los datos, simplemente se debe introducir el número correspondiente a la opción elegida. (Hay que tener cuidado con el orden en el que se realizan las cosas ya que no se ha implementado el control de errores. Por ejemplo, fallaría al crear una tarea antes de crear al menos un miembro ya que no hay miembros a los que asignar la tarea.)

Para salir del programa tienes dos opciones, la primera es salir guardando los datos en archivos CSV que se crearán en la raíz del proyecto y la segunda que es salir sin guardar. Los datos se cargan automáticamente al iniciar el programa, pero existe la opción de borrar los datos (de la sesión actual, cambios solo aplicables si se sale guardando los datos).