



DISEÑO Y MANTENIMIENTO DEL SOFTWARE

4º GRADO DE INGENIERÍA INFORMÁTICA

PRÁCTICA OBLIGATORIA

APLICACIÓN SCRUM

ALUMNOS:

Yi Peng Ji

Alicia Olivares Gil

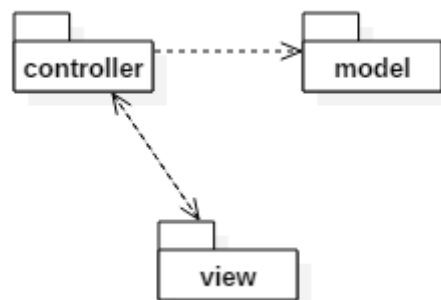
Contenido

1.	Estructura principal del programa. Patrón MVC.....	3
2.	Modelo	3
a)	Datos que maneja el programa.....	3
b)	Persistencia	5
c)	Patrón Estado para la clase Tarea	6
3.	Vista.....	6
4.	Controlador	7
a)	Patrón Estado para la clase Controlador.....	7

1. Estructura principal del programa. Patrón MVC

El programa se divide en 3 paquetes principales correspondiendo con los participantes del patrón arquitectónico Modelo-Vista-Controlador:

- **model**: contiene la estructura de los datos que maneja el programa y se encarga del acceso a la persistencia.
- **view**: corresponde con la interfaz de la aplicación, la parte del programa con la que interactúa el usuario.
- **controller**: responde a los eventos generados por la interacción del usuario con la vista y realiza peticiones al modelo. Se puede ver como un intermediario entre la vista y el modelo.

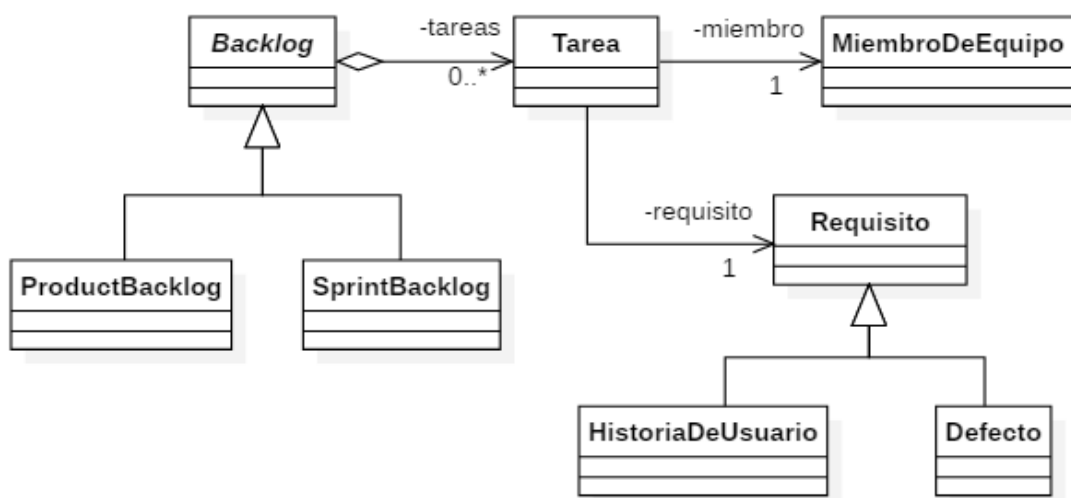


2. Modelo

En el **paquete model** se incluyen las entidades necesarias para representar los datos que emplea el programa y el acceso a la persistencia.

a) Datos que maneja el programa

La estructura básica de los datos se define con las siguientes relaciones entre entidades:



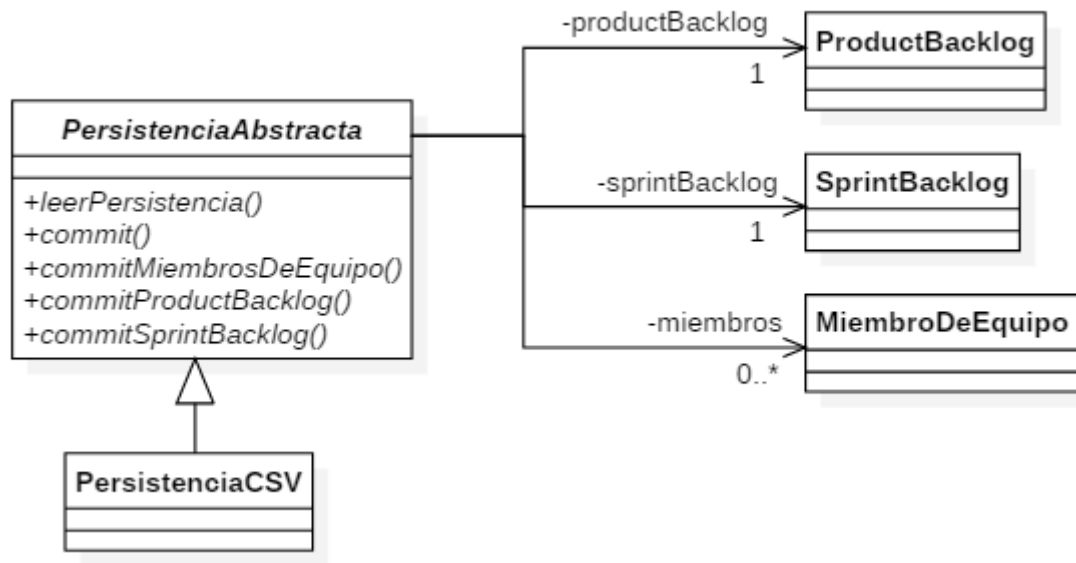
- Una clase abstracta **Backlog** que contiene un Mapa de Tareas y que implementa los mecanismos para añadir tareas y acceder a ellas.
- Una clase concreta **ProductBacklog** que extiende la clase Backlog y que contiene las tareas pendientes que no han sido añadidas al Sprint actual. Esta clase implementa el **patrón Singleton** ya que solo se necesita una instancia a la que se debe tener un acceso global.
- Una clase concreta **SprintBacklog** que extiende la clase Backlog y que contiene las tareas que han sido añadidas al Sprint actual. Además, contiene dos atributos adicionales:

- **descripcion:** Descripción del sprint.
- **fechalnicio:** Fecha en la que se inicia el sprint.

Esta clase también implementa el **patrón Singleton** ya que solo existirá un Sprint al mismo tiempo.

- Una clase **Tarea** con los siguientes atributos:
 - **idTarea:** Identificador entero de la tarea.
 - **coste:** coste entero de la tarea.
 - **beneficio:** Beneficio entero de la tarea.
 - **requisito:** Objeto de la clase Requisito asociado. Cada Tarea se asocia a un único requisito.
 - **miembro:** Objeto de la clase MiembroDeEquipo asociado, corresponde con el miembro al que se le asigna esa tarea. Cada tarea se asigna a un único miembro.
 - **estado:** Objeto de la clase EstadoTarea asociado, corresponde con el estado de la tarea dentro del ciclo de scrum (pendiente, en proceso, en validación, completada). Decidimos usar un objeto en lugar de un atributo entero o String para implementar el patrón Estado que gestione las transiciones de la tarea entre los distintos estados.
- Una clase **MiembroDeEquipo** con un atributo nombre que representa a un miembro del equipo de desarrollo.
- Una clase **Requisito** con los siguientes atributos:
 - **titulo:** título del requisito (o de la tarea, ya que la relación es 1 a 1).
 - **descripcion:** descripción del requisito (o de la tarea).
- Una clase **Defecto** que extiende la clase Requisito y añade un nuevo atributo:
 - **tarea:** identificador entero de la tarea anterior con la que se relaciona dicho defecto.
- Una clase **HistoriaDeUsuario** que extiende la clase Requisito y añade un nuevo atributo:
 - **actor:** Actor al que se asocia la historia de usuario (p.e. administrador, cliente, programador...).

b) Persistencia



Implementamos la persistencia mediante ficheros en **formato csv**. Para ello contamos con una clase **PersistenciaCSV** que extiende la clase abstracta **PersistenciaAbstracta**. La lectura y escritura en la persistencia se realiza en la implementación en **PersistenciaCSV** de los métodos abstractos de **PersistenciaAbstracta**. Para la interacción con los ficheros de persistencia empleamos la librería **opencsv-4.3.2**.

La clase **Persistencia CSV** implementa el **patrón Singleton** ya que solo se necesita un acceso a la persistencia al que acceder de forma global.

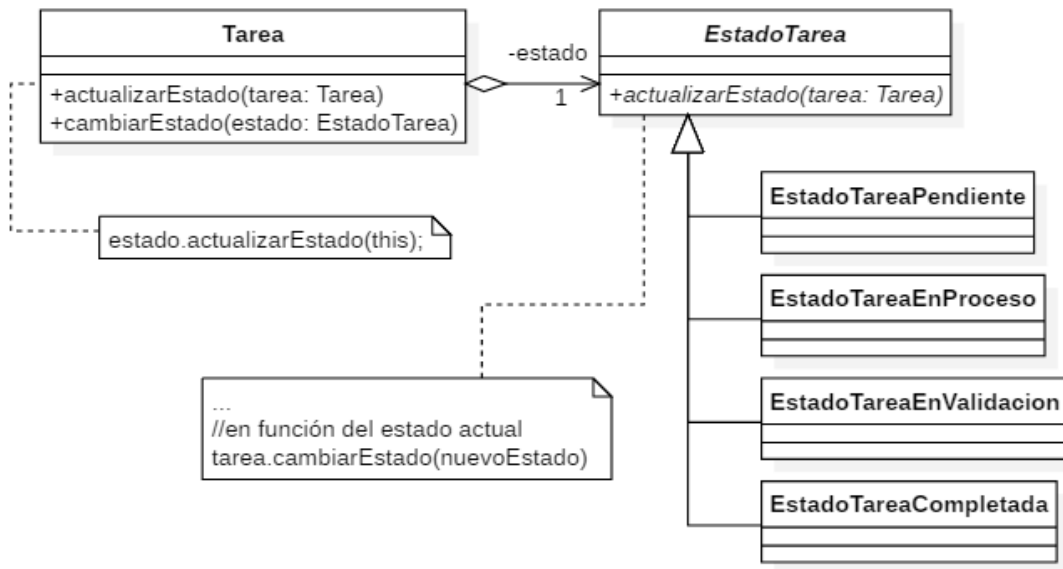
La clase **PersistenciaAbstracta** contiene las relaciones hacia los datos dinámicos que maneja el programa:

- **productBacklog**: contiene las tareas que no se han asignado a un sprint.
- **sprintBacklog**: contiene las tareas del sprint actual
- **miembros**: Mapa de todos los miembros del equipo, se les haya asignado o no una tarea.

Decidimos incluir la clase **PersistenciaAbstracta** por encima de **PersistenciaCSV** para permitir la opción de implementar nuevas formas de persistencia en el futuro. Para ello solo se debería añadir una clase concreta que extendiera la clase **PersistenciaAbstracta** e implementase sus métodos de forma adecuada a la nueva forma de persistencia.

c) Patrón Estado para la clase Tarea

Para gestionar las transiciones entre los estados de una tarea decidimos emplear el patrón Estado. Cada objeto Tarea tiene un **EstadoTarea** asociado que además de indicar su estado actual, define el siguiente estado al que se moverá al actualizar su estado. Todos los EstadoTarea concretos implementan el **patrón Singleton**, ya que, dado que no tienen estado interno, cada instancia puede ser compartida por varios objetos Tarea sin que perjudique al correcto funcionamiento.



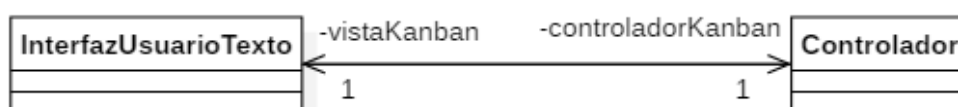
Permitimos las siguientes transiciones:



Si una tarea en validación pasa a estar en proceso o completada será decisión del usuario. El resto se moverán al siguiente estado definido.

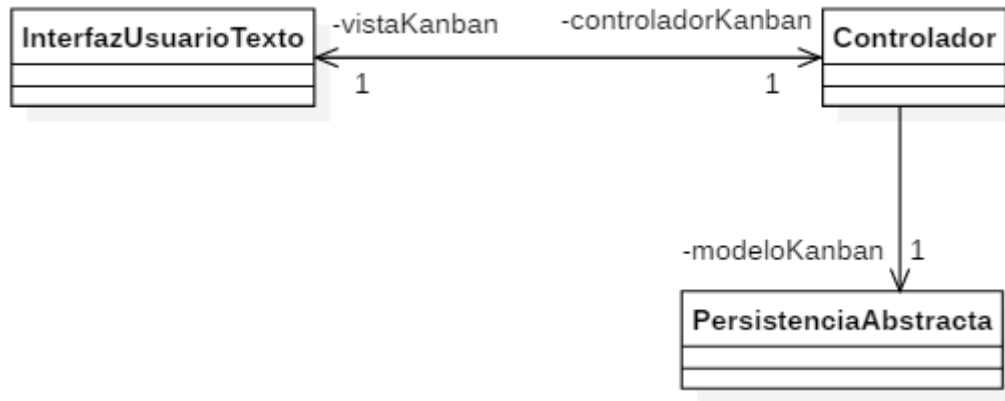
3. Vista

El paquete view contiene la implementación de la interfaz de usuario en modo texto mediante la clase **InterfazUsuarioTexto**. Esta clase tiene una referencia al controlador y un método por cada submenú a imprimir, además de métodos auxiliares para pedir las acciones al usuario.



La navegación es bidireccional ya que la vista hace peticiones a través de los métodos del controlador, y el controlador debe poder llamar a las funciones que imprimen los submenús.

4. Controlador



El paquete controller contiene la clase **Controlador**, encargada de dos cosas:

- transmitir las peticiones de la vista al modelo (datos dinámicos y persistencia) a través de algunos métodos definidos como `anadirNuevoSprint`, `moverTarea`, `anadirTareaSprint`, `anadirTarea`, `modificarTarea`...
- gestionar las transiciones entre los distintos submenús de la vista.

a) Patrón Estado para la clase Controlador

Para gestionar las transiciones entre los distintos submenús de la interfaz de texto decidimos emplear el **patrón Estado**. Definimos una clase **EstadoControlador** por cada submenú en la cual se definirá la transición al siguiente EstadoControlador, es decir, al siguiente submenú a mostrar.

Cada EstadoControlador concreto da su propia implementación de dos métodos:

- **mostrarMenu**: llama al método de la interfaz que muestra el submenú correspondiente a ese estado, el cual recibirá las acciones del usuario.
- **actualizarEstado**: modifica el estado del controlador al siguiente estado en función del estado actual y de las acciones del usuario.

La ejecución principal del programa consistirá en ejecutar estos dos métodos en ese orden hasta la finalización del programa.

Además, cada EstadoControlador concreto implementa el **patrón Singleton**. Esto no provoca problemas ya que durante la ejecución del programa solo existirá una instancia de Controlador (al ser también Singleton).

