


| <div>  <div> DEPARTAMENTO Lenguajes y Ciencias de la Computación </div> </div> <div> Programación Orientada a Objetos (17 de mayo de 2021) </div> | |
|---|--|
| APELLIDOS, Nombre | <div>TITULACIÓN Y GRUPO</div> <div>MÁQUINA</div> |

NOTAS PARA LA REALIZACIÓN DEL EJERCICIO:

- Al inicio del contenido de cada fichero realizado deberá aparecer un comentario con tus apellidos y nombre, titulación y grupo.
- Los diferentes apartados tienen una determinada puntuación. Si un apartado no se sabe hacer, *no debes pararte en él indefinidamente*. Puedes abordar otros.
- Está permitido:
 - Consultar los apuntes (CV), la API (Internet), la guía rápida de la API (CV).
 - Añadir métodos privados a las clases.
- No está permitido:
 - Intercambiar documentación con otros compañeros.
 - Recibir ayuda de otras personas. Se debe realizar personal e individualmente la solución del ejercicio propuesto.
 - Añadir métodos no privados a las clases.
 - Añadir variables o constantes a las clases.
 - Modificar la visibilidad de las variables, constantes y métodos que aparecen en el diagrama UML.
 - Modificar el código suministrado.
- Una vez terminado el ejercicio, debéis subir (a la tarea creada en el campus virtual para ello) un fichero comprimido de la carpeta src que hayáis realizado y usáis vuestros apellidos y nombre para el nombre del mismo (Apellido1Apellido2Nombre.rar o .zip).
- La evaluación tendrá en cuenta la claridad de los algoritmos, del código y la correcta elección de las estructuras de datos, así como los criterios de diseño que favorezcan la reutilización.
- Para la corrección del ejercicio se utilizarán programas de detección de copias/plagios.
- Con posterioridad a la realización del ejercicio, el profesor podrá convocar a determinado/as alumno/as para realizar entrevistas personales sincrónicas con objeto de comprobar la autoría de las soluciones entregadas.
- Una vez terminado el ejercicio subir, a la tarea creada en el campus virtual para ello, un solo archivo con los ficheros fuente (.java) que hayáis realizado.

Acaban de contratar a los alumnos de esta asignatura para el desarrollo de un importante proyecto. Para organizar bien el proyecto se quiere gestionar de forma ágil con un tablero Kanban que permite visualizar de forma sencilla el estado de las tareas. Este sistema se inventó por Toyota en los años 40 y es parte esencial de las nuevas metodologías Lean. El funcionamiento de tableros se suele hacer con post-its donde se divide una pizarra en columnas con el estado de las tareas y se van moviendo de columna conforme avanza la tarea.

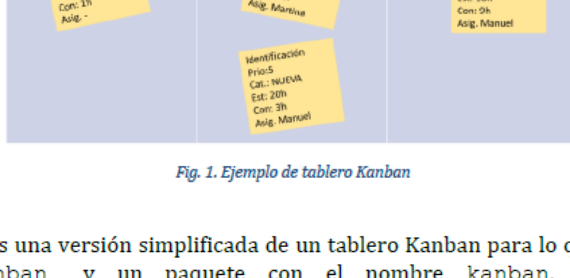


Fig. 1. Ejemplo de tablero Kanban

Implementaremos una versión simplificada de un tablero Kanban para lo que crearemos un proyecto prKanban y un paquete kanban que disponga de los siguientes atributos: KanbanException, Tarea y Kanban para la gestión básica. Para mostrar la información con distintos criterios incluiremos también la interfaz Criterio y dos implementaciones CriterioPrioridad y CriterioMaximoEsfuerzo.

- (0.25 ptos.) Clase KanbanException, que representa excepciones comprobadas para manejar las situaciones excepcionales que puedan producirse en las siguientes clases.

- (2.50 ptos.) Clase Tarea en el paquete kanban que disponga de los siguientes atributos:

- estado** (String) de las tareas del proyecto pueden ser: NOINICIADA, ENPROCESO, TERMINADA. (Se almacenará en mayúsculas)
- título** (String) Título de la tarea. (No puede cambiarse)
- prioridad** (int) de 1-5 (donde 1 es menos prioritario).
- horasEstimadas** (double). Horas estimadas para el desarrollo de la tarea. (No puede cambiarse)
- horasConsumidas** (double). Horas consumidas del desarrollo de la tarea.

- Un constructor al que se proporcionen como parámetros los valores iniciales de los atributos anteriores (en ese mismo orden). Debe validar todos los atributos, generando una KanbanException, si ocurre algo de lo siguiente:
 - El título es cadena vacía
 - El estado no es válido. Debe ser: NOINICIADA ENPROCESO TERMINADA (ver método auxiliar esEstadoVálido)
 - Prioridad no está entre 1 y 5.
 - Las horasConsumidas < 0
 - Las horasEstimadas <= 0

- Otro constructor al que se le proporcione como único parámetro el título de la tarea, que solo se llamaría desde dentro del paquete. El valor por defecto de los atributos que no se pasan como argumento será:
 - estado="NOINICIADA", prioridad=1, horasEstimadas=1, horasConsumidas=0

- Método estático


```
public static boolean esEstadoValido(String estado)
```

 que podrá usarse por otras clases y devolverá verdadero si el parámetro recibido es alguno de los estados válidos: NOINICIADA ENPROCESO TERMINADA.

- Implementar todos los *getter* y *setter* salvo el título y las horasEstimadas que se podrán consultar, pero no podrán modificarse una vez creada la tarea. Deberán validarse en los *setter* que el dato es correcto al igual que en el constructor.

- Implementar el método toString para que muestre la información con el formato del ejemplo siguiente:

```
Tarea: Título. estado prioridad:prioridad horas:(horasConsumidas/horasEstimadas)
Tarea: Análisis Requisitos. NOINICIADA prioridad:5 horas:(9.0/10.0)
```

- Se considerará que dos tareas son iguales si tienen el mismo título sin distinguir mayúsculas y minúsculas.

- (0.5 ptos.) Realizar una clase PruebaTarea que se sitúe en el paquete por defecto. Debe realizarse pruebas que sirva para verificar el constructor, la generación de excepciones, y los métodos toString y equals. Al menos debe incluir:
 - Crear dos tareas válidas y tres que den fallo (título vacío, prioridad fuera del rango 1-5, horasEstimadas negativas o cero, estado no válido...)

- Mostrar tareas correctas por pantalla
- Capturar excepción si el constructor no crea bien
- Comparar tareas

- (3.75 ptos.) La clase Kanban almacenará la información de un tablero Kanban. Todas las tareas que forman parte del tablero se añaden en el constructor y posteriormente no se podrán añadir ni eliminar tareas, facilitando esto la gestión y los recorridos de la estructura. Se guardarán todas las tareas en un array denominado tareas que se creará en el constructor con el tamaño de las tareas válidas que se reciban (y al que se podrá acceder a través del correspondiente *getter*). La clase dispondrá de:
 - Un constructor que recibe como argumento un array de String con el formato del ejemplo siguiente:

```
{"NOINICIADA;Análisis Requisitos;5;10;9","TERMINADA;Definición del proyecto;4;50;50",
...
"NOINICIADA;Carga datos;3;2;1"}
```

Cada elemento del array contiene información sobre una tarea que habrá que procesar para incluirla en la estructura. Si alguna tarea no es correcta (bien porque alguno de los datos no satisface los requisitos establecidos en el ejercicio 3 o porque el formato de la cadena de caracteres no es correcto), no se incluirá en el array. Es importante que al finalizar el almacenamiento de las tareas el tamaño del array tareas sea igual al número de tareas válidas que contiene.

- Un método


```
String resumenTareas()
```

 que devuelve una cadena de caracteres con el siguiente resumen que cuantifica el número de tareas por estado y el total de horas de cada tipo. Por ejemplo:

```
RESUMEN DE TAREAS
4 No iniciada. 2 En proceso. 1 Terminada.
Horas consumidas: 82.0. Horas estimadas:143.0
```

- Método toString() que devuelve una cadena de caracteres con todas las tareas separadas por coma y encerradas entre corchetes.

- (3 ptos.) Se quiere poder consultar las tareas que cumplan un determinado criterio. Se va a crear una interfaz llamada Criterio que define el método


```
public Tarea[] filtrar(Tarea[] tareas)
```

 que deberá ser implementada por las clases:


```
CriterioPrioridad
```

 y


```
CriterioMaximoEsfuerzo
```

.

- Crear la interfaz Criterio con el método filtrar.

- Implementar en la clase Kanban un método que devuelva un array de tareas seleccionadas según el filtro realizado por un objeto que implemente Criterio y que se pasa como argumento:


```
public Tarea[] seleccionar(Criterio criterio).
```

- Crear la clase CriterioPrioridad que implementa la interfaz Criterio filtrando las tareas con una determinada prioridad. Para ello, esta clase debe incluir un atributo prioridad (int) que debe inicializarse en el constructor e implementar el método filtrar de la interfaz, así como un método para modificar el valor de la prioridad (setter).

- Crear la clase CriterioMaximoEsfuerzo que implemente la interfaz Criterio, donde el método filtrar devolverá las tareas que tengan el mayor número de horas estimadas.

Diagrama de clases:

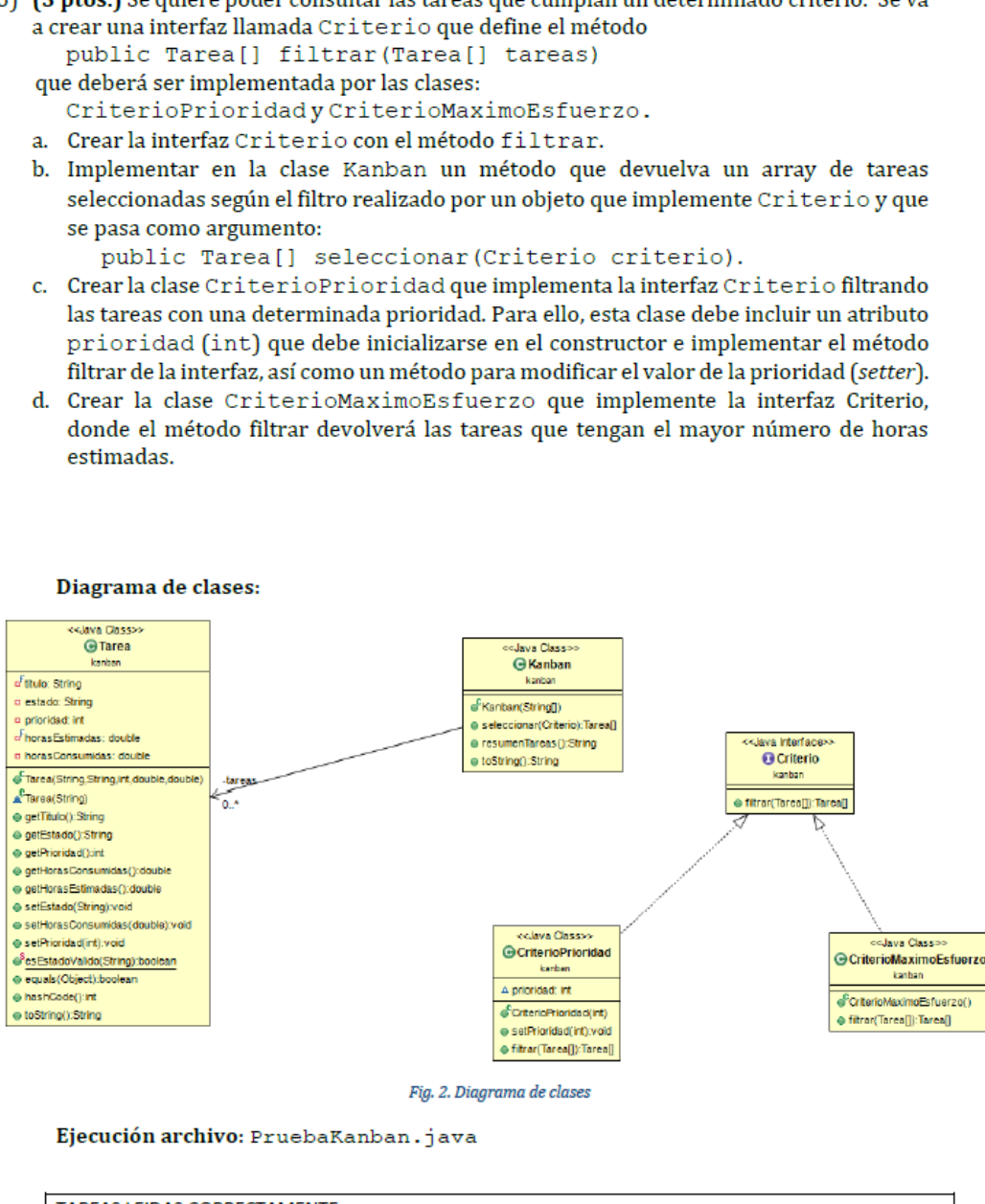


Fig. 2. Diagrama de clases

Ejecución archivo: PruebaKanban.java

TAREAS LEIDAS CORRECTAMENTE:

[Tarea: Análisis Requisitos. NOINICIADA prioridad:5 horas:(9.0/10.0), Tarea: Definición del proyecto. TERMINADA prioridad:4 horas:(50.0/50.0), Tarea: Carga datos. NOINICIADA prioridad:3 horas:(1.0/2.0), Tarea: Diseñar Base de Datos. ENPROCESO prioridad:3 horas:(2.0/10.0), Tarea: App Móvil. ENPROCESO prioridad:4 horas:(19.0/50.0), Tarea: Identificación. ENPROCESO prioridad:1 horas:(3.0/20.0)]

LISTADO POR PRIORIDADES:

Prioridad 1
[Tarea: Identificación. ENPROCESO prioridad:1 horas:(3.0/20.0)]

Prioridad 2
[]

Prioridad 3
[Tarea: Carga datos. NOINICIADA prioridad:3 horas:(1.0/2.0), Tarea: Diseñar Base de Datos. ENPROCESO prioridad:3 horas:(2.0/10.0)]

Prioridad 4
[Tarea: Definición del proyecto. TERMINADA prioridad:4 horas:(50.0/50.0), Tarea: App Móvil. ENPROCESO prioridad:4 horas:(19.0/50.0)]

Prioridad 5
[Tarea: Análisis Requisitos. NOINICIADA prioridad:5 horas:(9.0/10.0)]

LISTADO DE TAREAS CON MÁXIMO NÚMERO DE HORAS ESTIMADAS:

[Tarea: Definición del proyecto. TERMINADA prioridad:4 horas:(50.0/50.0), Tarea: App Móvil. ENPROCESO prioridad:4 horas:(19.0/50.0)]

RESUMEN DE TAREAS

2 No Iniciada. 3 En Proceso. 1 Terminadas
Horas consumidas: 84.0 Horas Estimadas: 142.0