



APELLIDOS, Nombre

TITULACIÓN Y GRUPO

MÁQUINA

### NOTAS PARA LA REALIZACIÓN DEL EJERCICIO:


- El ejercicio se almacenará en el directorio (espacio de trabajo) **C:\POO\EXJUNIO2016**. En caso de que no exista se creará, y si ya existiese, se borrará su contenido antes de comenzar.
- Al inicio del contenido de cada fichero deberá indicarse **el nombre del alumno, titulación, grupo y código del equipo** que está utilizando.
- 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.
- Los diferentes apartados tienen una determinada puntuación. Si un apartado no se sabe hacer, el alumno *no debe pararse en él indefinidamente*. Puede abordar otros.
- **Está permitido:**
  - Consultar la API.
- **No está permitido:**
  - Utilizar otra documentación electrónica o impresa.
  - Intercambiar documentación con otros compañeros.
  - Utilizar soportes de almacenamiento.
- Una vez terminado el ejercicio se subirá un **fichero comprimido** sólo con los ficheros **fuentes (.java)** que hayáis realizado a la tarea creada en el campus virtual para ello.

## Proyecto prExJunio2016

Se desea crear una aplicación para gestionar las peticiones de laboratorios por parte de las asignaturas de un centro docente. **Se han de crear**, en el paquete `prExJunio2016`, las clases: `AsignacionException`, `PeticionAsignacion`, `Asignaciones`, `AsignacionesConAlternativas` y `ControladorAsignacion`. También **se debe crear**, en el paquete “por defecto” la clase `PruebaPeticionAsignacion` para la prueba de algunas de las clases anteriores. Todas estas clases aparecen en el diagrama de clases que se suministra en el campus virtual. Por otro lado, **se suministran** en el campus virtual la interfaz `VistaAsignacion` y las clases `FranjaHoraria` y `PanelAsignacion`, que deben meterse en el paquete `prExJunio2016`. También **se suministran** `Prueba1`, `Prueba2` y `PruebaTotalGUI` para realizar pruebas completas de todas las clases y que deben meterse en el paquete “por defecto” del proyecto. Por último, **se suministran** dos ficheros, `peticiones.txt` y `peticionesca.txt`, con datos necesarios para realizar alguna de las operaciones que se indican más adelante. Estos ficheros deben almacenarse directamente en la carpeta del proyecto `prExJunio2016`. El diagrama de clases está disponible en el campus virtual.

Se proporciona en el campus virtual la clase `FranjaHoraria` con información sobre el día y la hora en la que se impartirá una asignatura. `DiaSemana` es un tipo enumerado con los valores {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES} y `HoraDia` es un tipo enumerado con los valores {PRIMERA, SEGUNDA, TERCERA, CUARTA, QUINTA, SEXTA}. Cada valor se *corresponde* con una franja horaria de clase. La clase dispone de un constructor para crear una franja horaria dados el día y la hora como cadenas de caracteres (`String`) y una representación (`toString()`). El constructor lanzará una excepción de tipo `AsignacionException` en caso de que las cadenas proporcionadas no se correspondan con un día o con una hora válidas. También proporciona métodos para obtener el día y la hora (`DiaSemana getDia()`, `HoraDia getHora()`). **NOTA: Los tipos enumerados redefinen los métodos `equals()`, `hashCode()` y `compareTo()`. Un valor de tipo enumerado es menor que otro si está definido antes. Por ejemplo, el valor LUNES es el menor valor del tipo `DiaSemana` y VIERNES es el mayor.**

A continuación, se describen cada una de las clases pedidas:

- 1) **(0.25 ptos.)** La clase `AsignacionException` se utilizará para tratar las diferentes situaciones excepcionales. Se trata de una excepción *no comprobada*. En general, se lanzará esta excepción si los parámetros de entrada de los métodos no contienen valores adecuados.
- 2) **(1.75 ptos.)** La clase `PeticionAsignacion` mantendrá información sobre una petición de asignación de laboratorio. En concreto, el nombre del profesor (`String`), el nombre de la asignatura (`String`) y la franja horaria (`FranjaHoraria`) en la que se impartirá la asignatura. La clase dispondrá de constructores y métodos necesarios para:
  - a. Construir un objeto de la clase dadas dos cadenas de caracteres (`String`), que son los nombres de la asignatura y del profesor, y la franja horaria en la que se quiere realizar la reserva.
  - b.  Obtener el nombre del profesor (`String getProfesor()`), el nombre de la asignatura (`String getAsignatura()`) y la franja horaria (`FranjaHoraria getFranja()`).
  - c. La representación de una petición de asignación (`String toString()`) viene dada por la franja horaria de la petición, el nombre de la asignatura y el nombre del profesor. El formato será como el de este ejemplo: (LUNES, PRIMERA) -> POO, Antonio Lopez.
  - d. Dos objetos de la clase `PeticionAsignacion` son iguales si coinciden la franja horaria de las peticiones. El nombre del profesor y el de la asignatura no se tienen en cuenta.
  - e. Define un orden natural para la clase `PeticionAsignacion` que ordene las peticiones primero por el día de la petición, según el orden natural definido para el tipo enumerado `DiaSemana`. Si el día es el mismo entonces las ordenará por la hora, según el orden natural definido para el tipo enumerado `HoraDia`.
- 3) **(0.5 ptos.)** Crea una aplicación (clase distinguida `PruebaPeticionAsignacion`) para probar la clase anterior. En esta aplicación se crean dos objetos de la clase `PeticionAsignacion`. En el primero la asignatura es "POO", el nombre del profesor "Juan Lopez" y reserva un laboratorio el "Lunes" a "primera" hora. En el segundo la asignatura es "FP", el nombre del profesor "Maria Gomez" y reserva un laboratorio también el "Lunes" a "primera" hora. Después se muestran por pantalla los dos objetos. Suponiendo que sólo hay un laboratorio, habrá conflicto si los dos profesores intentan reservar el mismo día a la misma hora. Para saber si hay conflicto se comprobará si los dos objetos son iguales y en ese caso se mostrará un mensaje indicando que hay conflicto. Si la franja horaria solicitada para alguna asignatura no es correcta se mostrará el mensaje de error "ERROR: Franja Horaria Incorrecta". La ejecución de la aplicación producirá la siguiente salida por pantalla:

```
(LUNES, PRIMERA) -> POO, Juan Lopez
(LUNES, PRIMERA) -> FP, Maria Gomez
Conflicto: LUNES a PRIMERA hora.
```

Cambiar ahora la franja horaria de la primera petición para solicitar una franja incorrecta indicando un día y/o hora incorrectos (por ejemplo, "Lune" en lugar de "Lunes"). Comprobar que en ese caso la salida es: ERROR: Franja Horaria Incorrecta

- 4) **(4 ptos.)** La clase `Asignaciones` se encargará de realizar la asignación de los laboratorios y de identificar los conflictos existentes en caso de que no haya laboratorios suficientes para una franja determinada. Para ello, almacenará la asignación de laboratorios en una correspondencia o asociación denominada *asignacion*, que asociará cada número de laboratorio (`Integer`) con el conjunto de peticiones a las que se les asignará ese laboratorio. Esta correspondencia se definirá como `SortedMap<Integer, SortedSet<PeticionAsignacion>>`. Los conflictos identificados se almacenarán en una lista de peticiones denominada *conflictos* que se definirá como `protected List<PeticionAsignacion>`. El número de laboratorios disponibles para realizar la asignación vendrá dado por el valor de la constante de instancia `NUM_LAB`. La clase dispondrá de dos constructores y varios métodos de instancia públicos:

- a. El constructor `Asignaciones(int)` recibirá como parámetro el número de laboratorios, inicializará la constante de instancia `NUM_LAB` y creará la correspondencia `asignacion` y la lista `conflictos`. La lista de conflictos será una lista vacía, pero la correspondencia `asignacion` se inicializará con una entrada por cada laboratorio disponible, donde la clave será el número de laboratorio (un número entre 1 y `NUM_LAB`) y el valor asociado a cada laboratorio será inicialmente un conjunto vacío. El constructor `Asignaciones(int, String)` recibe el número de laboratorios y un nombre de fichero y, tras crear la correspondencia de asignaciones y la lista de conflictos de la misma forma que en el constructor anterior, añadirá toda la información que contiene el fichero. Para ello invocará al método descrito en el apartado e. Si el número de laboratorios es menor que 1 se lanzará una excepción.
- b. El método `Set<Integer> buscarHuecos(FranjaHoraria franja)` devolverá un conjunto con todos los laboratorios que tiene libre la franja dada. Para ello se creará una petición de asignación indicando como nombre de la asignatura y del profesor la cadena vacía (`''''`) y la franja dada y se añadirán al conjunto todos los laboratorios que tienen esa franja libre. Un laboratorio tiene esa franja libre si su conjunto asociado no contiene ninguna petición con la misma franja (es decir, no existe otra petición igual).
- c. El método booleano `nuevaAsignacion(PeticionAsignacion pa)` asigna una petición a un laboratorio. Buscará los huecos libres para la franja horaria de esta petición y la asignará a cualquiera de ellos. El método devolverá `true` si ha sido posible realizar la asignación o `false` en otro caso.
- d. El método `void realizarAsignacion(String linea)` crea una petición de asignación con la información que hay en la cadena `linea` e intenta hacer una nueva asignación. La `linea` contiene la siguiente información:
- ```
P00#Antonio Lopez#Lunes#primera
```
- Es decir, el separador es el símbolo `#` y los datos son el nombre de la asignatura, el profesor, el día y la hora. Si no es posible hacer la asignación, la petición se añadirá a `conflictos`. Cualquier error de formato detectado provocará que se lance una excepción del tipo `AsignacionException` con el mensaje adecuado informando de tal situación.
- e. El método `void leerPeticionesDeFichero(String nf)` recibirá como parámetro el nombre de un fichero con todas las peticiones de laboratorio. Cada línea del fichero contendrá información de una petición concreta. El formato de cada línea es el mencionado anteriormente. En el campus virtual se suministra un fichero de ejemplo (`peticiones.txt`). Este método leerá los datos del fichero y por cada línea realizará una asignación.
- f. Se ha de redefinir el método `String toString()`. La representación de los objetos de la clase muestra toda la información contenida en la correspondencia `asignación` y toda la información contenida en la lista `conflictos`. El formato que se ha de utilizar es el que se muestra en el fichero `asignaciones.txt` proporcionado en el campus virtual. Para la realización de este método se debe usar un `StringBuilder`.
- g. Los métodos `void escribirAFichero(String)` y `void escribirAsignaciones(PrintWriter)` guardan la información de la correspondencia `asignacion` y de la lista `conflictos` en el fichero o en el flujo de salida dado como parámetro respectivamente. El formato de los datos será el mismo del apartado f.

Podéis probar la clase anterior utilizando la clase `Prueba1` y el fichero `peticiones.txt` proporcionados en el campus virtual. La salida que se ha de obtener en consola es la mostrada en el fichero `asignaciones.txt`, también disponible en el campus virtual.

5) (2 ptos.) La clase `AsignacionesConAlternativas` se comporta como la clase `Asignaciones`, con la diferencia de que cada petición puede venir con franjas alternativas. La clase dispondrá de un constructor y dos métodos de instancia:

- a. El constructor `AsignacionesConAlternativas(int,String)` recibe como primer parámetro el número de laboratorios, y como segundo el nombre del fichero con los datos de las peticiones de las asignaturas. En este caso, cada línea además de la asignatura y el profesor, podrá contener información para construir varias franjas horarias. Un ejemplo del formato de la línea es:  
`POO#Antonio Lopez#Lunes#primera#martes#segunda#jueves#primera`
- b. El método `Set<PeticionAsignacion> leerAlternativas(String linea)` es un método privado que leerá una línea como la anterior y creará un conjunto con todas las peticiones de asignación que se pueden crear usando el nombre de la asignatura, el nombre del profesor y variando la franja horaria. Así, desde la línea ejemplo anterior se podrán crear 3 peticiones de asignación. Cualquier error de formato detectado provocará que se lance una excepción del tipo `AsignacionException` con el mensaje adecuado.
- c. Redefinir el método `void realizarAsignacion(String linea)` para que desde una línea con la información de una petición con alternativas, cree el conjunto de peticiones alternativas e intente asignar una de ellas. Si todos los intentos fallan, deberá incluir cualquiera de las peticiones del conjunto en conflictos. Si el conjunto de peticiones está vacío deberá lanzar una `AsignacionException` informando que la línea está incompleta.

Podéis probar la clase anterior utilizando la clase `Prueba2` y el fichero `peticionesca.txt` proporcionados en el campus virtual.

6) (1.5 ptos.) La clase `ControladorAsignacion` controla e interactúa con el modelo (clase `Asignaciones`) y la vista (se proporcionan en el campus virtual la interfaz `VistaAsignacion`, la clase `PanelAsignacion` y el fichero `GUI.gif` con una imagen de la vista). El constructor debe:

- habilitar la parte de inicialización de la vista (campos para introducir el nombre del fichero con las peticiones, el checkbox de alternativas, el número de laboratorios y los botones inicio y reinicio) y
- mostrar un mensaje (método `mensaje(String)`) indicando al usuario que introduzca los datos de entrada y pulse el botón “Inicio”.

El comportamiento de la aplicación al pulsar los botones es el siguiente:

En primer lugar se borran los mensajes o errores que hubiera.

#### **Botón “Inicio”:**

- En caso de que no se introduzca el fichero de peticiones o el número de laboratorios sea menor que 1 se mostrará un mensaje de error y no se hará nada más.
- Si no se marca el checkbox de alternativas se creará un objeto de la clase `Asignaciones`. En otro caso, se creará un objeto de la clase `AsignacionesConRestricciones`.
- Se mostrará en el historial de mensajes la información de las asignaciones y conflictos.
- Se deshabilitará la zona de inicialización.
- Se mostrará un mensaje de que todo ha ido bien si ese es el caso.

**Botón “Reinicio”:** Borra el área del historial de mensajes y habilita la parte de inicialización de la vista.

**Botón “Guardar”:** Guarda la información sobre asignaciones y conflictos en el fichero indicado en el campo de texto a su derecha.

Cualquier excepción que se produzca se capturará y se mandará un mensaje de error (método `error(String)`).