PROGRAMACIÓN ORIENTADA A OBJETOS

(Prueba realizada el 7 de septiembre de 2012)

APELLIDOS, Nombre	TITULACIÓN Y GRUPO	
	MÁQUINA	

NOTAS PARA LA REALIZACIÓN DEL EJERCICIO:

- El ejercicio se almacenará en el directorio **C:\POO**. En caso de que no exista deberá crearse, y si ya existiese, deberá borrarse todo 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 el API.
- **No está permitido**:
  - Utilizar otra documentación electrónica o impresa.
  - Intercambiar documentación con otros compañeros.
  - Utilizar soportes de almacenamiento.
- Una vez terminado el ejercicio subir un fichero comprimido (.rar) sólo con los **fuentes**

Se desea llevar a cabo la gestión informatizada de la asignación de plazas en un colegio de educación infantil y primaria. Para ello, se creará un proyecto `prAsignacionPlazas` con las clases siguientes:

1) (***0.25 ptos.***) La clase `AsignacionException` se utilizará para tratar diferentes situaciones excepcionales tal y como se indica más adelante. Se trata de una excepción no comprobada.

2) (***1.5 ptos.***) La clase `Solicitante` debe mantener información sobre un solicitante de plaza. En concreto, el NIF del tutor legal del solicitante (`String`), el nombre del solicitante (`String`), los puntos (`double`) del solicitante después de ser baremado, y una indicación (`boolean`) de si ese solicitante tiene plaza o no. También dispondrá del constructor y los métodos necesarios para:
   a. *(0.25 ptos.)* Construir un objeto de la clase a partir del NIF, nombre y puntos del solicitante. La representación de un solicitante (`String toString()`) viene dada por esos tres datos, separados por espacios en blanco.
   b. *(0.25 ptos.)* Obtener el NIF (`String getNIF()`), obtener el nombre (`String getNombre()`), obtener los puntos (`String getPuntos()`), ver si se le ha asignado una plaza (`boolean getTienePlaza()`) y asignarle o quitarle una plaza al solicitante (`void setTienePlaza(boolean plaza)`).
   c. *(0.5 ptos.)* Dos objetos de la clase `Solicitante` son iguales si coinciden sus nombres y sus puntos. No se tendrán en cuenta las diferencias por mayúsculas o minúsculas.
   d. *(0.5 ptos.)* Un objeto de la clase `Solicitante` s1 es menor que otro s2 cuando el número de puntos de s1 sea mayor que el número de puntos de s2. Si el número de puntos es igual, entonces s1 será menor cuando su nombre sea menor alfabéticamente (independientemente de minúsculas y mayúsculas) que el nombre de s2.

3) (**3.50 *ptos.***) La clase `Colegio` almacenará la información de los solicitantes en una estructura `solicitudes` de tipo `SortedMap<String,SortedSet <Solicitante>>`, donde los posibles valores del campo clave de la estructura `solicitudes` serán "I3", "I4" o "I5", para hacer referencia a los cursos de infantil de 3, 4 ó 5 años, y "P1", "P2", "P3",…,"P6", para hacer referencia a los cursos de primaria entre 1º y 6º (no tiene por qué haber solicitudes para cada curso escolar). También dispondrá del constructor y los métodos necesarios para:

   a. *(1.75 ptos.)* Construir un objeto de la clase a partir del nombre del fichero (`String`) que contiene la información necesaria para crear la aplicación `solicitudes`. Los datos para probar el correcto funcionamiento del examen son los que aparecen en el fichero `solicitudes.txt` proporcionado en el campus virtual. Una parte de este fichero se muestra a continuación para ver su formato. Para cada curso para el que haya solicitudes se indica: (1) una línea con el nombre del curso (ej. I3 en la primera línea del fichero) y el número de alumnos en ese curso (ej. 3 en la primera línea del fichero), y (2) una línea con los datos de cada solicitante incluyendo NIF del tutor, número de puntos y nombre del solicitante (ej. si para el curso I3 se indicaba que había 3 solicitudes, en el fichero habrá 3 líneas, una para cada solicitante).

```
I3 3
11111111A 25 María Martín Arrebola
11111111B 26.5 Antonio Gómez Muñoz
11111111C 30 Antonio Pérez Gallardo
I4 1
22222222A 14 Eva López García
```

      Cualquier error de formato detectado (falta algún dato del solicitante, o bien el dato no es del tipo correcto), provocará el lanzamiento de una excepción del tipo `AsignacionException`.

   b. *(1 pto.)* Hacer la asignación de plazas (`void asignarPlazas(Map<String, Integer> plazas)`). Este método recibe una aplicación donde se indica el número de plazas libres para cada curso escolar (ver programa `Principal.java` proporcionado en el campus virtual). Si para un determinado curso escolar no existe una entrada en la aplicación es que el número de plazas libres para ese curso es 0. Para cada curso en el que haya plazas libres habrá que modificar la aplicación `solicitudes` para indicar para cada solicitante si tiene o no tiene plaza. La asignación de plazas se realizará de la forma siguiente: Se le asignará plaza primero a los solicitantes que tengan más puntos. En caso de empate se le asignará plaza primero al que su nombre sea alfabéticamente menor.

   c. *(0.25 puntos)* Hacer que todos los solicitantes queden sin plazas (`void limpiar()`).

   d. *(0.5 ptos.)* Mostrar asignaciones de plazas con orden natural. Dos métodos, uno para mostrar sobre un `PrintWriter` la información de los solicitantes organizados por curso (`void mostrarSolicitantes (boolean conPlaza, PrintWriter pw)`) y otro para almacenarla en un fichero (`void mostrarSolicitantes (boolean conPlaza, String nombreFichero)`). Si el parámetro `conPlaza` es `true`, se mostrarán los solicitantes a los que se les ha podido asignar plaza. Si es `false`, se mostrarán los solicitantes a los que no se les ha podido asignar plaza. El formato de salida es como el mostrado en los ficheros `conPlaza.txt` y `sinPlaza.txt` proporcionados en el campus virtual.

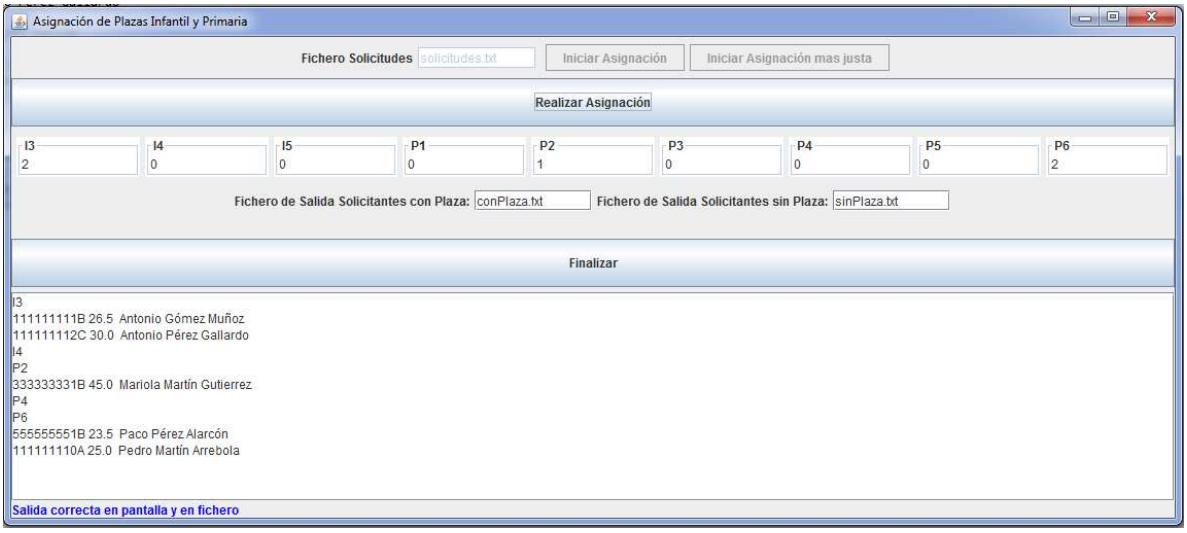
4) (***2.75 ptos.***) La clase `ColegioFamiliar` se comporta como la clase `Colegio`. La diferencia radica en que si dos hermanos piden plaza en el mismo colegio, en el mismo

o en distinto curso, según la asignación de plazas que se realiza en la clase `Colegio` puede ocurrir que uno de los hermanos consiga la plaza y el otro no. Para evitar esto, en esta clase la asignación de plazas tiene en cuenta el hecho de que dos o más solicitantes pueden ser hermanos (son solicitantes que tienen el mismo NIF de su tutor) y siempre se cumplirá que o todos los hermanos consiguen plaza o ninguno de ellos la consigue. Para ello, dispondrá del constructor y los métodos necesarios para:

a. *(0.25 ptos.)* Construir un objeto de la clase a partir del nombre del fichero (`String`) que contiene la información necesaria para crear la aplicación `solicitudes`. El formato del fichero es el mismo que el usado para crear un objeto de la clase `Colegio`.
b. *(2.5 ptos.)* Una redefinición del método `public void asignarPlazas(Map<String,Integer> plazas)` que se comporte de la siguiente forma: antes de asignar una plaza a un solicitante hay que localizar a todos sus hermanos (incluyendo al solicitante) y comprobar si sería posible asignarles plaza a todos (no se tendrá en cuenta la puntuación de los hermanos del solicitante, sólo si hay plaza o no disponible para todos ellos). Si es posible, se le asigna la plaza a todos ellos; en otro caso, no se le asigna a ninguno. Para ello, se deben implementar los siguientes métodos auxiliares:
    1. `SortedMap<String,List<Solicitante>> buscarHermanos ( String nif)` que recibe un NIF y devuelve una aplicación donde el campo clave es el nombre de los cursos y el campo valor es la lista de hermanos que han solicitado plaza en ese curso (todos los hermanos tienen el mismo NIF de su tutor).
    2. `boolean hayPlazas(SortedMap<String,List<Solicitante>> hermanos, Map<String,Integer> plazas)`, que recibe dos valores: el primero es la aplicación con la información de los hermanos obtenida con el método anterior; el segundo es otra aplicación donde el campo clave son los cursos y los valores son el número de plazas disponibles en ese curso. El método devuelve `true` si es posible asignarle una plaza a cada uno de los hermanos indicados en el primer parámetro, y `false` en caso contrario.
    3. `void asignarPlazaHermanos (SortedMap<String, List <Solicitante>> hermanos, Map<String,Integer> plazas)`, que recibe también la información de los hermanos y las plazas disponibles en cada curso y, suponiendo como precondición que hay plaza para todos los hermanos hace las asignaciones de plazas, reduciendo el número de plazas disponibles de acuerdo a las asignaciones realizadas.

5) (***2 ptos.***) La clase `ControladorAsignacion` controla e interactúa con el modelo (compuesto por las clases anteriormente descritas) y la vista (se proporcionan en el campus virtual la interfaz `VistaAsignacion` y la clase `PanelAsignacion`). El constructor debe habilitar la parte de inicialización de la vista (introducción del fichero de solicitudes, y los botones de iniciar la asignación) y mostrar un mensaje en la parte baja de la misma indicando al usuario que introduzca el nombre del fichero y pulse el botón iniciar que desee. El resto de la vista estará deshabilitado. La pulsación de uno de los botones de iniciar hará que se cree un objeto de la clase `Colegio` o de la clase `ColegioFamiliar`, se deshabilite la zona de inicialización, se habilite el resto de la vista y se muestre un mensaje al usuario invitándolo a realizar la asignación. Cada vez que el usuario pulse el botón "Realizar Asignación" se procederá a realizar la asignación a todos los solicitantes y todos los cursos. El resultado se mostrará en el área de texto y se escribirá en el fichero de salida especificado. El botón "Finalizar" deshabilita esta zona de proceso, limpia las zonas de texto y habilita la parte de inicialización de la vista para poder empezar de nuevo con otros ficheros de entrada. Si

al pulsar un determinado botón, no se han introducido los datos necesarios en los campos de texto, se debe mostrar un mensaje de error en la parte baja de la vista.



Las clases principales con ejemplos de uso de las clases anteriormente descritas (SSin GUIs y con GUIs) están disponibles en el campus virtual.