NOTAS PARA LA REALIZACIÓN DEL EJERCICIO:

- El ejercicio se almacenará en el directorio (espacio de trabajo) **C:\POO\EXSEPT2016**. 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 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 **fuente (.java)** que hayáis realizado a la tarea creada en el campus virtual para ello.

Proyecto prExSept2016

Se desea crear una aplicación para manipular la información relacionada con equipos y jugadores de una competición de pádel. **Se han de crear**, dentro de un paquete denominado `prExSept2016`, las siguientes clases: `CompeticionException`, `Jugador`, `Equipo`, `OrdenAltEquipo`, `Competicion`, `CompeticionMaster`, `ControladorCompeticion`. También **se debe crear**, en el paquete "por defecto" del proyecto, la clase denominada `PruebaEquipo` 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 `VistaCompeticion` y la clase `PanelCompeticion`, que deben meterse en el paquete `prExSept2016`. También **se suministran** `PruebaJugador`, `PruebaTotal` y `PruebaTotalGUI` para realizar pruebas de las clases. Estas clases de prueba deben meterse en el paquete "por defecto" del proyecto. Por último**, se suministra** un fichero, `equipos.txt`, con datos necesarios para realizar alguna de las operaciones que se indican más adelante. Este fichero debe almacenarse directamente en la carpeta del proyecto `prExSept2016`.

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

1) (**0.25 *ptos.***) La clase `CompeticionException` 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) (**0.75 *ptos.***) La clase `Jugador` mantendrá información sobre un jugador de pádel. En concreto, su nombre (`String`), el número de partidos jugados (`int`), y el número de partidos ganados (`int`). La clase dispondrá de constructores y métodos necesarios para:
   a. Construir un objeto de la clase dados los valores para los tres datos descritos anteriormente. Si el número de partidos jugados o partidos ganados es menor que cero, se lanzará una excepción del tipo `CompeticionException` con el mensaje correspondiente. Si el número de partidos ganados es mayor que el número de partidos jugados, se lanzará una excepción del tipo `CompeticionException` con el mensaje correspondiente.
   Construir un objeto de la clase dado solamente el nombre. En este caso, tanto el número de partidos jugados como el de partidos ganados se inicializará a 0.
   b. Obtener el nombre (`String getNombre()`), partidos jugados (`int getPJugados()`) y partidos ganados (`int getPGanados()`). Incrementar el número de partidos jugados y ganados

(`void increPartidos(int,int)`). En este último método hay que controlar los mismos tipos de errores que se detallaron en el apartado a.

c. La representación de un jugador (`String toString()`) viene dada por su nombre, partidos jugados y partidos ganados separados por el carácter ':'. Ej: Juan Diaz:2:1.

d. Dos objetos de la clase `Jugador` son iguales si coinciden sus nombres, ignorando mayúsculas y minúsculas. El número de partidos jugados y ganados no se tienen en cuenta.

3) Utiliza la clase distinguida `PruebaJugador` suministrada para probar la clase anterior. En esta aplicación se crean dos objetos de `Jugador`. El primero con nombre "Ana Romero", 2 partidos jugados y 1 partido ganado. El segundo con nombre "ana romero", 2 partidos jugados y 0 partidos ganados. Después se incrementa en 1 tanto los partidos jugados como los ganados del segundo objeto, y se muestra por pantalla el contenido de ambos objetos. A continuación, se comprueba si ambos objetos son iguales, indicándolo por pantalla. La ejecución de la aplicación producirá la siguiente salida por pantalla:

```
Jugador 1 = Ana Romero:2:1
Jugador 2 = ana romero:3:1
Los jugadores son iguales
```

4) (***1 pto.***) La clase `Equipo` mantendrá información sobre un equipo de pádel para la competición. En concreto, su nombre (`String`), su categoría (`int`), y los puntos conseguidos (`int`). La clase dispondrá de constructores y métodos necesarios para:

a. Construir un objeto de la clase dados los valores para los tres datos descritos anteriormente. Si la categoría no es un número comprendido en el rango de 1 a 5 (ambos inclusive), se debe lanzar una excepción de tipo `CompeticionException` informando de tal situación. También se lanzará una excepción de este tipo si el número de puntos es menor que cero.

b. Obtener el nombre (`String getNombre()`), la categoría (`int getCategoria()`) y los puntos (`int getPuntos()`). Modificar la categoría del equipo (`void setCategoria(int)`). Incrementar el número de puntos del equipo (`void increPuntos(int)`). En estos dos últimos métodos hay que controlar los mismos tipos de errores que se detallaron en el apartado a

c. La representación de un equipo (`String toString()`) viene dada por su nombre, categoría y puntos en la clasificación, separados por el carácter ':'. Ej: Parque Litoral:3:10.

d. Dos objetos de la clase `Equipo` son iguales si coinciden sus nombres, ignorando mayúsculas y minúsculas, y también coinciden sus categorías y puntos en la clasificación.

e. El *orden natural* de dos objetos de la clase `Equipo` se realiza de la siguiente forma: es menor aquel equipo con un número menor de categoría. En caso de categorías iguales, será menor aquel equipo con un número mayor de puntos en la clasificación (¡ojo, es menor el que más puntos tiene!). Por último, es caso de igualdad en puntos, se aplica el orden natural de sus nombres, ignorando mayúsculas y minúsculas.

5) (***0.5 ptos.***) La clase `OrdenAltEquipo` proporciona un *orden alternativo* para los objetos de la clase `Equipo`. Este orden alternativo se realiza de la siguiente forma: es menor aquel equipo con un número menor de categoría. En caso de categorías iguales, se aplica el orden natural de sus nombres, ignorando mayúsculas y minúsculas. Por último, en caso de igualdad de nombres, será menor aquel equipo con un número mayor de puntos en la clasificación (¡ojo, es menor el que más puntos tiene!).

6) (***0.5 ptos.***) Crea una aplicación (clase distinguida `PruebaEquipo`) para probar las dos clases anteriores. En esta aplicación se creará un conjunto ordenado vacío de objetos de la clase `Equipo` (la ordenación seguirá el *orden natural* de los objetos). Se añadirán al conjunto tres equipos: el primero con nombre "Parque Litoral", con categoría 3 y con 10 puntos; el segundo con nombre "Centro Padel", con categoría 2 y con 6 puntos; el tercero con nombre "Padel Palo", con categoría 3 y con 8 puntos. A continuación se mostrará por pantalla el contenido del conjunto. Después crearemos otro conjunto ordenado vacío de objetos de la clase `Equipo`, pero esta vez teniendo en cuenta que los objetos se ordenarán dentro del conjunto atendiendo al *orden alternativo* que se especifica en la clase `OrdenAltEquipo`. Se añadirán a este nuevo conjunto los tres equipos anteriores. Finalmente, se mostrará por pantalla el contenido de este segundo conjunto. La ejecución de la aplicación producirá la siguiente salida por pantalla:

```
[Centro Padel:2:6, Parque Litoral:3:10, Padel Palo:3:8]
[Centro Padel:2:6, Padel Palo:3:8, Parque Litoral:3:10]
```

7) (4.7***5*** ***ptos.***) La clase `Competicion` almacenará la información relativa a una determinada competición de pádel en una correspondencia o asociación denominada `competicion`, que asocia cada equipo participante (objeto de la clase `Equipo`) con la lista de jugadores pertenecientes a ese equipo (objetos de la clase `Jugador`). Esta correspondencia se definirá como `SortedMap<Equipo, List <Jugador>>)`. Los equipos estarán ordenados atendiendo a su *orden natural*. La clase dispondrá de dos constructores y varios métodos de instancia públicos:

a. El constructor `Competicion()` creará la correspondencia `competicion` vacía. El constructor `Competicion(String)` recibe un nombre de fichero y, tras crear la correspondencia vacía, añade a ella toda la información que contiene el fichero. Para ello invocará al método descrito en el apartado b.

b. El método `void leerDatos(String)` almacena en la correspondencia `competición` los datos de los equipos y jugadores contenidos en el fichero cuyo nombre recibe como parámetro. El formato de los datos se puede observar en el fichero suministrado como ejemplo en el campus virtual (`equipos.txt`). Para cada equipo aparece una línea con los datos propios del equipo (nombre, categoría y puntos) en primer lugar y a continuación los datos propios de cada jugador (nombre, partidos jugados y partidos ganados) perteneciente a ese equipo. Tanto los datos de un equipo como los de un jugador están separados por dos puntos (el carácter ':'). A continuación aparece una línea de dicho fichero con información relativa al equipo "Padel Palo":

```
Padel Palo:3:8:Rosa Lima:2:1:Jose Atencia:2:1:Manuel Rodriguez:2:2:Barbara Avila:2:2
```

Con los datos propios de un equipo se creará un objeto de la clase `Equipo`. Posteriormente, se creará una lista de objetos de la clase `Jugador` vacía y se procederá a almacenar en ella cada uno de los jugadores creados con los datos que aparecen en la línea. Finalmente se invocará al método descrito en el apartado c (`insertaJugadoresEquipo`) para llevar a cabo la asociación del equipo a la lista de jugadores correspondiente.
Cualquier error de formato detectado (falta algún dato o bien el dato no es del tipo correcto), provocará el lanzamiento de una excepción del tipo `CompeticionException` con el mensaje adecuado informando de tal situación.

c. El método `void insertaJugadoresEquipo(Equipo, List<Jugadores>)` asocia en la correspondencia `competicion` el equipo recibido como primer parámetro con la lista de jugadores recibida como segundo parámetro. Si el equipo ya estuviese en la correspondencia se añadirán los jugadores indicados en el segundo parámetro a la lista de jugadores que ya tuviera previamente asignada, sin repetir los que ya estuvieran dentro de esa lista.

d. Se ha de redefinir el método `String toString()`. La representación de los objetos de la clase muestra toda la información contenida en la correspondencia `competicion`. El formato que se ha de utilizar es el que se muestra en el ejemplo del fichero suministrado "SalidaConsolaEjecucion.pdf". Se debe utilizar la clase *StringBuilder*.

e. Los métodos `void escribirFichero(String)` y `void escribir(PrintWriter)` guardan la información de la correspondencia `competicion` en el fichero o en el flujo de salida dado como parámetro respectivamente. El formato de los datos será el mismo del apartado d.

f. El método `void increPartidos(String,int,int)` incrementa el número de partidos jugados (segundo parámetro) y partidos ganados (tercer parámetro) al jugador almacenado en la correspondencia `competicion` cuyo nombre se recibe como primer parámetro. Si el jugador no está almacenado, se lanzará una excepción del tipo `CompeticionException` con el mensaje adecuado informando de tal situación. Nota: un jugador no puede pertenecer a varios equipos.

8) (***1.25 ptos.***) La clase `CompeticionMaster` *se comporta como la clase* `Competicion`, con la diferencia de que a la hora de añadir jugadores a los equipos, sólo se considerarán aquellos jugadores que han ganado un número de partidos igual o superior a un determinado umbral (`int`). La clase dispondrá de un constructor y un método de instancia:

a. El constructor `CompeticionMaster(String, int)` recibe como primer parámetro el nombre del fichero con los datos de equipos y jugadores para almacenar en la correspondencia. El

segundo parámetro es el valor umbral establecido para el número de partidos ganados exigido para que un jugador forme parte de la competición.

b. El método `void insertaJugadoresEquipo(Equipo, List<Jugadores>)` será una redefinición del método correspondiente de la clase `Competicion`. En este caso, los jugadores que no hayan ganado un número de partidos igual o superior al umbral establecido en el constructor, no serán incluidos en la lista asociada al equipo dentro de la correspondencia.

9) (***1 pto.***) La clase `ControladorCompeticion` controla e interactúa con el modelo (clase `Competicion`) y la vista (se proporcionan en el campus virtual la interfaz `VistaCompeticion` y la clase `PanelCompeticion`). El constructor debe habilitar la parte de inicialización de la vista (introducción del fichero con los datos de equipos y jugadores y el botón de inicio) 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. El resto de la vista estará deshabilitado. La pulsación del botón "Inicio" hará que se cree un objeto de la clase `Competicion` (pasándole el nombre del fichero introducido), se deshabilite la zona de inicialización y se habilite el resto de la vista. Cada vez que el usuario pulse algunos de los demás botones se procederá a realizar la acción correspondiente. El botón "Guardar" guarda la información sobre equipos y jugadores en el fichero indicado en el campo de texto a su derecha. En caso de que dicho campo contenga la cadena vacía, la información se imprime en el área de texto del panel. El botón "Incrementar Partidos" hará que se lea el nombre del jugador del campo de texto de su derecha y el número de partidos jugados y partidos ganados de los campos de texto que aparecen debajo. Si todo es correcto se realizará la operación de incremento de partidos sobre el modelo. Tras cada operación se debe mostrar un mensaje de confirmación o error en la parte baja de la vista.