DEPARTAMENTO LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN	**PROGRAMACIÓN ORIENTADA A OBJETOS** (Prueba realizada el 8 de junio de 2015)	
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 sólo con los ficheros **.java** que hayáis realizado a la tarea creada en el campus virtual para ello.

Se desea desarrollar una aplicación para la gestión de pacientes a los que atiende el servicio de maternidad de un hospital. Para ello, se creará un proyecto `prMaternidad` con las clases siguientes:

1) (***0.25 ptos.***) La clase `MaternidadException` 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) (***2.25 ptos.***) La clase `Persona` mantendrá información sobre un paciente ingresado en el área de maternidad del hospital. En concreto, su nombre (`String`), un código (`int`) que se considerará único y la habitación (`int`) asignada a ese paciente. La clase dispondrá de constructores y métodos necesarios para:
   a. Construir un objeto de la clase dados los valores para las tres variables descritas anteriormente. Si el código proporcionado no es un número positivo mayor que cero o la habitación dada es negativa se debe lanzar una excepción de tipo `MaternidadException` informando de tal situación.
   b. Obtener el nombre (`String getNombre()`), el código (`String getCodigo()`) y el número de habitación (`String getHabitacion()`).
   c. La representación de una persona (`String toString()`) viene dada por su nombre, código y número de habitación separados por el carácter ':'.
      Ej: García, Ana:1002:132
   d. Dos objetos de la clase `Persona` son iguales si coinciden sus códigos, que recordemos que son únicos.
   e. Los objetos de la clase `Persona` se ordenan de forma natural por código.
   f. Se proporcionará una ordenación alternativa, que organice los objetos primero por nombre, ignorando mayúsculas y minúsculas, y, en caso de igualdad, por código. Esta ordenación deberá definirse en una clase aparte, llamada `OrdAlt`.

3) (***4 ptos.***) La clase `Maternidad` almacenará la información de los pacientes del servicio de maternidad en una estructura `pacientes` de tipo `SortedMap<Persona, SortedSet <Persona>>`, de tal manera que cada madre ingresada esté asociada al conjunto de bebés a los que ha dado a luz. La madre y los bebés suelen estar en la misma habitación, pero hay veces en qué ello no ocurre así. Por ejemplo, si alguno de los bebés debe ir a la incubadora (habitación con código 0).
La clase debe proporcionar los siguientes constructores y métodos públicos:

a. El constructor `Maternidad()` crea la estructura de datos. Se debe tener en cuenta que queremos ordenar las madres por nombre (orden alternativo mencionado anteriormente). Los bebés asociados a cada madre se ordenarán por código (orden natural).
El constructor `Maternidad(String)` recibe un nombre de fichero y, tras crear la estructura de datos, añade toda la información que contiene el fichero (ver apartados siguientes).

b. Los métodos `addPacientesFichero(String)` y `addPacientes(Scanner)` añaden información sobre pacientes a la estructura de datos. El primero lee la información de un fichero. El segundo desde el flujo de entrada proporcionado. Se supondrá que la información sobre cada madre y sus bebés se encuentra en una línea de texto, donde el carácter `'#'` separa la información de distintas personas. Para cada persona, sus datos particulares se separan con el carácter `':'`. Por ejemplo, la siguiente línea indica que la persona Ana García es madre de dos niños, Pedro y Nuria.

```
García,Ana:1002:132#López,Pedro:1034:132#López,Nuria:1036:0
```

Cualquier error de formato detectado (falta algún dato personal, o bien el dato no es del tipo correcto), provocará el lanzamiento de una excepción del tipo `MaternidadException`.
Se proporciona el fichero de datos `pacientes.txt` para probar el correcto funcionamiento de la solución implementada.

c. El método `addMadreBebes(Persona madre, Collection<Persona> bebes)` registra la madre y sus bebes en el servicio de maternidad. Nótese que la colección `bebes` puede estar vacía, lo que indica que dicha madre aún no ha dado a luz. Si la madre ya estuviese registrada se añadirán los bebés indicados por el segundo parámetro al conjunto de bebés que ya tenía previamente.

d. Se ha de redefinir el método `String toString()`. La representación de los objetos de la clase muestra toda la información contenida en la estructura de datos. El formato que se ha de utilizar es el que aparece en el fichero `pacientes.txt`. Es decir, en cada línea la información de una madre y sus bebés.

e. Los métodos `escribirFichero(String)` y `escribir(PrintWriter)` guardan la información de la estructura de datos en el fichero o en el flujo de salida dado respectivamente.

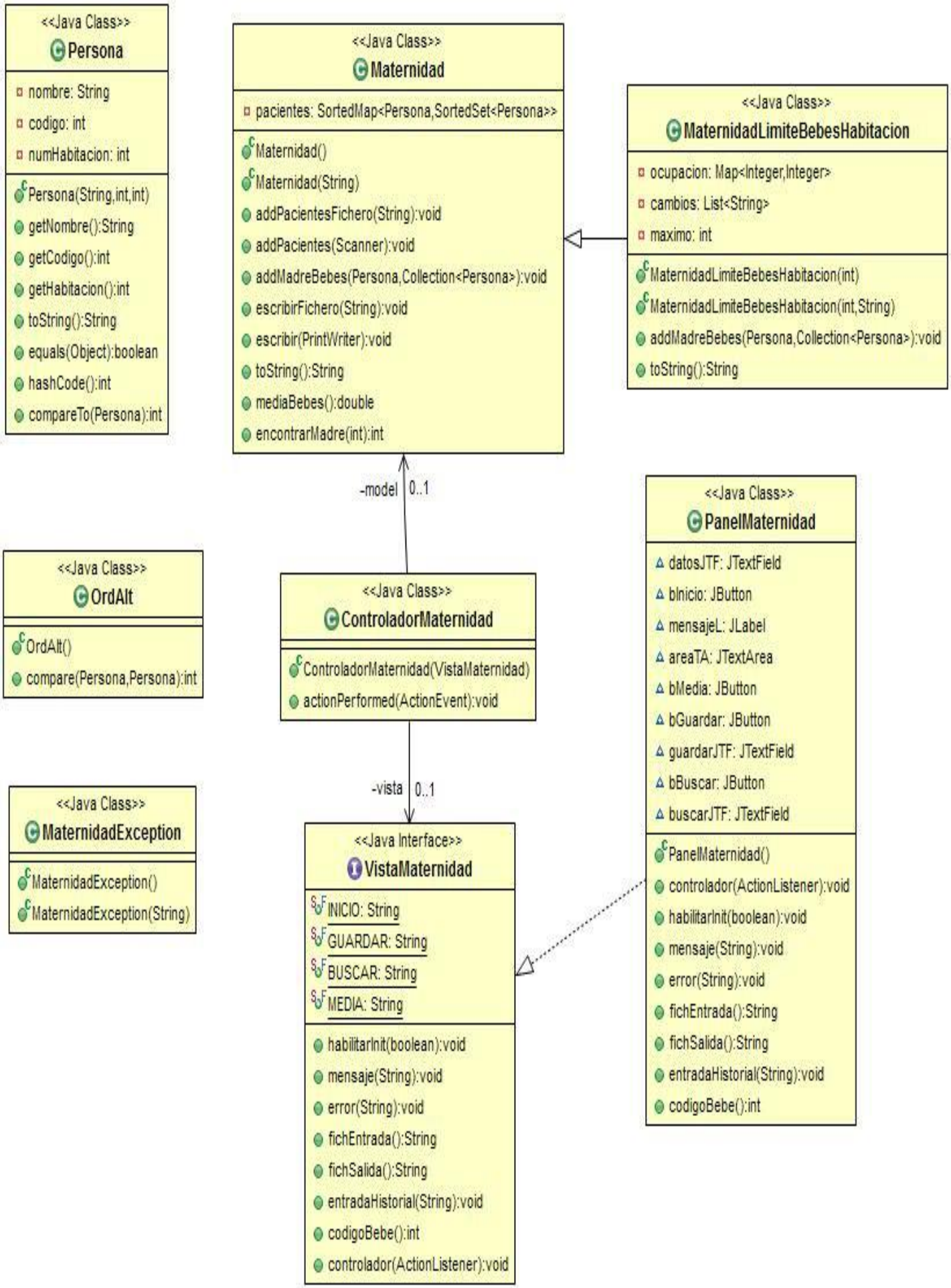
f. El método `double mediaBebes()` calcula el número medio de bebés por madre que están registrados en el sistema. Aquellas madres que no tienen hijos asociados **no** se tienen en cuenta en el cómputo.

g. El método `int encontrarMadre(long codigoBebe)` devuelve el número de habitación de aquella persona que es madre del bebé cuyo código se pasa como parámetro. En caso de no encontrarse la madre, se debe lanzar una excepción de tipo `MaternidadException` informando de tal situación.

4) (***2 ptos.***) La clase `MaternidadLimiteBebeHabitacion` se **comporta como la clase** `Maternidad`, con la diferencia de que va a detectar aquellas habitaciones que tienen un número de bebés que supera el valor máximo definido en la variable llamada `maximo` (`int`). Para cada habitación que supere el máximo permitido se guardará un mensaje informando de tal situación en la variable `cambios` (`List<String>`). Con el propósito de saber la ocupación actual de cada habitación esta clase cuenta con una variable `ocupacion` (`Map<Integer,Integer>`) que asocia cada habitación con el número de bebés actualmente asignados a ella. La variable `ocupación` se va actualizando cada vez que un nuevo bebé es registrado en el sistema y será consultada para ver si se supera el mencionado límite.
   a. Los constructores `MaternidadLimiteBebeHabitacion(int maximo)` y `MaternidadLimiteBebeHabitacion(int maximo, String nombreFichero)` deben crear todas las estructuras de datos: en el primer caso se crean vacías; en el segundo caso se añade toda la información del fichero cuyo nombre se pasa como segundo parámetro. El formato de dicho fichero es el mismo que el utilizado para la clase `Maternidad`.
   b. Redefinir `addMadreBebes(Persona madre, Collection<Persona> bebes)`. La nueva versión el método debe añadir la información de pacientes a la estructura de datos `paciente` sin tener en cuenta el límite de bebés. En caso de que se detecte que ello hace que alguna habitación supere dicho límite, se deberá añadir un mensaje informativo donde conste el número de habitación al listado de cambios (`cambios`) que han de hacerse. Se debe tener en cuenta que la incubadora (habitación con código 0) no estará sujeta a dicho límite.
   c. Redefinir `String toString()` para que a la información sobre las madres y bebés a las que atiende el servicio de maternidad mostrada en la clase `Maternidad`, añada todos los mensajes contenidas en la lista de cambios.

5) (***1.5 ptos.***) La clase `ControladorMaternidad` controla e interactúa con el modelo (clases `Maternidad` y `Persona`) y la vista (se proporcionan en el campus virtual la interfaz `VistaMaternidad` y la clase `PanelMaternidad`). El constructor debe habilitar la parte de inicialización de la vista (introducción del fichero de pacientes, 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 `Maternidad` (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 pacientes 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 "Buscar Madre" hará que se lea el código del bebé del campo de texto a su derecha y se muestre en el área de texto la habitación en que se encuentra su madre. El botón "Media" muestra en el área de texto el resultado de calcular la media de bebés. Tras cada operación se debe mostrar un mensaje de confirmación o error en la parte baja de la vista.

Las clases `Main` y `MainGUI` se proporcionan en el campus virtual, para que se pueda probar el funcionamiento de las distintas clases a implementar.


<<Java Class>>
Persona
nombre: String
codigo: int
numHabitacion: int
Persona(String,int,int)
getNombre():String
getCodigo():int
getHabitacion():int
toString():String
equals(Object):boolean
hashCode():int
compareTo(Persona):int
<<Java Class>>
Maternidad
pacientes: SortedMap<Persona,SortedSet<Persona>>
Maternidad()
Maternidad(String)
addPacientesFichero(String):void
addPacientes(Scanner):void
addMadreBebes(Persona,Collection<Persona>):void
escribirFichero(String):void
escribir(PrintWriter):void
toString():String
mediaBebes():double
encontrarMadre(int):int
<<Java Class>>
MaternidadLimiteBebesHabitacion
ocupacion: Map<Integer,Integer>
cambios: List<String>
maximo: int
MaternidadLimiteBebesHabitacion(int)
MaternidadLimiteBebesHabitacion(int,String)
addMadreBebes(Persona,Collection<Persona>):void
toString():String
-model
0..1
<<Java Class>>
OrdAlt
OrdAlt()
compare(Persona,Persona):int
<<Java Class>>
ControladorMaternidad
ControladorMaternidad(VistaMaternidad)
actionPerformed(ActionEvent):void
<<Java Class>>
PanelMaternidad
datosJTF: JTextField
bInicio: JButton
mensajeL: JLabel
areaTA: JTextArea
bMedia: JButton
bGuardar: JButton
guardarJTF: JTextField
bBuscar: JButton
buscarJTF: JTextField
PanelMaternidad()
controlador(ActionListener):void
habilitarInit(boolean):void
mensaje(String):void
error(String):void
fichEntrada():String
fichSalida():String
entradaHistorial(String):void
codigoBebe():int
-vista
0..1
<<Java Class>>
MaternidadException
MaternidadException()
MaternidadException(String)
<<Java Interface>>
VistaMaternidad
INICIO: String
GUARDAR: String
BUSCAR: String
MEDIA: String
habilitarInit(boolean):void
mensaje(String):void
error(String):void
fichEntrada():String
fichSalida():String
entradaHistorial(String):void
codigoBebe():int
controlador(ActionListener):void