Examen Programación II – Junio 2021

1. Planteamiento

Partimos de la miniagenda que has recibido los días anteriores. La modificación principal realizada en ese sistema es que ahora al entrar se pide una identificación de usuario (un string único para cada usuario) que permitirá realizar alguna de las tareas. También hay un nuevo botón que permite ese login en cualquier momento. El código ya está programado y funcionando (clase principal `GestorAgenda`).

2. Tareas

Sobre este código de partida, se pide que realices las siguientes tareas (puntuación s/10 entre corchetes, núm. aprox. de líneas entre llaves, y **algunos vídeos de apoyo a las tareas en ALUD**):

1. Herencia, polimorfismo [3] {70 l} [`CitaMedica, EspecialidadMedica, GestorAgenda, EspacioAgenda`]

1a [1] Crea una nueva clase **`CitaMedica`** en la jerarquía de `EspacioAgenda` que implemente `Editable` y permita crear nuevas citas específicas de salud, con color de fondo verde. Además de los elementos comunes, la cita médica debe tener un atributo String con el nombre del médico, y su especialidad sanitaria. Elige el mejor sitio posible **para reutilizar código** de cara a hacer esta herencia.

1b [0,5] Define la especialidad con un nuevo tipo enumerado EspecialidadMedica con los valores *GENERAL, OTORRINO, RESPIRATORIO, DIGESTIVO, TRAUMA, INTERNA, DERMATOLOGIA*.

1c [0,5] Incluye en la clase un método `equals` que compare adecuadamente dos citas médicas (misma descripción, fecha, duración, médico y especialidad), así como un método `toString` que devuelva descripción, médico y especialidad. La edición de cita médica debe pedir interactivamente (`JOptionPane`) médico, descripción y especialidad. Reutiliza en estos métodos todo el código posible.

1d [0,5] Añade a la inicialización de `EspacioAgenda` un tipo de espacio *"Cita médica"* y modifica el método *`crearNuevoEspacio`* para que se inicialice una nueva cita médica con especialidad por defecto *`GENERAL`*.

1e [0,5] Comprueba que todo funciona añadiendo en la inicialización de datos (método `initDatos` del `GestorAgenda`) una nueva cita médica con la médica Sandra de dermatología, 2 horas después de la hora actual. Puedes reiniciar los datos siempre que quieras borrando el fichero de datos *GestorAgenda.dat*.

2. Estructuras de Datos [2,5] {15 l} [`GestorAgenda`] Se pretende que el sistema permita guardar agendas independientes de cada usuario. Como todo el funcionamiento actual parte de una lista de espacios de agenda (`listaAgenda`), vamos a crear un mapa que asocie a cada usuario (String) una lista de espacios, de modo que al cambiar de usuario simplemente haya que cambiar la lista que el sistema maneja.

2a [0,5] Crea un nuevo atributo `mapaAgendas` que permita este trabajo e inicialízalo vacío.

2b [1,3] Observa que hay un método `login()` que se lanza tanto al inicio como cuando se pulsa el botón de la izquierda con el icono de los usuarios. Programa ese método para que borre la agenda actual y limpie la ventana (puedes usar el método ya existente) y reasigne la lista de espacios (`listaAgenda`) desde el mapa, bien con la lista ya existente si el usuario existe, o con una lista nueva vacía si el usuario no existía. Ojo, necesitarás llamar al método `reiniciaVentanaDeAgenda()` para que se reinicie ventana y panel al cambiar esa lista. Comprueba que funciona cambiando de usuario y viendo que las agendas son diferentes y se mantienen.

2c [0,7] Para comprobar que todo está yendo bien, añade un código a este método `login` que visualice en consola los usuarios actualmente en el mapa y sus agendas, con este formato de ejemplo:

```
Usuarios actualmente en agenda:
a - 3 espacios: [ordenar OTROS, limpiar HOGAR, padel DEPORTE]
b - 2 espacios: [tele OTROS, cine con Eva]
```

3. Ficheros y eventos [2] {30 l} [`GestorAgenda`] Queremos que la agenda sea persistente, de modo que cuando cerramos el programa se guarde la información y cuando se vuelva a abrir se cargue. Tienes preparados dos métodos para hacer la carga y el guardado de los ficheros, si has hecho la tarea 2 puedes guardar y cargar el mapa, y si no la has hecho guarda y carga la lista.

Para llamar al método `cargaDatosFichero` hazlo modificando el método `initDatos` de modo que se inicialicen los datos con valores de prueba solo si no se puede cargar el fichero.

Para llamar al `guardaDatosFichero`, debes añadir un evento a la ventana en el método `lanza`.

4. Componentes, layouts, eventos [2] {20 l} [`VentanaAgenda`] Verás que no cabe mucha información en cada recuadro de la agenda. Se propone que al pasar por encima de cada uno de estos espacios de agenda salga en la parte inferior de la ventana el detalle de fecha, hora y duración (a la izquierda) y su descripción (centrada).

Para ello deberás añadir un evento de ratón a cada espacio de agenda, lo puedes hacer en el método `anyadirGestorRaton` al que el programa llama cada vez que se crea un espacio.

Crea además en el constructor de la ventana atributos para un panel inferior, dos etiquetas que vayan dentro de él ajustadas a izquierda y centro, e inicializa esos componentes con el tipo de letra de mensajes ya existentes para que se vean como aparece en el vídeo de esta tarea.

5. Hilos [2,5] **Elige solo una** de las dos opciones siguientes para añadir un hilo a la implementación actual (observa el funcionamiento pretendido en los vídeos correspondientes):

- [A] [`GestorAgenda`] {10 l} Verás un método, `recalculaEspacioEnSuOrigen`, que mueve un espacio de agenda a su sitio original cuando el drag que se realiza lo coloca fuera de la ventana. Realiza esto mismo, pero con un hilo que haga una animación de 1 segundo (aprox.) moviendo la posición x,y cincuenta veces, con pausas de 20 milisegundos. Tienes código comentado con cálculos ya realizados de coordenadas, listos para hacer la animación.
- [B] [`VentanaAgenda`] {20 l} Añade un evento de ratón (al final del constructor) para que al hacer click a la vez que se pulsa la tecla Ctrl en la etiqueta de mensaje superior (`lMensaje`) se lance un hilo que muestra en esa etiqueta en azul todos los elementos de la agenda uno detrás de otro (2 segundos cada uno) y finalmente acaba volviendo a mostrar en negro el mensaje que originalmente estaba.

- Tienes en ALUD los ficheros con los que se va a trabajar en un fichero comprimido. Crea un proyecto en eclipse y descomprime ese fichero en la carpeta de proyecto sin cambiar los paquetes.
- En algunos casos verás un comentario `// Tarea n` en la parte de código correspondiente a la tarea (número *n*) a completar. Si escribes código en algún otro lugar, añade el mismo comentario.
- Verás también en ALUD una serie de vídeos que puedes revisar para entender mejor lo que se pretende en las tareas 1, 2, 4 y 5.
- Al acabar, haz .zip y entrega en ALUD (**asegúrate de que incluyes los ficheros java modificados**).