RELACION DE EJERCICIOS DE PROGRAMACIÓN.

1. Se quiere simular el comportamiento de una máquina fotográfica. Se tiene para ello:

- Un carrete tiene capacidad para un número determinado de fotografías (12, 24, 36). Sobre un carrete se puede hacer ***nueva*** lo que hace que se incremente el número de fotografías ya hechas, hasta que se finalice el carrete en cuyo caso no se permite realizar nuevas fotografías. Se tiene también los métodos ***sePuedeHacer*** que nos devuelve verdadero si quedan fotografías por hacer, **estaLLeno** que devuelve verdadero si el carrete se ha llenado y **numeroFotos** que nos devuelve el número de fotos que se han hecho actualmente del carrete. Un carrete se puede **revelar**. Este proceso hace que se obtenga fotografías válidas y no válidas. Las fotografías no válidas son aquellas que no se hayan hecho en el momento del revelado y las que salen veladas (se tiene un 2% de probabilidad de que una foto se vele).

- Una cámara fotográfica cuando se crea no tiene carrete. Se tendrá la función **ponerCarrete** que nos permitirá poner un carrete a la cámara fotográfica. Se tendrá el método **sePuedeHacerFoto** que devolverá verdadero si la cámara tiene un carrete y se puede hacer fotos con el carrete. Se tendrá el método **hayCarrete** que nos devolverá verdadero si tenemos un carrete puesto. Se podrá lanzar una **nuevaFotografía** siempre que se pueda hacer fotos y que hará una nueva fotografía en el carrete. En cualquier momento se podrá **quitarCarrete** (siempre que haya un carrete puesto). Al quitar el carrete no se pueden hacer fotografías hasta que se ponga otro carrete. Cuando se quita el carrete se **revela** y se almacena el número de fotografías válidas y veladas.

En cualquier momento se quiere saber: el número de carretes que se han puesto a la cámara fotográfica (en principio será 0), el número de fotografías válidas y veladas que se han hecho con la cámara y el número de fotografías que se han hecho del carrete actual.

Implementad las clases **Carrete** y **CamaraFotografica**.

2. Implementad la siguiente jerarquía de clases:

- Un coche es un vehículo de cuatro ruedas, que puede tener 2, 3 o 5 puertas. Se conoce de él el color, la marca y el modelo.

- Un camión es un vehículo con un número de ruedas que varía entre 6 y 20. Un camión tiene únicamente 2 puertas y de él se conoce el color, la marca y el modelo. Además para un camión interesa saber el número de ejes que tiene y el peso neto y con carga máximo.

Todos los atributos pueden consultarse en cualquier momento. Una vez establecidos los atributos no pueden modificarse. Cada clase tiene un

método descríbete que muestra en pantalla una descripción completa del vehículo.

Haced un programa que permita crear de forma dinámica un array de 20 vehículos y que nos permita para cada uno de los vehículos consultar todos sus atributos individualmente y mostrar la descripción completa.

3. Se tiene una tienda de informática en la que se venden **ordenadores**. Todo ordenador es de un fabricante y un modelo y tiene un precio de venta a público que se calculará a partir de sus componentes. Los ordenadores pueden ser de **sobremesa** y **portátiles**. Un ordenador de sobremesa está formado por un **monitor**, una **CPU** y un **teclado**. El monitor puede ser LCD o analógico, de 14", 15" o 17". La CPU debe guardar una descripción de las características de la misma: velocidad del micro, capacidad del HD, etc. El teclado puede ser de 102 teclas o de 101, inalámbrico ó con cable. Cada uno de los componentes tiene un precio. Los ordenadores portátiles tienen un monitor de 14" LCD, un teclado de 101 teclas con cable y tiene una CPU con iguales características que los ordenadores de sobremesa. Todas las características de las clases descritas anteriormente no pueden ser accedidas directamente, por lo que sólo pueden establecerse mediante el constructor o los métodos apropiados.

Haced un programa de prueba con el siguiente menú:

1.- Crear ordenador
2.- Mostrar características del ordenador
3.- Mostrar precio del ordenador
3.- Salir

Mostrar características del ordenador se hará mediante llamadas a los métodos de los objetos mostrando los valores que te devuelvan.

4. Se tiene la clase **SerVivo**. Para todo ser vivo sabemos el nombre *científico* y el nombre *vulgar*. Además se le puede preguntar *describete*() y pondrá una descripción completa de las características que tiene. Los nombres se deben poder modificar en cualquier momento con los métodos apropiados.

Sabemos que los seres vivos se pueden dividir en dos grandes grupos: **Animales** y **Vegetales**. Los animales tienen la característica de movilidad de la que no gozan los vegetales, es por ello que al describirse, los animales dicen puede moverse mientras que los vegetales dicen no puede moverse como descripción adicional.

Los animales tienen un numero de patas que se indican al crearse. Este número de patas se puede mostrar y modificar en cualquier momento y aparece cuando se llama a *descríbete*().

Los vegetales tienen un numero de partes que varía de unos a otros (raices, tallo, hojas, etc). Al crear el vegetal se da el numero de partes y una descripción de cada una de las partes (usar un array de cadenas de

caracteres). Estas partes pueden ser mostradas en cualquier momento y aparecen cuando se llama a *descríbete*().

Los animales pueden dividirse a su vez en **Mamíferos**, **Reptiles** y **Aves**. Todos ellos muestran en *descríbete()* la familia a la que pertenecen.

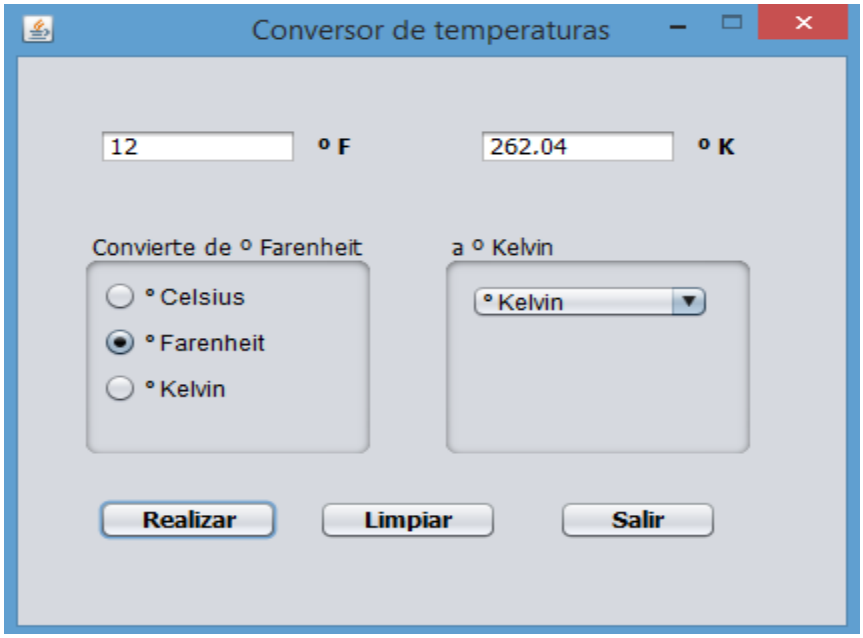
Todos los mamíferos tienen cuatro patas que no pueden modificarse nunca.

Los reptiles tienen la característica de sangre fría que los distingue del resto de familias.

Las aves pueden volar.

Implementad toda la jerarquía de clases y comprobar su funcionamiento.

5. Diseñad el siguiente GUI en Java, con el siguiente comportamiento:



Para ello, se introducirá en el *textField* de la izquierda la temperatura a convertir:

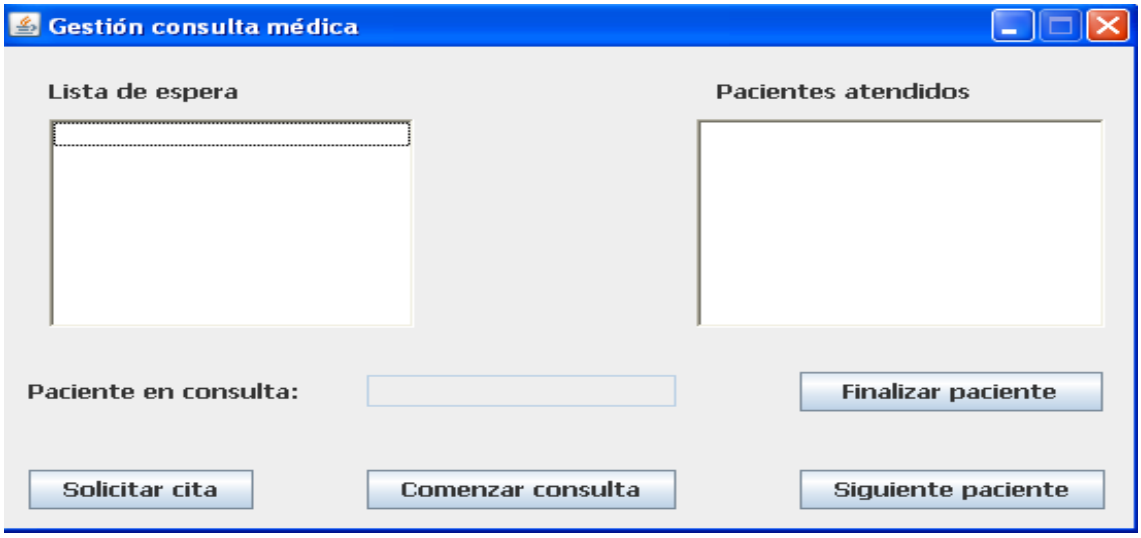
- Cada vez que se seleccione un nuevo *Checkbox* deberá modificar el *label* de la izquierda con el tipo de temperatura correspondiente. También modificará el título del *panel* que contiene a los *Checkbox*.
- Cada vez que se seleccione un nuevo ítem del *list* deberá modificar el *label* de la derecha con el tipo de temperatura correspondiente. También modificará el título del *panel* que contiene el *list*.
- Cuando se pulse el botón **Limpiar** limpiará los dos *textField*; seleccionará el *Checkbox* de etiqueta ***Celsius*** y el ítem del *list* de etiqueta ***Farenheit***. También dejará el foco en el *textField* izquierdo.
- Cuando se pulse el botón **Salir** finalizará la aplicación.

- El botón **Realizar**, realizará la conversión de la temperatura seleccionada en los *Checkbox* a su correspondiente en el *list*, teniendo en cuenta las siguientes fórmulas:

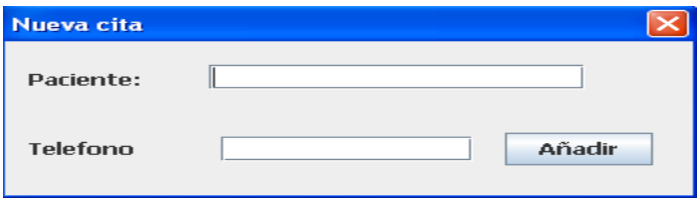
  celsius = (farenheit * 9 / 5) + 32 = kelvin + 273.15

El programa deberá controlar la excepción que se produzca si en el *textField* izquierdo se introdujera un dato erróneo o no se introdujera nada, visualizando un mensaje de error en una ventana de diálogo. El *textField* derecho será no editable.

6. Se quiere gestionar una consulta médica. Para cada *nueva cita* se desea almacenar información del *nombre* y *teléfono* del paciente. Para ello cread la clase ***Paciente*** con los apartados correspondientes para almacenar un paciente, y para mantener todas las citas usar una ***Colección***. Haced un programa cuya ventana principal sea la siguiente:



- Cuando se pulsa **Solicitar cita** se abre una ventana de diálogo para añadir un nuevo paciente a la ***colección***. Permitirá introducir cuantos pacientes se quiera, hasta cerrar la ventana de diálogo. Al pulsar **Añadir**, añadirá el paciente a la ***colección***, limpiará los dos *textField* y situará el foco en el *textField* **Paciente**.

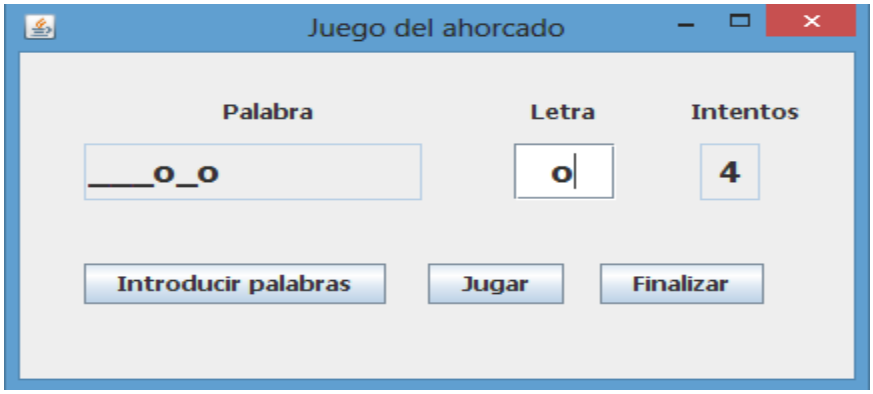


- Cuando se pulsa **Comenzar consulta** borrará todos los ítems de los dos *list* y luego copiará todos los pacientes almacenados en la ***colección*** al *list* **Lista de espera**, empezando por el último paciente en entrar en consulta y finalizando con el primero.
- Cuando se pulsa **Siguiente paciente** copiará el último ítem del *list* **Lista de espera** en el *textField* **Paciente en consulta** eliminándolo

del *list* y de la ***colección***. Deberá comprobar que queden aún pacientes. En caso de que no queden más pacientes deberá visualizar el correspondiente mensaje de error en una ventana de diálogo.

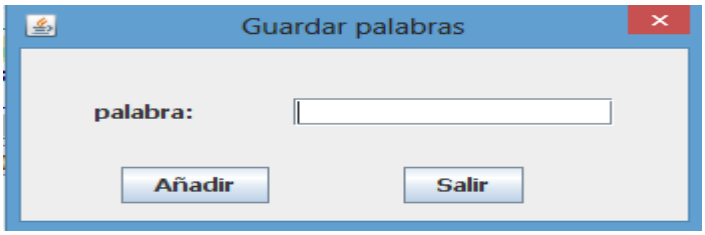
- Cuando se pulsa **Finalizar paciente**, borrará el paciente del *textField* **Paciente en consulta** y lo añadirá al *list* **Pacientes atendidos**.

7. Diseñad el siguiente GUI en Java, para jugar al **ahorcado**:



con el siguiente comportamiento:

- Cuando se pulse el botón **Introducir palabras** nos mostrará la siguiente ventana de diálogo (modal) que nos permitirá introducir cuantas palabras deseemos, almacenándolas en un objeto ***Colección*** (*Vector*, *ArrayList*…).

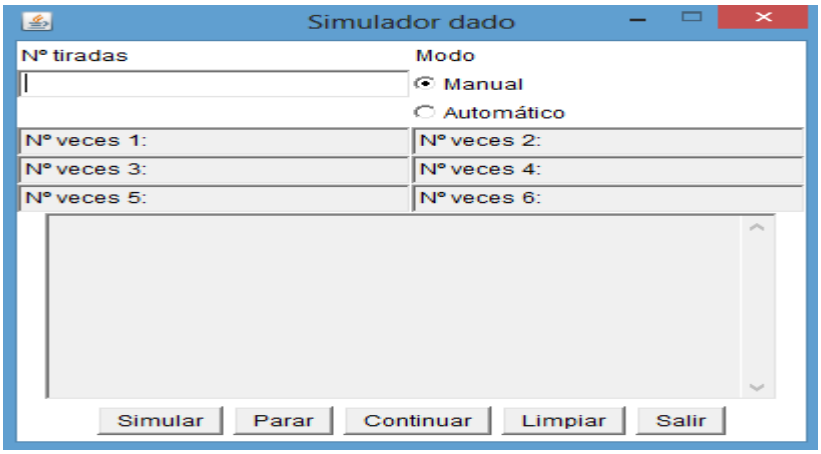


- Cuando se pulse el botón **Jugar**, comenzará una nueva partida. Para ello extraerá una nueva palabra, aleatoriamente, de la ***Colección,*** y mostrará en el *TextField* **Palabra** tantos guiones como caracteres tenga la palabra. También inicializará el número de intentos a 5, mostrándolos en *TextField* correspondiente. Ambos *TextField* serán no editables.

- Cada vez que introduzcamos un carácter en el *TextField* **Letra** deberá borrar el carácter que hubiera, mostrando únicamente el nuevo carácter. Al mismo tiempo deberá buscar dicho carácter en la palabra sustituyendo el guión por el carácter en cada coincidencia. También deberá decrementar el nº de intentos mostrándolos en el *TextField* correspondiente, siempre que dicho carácter no esté contenido en la palabra a buscar. Este proceso se repetirá hasta que acertemos la palabra o se agoten el nº de intentos. En cualquier caso mostrará una ventana de diálogo (modal) con el mensaje que corresponda. Finalmente mostrará otra ventana de diálogo (modal) preguntándonos si deseamos

volver a jugar. En caso negativo finalizará la aplicación. En caso afirmativo comenzará una nueva partida.

8. Diseñad el siguiente GUI, con el siguiente comportamiento:



- La ventana se debe poder cerrar y no puede ser redimensionada.
- Los *textField* “Nº veces” y el *textArea* deben ser no editables.
- Cuando se pulsa el botón *Simular* comprobará el modo seleccionado:

    - si el modo seleccionado es *manual*, habilitará los botones *Parar* y *Continuar* y simulará el lanzamiento de un dado las veces indicadas en “Nº tiradas”, utilizando para ello un **Thread**. Visualizará el número obtenido en el *textArea* correspondiente e incrementará el contador correspondiente. Con el botón *Parar* permitirá parar la ejecución del proceso y con el botón *Continuar* reemprenderá la ejecución del mismo.

    - si el modo seleccionado es *automático* inhabilitará los botones *Parar* y *Continuar*, leerá el valor del *textField* “Nº tiradas”, controlando los posibles errores, y repetirá el proceso de simular el lanzamiento de un dado, el número de tiradas introducido, visualizando los valores obtenidos en el *textArea* e incrementando los contadores correspondientes. Mostrará cada nuevo valor transcurridos 0.5 segundos.

- Cuando se pulsa el botón *Limpiar* debe limpiar el *textField* “Nº tiradas”y el *textArea*; poner los contadores “Nº veces” a 0; cambiar el modo a “manual”; y cambiar el foco al *textField* “Nº tiradas”.
- Cuando se pulsa el botón *Salir* debe finalizar el programa.