CUESTIONES Y EJERCICIOS

1. Descarga del repositorio el proyecto *U5.Extra.BDEmpleado.* En primer lugar, observa la clase *Empleado*. A continuación, observa que la clase *BDEmpleado* simula una base de datos que almacena datos de los empleados de una empresa, pero que puede sobrecargarse "aleatoriamente" y lanzar un java.sql.SQLException (una "checked" exception) indicando que pasó demasiado tiempo sin obtener la respuesta a una consulta.

   Tienes que darle código a los métodos privados que dan apoyo al método main, siguiendo los siguientes pasos:

   a) Cada método le pedirá al usuario la información que necesite por teclado para poder invocar al método correspondiente de la BDEmpleado.

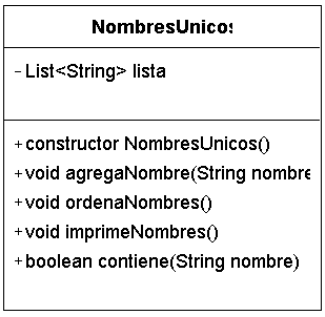
   b) A continuación, creará un objeto de tipo BDEmpleado, intentará realizar la conexión a la base de datos, llamar al método "selectXXX" que corresponda e imprimir el resultado correspondiente o algún mensaje que indique que no se encontraron registros que cumplan con las especificaciones indicadas por el usuario.

   c) Tanto si hubo fallo como si no lo hubo, se debe desconectar de la base de datos.

   d) En el caso de que haya producido una SQLException se debe imprimir el mensaje "Parece que la BD está sobrecargada... ¿Quieres reintentar la operación? (S/N)". A continuación, se leerá la respuesta del usuario. Si el usuario teclea 's' o 'S' entonces se repetirá el proceso desde el paso a). En caso contrario el método terminará.

   NOTA: Se recomienda el uso de un bucle *do… while* y una variable lógica que actúe como "bandera" indicando si se debe repetir la operación o no.

2. Vamos a crear la clase *NombresUnicos* correspondiente al siguiente diagrama UML, teniendo en cuenta que:

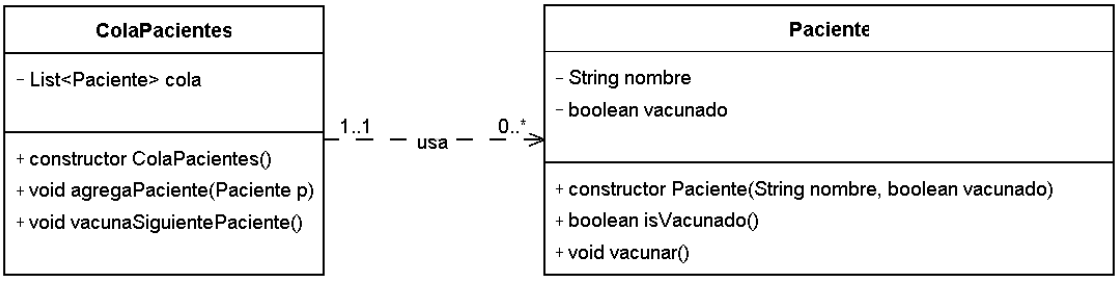


   - La clase usará una lista de cadenas para almacenar nombres de personas SIN DUPLICADOS.
   - El constructor de la clase debe inicializar la propiedad *lista*.
   - El método *agregaNombre* recibirá una cadena como parámetro y la añadirá a la lista solo en el caso de que no existiese es nombre previamente en la lista. Para ello se considerarán iguales dos nombres independientemente de la capitalización de las letras. Ej: Ana = ANA = ana
   - Además, el método *agregaNombre* lanzará una *NombreDuplicadoException* en el caso de que se intente guardar un nombre que ya había sido almacenado previamente.
   - El método *ordenaNombres* ordenará alfabéticamente la lista, sin imprimir nada.
   - El método *imprimeNombres* utilizará un iterador para imprimir los nombres en el siguiente formato:
     1. Ana
     2. Juan
     3. Pedro

- El método *contiene* devolverá un lógico que indica si la cadena que se recibe como parámetro está contenida o no en la lista.

Por último, realiza una clase *PruebaNombresUnicos* con un método *main* que cree un objeto de la clase *NombresUnicos* y pruebe todos sus métodos, capturando las excepciones que se produzcan.

3. Vamos a crear las clases correspondientes al siguiente diagrama UML:



Teniendo en cuenta que:

- En la clase Paciente, el método "vacunar" cambia el valor de la propiedad *vacunado* a *true* si es que el paciente no estaba vacunado. En caso contrario se debe lanzar una excepción *PacienteVacunadoException* con un mensaje descriptivo.
- En la clase ColaPacientes se utilizará una lista para modelar la cola de los pacientes para recibir la vacuna. Esto implica que siempre añadiremos un nuevo paciente al final de la cola.
- El constructor de la clase ColaPacientes debe inicializar la propiedad tipo List.
- El método *vacunaSiguientePaciente* comprobará si hay pacientes en la cola, lanzando una *ColaVaciaException* en caso de que esté vacía. En el caso de que sí haya pacientes, tomará el primer paciente de la cola y lo vacunará sin preguntar nada. En el caso de que se lance una *PacienteVacunadoException* deberá capturarse y escribirse el siguiente mensaje en la pantalla "¡Hay que estar aburrido para querer vacunarse 2 veces!". Tanto si el paciente se vacunó correctamente como si no se debe extraer el paciente de la cola dejándola lista para atender al siguiente.
- Todas las excepciones deben heredar de Exception.

Realiza también una clase PruebaColaPacientes con un método *main* que cree un objeto ColaPacientes y le agregue 4 pacientes, 3 de los cuales estarán sin vacunar y 1 vacunado.

Después se intentarán vacunar a todos los pacientes de la cola, para ello debes realizar un bucle tipo *while(true)* en el que se atenderán a todos los pacientes de la cola llamando al método *vacunaSiguientePaciente* en cada iteración. Cuando se lance la excepción *ColaVaciaException* se imprimirá el mensaje "Todos los pacientes están vacunados" y terminaremos el programa.