Examen Programación III

- Tienes en ALUD ("Exámenes y Evaluación") los ficheros de partida en un archivo comprimido. Crea un proyecto en Eclipse y descomprime las carpetas `src, test` de ese fichero en el proyecto.
- Para entregar el examen en ALUD, comprime las carpetas `src, test`, en un fichero.zip o .rar. Asegúrate de que tiene los ficheros que has modificado.
- Tiempo de examen: 2,5h (+ 1,5h opcional para las competencias 1 y 2).

Has estado revisando estos días un programa preparatorio con un chat simulando una IA conversacional (*ChatNoGPT*). Se ha incorporado a ese código una entidad adicional, `Conversacion`, que permite agrupar las frases de cada sesión diferenciada de conversación de un usuario con el sistema. Para ello está añadida ya la clase *Conversacion*, y modificada la estructura de la BD con la nueva tabla *conversacion* que tiene una clave externa de usuario (cada usuario puede tener varias conversaciones); y la clave externa *idConv* se ha añadido a la tabla frase (cada conversación tiene n frases). La BD de prueba que se genera al ejecutar *Main* la primera vez tiene datos de prueba de dos conversaciones del usuario "a" (password "a").

Tareas

Se pide que realices las siguientes tareas (su puntuación sobre 10 entre corchetes):

T1. Base de datos [4 – C3:10%] [`GestorPersistencia.java`, 35 líneas de código]
Verás un método *cargaConversaciones* en el gestor de persistencia que devuelve unos datos de prueba. Tienes que hacer que en su lugar devuelva desde la base de datos las conversaciones del usuario correspondiente, teniendo en cuenta las siguientes consideraciones:

- Cada usuario tiene de 0 a n conversaciones, y cada conversación tiene de 0 a n frases.
- Las conversaciones deben estar ordenadas por identificador.
- En cada conversación, las frases deben estar ordenadas por fecha.
- **Resolverlo haciendo SELECT independientes en las tablas de conversación y frase.**
- **No llamar a ningún otro método de la clase para solucionar la tarea** (es decir, codificar todo lo necesario en este método).

Todo debería seguir funcionando igual, pero ahora la tabla izquierda se cargará de la base de datos de conversaciones reales, no con los datos de prueba fijos.

T2. JUnit [4 – C4:10%] [`GestorPersistenciaTest.java`, 30 líneas de código]
Encontrarás una clase de JUnit 4 **`GestorPersistenciaTest.java`**, con un método que tienes que programar para comprobar que el método `cargaConversaciones()` de la clase `GestorPersistencia` correspondiente a la tarea T1 funciona correctamente.
Como ese método tiene un funcionamiento de prueba, puedes resolver esta tarea aunque no hayas resuelto la tarea 1; y también puedes utilizar esta tarea para comprobar que la tarea 1 programada funciona correctamente. Para pasar la comprobación de forma correcta esta prueba de unidad debe:

- Inicializar un gestor de persistencia con una base de datos de test (**utilizar un nombre de BD diferente al de la aplicación**) y prepararlo con los datos de prueba (método *crearTablasBD*).
- Probar los datos correspondientes al usuario "a".
- Para cada conversación:
  - Comprobar que el usuario de la conversación es el usuario "a".
  - Comprobar que el id es posterior al id de la anterior conversación.
  - Comprobar que la fecha es posterior a la fecha de la anterior conversación.
- Para cada frase dentro de cada conversación:
  - Comprobar que el idConv de cada frase coincide con el id de la conversación.
  - Comprobar que el primer emisor de cada conversación es el usuario "a", y que se irán alternando "a" con `Main.NOMBRE_CHAT`, frase a frase.
  - Comprobar que el id es posterior al id de la anterior frase.
  - Comprobar que la fecha **al menos 100 msg posterior** a la fecha de la anterior frase.

T3. Recursividad [2 – C5:5%] [`MainT3.java`, 20 líneas]

Ya que el funcionamiento de nuestro bot no es una generación real sino una elección aleatoria entre distintas opciones, queremos hacer algunos cálculos de cuántas conversaciones posibles pueden generarse. Para ello tienes preparada la clase *MainT3* que se ejecuta independientemente de las ventanas y permite hacer cálculos sobre las frases de respuesta posibles de *ChatNoGPT*.

Programa un algoritmo recursivo que genere todas las combinaciones de "n" respuestas consecutivas que puede dar el bot en una conversación (simplemente una lista de sus frases, sin considerar interacciones con el usuario).

Queremos calcular también las conversaciones que no incluyan ninguna pregunta (habrás observado que la mayor parte de las veces *ChatNoGPT* pregunta explícitamente alguna cosa). Para ello tienes ya definido un método *compruebaPosibilidades* que informa cuándo una lista de respuestas no contiene ninguna pregunta ni ninguna frase repetida.

Completa el algoritmo recursivo haciendo que cuente tanto las combinaciones totales como las combinaciones sin repetición ni pregunta, y sacando estas por consola. Intenta que los contadores sean parámetros de la llamada recursiva.

Para 5 frases, por ejemplo, la consola debería mostrar:

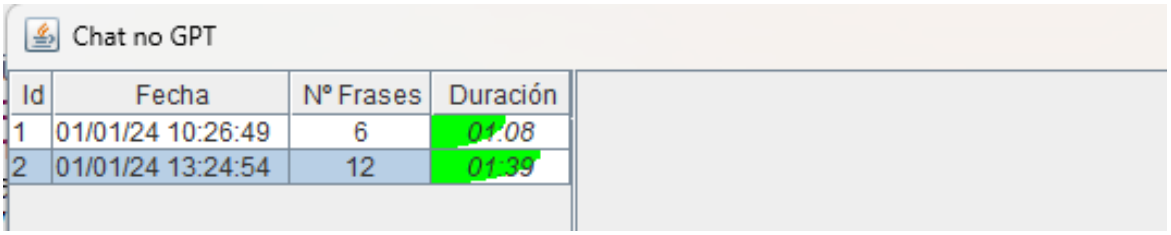
```
[Eso suena interesante. Cuéntame más., Me pregunto qué te llevó a esa conclusión., Sería
    interesante que me dieras un ejemplo de eso., Es relevante. Cuéntame más al respecto.,
    Tienes que sentir orgullo por cómo te has comportado.]
...
...   (dos mil y pico líneas)
...
[Me interroga lo que me comentas, no sé interpretarlo., Me interesa el impacto que crees que
    eso tendrá a largo plazo., Tienes que sentir orgullo por cómo te has comportado., Es
    relevante. Cuéntame más al respecto., Sería interesante que me dieras un ejemplo de eso.]
Combinaciones totales: 6240321451
Combinaciones sin pregunta: 2520
```

T3 Extra. [1] Verás que con la combinatoria de solo 5 frases el programa puede tardar unos segundos o minutos en ejecutarse. Si quieres intentar realizar la poda exponencial para que solo se sigan explorando aquellas opciones que cumplen la no repetición y no pregunta, se debería ejecutar mucho más rápido y podrás llegar a descubrir cuántas combinaciones de 6 frases sin pregunta. (¿Y de 7? ¿tiene sentido?) Responde en los comentarios de tu código.

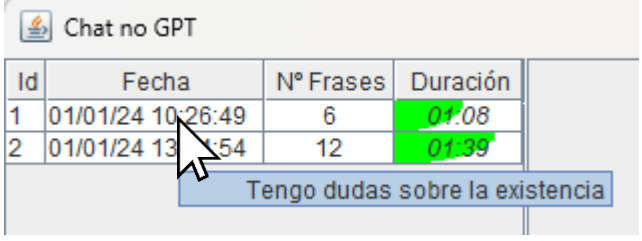
PARTE 2 – RECUPERACIÓN/MEJORA DE COMPETENCIAS PREVIAS

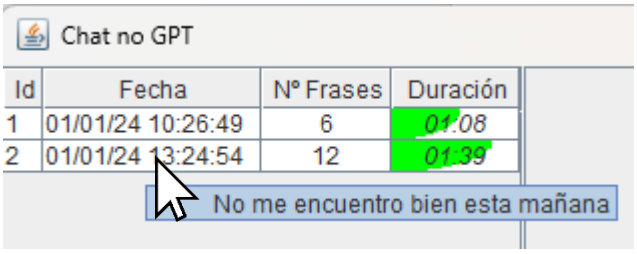
T4. Swing [C1: 10%] [`GestorUsuarios.java` (120 líneas)]
Verás en el método `cargaTablaConversaciones()` que se define un modelo de datos mínimo para la tabla de conversaciones que hay a la izquierda de la ventana. Mejora esta tabla cambiando ese código por la llamada al método `defineTablaConversaciones`, que deberás programar siguiendo las siguientes indicaciones:

- Crea un modelo personalizado de datos de tabla con una clase nueva (defínela abajo como clase interna de la misma clase). Considerando:
  - Recibe y guarda la lista de conversaciones
  - Tiene tantas filas como conversaciones
  - Tiene 4 columnas, nombradas "*Id*", "*Fecha*", "*Nº Frases*", "*Duración*"
  - Los valores de cada una de esas columnas son de los tipos *String, Long, Integer, Integer*
  - No es editable
  - El valor que devuelve en las columnas es el siguiente:
    - Columna 0: id de la conversación convertido a string
    - Columna 1: fecha de la conversación en milisegundos
    - Columna 2: número de frases que contiene la conversación
    - Columna 3: número de segundos que dura la conversación (diferencia de tiempo entre primera y última frase)
- La anchura preferida de las columnas debe ser 20, 110, 65, 65 píxeles (respectivamente)
- El renderer de la columna 0 debe ser el renderer por defecto.
- El renderer del valor Long de la columna 1 debe visualizar la fecha de la conversación en formato dd/mm/aa hh:mm:ss.
- El renderer del valor Integer de la columna 2 debe estar centrado.
- El renderer del valor Integer de la columna 3 debe ser una etiqueta con visualización personalizada donde:
  - El texto esté centrado.
  - El tipo de letra sea Arial itálica de 12 puntos de tamaño.
  - En el fondo se dibuje un cuarto de óvalo verde, centrado en la esq uina superior izquierda, con radio vertical = altura del label, radio horizontal = segundos dividido por 2 (el tamaño horizontal es dependiente de la duración).



- Al pasar el ratón por encima de la tabla, debe aparecer un tooltip donde se indica la primera frase de la conversación correspondiente a la fila.





T5. Estructuras de Datos [C2: 10%] [`MainT5.java` (25 líneas)]
Completa el método `procesoT5()` para mostrar todas las repuestas que en nuestro sistema hacen los usuarios a cada una de las frases lanzadas por *ChatNoGPT*. Sigue los siguientes pasos:

- Ordena la lista de frases por identificador de conversación en primer lugar, por fecha en segundo.
- Recorre la lista y crea un mapa en el que cada una de las frases lanzadas por ChatNoGPT se asocie a todas las respuestas que ha dado cualquier usuario a esa frase.
- Diseña la estructura para que permita que las frases de ChatNoGPT estén en orden alfabético, y que las respuestas que dan los usuarios estén sin repetición y también en orden alfabético.
- Visualiza como resultado en la consola cada una de las frases de ChatNoGPT (en orden alfabético) seguida de todas las respuestas que han dado los usuarios (en orden alfabético), con el siguiente formato (basado en los datos de ejemplo). Imprime solo las frases de ChatNoGPT que han tenido alguna respuesta:

```
Eso es bastante curioso. ¿Puedes explicar por qué?
      Porque ayer fue una noche de locura
Eso suena como un gran cambio ¿Cómo lo enfrentaste?
      Con mucha entereza
      Pues mareado, desconectado, desanimado...


...

¿Cómo mantienes tu motivación en situaciones así?
      Pues a base de refrescos de cola sin azúcar
```