Universidad Metropolitana

Departamento de Gestión de Proyectos y Sistemas.
Estructuras de datos (BPTSP06).
Trimestre 2425-1.

Prof. Fernando Torre, Antonio Guerra y Christian Guillén-Drija

Proyecto: Árbol Genealógico

Preliminares

En el siguiente proyecto, usted tendrá que realizar una investigación documental que le permita obtener información sobre el contexto del problema. En tal sentido, se sugiere que comience por realizar la siguiente lectura relativa a Hash Tables, pero tome en cuenta que es solo un recurso inicial que debe ser complementado con la búsqueda autónoma de información por parte de los integrantes del equipo de trabajo:

https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/

El problema

La Mano del Rey ha encontrado algunas discrepancias llamativas en el Registro de Linajes que desea analizar. Sin embargo, el registro se encuentra escrito en un lenguaje antiguo y críptico. Para ayudar a descifrarlo, el Rey le ha ordenado al Escriba Real, transcribirlo en un formato más legible que permita ver los árboles genealógicos de cada linaje (apellido o casa). Por desgracia, el formato estructurado que el Escriba conoce es el formato JSON, por lo que sigue siendo indescifrable para La Mano del Rey.

Se le ha encomendado a su equipo la tarea de realizar un visor para estos JSON que le permita a La Mano del Rey navegar entre los registros. En particular, su solución debe permitirle, dado, el nombre de un integrante de cierto linaje, conocer el registro de su padre o representante (si este existe) y cuáles son los registros de sus hijos o dependientes (para aquellos que tengan registros).

Debido a que los linajes del Reino son patrilineales (es decir, solo los herederos varones son elegibles para asumir el título de su padre) debe considerarse solo el padre, y no la madre, en esta solución.

Requerimientos Funcionales

Debe realizar un programa en Java utilizando la librería de interfaces gráficas de Java Swing que permita:

1. **Cargar un nuevo árbol genealógico:** El programa debe, en cualquier momento, poder cargar un archivo con un nuevo árbol genealógico. Si ya hay un árbol cargado, el programa deberá reemplazarlo con el siguiente a analizar. El sistema debe indicar el nombre del Linaje cargado en memoria y debe mostrar de forma gráfica dicho árbol. Se sugiere utilizar GraphStream (https://graphstream-project.org/).
2. **Ver Registro:** El programa debe ser capaz de mostrar todos los datos del integrante del linaje que se está revisando, el cual debe corresponder con un nodo del árbol. Esto deberá realizarse cuando el usuario haga clic sobre alguno de los nodos del árbol. La búsqueda de la información correspondiente al nodo debe tener una complejidad lo más cercano a O(1).
3. **Buscar por nombre:** esta búsqueda se deberá hacer de la siguiente forma:
    a. Primero, el programa debe poder enumerar todos los registros que coincidan con una cadena de búsqueda correspondiente al nombre o mote dado (dicha operación no debe tener una complejidad mayor a O(n).
    b. Una vez enumerados, debe poderse escoger alguna de las opciones presentadas para ver el detalle. Nótese que los nombres no son únicos (el hijo de William se llama “William”, pero se distingue por tener el numeral “segundo” de su nombre), pero los motes sí son únicos. Una vez seleccionado un nombre, entonces el sistema mostrará de forma gráfica el árbol correspondiente a su descendencia, comenzando con él. El mecanismo de búsqueda del integrante, deberá tener una complejidad lo más cercano a O(1).
4. **Mostrar antepasados:** Dado el nombre de un integrante del linaje, el sistema deberá mostrar la lista de nombres de sus antepasados de forma ordenada. El programa deberá presentar dicha información mediante un grafo (lista), y una vez más, el usuario podrá hacer clic sobre los nodos para observar la información correspondiente a ese antepasado.
5. **Buscar por Título:** El programa debe poder enumerar todos los registros tenedores de un mismo título nobiliario específico. Una vez enumerados, debe poderse escoger alguno de estos registros para ver el detalle.
6. **Lista de integrantes de una generación:** El usuario podrá solicitar al programa la lista de los integrantes de una generación dentro del linaje.

Formato de Archivo

Los archivos se presentan en un archivo JSON con en el siguiente formato:

```
{
    <Nombre de la casa>:[
        {
            <Nombre Completo₁>:[
                {"Of his name":<Numeral₁>},
                {"Born to":<Padre₁>},
                {"Born to":<Madre₁>},
                {"Known throughout as":<Mote₁>},
                {"Held title":<Título nobiliario₁>},
```

```
            {"Wed to":<Esposa_1>},
            {"Of eyes":<Color_1a>},
            {"Of hair":<Color_1b>},
            {"Father to":[
                    <Nombre 1>,
                    ...
                    <Nombre n_1>
                ]
            },
            {"Notes":<Comentarios sobre su vida>},
            {"Fate":<Comentarios sobre su muerte>}
        ]
    },
    ...
    {
        <Nombre Completo_m>:[
            {"Of his name":<Numeral_m>},
            {"Born to":<Padre_m>},
            {"Born to":<Madre_m>},
            {"Known throughout as":<Mote_m>},
            {"Held title":<Título nobiliario_m>},
            {"Wed to":<Esposa_m>},
            {"Of eyes":<Color_ma>},
            {"Of hair":<Color_mb>},
            {"Father to":[
                    <Nombre 1>,
                    ...
                    <Nombre n_m>
                ]
            },
            {"Notes":<Comentarios sobre su vida>},
            {"Fate":<Comentarios sobre su muerte>}
        ]
    },

}
```

Donde las líneas en gris son opcionales, *n* es el número de hijos y *m* es el número de registros. Nótese que la combinación “<Nombre completo>, <numeral>” será única, por lo que puede ser usada para identificar unívocamente a los padres. Los motes, al también ser únicos, pueden ser usados para identificar unívocamente a los padres. Los hijos se enumeran solo con su nombre de pila; debe usarse la información del padre en el registro correspondiente para encontrar si el registro de ese hijo existe. Debido a que el árbol genealógico muestra múltiples generaciones, puede haber múltiples tenedores del mismo título nobiliario a lo largo de la historia.

Los archivos de entrada a analizar son los siguientes:

- Baratheon.json
- Targaryen.json

Requerimientos técnicos

1. Debe tener una clase Árbol (Tree), la cual debe almacenar el árbol genealógico.
2. Debe tener una clase Tabla de Dispersión (Hash Table), la cual debe poder encontrar los nombres, motes O(1)
3. Puede utilizar cualquier otra estructura auxiliar de ser necesario para mejorar los tiempos de respuesta del programa. Sin embargo, **NO podrá utilizar librerías para la implementación de alguno de tipos de datos abstractos.**
4. La aplicación debe ofrecer una interfaz gráfica al usuario. No se permite la entrada ni la salida de información por consola.
5. El programa debe poder cargar un archivo de texto para la lectura de datos.
6. Debe documentar el proyecto con Javadoc.
7. Los equipos de trabajo deberán utilizar **GitHub** para el control de versiones y facilitar el trabajo en equipo de manera asíncrona. De esta forma, podrán comenzar a crear su portafolio de trabajos, elemento que puede ser importante a la hora de buscar trabajo. En el registro se deberá reflejar la participación activa y significativa de los integrantes.

Consideraciones

- Los equipos pueden tener como **máximo 3 personas.**
- Los proyectos que no tengan interfaz gráfica, serán calificados con **0 (cero)**.
- Los proyectos que no tengan documentación alguna, serán calificados con **0 (cero)**.
- Los proyectos que implementen el tipo de dato abstracto usando librerías como java.util, serán calificados con **0 (cero)**.
- Los proyectos que sean iguales o parecidos, serán calificados con **0 (cero)**.
- Los programas que “no compilen” o “no corran”, serán calificados con **0 (cero).** Tenga en cuenta que el hecho de que el programa muestre una interfaz gráfica de usuario sin funcionalidad alguna, será considerado como que ***no*** corre.
- Los registros de Github que tengan todos los commits de un solo integrante o en un solo día, serán considerados como ***no válidos***, pues no reflejan la participación activa y significativa de todos los integrantes del equipo (punto 7 del apartado de requerimientos técnicos).
- Los proyectos **podrán ser sometidos a defensa**, es decir, el facilitador convocará al equipo para una revisión si lo considera necesario.

Criterios de evaluación

- *Funcionalidad:* Capacidad para proporcionar las funcionalidades que satisfacen las necesidades explícitas e implícitas bajo unas ciertas condiciones: **40%**

    - *Adecuación:* El programa ofrece todas funcionalidades que respondan a las necesidades, tanto explícitas (contenidas en el documento descriptivo del proyecto) como implícitas; entendiendo como necesidades implícitas, aquellas que, no estando descritas en el documento, surgen como resultado de un concienzudo análisis del problema planteado y que aseguran el correcto funcionamiento del programa.

    - *Exactitud*: El programa genera los resultados o efectos correctos o acordados, con el grado necesario de precisión.

- *Fiabilidad*: Capacidad para mantener un nivel especificado de prestaciones cuando se usa bajo ciertas condiciones: **20%**

    - *Madurez*: El programa no presenta fallas originadas por errores de programación, análisis o diseño. **(10%)**

    - *Tolerancia a fallos*: El programa responde adecuadamente al manejo inadecuado del usuario; es decir, mantiene su correcto funcionamiento aun cuando el usuario introduzca datos erróneos o manipule inadecuadamente las interfaces de usuario. **(10%)**

- *Usabilidad*: Capacidad del proyecto para ser entendido, aprendido, usado y al mismo tiempo, ser atractivo para el usuario, cuando se usa bajo condiciones específicas: **10%**

    - *Comprensibilidad*: El programa ofrece una interfaz de fácil comprensión, facilitando su aprendizaje y correcta utilización. El programa emite mensajes de alerta cuando se introducen valores erróneos. Existen elementos informativos que indican al usuario como operar el programa. **(5%)**

    - *Capacidad de ser atractivo*: El diseño de la interfaz de usuario, esto es: disposición de controles, esquema de colores, utilización de cajas de diálogo y demás elementos gráficos; es atractivo para el usuario. **(5%)**

- *Eficiencia*: Capacidad para proporcionar prestaciones apropiadas, relativas a la cantidad de recursos utilizados, bajo condiciones determinadas: **30%**

    - *Estructuras de datos*: Utiliza eficientemente las estructuras de datos para la solución del problema dentro de valores óptimos de complejidad temporal y espacial.