PRÁCTICA 4

La práctica corresponde al:
Tema 5: Estructuras Arbóreas

Peso en la nota final:
6%

Objetivos

Se espera que el alumno se familiarice con la implementación interna de algunas de las estructuras de datos que se incluyen en el *Java Collections Framework*, así como el contenido completo de una de las ramas de la jerarquía y las posibilidades que existen para extender la jerarquía con nuevas implementaciones.
En el caso de la práctica actual se trabajará sobre las estructuras conocidas como **Set** (o Conjunto en español), empleando una implementación de éstas para incluirse dentro de las librerías de Java como un *SortedSet*.
Al igual que en anteriores prácticas se trabajará con la Genericidad de Java, que permite definir los tipos de datos a almacenar en una colección. Es una característica muy importante que mantienen todos los componentes del *Java Collections Framework*.
Se incorporará de manera implícita a la forma de trabajo y evaluación de las prácticas. Esto implica que se valorará como un aspecto más de las respuestas aportadas.

Conceptos Teóricos

Caché LRU (Least Recently Used)

Una caché LRU es un tipo de estructura de datos que almacena elementos en función de su uso reciente. El algoritmo LRU reemplaza el elemento menos recientemente utilizado cuando la caché alcanza su capacidad máxima. Esto significa que cada vez que se accede a un elemento, se actualiza su posición en la caché, manteniendo los elementos más recientemente utilizados en posiciones más accesibles. Este tipo de caché es útil para mejorar el rendimiento de sistemas que requieren acceso rápido a datos frecuentemente utilizados[1].

Interfaces de Set y SortedSet en Java

Set: La interfaz Set en Java es parte del *Java Collections Framework* y representa una colección que no permite elementos duplicados. Implementaciones comunes de Set incluyen `HashSet`, `LinkedHashSet` y `TreeSet`. Los conjuntos son útiles cuando se necesita garantizar que no haya duplicados en una colección de elementos.

SortedSet: La interfaz `SortedSet` extiende Set y proporciona una colección que mantiene sus elementos en orden. Este orden puede ser el orden natural de los elementos (definido por la implementación de Comparable) o un orden definido por un `Comparator` proporcionado en el momento de la creación del conjunto. `SortedSet` incluye métodos

[1] https://en.wikipedia.org/wiki/Cache_replacement_policies

adicionales para aprovechar el orden, como first(), last(), headSet(), tailSet(), y subSet(). Implementaciones comunes de `SortedSet` incluyen `TreeSet`.

Para más detalles, puedes consultar la documentación oficial de Java

Trabajo del Alumno

Se solicita al alumno la implementación de un nuevo tipo de datos en java que permita implementar un conjunto de datos que deberá funcionar como una caché de datos, con un algoritmo de reemplazo LRU que tendrá en cuenta el momento de ACCESO a cada dato (no solo el momento de inserción).
Para realizar esta tarea, la clase deberá implementar el interfaz `java.util.SortedSet`. Se recomienda así mismo que se extienda la clase `java.util.AbstractSet`, para sobrescribir solo los métodos necesarios. La clase creada deberá permitir almacenar objetos de cualquier clase (será por tanto una clase genérica).
De esta forma, el conjunto tendrá un tamaño máximo que no se podrá superar. Al almacenar o acceder a un elemento se adjuntará al mismo una información de acceso que permitirá mantener un orden de acceso entre todos. Se podrá así identificar los elementos menos utilizados para que si al insertar se va a superar el tamaño máximo, se puedan eliminar éstos para mantener el tamaño. Se considerará un conjunto ordenado y el orden a respetar para los elementos contenidos será el del último acceso.

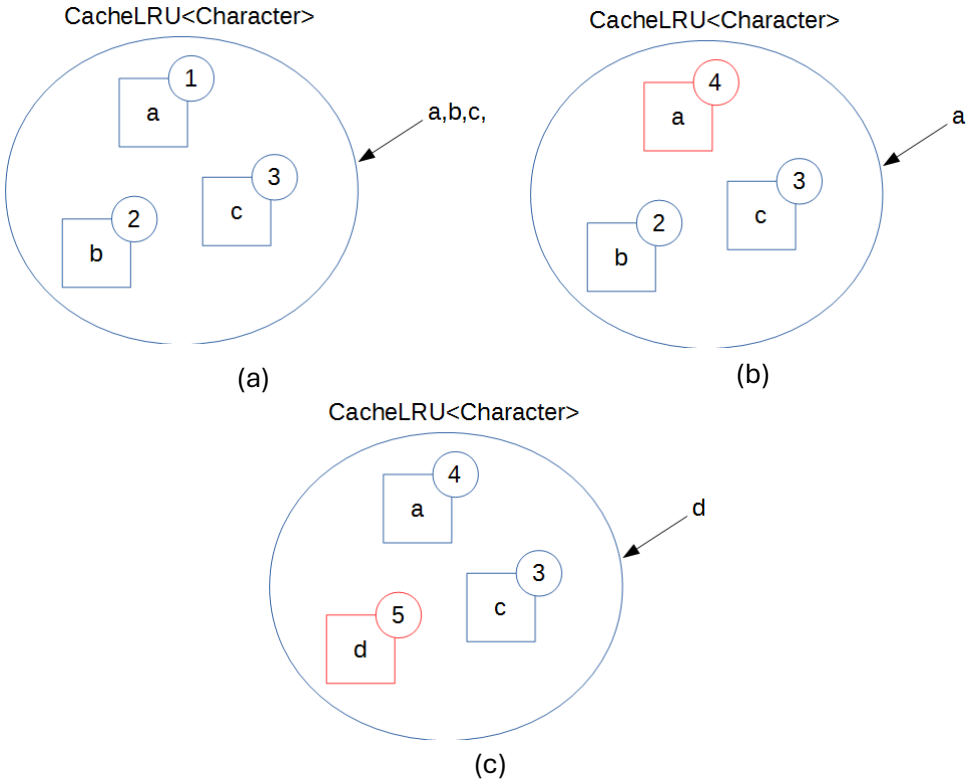
1. Programa

1. Se definirá la clase ConjuntoLRU, para que extienda la clase abstracta AbstractSet e implemente el interfaz SortedSet
2. Como se pretende que esta caché tenga un tamaño fijo, se deberá implementar un **constructor** que reciba como parámetro el número máximo de elementos que puede almacenar. Se deberá restringir el acceso al constructor por defecto para que no se pueda inicializar sin tamaño.

   Puesto que el orden que se pretende mantener es el propio orden en el que se han incluido o consultado los elementos contenidos, NO es necesario definir el comparador que sería necesario en otros casos.
3. Se dispondrá de un contador de accesos general para toda la clase. Este se incrementará en 1 unidad cada vez que se realice un acceso al contenido de la misma. Para implementar el algoritmo LRU, cada elemento almacenado tendrá asociado un número que será el valor de ese contador la última vez que se accedió a él. De esa forma, el elemento cuyo valor asociado sea el menor en el conjunto será el elemento accedido hace más tiempo, mientras que aquél que tenga el número más alto será el último al que se accedió.

4. Ya que se desea que esta clase sea modificable, se deberán sobrescribir al menos los métodos `add` y `remove`.

   El método de `add` permitirá asociar a cada elemento el indicador de inserción. En caso de que se intente añadir un elemento que ya se encuentra en el conjunto, no se almacenarán duplicados, pero se actualizará el indicador de acceso para indicar que ha sido accedido nuevamente.



En la figura se puede ver como:

a) se insertan sucesivamente “a”, “b” y “c”
b) se inserta de nuevo la “a”: no se incluye un duplicado, sino que actualiza el indicador del último acceso
c) se inserta “d”, suponiendo que el tamaño máximo definido es de 3. En este caso se elimina el que se accedió más anteriormente en el tiempo (en ese caso la “b”), para dejar espacio al nuevo elemento.

5. Como se debe también realizar la actualización de en qué momento se accede a cada elemento, se debe así mismo sobrescribir el `Iterator` asociado, para que según se vayan accediendo a los elementos, se actualice la información correspondiente a su acceso.
6. Finalmente, al ser un conjunto ordenado, se deberá poder acceder a los elementos en función de su último acceso, a través de los métodos: `first`, `last`, `headSet`, `tailSet` y `subSet`.
   La consulta en función del momento de acceso: `first` devolverá el elemento que se accedió hace más tiempo, mientras que `last` devolverá el último elemento al que se ha accedido.
   El significado que se da al parámetro o parámetros que se emplean en `headSet`, `tailSet` y `subSet` será el de p.ej. `headSet`: “obtener los elemento que se han accedido por última vez hace más tiempo que el que se facilita como parámetro” y de forma análoga en los otros dos.
   En caso de que los elementos sobre los que se consulten no estén incluidos en el Set se lanzará la excepción `IllegalArgumentException`.

Consideraciones:

- Se debe implementar la genericidad en la clase entregada.
- Se deben implementar todos los métodos solicitados en el apartado anterior y opcionalmente los que el alumno crea conveniente.
- El código entregado deberá estar debidamente comentado, siguiendo la especificación javadoc. Se solicita también la inclusión de comentarios en línea que describan las partes que el alumno considere más interesantes.

Se recomienda revisar la documentación oficial de Java:
https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/Set.html
https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/SortedSet.html
https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/NavigableSet.html

2. Informe

Se pide al alumno que, una vez completada la clase a programar, realice el un informe sencillo en el que se incluya un razonamiento de la complejidad algorítmica de cada uno de los métodos que se han implementado. No es necesario realizar un análisis completo y detallado de cada método (es decir, implementar las pruebas de tiempo de ejecución para el análisis de complejidad), pero sí incluir un razonamiento que relacione el código fuente que se facilita y a la complejidad que se considera que tiene el mismo.

La extensión del informe será de aproximadamente 5 páginas (incluyendo portadas, índices, gráficas…etc.).

Condiciones de Entrega

- La práctica se realizará en grupos de 1 o 2 personas.
- La entrega se hará SOLAMENTE por medio de la plataforma UBUVirtual. Cada miembro del grupo deberá subir su propia copia de la solución. No se admitirán soluciones fuera de plazo o por otros medios de entrega (e-mail, mensajería interna, …).
- La entrega incluirá la información de quienes son los autores de la misma, tanto en los ficheros de código fuente como en los documentos asociados.
- Cada entrega consistirá en un fichero comprimido (formato .zip), con la estructura de nombre
  “`Apellidos1Nombre1_Apellidos2Nombre2`”
- Qué deberá incluir el fichero comprimido:
  - Código fuente de la solución (dentro de la estructura de paquetes necesaria)
  - Documentación en formato Javadoc
  - Informe en formato PDF (en aquellas prácticas que se solicite).
- No hace falta entregar ficheros binarios

Criterios de Evaluación

- **Corrección del funcionamiento**. Se valorará como aspecto más importante que el funcionamiento de la estructura de datos implementada cumpla con todas las condiciones solicitadas, tanto en la documentación oficial como en el enunciado. Se facilitan al alumno un subconjunto de los tests que se emplearán en la corrección, pero se aconseja que el alumno realice de forma adicional los que crea conveniente.
- **Complejidad algorítmica**: Como parte fundamental de los conocimientos de la asignatura, se considerará la complejidad algorítmica de las soluciones proporcionadas el segundo aspecto más importante a valorar en los ejercicios. Penalizará en la nota aquellas soluciones que, a pesar de proporcionar salidas correctas, lo hagan con una complejidad mucho mayor de la estrictamente necesaria.
- **Informe**: se tendrá en cuenta la corrección del informe de complejidad algorítmica de la solución. No es necesario un informe excesivamente extenso, sino que se centre en la comparación de la complejidad algorítmica solicitada. La extensión aproximada es de 5 páginas.
- **Corrección del código**. Ausencia de *warnings* u otros elementos no deseables como variables no utilizadas, código que no se emplea, catch vacíos, ausencia de marcas sobre los métodos (*@Override*, @...) etc.
- **Documentación**. Las explicaciones y razonamientos que se incluyan a modo de comentarios en el código que entrega el alumno. Se solicita que el alumno entregue adicionalmente la documentación generada en formato javadoc, junto con los ficheros de código fuente.