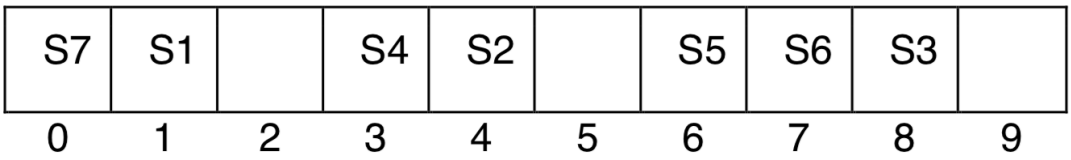
Ejercicio 1

1. Los métodos asociados en Java a la clase Iterator del API estándar de Java: Seleccione una:

   a. Solamente permiten al usuario recorrer los elementos en la estructura de datos en orden directo e inverso
   b. Permiten al usuario recorrer los elementos en la estructura de datos y opcionalmente realizar operaciones de borrado
   c. Solamente permiten al usuario recorrer los elementos en la estructura de datos en orden directo, inverso y opcionalmente realizar operaciones de borrado
   d. Solamente permiten al usuario recorrer los elementos en la estructura de datos

2. Sea la tabla hash de tamaño 10 que aparece en la figura y donde en cada posición solo hay espacio para insertar un único registro. Los registros con claves S1 a S7 ya han sido insertados en la tabla usando direccionamiento abierto con prueba lineal. ¿Cuál es el máximo número de accesos a la tabla necesario para buscar una clave que no está presente?



   a. 1
   b. 3
   c. 4
   d. 2

3. ¿Qué caso de uso se ajusta mejor al uso de una estructura de datos cola? Seleccione una:

   a. Cualquier aplicación que no precise respetar el orden de llegada de los elementos
   b. Implementar un buffer de teclado
   c. Evaluar si una expresión aritmética está escrita correctamente
   d. Comprobar la existencia de un valor en una colección de valores

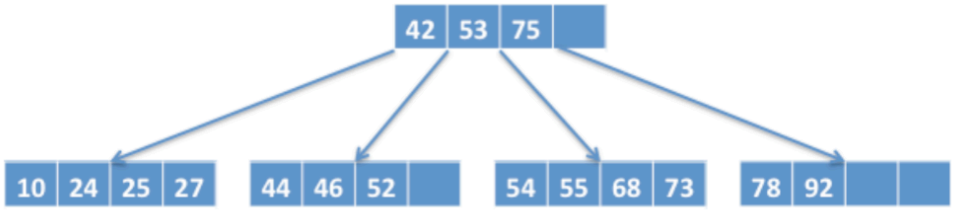
4. En un árbol binario siempre ocurre que: Seleccione una:

   a. Todo nodo tiene como máximo dos hijos.
   b. Todo nodo hijo izquierdo tiene una clave menor que su padre y todo hijo derecho tiene una clave mayor o igual que la de su padre.

c. En el camino desde el nodo raíz hasta cualquier nodo hoja, la clave de cada nodo es mayor o igual que la clave de su padre.
d. Todo nodo que no sea hoja tiene hijos cuyos valores de claves son menores o iguales que las de sus padres.

5.

b) Dado el árbol B de orden 5, se pide insertar secuencialmente las siguientes claves: 13, 50, 76, 62 y 77, dibujando el árbol B resultante tras insertar cada una de las claves.



Estructuras de Datos Avanzadas
Grado en Ingeniería Informática
Examen Ordinario del 15 de junio de 2017

A

Nombre y apellidos del alumno:_____

Firma de comprobación de asistencia al examen:

Normas generales:

- La duración del examen es de 2 horas (más 10 minutos para iniciar el entorno informático).
- El alumno debe ejecutar la aplicación **ExamWatcher** (disponible en la pestaña otros recursos del campus virtual) durante el examen. Tras descargarla y ejecutarla el alumno debe escribir su nombre en el campo correspondiente de la aplicación. Dicha aplicación monitoriza las acciones del usuario y por ello estará activa durante todo el examen. Cuando termine el examen, la aplicación tiene un botón para generar un fichero ZIP con el directorio en el que esté el código del alumno. El alumno debe subir dicho ZIP a la actividad examen práctico de Junio del campus virtual.
- Desde el campus virtual hay que descargar un archivo ZIP que contiene el **esqueleto** de los dos ejercicios del examen así como los test unitarios correspondientes. Se recomienda crear un proyecto de Netbeans vacío; copiar dentro de la carpeta del proyecto la carpeta src y la carpeta test proporcionadas; añadir las librerias JUnit y Hamcrest; y finalmente cerrar y volver a abrir Netbeans.

Ejercicio 1 [3.5 puntos]

Se desea desarrollar un sistema para la evaluación de expresiones aritméticas que se introduzcan en modo texto. Para tal fin, en el paquete `examenjunio2017a.Ejercicio1`, el proyecto que se proporciona al alumno contiene la clase `ArithmeticEvaluator` que el alumno debe completar.

La clase `ArithmeticEvaluator` permite crear objetos que evalúan expresiones aritméticas, en formato de cadena, devolviendo un resultado numérico. Solo se contempla el uso de los operadores +,- y / y la existencia de números naturales. Por ejemplo, si un objeto de la clase `ArithmeticEvaluator` recibe "1+12/2" devolvería 7. Obsérvese que la operación de producto tiene prioridad sobre la suma y la resta, y que entre sumas y restas tiene precedencia el operador más a la izquierda. Para construir la clase `ArithmeticEvaluator` es obligatorio utilizar un objeto de la clase `LinkedBinaryTree <String>`.

La clase se utiliza en dos fases: la fase de construcción del árbol y la fase de evaluación de árbol.

- Construcción del árbol: se coloca la cadena de la expresión en el nodo raíz, se busca el operador de menor prioridad más a la derecha, la subcadena anterior al operador se lleva al nodo hijo izquierdo, y la subcadena posterior al operador se lleva al hijo derecho. El proceso se repite con cada hijo hasta que no se encuentran operadores. Se recomienda usar los métodos `lastIndexOf` y `subString` de la clase `String`.
- Evaluación del árbol: mediante un recorrido recursivo que parte de la raíz se evalúan los dos hijos de un nodo y finalmente se evalúa el nodo padre que opera sobre los hijos. El proceso se repite recursivamente hasta obtener el resultado de evaluar el árbol completo.

En particular se pide:

a) [0.5 puntos] En la clase `ArithmeticEvaluator`, **definir la propiedad privada de tipo árbol** descrita e implementar el método `getTree` para que devuelva dicho árbol.

b) [1.5 puntos] **Implementar el método constructor** de la clase `ArithmeticEvaluator`. Dicho método constructor recibe una expresión aritmética y rellena el árbol.

c) [1.5 puntos] **Implementar el método `evaluate`** en la clase `ArithmeticEvaluator`. Dicho método recorre el árbol de manera recursiva y devuelve un valor numérico de tipo entero con el resultado de dicha evaluación.

NOTA: Se proporcionan test unitarios para comprobar el funcionamiento de cada parte. La clase `ArithmeticEvaluator` podrá contener en su interior cualquiera de las estructuras de datos de la clase `Colletion` o de las estudiadas durante el curso, valorándose especialmente lo adecuado de la elección que se realice. Se recuerda que, por lo general, es un mal diseño tener métodos que compartan mucho código o no introducir métodos privados auxiliares cuando sea necesario. Las soluciones recursivas no serán penalizadas.

Ejercicio 2 [3.5 puntos]

`SCHashDictionary` es una estructura de datos de tipo tabla hash que utiliza listas para resolver las colisiones. Cuando se inserta un elemento en un `SCHashDictionary` se utiliza una función de hash para localizar la posición del array en la que hay que insertar el elemento. Las colisiones se resuelven gracias a que en cada posición del array hay una lista.

El problema de `SCHashDictionary` es que en caso de colisiones la complejidad media de inserción y de búsqueda es W(n). Por ello, se desea sustituir dichas listas por otra estructura de datos que en caso de colisión tengan una complejidad media de inserción y de búsqueda de W(1).

En el paquete `examenjunio2017a.Ejercicio2` contiene la clase `SCW1HashMap`, que es una simplificación de la clase `SCHashDictionary` realizada en prácticas. Las simplificaciones consisten en que:

- Se pide un Map en vez de un Dictionary.
- El bucket (o array interno) tiene un tamaño fijo de 100 posiciones y no es necesario considerar el caso de que se llene, ni por tanto ningún tipo de rehash.
- No es iterable, ni contempla ningún tipo de iteradores.
- La función de hash consiste en obtener el hash de la clave (mediante el método `hashCode`) y hacer módulo 100.

Se pide:

a) [1.5 puntos] **Crear el constructor y las propiedades necesarias** en `SCW1HashMap` para que se permitan inserciones y borrados con una complejidad media de W(1) en caso de conflicto.

b) [1.5 puntos] **Crear los métodos put y get** de `SCW1HashMap` para que la estructura de datos funcione y pase todos los test correspondientes.

c) [0.5 puntos] **Crear el método remove** de `SCW1HashMap` para que la estructura de datos funcione y pase todos los test de la clase.

NOTA: Se proporcionan test unitarios para comprobar el funcionamiento de cada parte. Se valorará un buen diseño (de clases y métodos) para obtener la máxima calificación. Se recuerda que, por lo general, es un mal diseño tener métodos que compartan mucho código o no introducir métodos privados auxiliares cuando sea necesario. La clase `SCW1HashMap` podrá contener en su interior cualquiera de las estructuras de datos de la clase `Colletion` o de las estudiadas durante el curso, valorándose especialmente lo adecuado de la elección que se realice.