NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Segundo examen parcial

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 90 minutos
Puntuación máxima: 7 puntos
Fecha: 3 mayo 2018

Instrucciones para el examen:

- No se permite el uso de libros o apuntes, ni tener teléfonos móviles u otros dispositivos electrónicos encendidos. Incumplir cualquiera de estas normas puede ser motivo de expulsión inmediata del examen.
- Rellena tus datos personales antes de comenzar a realizar el examen.

Problema 1 (2,5 / 7 puntos)

Como parte de una aplicación de gestión de citas del Hospital Universitario Carlos III, necesitamos usar las siguientes estructuras de datos: una lista doblemente enlazada y una cola. Su labor es programar los métodos que se indican en los apartados sucesivos. Para ello **puede utilizar como referencia, el código adjunto** (donde puede suponer implementados todos los métodos excepto los requeridos en el enunciado). Este código incluye la clase `DLNode<E>`, que representa un nodo y la clase `DLinkedList<E>`, que representa la lista doblemente enlazada.

Apartado 1 (1,5 puntos)

Implemente el método `insert(E info, int position)`, que inserta un elemento con la información correspondiente (`info`) en la posición indicada y no devuelve nada. Además, el parámetro `position` podrá tomar valores positivos y negativos para especificar la posición donde insertarlo. Sólo deberás insertar un nuevo elemento si el valor de `position` está en el rango adecuado. Observa la figura 1 para comprender qué rango es y dónde debes añadir el nuevo nodo de acuerdo con el valor del parámetro `position` que recibe el método.

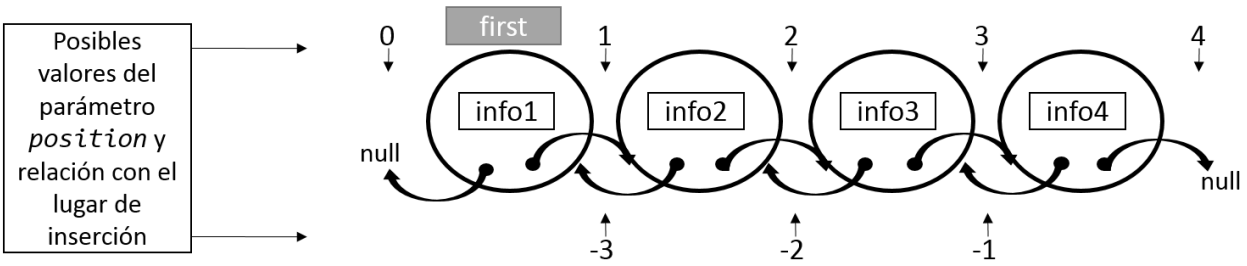


Figura 1: Ejemplo con posibles valores del parámetro *position*, para una lista de 4 elementos, y relación con el lugar de inserción deseado en la lista mediante el método *insert*.

NOTA: Al recibir un valor negativo de `position` (dentro del rango), puedes buscar su transformación en el valor positivo equivalente. Es decir, aquel valor positivo de `position` que lo insertaría en la misma posición.

Apartado 2 (1 punto)

Haciendo uso de la clase `DLinkedList<E>` y de los métodos que contiene (puede hacer uso del método del apartado anterior, aunque no lo haya completado), programe una estructura de datos que implemente únicamente el funcionamiento de una **cola**. Esta clase (`Queue<E>`) tendrá un único constructor que no recibe ningún parámetro y dos métodos: `enqueue(E info)` para insertar elementos y `dequeue()` para sacarlos. Sea minimalista y use el mínimo código.

```
public class DLNode<E> {

 private E info;
 private DLNode<E> next;
 private DLNode<E> prev;

 public DLNode (E info) {
      this.info = info;
      this.next = null;
      this.prev = null;
 }

 public DLNode() {this(null);}
 public E getInfo() {return this.info;}
 public void setInfo(E info) {this.info = info;}
 public DLNode<E> getNext() {return this.next;}
 public DLNode<E> getPrev() {return this.prev;}
 public void setNext(DLNode<E> n) {this.next = n;}
 public void setPrev(DLNode<E> p) {this.prev = p;}
}
```

```
public class DLinkedList<E> {

  private DLNode<E> first;

  public int size() {...}
  public E extractFirst(){...}

  public void insert(E info, int
position){
   // APARTADO 1.1:
  }

}
```

Problema 2 (3 / 7 puntos)

En los MOOCs (Massive Open Online Course), las interacciones sociales principalmente ocurren en los foros del curso. En este problema, queremos centrarnos en ver qué habilidades aparecen más en los mensajes de los foros de discusión, ya que la identificación correcta de las mismas puede ayudar a identificar problemas de los alumnos durante su aprendizaje. Las habilidades del curso pueden ser modeladas en un árbol, de modo que una habilidad pueda tener subhabilidades. Por ejemplo, "herencia" y "polimorfismo" pueden estar relacionadas con "encapsulación". Un ejemplo de árbol de habilidades es el siguiente:

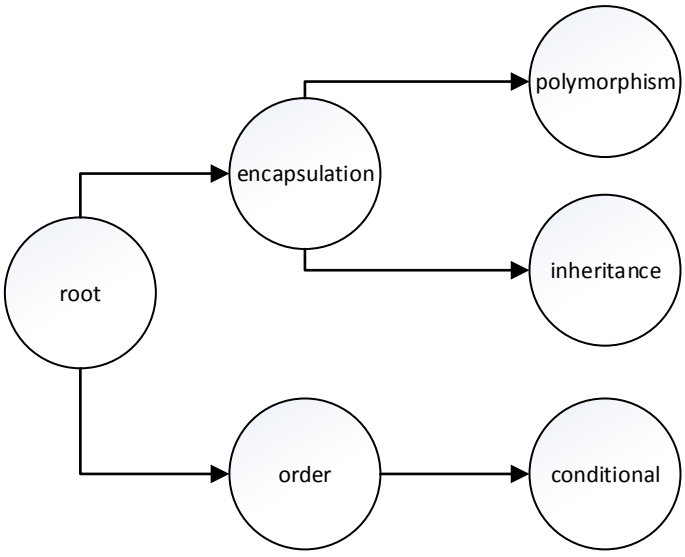
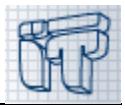


Figura 2: Ejemplo de árbol de habilidades de un curso

Para modelar estas habilidades, la clase `Skill` ha sido implementada. Esta clase tiene tres atributos: (1) `name`, (2) `mentions` y (4) `subskills`.

`name` indica el nombre de la habilidad (ej., polimorfismo), `mentions` indica el número de mensajes que contienen la habilidad (no necesitas preocuparte por la manera en la que se obtiene este valor) y `subskills`, que es un `ArrayList`, contiene las subhabilidades de una cierta habilidad. Además, hay una clase `SkillsTree` para modelar el árbol, que solo contiene la referencia a la raíz, que es un nodo por defecto que se utiliza para construir el árbol. En esta clase, puede suponer implementado el método `insert`, que inserta una habilidad al árbol a partir de otra habilidad dada, especificada por su nombre.



Para la realización del ejercicio, deberá hacer uso de estas clases, cuyo código aparece parcialmente al final del enunciado.

Apartado 1 (2 puntos)

Se quiere saber qué habilidades reciben más menciones en los mensajes y, para ello, las habilidades fundamentales son las hojas (ya que las de nivel superior agrupan varias habilidades de nivel inferior). Implemente el método `obtainLeafSkills()`, usando recursión, que devuelve un `ArrayList` con todas las habilidades hoja del árbol.

Apartado 2 (1 punto)

De cara a probar el método anterior, implementa un método `main` que cree un árbol como el de la figura, obtenga el `ArrayList` resultante de llamar al método del apartado 1, e imprima por pantalla el nombre de las habilidades. Para el número de menciones de cada habilidad, puedes considerar los valores de la siguiente tabla.

Habilidad	**Menciones**
Root	29
Encapsulation	14
Polymorphism	10
Inheritance	4
Order	5
Conditional	5

```
public class Skill {
 private String name;
 private int mentions;
 private ArrayList<Skill> subskills;

 public Skill(String name, int mentions){
    this.name = name;
    this.mentions = mentions;
    subskills = new ArrayList<Skill>();
 }

 public String getName() {return name;}
 public int getMentions() {return mentions;}
 public ArrayList<Skill> getSubskills() {
    return subskills;
 }
 public void addSubskill(Skill s){
    subskills.add(s);

 }
}
```

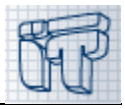
```
public class SkillsTree {
 private Skill root;

 public SkillsTree() {
     root = new Skill("root", 0);
 }

 public void insert(Skill s, String
parent)(){...}

 public ArrayList<Skills>
obtainLeafSkills(){
  // APARTADO 2.1
 }
 public   static   void   main(String
args[]){

  // APARTADO 2.2
 }

}
```

NOTA: La clase `ArrayList<E>` tiene los siguientes métodos, algunos de los cuales pueden serte de utilidad:

- `boolean add(E e)`
- `void add(int index, E element)`
- `E get(int index)`
- `boolean isEmpty()`
- `boolean remove(Object o)`
- `int size()`

Problema 3 (1,5 puntos)

En el contexto del problema anterior, una vez obtenido el `ArrayList` con los nodos hoja, se pide ordenar dichas habilidades de forma descendente en función del número de menciones. Para ello, se pide implementar el método `static void selectionSort(ArrayList<Skill> skills)`, que ordene el `ArrayList` de habilidades utilizando el algoritmo `SelectionSort`. Se debe usar la versión del algoritmo que busca la habilidad con más menciones entre las no ordenadas y la intercambia con la primera habilidad no ordenada.



SOLUCIONES DE REFERENCIA (Varias soluciones a cada uno de los problemas son posibles)

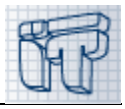
PROBLEMA 1

Apartado 1 (1,5 puntos)

```
// Método insert(E info, int position)
public void insert(E info, int position) {
	int listSize = size();
	if ((position <= listSize && position > -listSize) || (position == 0 &&
listSize == 0)) {
		DLNode<E> n = new DLNode<E>(info);

		if (position < 0) {
			position += listSize;
		}
	// En aux queremos tener el nodo anterior al que vayamos a insertar
	// Esto dará problemas si position es 0 con lo que distinguimos ese caso
		if (position == 0) {
			n.setNext(this.first);
			if (this.first != null) {
				this.first.setPrev(n);
			}
			this.first = n;
		} else {
			DLNode<E> aux = this.first;
			int counter = 0;
			while(counter < position-1) {
				counter++;
				aux = aux.getNext();
			}
			n.setNext(aux.getNext());
			n.setPrev(aux);
			aux.setNext(n);
			if (n.getNext() != null) {
				n.getNext().setPrev(n);
			}
		}
	}

}
```

Apartado 2 (1 punto)

```
public class Queue<E> {
	private DLinkedList<E> list;

	public Queue() {
		this.list = new DLinkedList<E>();
	}

	public void enqueue(E info) {
		list.insert(info, list.size());
	}

	public E dequeue() {
		return list.extractFirst();
	}
}
```



PROBLEMA 1

Apartado 1 (1,5 puntos)

- 0,25: Comprobación de que el parámetro *position* está dentro del rango correcto.
- 0,2: Manejo adecuado de valores negativos del parámetro *position*.
- 0,15: Creación del nuevo nodo.
- 0,3: Inserción del nodo para el caso en el que *position* es 0.
- 0,6: Inserción del nodo para el caso en el que *position* es distinto de 0.
    - 0,3: Recorrer la lista hasta llegar a la posición adecuada.
    - 0,3: Enlace del nuevo nodo en la posición adecuada.
- Los errores significativos están sujetos a sanciones adicionales

Apartado 2 (1 punto)

- 0,25: Atributo (lista enlazada doble).
- 0,25: Constructor sin parámetros que inicializa la lista.
- 0,25: Método enqueue.
- 0,25: Método dequeue.
- Los errores significativos están sujetos a sanciones adicionales

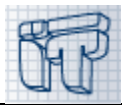
PROBLEMA 2

Apartado 1 (2 puntos)

```
public ArrayList<Skill> obtainLeafSkills(){
            return obtainLeafSkills(root);
}

public ArrayList<Skill> obtainLeafSkills(Skill root_skill){
            ArrayList<Skill> skills = new ArrayList<Skill>();
            if(root_skill.getSubskills().size() > 0){
                  for(int i=0; i<root_skill.getSubskills().size();i++){
                        ArrayList<Skill> aux =
obtainLeafSkills(root_skill.getSubskills().get(i));
                        for (int j = 0; j < aux.size();j++){
                              skills.add(aux.get(j));
                        }
                  }
            } else {
                  skills.add(root_skill);
            }
            return skills;
}
```

Apartado 2 (1 punto)

```
public static void main(String args[]){
            SkillsTree t = new SkillsTree();
            t.insert(new Skill("encapsulation",14), "root");
            t.insert(new Skill("polymorphism",10), "encapsulation");
            t.insert(new Skill("inheritance",4), "encapsulation");
            t.insert(new Skill("order",5), "root");
            t.insert(new Skill("conditional",5), "order");
            ArrayList<Skill> skills = t.obtainLeafSkills();
            for(int i=0;i<skills.size();i++){
                  System.out.println(skills.get(i).getName());
            }
}
```



PROBLEMA 3 (1,5 puntos)

```
public static void selectionSort(ArrayList<Skill> skills){
            for(int i = 0; i < skills.size(); i++){
                  int m = i;
                  for (int j = i; j < skills.size(); j++){
                        if (skills.get(j).getMentions() >
skills.get(m).getMentions()){
                              m = j;
                        }
                  }
                  swap(skills, i, m);
            }
}

public static void swap(ArrayList<Skill> skills, int i, int j){
      Skill aux = skills.get(i);
      skills.set(i, skills.get(j));
      skills.set(j, aux);
}
```

PROBLEMA 2

Apartado 1 (2 puntos)

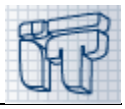
- 0,2: Llamada al método auxiliar
- 0,1: Declaración del método auxiliar
- 0,4: Caso base
  - 0,2: Identificación del caso base cuando la habilidad no tiene subhabilidades
  - 0,2: Añadir habilidad hoja al ArrayList
- 1,1: Caso recursivo
  - 0,2: For para recorrer las habilidades hijas
  - 0,5: Llamada recursiva al método
  - 0,4: Añadir al ArrayList las habilidades hijas. Si usa add() directamente en la llamada recursiva o introduce algún return en la llamada recursiva, entonces 0.
- 0,2: Declaración y devolución correcta del ArrayList con las habilidades hoja.
- Los errores significativos están sujetos a sanciones adicionales

Apartado 2 (1 punto)

- 0,15: Creación del objeto de tipo SkillsTree
- 0,25: Inserción de las habilidades al árbol
  - 0,05 cada inserción. Una inserción será incorrecta si, aunque la línea de código esté bien, no está en una posición correcta en el código (ej., se inserte un nodo hoja colgando de su padre si su padre no se ha creado todavía).
- 0,25: Llamada al método obtainLeafSkills() y almacenamiento del resultado en un ArrayList
- 0,35: Recorrer el ArrayList e imprimir los nombres de las habilidades
- Los errores significativos están sujetos a sanciones adicionales

PROBLEMA 3 (1,5 puntos)

- 0,9: Método selectionSort()
  - 0,2: Primer bucle correcto
  - 0,3: Segundo bucle correcto
  - 0,2: Bloque if correcto. Si el orden es ascendente en lugar de descendente, máximo 0,15.
  - 0,2: Llamada correcta y en su sitio del swap
- 0,6: Método swap()



  - 0,2: Almacenar la habilidad i en una variable auxiliar
  - 0,2: Asignación de la habilidad i
  - 0,2: Asignación de la habilidad j
- Los errores significativos están sujetos a sanciones adicionales