Computer Science Degree

Data Structures & Algorithms, Group 89M, 2014/2015

12th March 2015

Name and Surname:

…...

PROBLEM 1 (1 point) – Object Oriented Programming

A company is interested in developing an online store to sell different types of contents. All products have the following properties:

- Name or title of the content.
- Supplier. The company producing the content.
- Age rate. Minimum age to be able to consume this content
- Price. The amount to be paid by users wanting the content.

There can be different types of contents: e-books, music, films and applications. Only registered users can get access to the online store. Each user has the following features:

- Email
- Name
- Surname
- Identification Number (DNI in Spain)
- Collection of titles of contents acquired by this user

Provide the needed data structures to represent these elements. More specifically:

1. Provide all needed classes and constructors for them
2. Define and implement a method 'acquire' to register a new acquisition by a given user. The method takes a content as input and updates the set of contents acquired by a given user.
3. Include a structure containing all users registered at the online store
4. Define a method to search for the surnames of users that have consumed a given content. The method takes as input the title of the content whose consumers must be obtained.

The proposed solution must be designed under the Object Oriented Paradigm principles, that is,

pursuing robust, reusable and adaptable software.

Notes:

- If needed, you can use any of the ADTs studied. Their code must not be provided.
- It is not needed to implement getters and setters.
- A main method or Test class is not needed.

Solution:

```
        public class Content {

        public enum ContentType { APP, EBOOK, FILM, MUSIC};

        String title; //Name or title
        String supplier; //Supplier
        int    age_rate;    // Minimum age to be able to consume this content
        double price;       //Price
        ContentType type;   //The type of the content

        public Content(){
                title = "";
                supplier = "";
                age_rate=-1;
                price=-1;
                type = null;

        }

        public Content(String title, String supplier, int age, double price, ContentType
                                                                                   type){
                this.title = title;
                this.supplier = supplier;
                this.age_rate=age;
                this.price=price;
                this.type = null;

        }

}


public class User {

        String email;//     email
        String name; //      Name
        String surname;     //      Surname
        String identification_number; // NIF in Spain
        DList<String> collection; //Collection of titles of contents acquired by this user

        public User(String email, String name, String surname,String id){
                this.email = email;
                this.name = name;
                this.surname = surname;
                this.identification_number = id;

                collection = new DList<String>();
        }
```

```
	public void acquire(Content content){
		if(content != null){
			collection.addFirst(content.title);
		}
	}

}


public class SetOfUsers {

	DList<User> list_of_users;

	public SetOfUsers(){
		list_of_users = new DList<User>();
	}

	public DList<String> searchBuyers(String title){

		DList<String> result = new DList<String>();

		for(int i=0; i < list_of_users.getSize(); i++){
			User user = list_of_users.getAt(i);
			DList<String> user_contents = user.collection;
			if(user_contents != null){
				for(int j=0; j < user_contents.getSize(); j++){
					String current_title = user_contents.getAt(j);
					if(current_title.equals(title))
						result.addFirst(user.surname);
				}
			}
		}
		return result;
	}
}
```

PROBLEMA 2 (1 point)- Implementing a double ended queue.

Given the following classes:

```
public class DNode<E> {

    DNode<E> previousNode = null;
    DNode<E> nextNode = null;

    E elem;

    public DNode<E> getPreviousNode() {
       return previousNode;
    }

    public DNode<E> getNextNode() {
       return nextNode;
    }

    public E getElement() {
       return elem;
    }

    public void setElement(E e) {
       this.elem = e;
    }

    public DNode(E elem) {
       this.elem = elem;
    }

}


public class DQueue<E> implements IDQueue<E> {
        DNode<E> header;
        DNode<E> tailer;

        public DQueue() {
                header = new DNode<E>(null);
                tailer = new DNode<E>(null);
                header.nextNode = tailer;
                tailer.previousNode = header;
        }

        public boolean isEmpty() {
                return (header.nextNode == tailer);
        }

}
```

Implement the following methods in the DQueue class:

1. size. Returns the number of elements in the queue
2. addFirst. Adds an element to the beginning of the queue
3. getLast. Returns the element in the last position of the queue
4. removeLast. Removes the element in the last position of the queue

Solution:

```java
public int size() {
	int size = 0;
	for (DNode<E> nodeIt = header.nextNode; nodeIt != tailer; nodeIt =
								nodeIt.nextNode) {
		++size;
	}
	return size;
}


public void addFirst(E elem) {
	DNode<E> newNode = new DNode<E>(elem);
	newNode.nextNode = header.nextNode;
	newNode.previousNode = header;
	header.nextNode.previousNode = newNode;
	header.nextNode = newNode;
}


public E getLast() {
	if (isEmpty()) {
		System.out.println("DQueue: Queue is empty");
		return null;
	}
	return tailer.previousNode.getElement();
}

public E removeLast() {
	if (isEmpty()) {
		System.out.println("DQueue: Queue is empty");
		return null;
	}
	E e=getLast();
	tailer.previousNode = tailer.previousNode.previousNode;
	tailer.previousNode.nextNode = tailer;
	return e;
}
```