Examen POO Session 1 - Corrigé

QUESTION 1. (8,75 pts)

```
public class ListeTableau {
     protected static final int MAX_TAILLE = 100;
     // protected int maxtab; // Taille maximale
     protected int taille; // nombre d'élements du tableau
     protected Object[] tableau;

     // constructeur
(0,5 pt) public ListeTableau() {
          tableau = new Object[MAX_TAILLE];
          this.taille = 0;
     }
(0,25 pt) public int getTaille() {
          return taille;
     }
(0,25 pt) public boolean listeVide() {
          return taille == 0;
     }
(0,25 pt) public boolean listePleine() {
          return taille == MAX_TAILLE;
     }
(0,5 pt) public Object extremiteGauche() {
          if (!listeVide())
               return tableau[0];
          return null;
     }
(0,5 pt) public Object extremiteDroite() {
          if (!listeVide())
               return tableau[taille - 1];
          return null;
     }
(01 pt) public void ajouterFin(Object element) { // ajout ‡ la fin
          if (!listePleine()) {
               tableau[taille] = element;
               taille++;
          }
     }
(01 pt) public void supprimFin() { // supprimer le dernier element
          if (!listeVide())
               taille--;
     }

(02 pt) public void ajouterDebut(Object element) { // ajouter au debut
          if (!listePleine()) {// decalage ‡ droite
```

```
                for (int i = taille - 1; i >= 0; i--)
                    tableau[i + 1] = tableau[i];
                tableau[0] = element;
                taille++;
            }
        }
```

(02 pt)
```
        public void supprimDebut() { // supprimer le premier element
            if (!listeVide()) {
                // decalage ‡ gauche
                for (int i = 0; i < taille - 1; i++)
                    tableau[i] = tableau[i + 1];
                taille--;
            }
        }
```

(0,5 pt)
```
        public void afficheTableau() {
            System.out.print("[");
            for (int i = 0; i < taille - 1; i++)
                System.out.print(tableau[i] + ", ");
            System.out.println(tableau[taille - 1] + "]");

        }
    }// fin classe ListeTableau
```

QUESTION 2. (10,5 pts)

Classes d'exception :

(01 pt)
```
    public class DequePleineException extends Exception {
        public DequePleineException(String message){
            super(message);
        }
    }
```

(01 pt)
```
    public class DequeVideException extends Exception {
    public DequeVideException(String message){
        super(message);
    }
    }
```

Classes DequeTableau:

(0,5 pt)
```
    public class DequeTableau extends ListeTableau implements Deque{
```

(0,5 pt)
```
        public DequeTableau(){
            super();
        }
```

(02 pt)
```
public void dedequer(int sens) throws DequeVideException{
        if (estVide()) throw new DequeVideException("La Deque est
vide");
        else if (sens==gauche) supprimDebut();
            else supprimFin();
    }
```

(02 pt)
```
public void endequer(Object o,int sens) throws
DequePleineException{
        if (estPleine()) throw new DequePleineException("La deque est
pleine");
        else if (sens==gauche) ajouterDebut(o);
            else ajouterFin(o);
    }
```

(02 pt)
```
public Object extremite(int sens) throws DequeVideException{
        if (estVide()) throw new DequeVideException("La deque est
vide");
        else if (sens==gauche) return extremiteGauche();
            else return extremiteDroite();

    }
```

(0,5 pt)
```
public boolean estVide() {
        return listeVide();
    }
```

(0,5 pt)
```
public boolean estPleine(){
        return  listePleine();
    }
```
(0,5 pt)
```
public void afficheDeque() throws DequeVideException {
    if (estVide()) throw new DequeVideException("La deque est vide");
        super.afficheTableau();
    }
}
```

QUESTION 3. (3 pts)

```
public class Test {
    public static void main(String[] args) {
        int n = args.length;
        Deque d=new DequeTableau();
        int sens=1;
        try {
        for(int i=0;i<n;i++){
            int x = Integer.parseInt(args[i]);
                d.endequer(x, sens);
                sens*=-1;
        }
```

```
            d.afficheDeque();

            sens=-1;
            for(int i=0;i<n+2;i++)
                        d.dedequer(sens);sens*=-1;
                  d.afficheDeque();
            }
            catch (DequePleineException e) {
                  System.out.println(e.getMessage());
            }
            catch (DequeVideException e) {
                  System.out.println(e);
            }
      }
```

QUESTION 4. (4 pts)

```
import java.util.Vector;
public class ListeVecteur {
      protected static final int MAX_TAILLE = 100;
      protected Vector tableau;
// constructeur
      public ListeVecteur() {
            tableau = new Vector();
      }
      public int getTaille() {
            return tableau.size();
      }
      public boolean listeVide() {
            return tableau.size() == 0; // return tableau.isEmpty()
      }
      public boolean listePleine() {
            return tableau.size() == MAX_TAILLE;
      }
      public Object extremiteGauche() {
            if (!listeVide())
                  return tableau.get(0);
            return null;
      }
      public Object extremiteDroite() {
            if (!listeVide())
                  return tableau.get(tableau.size() - 1);
            return null;
      }
      public void ajouterFin(Object element) { // ajout ‡ la fin
            if (!listePleine()) {
                  tableau.add(element);
            }
      }
```

```
public void supprimFin() { // supprimer le dernier element
          if (!listeVide())
               tableau.remove(tableau.size()-1);
     }
     public void ajouterDebut(Object element) {
          if (!listePleine()) {
               tableau.add(0,element);
          }
     }

     public void supprimDebut() {
          if (!listeVide()) {
               tableau.remove(0);
          }
     }

     public void afficheTableau() {
          System.out.print("[");
          for (int i = 0; i < tableau.size() ; i++)
               System.out.print(tableau.elementAt(i) + ", ");
          System.out.println("]");
     }
}
```

En donnant cette solution, aucune modification ne sera apportée à l'interface Deque et à la classe DequeTableau qui l'implémente.

Une autre solution serait de supprimer *MAX_TAILLE* et ainsi la notion de Deque pleine n'aura plus lieu d'être. Ceci conduira à supprimer la méthode estPleine et tous les tests de deque pleine ainsi que l'exception DequePleineException.

Le barème est sur 26,25 points.
Le total est ramené à 20 en multipliant la note par 0,76 (20/26,25)