

## Algorithmique avancée

### TD n° 2

#### 1. Copie avec purge d'une liste :

On utilise des listes définies à la page 7 du cours numéro 1.

a) Ecrire un algorithme **copierPurger**(Liste @listeSource, Liste @listeDestination) qui copie les éléments de la listeSource *sans répétition* dans la listeDestination (initialement vide). La listeSource n'est pas modifiée par l'algorithme (sauf éventuellement son curseur).

b) Traduire cet algorithme en programmation objet.

#### 2. Listes doublement chaînées :

a) Dessiner une liste doublement chaînée de trois éléments.

b) Donner les algorithmes des opérations **insérer**(CelluleDC @nouvelleCel, CelluleDC @repereCel) et **supprimer**(CelluleDC @repereCel) dans une liste doublement chaînée.

Le lieu d'insertion ou de suppression est après la cellule repère (comme dans le cas de listes simplement chaînées).

#### 3. Pile et File :

On considère les interfaces suivantes :

**public interface Pile**

```
{ public void empiler(Object item);
  public Object depiler() throws Exception;
  public boolean estVide();
}
```

**public interface File**

```
{public void stocker(Object item); // en queue
 public Object prelever() throws Exception; // en tête
 public boolean estVide();
}
```

On fournit la classe **cellule** :

```
public class Cellule
  {private Object elem;
  private Cellule suivant; // remarquer la structure récursive
  public Cellule () { this(null);}
  public Cellule ( Object o )
    { elem = o;
      this.suivant = null;
    }
  public Object getElem() { return(elem); }
  public Cellule getSuivant() { return(suivant); }
  public void setElem(Object o) {elem = o;}
  public void setSuivant(Cellule s) { suivant = s;}
  public String toString() { return (elem.toString()); }
} // end class Cellule
```

Rédigez les classes **PileCellule** et **FileCellule** implémentant les interfaces **Pile** et **File** en utilisant des objets de type **Cellule** en tant que conteneurs pour les éléments stockés.