```java
package util;

import java.util.EmptyStackException;

/**
 * Represente une pile geree manuellement.
 * <p>
 *
 * @author Sacha Bron
 * @author Valentin Minder
 * @date 13.11.2014
 */
public class Pile {

    /**
     * Reference vers le sommet de la pile.
     * <p>
     * Valeur par defaut: null (si la pile est vide).
     */
    private ObjectContainer head = null;
    /**
     * Taille courante de la pile.
     * <p>
     * Valeur par defaut: 0 (la pile est vide).
     */
    private int size = 0;

    /**
     * Ajoute un element au sommet de la pile.
     *
     * @param data
     *             l'element a ajouter.
     * @return l'element ajouté au sommet.
     */
    public Object empile(Object data) {
        // la nouvelle tete est le container de data donne avec reference sur
        // l'ancienne tete
        head = new ObjectContainer(data, head);
        size++;
        return data;
    }

    /**
```

```
45      * Retourne et supprimer l'element au sommet de la pile.
46      *
47      * @return le premier element de la pile.
48      * @throw {@link EmptyStackException} si la pile est deja
  vide.
49      */
50     public Object depile() {
51         ObjectContainer pop = head;
52         Object dataHead = null;
53         if (pop != null) {
54             dataHead = pop.getData();
55             head = head.getPred();
56             size--;
57         } else {
58             throw new EmptyStackException();
59         }
60         return dataHead;
61     }
62 
63     /**
64      * Retourne une reprentation sous forme de String de l'etat de
  la pile.
65      * <p>
66      * Ex: "[ &lt;Obj1&gt; &lt;Obj2&gt; ]" ; "[ ]"
67      *
68      * @return string representant le contenu de la pile.
69      */
70     @Override
71     public String toString() {
72         Object[] array = toArray();
73         String s = "[ ";
74         for (int i = 0; i < array.length; i++) {
75             s += "<" + array[i] + "> ";
76         }
77         s += "]";
78         return s;
79     }
80 
81     /**
82      * Retourne une reprentation sous forme de tableau de l'etat
  de la pile.
83      * <p>
84      * Le somment de la pile est en position 0 de la pile. La
  taille du tableau
```

```
85      * correspond à la taille de la pile (y compris si la pile est
  vide).
86      *
87      * @return un tableau representant la pile.
88      */
89     public Object[] toArray() {
90         Object[] arrayRepresentation = new Object[size];
91         ObjectContainer current = head;
92         int indexArray = 0;
93         while (current != null) {
94             arrayRepresentation[indexArray] = current.getData();
95             current = current.getPred();
96             indexArray++;
97         }
98         return arrayRepresentation;
99     }
100
101    /**
102     * Retourne un @link {@link Iterateur} sur objet sur la pile.
103     *
104     * @return un iterateur sur la pile.
105     */
106    public Iterateur iterateur() {
107        Iterateur iter = new Iterateur() {
108
109            /**
110             * Reference vers l'objet en cours de manipulation
  dans cet
111             * iterateur.
112             */
113            private ObjectContainer nextContainer = head;
114
115            public Object suivant() {
116                if (nextContainer == null) {
117                    throw new NoSuchElementException();
118                } else {
119                    Object nextData = nextContainer.getData();
120                    nextContainer = nextContainer.getPred();
121                    return nextData;
122                }
123            }
124
125            public boolean possedeSuivant() {
126                return nextContainer != null;
```

```
127             }
128         };
129         return iter;
130     }
131 
132     /**
133      * Retourne la taille de la pile.
134      *
135      * @return taille de la pile.
136      */
137     public int getSize() {
138         return size;
139     }
140 
141     /**
142      * Represente un conteneur compose d'une donnee (data) et
    d'une reference
143      * vers son predecesseur.
144      *
145      * @author Sacha Bron
146      * @author Valentin Minder
147      * @date 13.11.2014
148      */
149     private class ObjectContainer {
150 
151         /**
152          * Donnee contenue par le conteneur.
153          */
154         private Object data;
155 
156         /**
157          * Reference vers le conteneur predecesseur.
158          */
159         private ObjectContainer pred;
160 
161         /**
162          * Constructeur d'un conteneur
163          *
164          * @param data
165          *             la donnee a contenir
166          * @param pred
167          *             la reference vers le predecesseur
168          */
169         public ObjectContainer(Object data, ObjectContainer pred)
```

```
    {
170             this.data = data;
171             this.pred = pred;
172         }
173 
174         /**
175          * Retourne la donnee representee par le conteneur.
176          *
177          * @return la donne
178          */
179         public Object getData() {
180             return data;
181         }
182 
183         /**
184          * Retourne la reference vers le conteneur predecesseur.
185          *
186          * @return le predecesseur
187          */
188         public ObjectContainer getPred() {
189             return pred;
190         }
191 
192         /**
193          * Retourne la reprensation sous forme de String de ce
    conteneur.
194          * <p>
195          * Il s'agit de l'appel toString() sur les donnees
    contenues.
196          */
197         @Override
198         public String toString() {
199             return data.toString();
200         }
201     }
202 }
203 
```