

Templates.java

```
2  * Licensed to the Apache Software Foundation (ASF) under one or more
17 package com.java;
18
19 import static org.testng.Assert.*;
20
21 import static java.lang.System.*;
22
23 import java.io.BufferedReader;
24 import java.io.FileReader;
25 import java.io.IOException;
26 import java.util.ArrayList;
27 import java.util.Arrays;
28 import java.util.Collection;
29 import java.util.Iterator;
30 import java.util.List;
31
32 import org.testng.annotations.Test;
33
34 /**
35  * <b>Templates</b> are a structured description of <b>coding patterns</b>.
36  *
37  * @author Bert Lee
38  * @version 2013-6-2
39  */
40 public class Templates {
41
42     /**
43      * <b>arrayadd</b> Template.
44      *
45      * <pre>
46      * 参考{@link Arrays#copyOf(Object[], int)}实现.
47      * </pre>
48      *
49      * @param a
50      * @param e
51      */
52     public void arrayadd(Object[] a, Object e) {
53         Object[] result = new Object[a.length + 1];
54         // array copy - faster
55         System.arraycopy(a, 0, result, 0, a.length);
56         result[a.length] = e;
57     }
58
59     private String name;
60
61     /**
62      * Generate <b>hashCode()</b> and <b>equals(Object)</b>.
63      *
64      * <pre>
65      * 参考{@link Arrays#hashCode(Object[])}实现.
66      * </pre>
67      */
68     @Override
69     public int hashCode() {
70         final int prime = 31;
71         int result = 1;
72         result = prime * result + ((name == null) ? 0 : name.hashCode());
73         return result;
74     }
75
76     /**
```

Templates.java

```

77     * <b>equals</b>, <b>instanceof</b>, <b>cast</b> Template.
78     *
79     * <pre>
80     * 参考《Effective Java》实现, p-36
81     * </pre>
82     */
83     @Override
84     public boolean equals(Object obj) {
85         // 性能优化: 比较操作有可能很昂贵
86         if (this == obj)
87             return true;
88
89         if (! (obj instanceof Templates))
90             return false;
91
92         // cast
93         Templates other = (Templates) obj;
94         return name == null ? other.name == null : name.equals(other.name);
95     }
96
97     /*
98     * Generate
99     */
100 // @Override
101 // public boolean equals(Object obj) {
102 //     // 性能优化: 比较操作有可能很昂贵
103 //     if (this == obj)
104 //         return true;
105 //
106 //     if (obj == null)
107 //         return false;
108 //     if (getClass() != obj.getClass())
109 //         return false;
110 //
111 //     Templates other = (Templates) obj;
112 //     return name == null ? other.name == null : name.equals(other.name);
113 // }
114
115 /**
116  * <b>try</b> or <b>catch</b> Template.
117  *
118  * @param s
119  * @param defaultValue
120  * @return
121  */
122 public static int parseInt(String s, int defaultValue) {
123     try {
124         int i = Integer.parseInt(s, 10);
125         return i;
126     } catch (NumberFormatException nfe) {
127         return defaultValue;
128     }
129 }
130
131 private String email;
132 private String phone;
133
134 /**
135  * <b>toString</b> Template.
136  */
137 @Override

```

Templates.java

```

138 public String toString() {
139     StringBuilder sb = new StringBuilder();
140     sb.append("Templates [")
141         .append("name=").append(name)
142         .append(", email=").append(email)
143         .append(", phone=").append(phone)
144         .append(']');
145     return sb.toString();
146 }
147
148 /*
149  * Generate
150  */
151 // @Override
152 // public String toString() {
153 //     StringBuilder sb = new StringBuilder();
154 //     sb.append("Templates [")
155 //         .append("name=").append(name)
156 //         .append(", email=").append(email)
157 //         .append(", phone=").append(phone)
158 //         .append(']');
159 //     return sb.toString();
160 // }
161
162 /**
163  * <b>foreach</b> Template.
164  *
165  * @param a
166  * @return
167  */
168 public static int sum(int[] a) {
169     int sum = 0;
170     for (int i : a) {
171         sum += i;
172     }
173     return sum;
174 }
175
176 /**
177  * <b>static_method</b>, <b>while</b>, <b>finally</b>, <b>if</b> Template.
178  *
179  * @param fileName
180  * @return
181  * @throws IOException
182  */
183 public static String bufferedRead(String fileName) throws IOException {
184     BufferedReader in = null;
185
186     try {
187         in = new BufferedReader(new FileReader(fileName));
188
189         StringBuilder content = new StringBuilder(1024); // 2KB
190         String l = null;
191         while ((l = in.readLine()) != null) {
192             content.append(l);
193         }
194         return content.toString();
195     } finally {
196         if (in != null) {
197             in.close();
198         }
199     }
200 }

```

Templates.java

```

199     }
200 }
201
202 /**
203  * <b>main</b>, <b>sysout</b> Template.
204  *
205  * @param args
206  */
207 public static void main(String[] args) {
208     out.println("Hello world!");
209 }
210
211 /**
212  * <b>method</b>, <b>new</b>, <b>list</b>, <b>for-iterate</b> Template.
213  */
214 public void loop() {
215     List<String> list = new ArrayList<String>();
216     list.add("1");
217     list.add("2");
218     list.add("3");
219
220     for (Iterator<String> it = list.iterator(); it.hasNext();) {
221         String s = it.next();
222         if ("2".equals(s)) {
223             it.remove(); // 只有需要删除操作时，才会使用迭代器
224         }
225     }
226 }
227
228 /**
229  * <b>static_final</b> Template.
230  */
231 public static final long INVALID_ID = -1;
232
233 /**
234  * <b>switch</b> Template.
235  *
236  * @param s
237  */
238 public void select(Selector s) {
239     switch (s) { // Ctrl + 1
240         case BLUE:
241             // do what
242             break;
243         case GREEN:
244             //
245             break;
246         case RED:
247             //
248             break;
249         default:
250             //
251             break;
252     }
253 }
254 static enum Selector {
255     RED("red"),
256     GREEN("green"),
257     BLUE("blue"),
258     ;
259 }

```

Templates.java

```

260     @Override
261     public String toString() {
262         return value;
263     }
264
265     private String value;
266
267     Selector(String value) {
268         this.value = value;
269     }
270 }
271
272 /**
273  * <b>test</b>, <b>array</b>, <b>for</b> Template.
274  */
275 @Test
276 public void unitTest() {
277     List<Integer> expected = new ArrayList<Integer>(3);
278     for (int i = 0; i < expected.size(); i++) {
279         expected.add(Integer.valueOf(i));
280     }
281
282     Integer[] actual = toArray(expected);
283
284     assertEquals(actual.length, expected.size());
285     // for
286     for (int i = 0; i < actual.length; i++) {
287         assertEquals(actual[i], expected.get(i));
288     }
289 }
290
291 /**
292  * <b>toArray</b> Template.
293  *
294  * @param c
295  * @return
296  */
297 public static Integer[] toArray(Collection<Integer> c) {
298     return c.toArray(new Integer[c.size()]);
299 }
300 }
301

```