**Generics Demo**

In code written before Java 1.5, all items put into a Collection were reduced to type Object. When you accessed the item, you had to remember to cast the Object explicitly. There was always the danger that you would forget to cast, or to put in an object of the wrong type which later would throw an exception during the cast. For example,

List myList = new ArrayList(); //old style, non-generic

myList.add("TJHSST");

myList.add(1.0);

String str0 = (String)myList.get(0); //cast

String str1 = (String)myList.get(1); //throws ClassCastException

If you remember, our ListNode object stored Objects, which meant you had to cast when you accessed the items. In other words, ListNode is not generic. Neither is TreeNode.

Since Java now includes *generic types*, we can write a generic resource class, using <E>, and supply the specific data type, <String>, <Widget>, or <Whatever>, in the driver during run-time.

The compiler performs the necessary type checking on the generic type and guarantees that a ClassCastException for the specific type, whatever it is, is never thrown at runtime. The generic code is thus type-safe.

1 //name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 2 import java.util.\*;  
 3   
 4 public class Generics\_Demo  
 5 {  
 6 public static void main(String[] args)  
 7 {  
 8 /\*\*\*\*\*\*\*\*\*\*\*\*\* non-generic data structures \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 9   
10 List myList = new ArrayList(); //old style, non-generic  
11 myList.add("this is a string"); //put a String in the arrayList  
12 // String str = myList.get(0); //error: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
13   
14 //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//Fix it.  
15   
16 //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//Fix it by casting.  
17   
18 //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//another way to cast.  
19   
20 //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//Stack Overflow's way to change obj to a String  
21   
22 //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//every Object has a toString()  
23 //System.out.println( str );  
24   
25 List theList = new ArrayList(); //old style, non-generic  
26 theList.add(3); //put an int in the arrayList   
27 //int x = theList.get(0); //error: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
28   
29 //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//Fix it.  
30   
31 //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//Fix it by casting.  
32   
33 //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ //Stack Overflow's way to change obj to an Integer  
34   
35 //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//every Object has a toString()  
36   
37 // int square = x \* x; // it behaves like an Integer  
38 // System.out.println( square );   
40   
41 /\*\*\*\*\*\*\*\*\*\*\*\*\* generic data structures \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/   
42   
43 List<String> stringList = new ArrayList<>(); // ArrayList<E>  
44 stringList.add("this is a string");  
45 String str2 = stringList.get(0); //it "remembers" the data is a String  
46 String str3 = str2.substring(1,2); //all String methods are available without casting  
47 System.out.println( str3 );  
48   
49 LinkedList<Integer> ints = new LinkedList<>(); // LinkedList<E>  
50 ints.add(3);  
51 Integer y = ints.getFirst(); //it "remembers" the data is an Integer  
52 Integer square = y \* y; //no need to cast  
53 System.out.println( square );  
54   
55 /\*\*\*\*\*\*\*\*\*\*\*\*\* ListNode \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/   
56   
57 ListNode<Integer> s = new ListNode<>(4, null); //uses the generic ListNode<E>, see below  
58 ListNode<Integer> t = new ListNode<>(5, s);  
59 Integer num = t.getNext().getValue(); //what type does getNext() return?  
60 Integer sq = num \* num;   
61 System.out.println( sq );  
62 }  
63   
65 static class ListNode<E> //write the generic ListNode<E> class  
66 {  
67 /\* two private fields \*/  
68   
69   
70 /\* one two-arg constructor \*/  
71   
72   
73   
74   
75   
76 /\* 2 accessor methods \*/   
77   
78   
79   
80   
81   
82   
83   
84   
85 /\* 2 modifier methods \*/   
86   
87   
88   
89   
90   
91   
92   
93   
94 }  
95 }  
96