

A Java generic set interface and class

Assignment 3 for CSC 243, Spring, 2010, Dr. Dale E. Parson

<http://faculty.kutztown.edu/parson/spring2010/CSC243Spring2010.html>

Assignment 3 is due 11:59 PM on Tuesday April 6.

Use **gmake** **turnitin** from your **games2010rev3a** directory to turn it in.

Make sure to **gmake javadoc** to test your javadoc requirements after getting your code working.

Perform the following steps to copy and inspect my initial code handout.

```
cp ~parson/JavaLang/games2010rev3a_assign3.zip ~/JavaLang
cd ~/JavaLang
/bin/unzip games2010rev3a_assign3.zip
cd ./games2010rev3a
gmake clean test
```

These tests will fail because you need to complete an interface definition and then write a class that implements that interface. There are three coding parts to this assignment, all involving files in subdirectory **games2010rev3a/scrabble** (package `games2010rev3a.scrabble`). The first two are need before you can get past compilation errors, and the third part is required testing that you must add. Please re-read the following requirements before turning this in.

Once you have compilation working, **gmake** **testboard** will run the tests related to your code changes. Running that without “clean” saves some time, once you are down to editing a file at a time. Make sure to run all tests (**gmake** **clean test**) before running **gmake** **turnitin**.

PART I: `games2010rev3a/scrabble/MySetInterface.java`

```
1 // STUDENT 1. Add the following items to this interface definition.
2 // 1a. Add a “package statement” for games2010rev3a.scrabble
3 // 1b. Add Javadoc comments above the declaration of MySetInterface
4 // describing it as a set of objects of generic type E.
5 // 1c. Add Javadoc comments above the declaration of each method
6 // describing its operation in a summary line, followed by
7 // a @param tag for each parameters, a @return tag for the
8 // return value, and an @exception tag for add’s exception.
9 // Replace ALL of my “//” comments with your Javadoc.
10
11 public interface MySetInterface<E> {
12
13 // Add e to the set if it is not already in the set.
14 // Return true if e was NOT in the set (add succeeds), false if it was.
15 // In either case e will be in the set UNLESS e is the null value,
16 // in which case this functions must throw java.lang.NullPointerException
17 // with a message.
18 boolean add(E e) throws NullPointerException ;
19
```

```
20 // Return true of the set contains Object o, else false.
21 // Note that if o is null or a type incompatible with E,
22 // then contains returns false.
23 boolean contains(Object o);
24
25 // Return a COPY of the set as an array of elements.
26 // If the set is empty then the return array has length 0.
27 // The caller is able to modify the returned array without
28 // changing the contents of the set that returns it.
29 E[] toArray();
30
31 }
```

PART II: `games2010rev3a/scrabble/MySetClass.java`

```
1 // STUDENT 2 define class MySetClass<E> that implements interface
2 // MySetInterface<E> in the same package as MySetInterface<E>.
3
4 // Define private fields for an array of type E to hold the elements,
5 // and an integer count field that states how many elements are actually
6 // being used in the array.
7
8 // Also define a symbolic constant integer field called GROWSIZE
9 // that is initialized to 1024.
10
11 // Define a 0-parameter constructor that allocates GROWSIZE elements for the
12 // array and initializes count to 0. LOOK AT
13 // /export/home/faculty/parson/JavaLang/mapgame/Mymap.java (we have
14 // gone over listings in class) to see how to allocate an array
15 // of generic typed elements using Java type casting. The compiler will
16 // give a warning for this type cast, as we have gone over in class.
17
18 // Define an add method as specified in MySetInterface<E> that
19 // checks for the exception condition and conditionally throws an exception.
20 // If it does not throw an exception, add must search all elements
21 // in the array using the .equals() method to compare parameter e to
22 // each array element. As soon as it finds a match, return false.
23 // If it does not find a match, then add e to the array.
24 // If you need to grow the array because it is not big enough to hold
25 // another element, allocate an array grown by GROWSIZE elements and
26 // copy the former array into it.
27 // LOOK AT
28 // /export/home/faculty/parson/JavaLang/sequence_aclass/sequence_arrayimpl.java
29 // for an example of growing an array and copying elements.
30 //
31 // Define a contains method as specified in MySetInterface<E> that
```

```

32 // returns true if Object o is in the set, else returning false.
33
34 // Define a toArray method as specified in MySetInterface<E> that
35 // returns a copy of the set's array of length count.
36
37 // Write implementation Javadoc comments for the class and each method.
38
39 // Use appropriate private or public protection for each method and field.

```

PART III: games2010rev3a/scrabble/ScrabbleBoard.java

```

984 private static void validateSpelling(String [] wordarray, String dictname) {
985     int status = OK ;
986     Scanner dictscanner = null ;
987     try{
988         dictscanner = new Scanner(new File(dictname));
989     } catch (java.io.FileNotFoundException fx) {
990         System.err.println("Dictionary file " + dictname + " not found.");
991         System.exit(BADFILE);
992     }
993     MySetInterface<String> dictionary = new MySetClass<String>();
994     while (dictscanner.hasNextLine()) {
995         String line = dictscanner.nextLine();
996         dictionary.add(line.toUpperCase());
997     }
998     for (int i = 0 ; i < wordarray.length ; i++) {
999         String [] fields = wordarray[i].split(":");
1000         if (! dictionary.contains(fields[0].toUpperCase())) {
1001             System.err.println(fields[0] + " not found in dictionary.");
1002             status = BADFILE ;
1003         }
1004     }
1005     // STUDENT 3: The above lines test a subset of the methods
1006     // specified in MySetInterface and implemented in MySetClass.
1007     // DO NOT CHANGE ANY OF THE ABOVE CODE IN THIS METHOD!
1008     // Add some lines BELOW to test your exception detection in
1009     // dictionary.add and also to test dictionary.toArray.
1010     // Write output for your added tests to System.err.
1011     // This means that you will have diffs in testboarddict2010.dif and
1012     // testboarddict2010_2.dif.
1013     //
1014     // When you are satisfied that testboarddict2010.out is correct
1015     // (i.e., all the diffs are what you expected to print below),
1016     // then you can
1017     // cp testboarddict2010.out testboarddict2010.ref
1018     // and re-run "gmake testboard" to test your update.

```

```

1019 //
1020 // When you are satisfied that testboarddict2010_2.out is correct
1021 // (i.e., all the diffs are what you expected to print below),
1022 // then you can
1023 // cp testboarddict2010_2.out testboarddict2010_2.ref
1024 // and re-run "gmake testboard" to test your update.
1025 //
1026 // I will be adding my own tests for this part as part of grading.
1027 }
1028 }

```

