

Java classes, interfaces and exceptions

Assignment 2 for CSC 243, Spring, 2010, Dr. Dale E. Parson

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

Assignment 2 is due 11:59 PM on Thursday March 18.

Use **gmake turnitin** from your **games2010rev3** directory to turn it in.

Javadoc for games2010rev3 preceding the assignment is at

<http://bill.kutztown.edu/~parson/javadoc/>.

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

```
cp ~parson/JavaLang/games2010rev3_assign2.zip ~/JavaLang
cd ~/JavaLang
/bin/unzip games2010rev3_assign2.zip
cd ./games2010rev3
gmake clean test
```

This test incorporates the original testtuyi, the first assignment's testboard, and this assignment's testspell. Initially the above line will fail to compile. After you eliminate all compilation errors, invoking **gmake testspell** will test only the assignment 3 portions of the code. Here are what the initial compilation failures look like:

```
cd scrabble && gmake build
gmake[1]: Entering directory `/export/home/faculty/parson/JavaLang/games2010rev3/scrabble'
/bin/bash -c "javac -g ScrabbleBoard.java"
ScrabbleBoard.java:362: cannot find symbol
symbol : class SpellingException
location: class games2010rev3.scrabble.ScrabbleBoard
    throws GameException, SpellingException {
        ^
/export/home/faculty/parson/JavaLang/games2010rev3/scrabble/ScrabbleGame.java:411: cannot
find symbol
symbol : method initGlobalDictionary(java.lang.String)
location: class games2010rev3.scrabble.ScrabbleGame
    initGlobalDictionary(keyvalue[1]);
        ^
/export/home/faculty/parson/JavaLang/games2010rev3/scrabble/ScrabbleGame.java:413: cannot
find symbol
symbol : method initLocalDictionary(java.lang.String)
location: class games2010rev3.scrabble.ScrabbleGame
    initLocalDictionary(keyvalue[1]);
        ^
/export/home/faculty/parson/JavaLang/games2010rev3/scrabble/ScrabbleGame.java:759: cannot
find symbol
symbol : class SpellingException
location: class games2010rev3.scrabble.ScrabbleGame
    } catch (SpellingException badspell) {
```

```
^
/export/home/faculty/parson/JavaLang/games2010rev3/scrabble/ScrabbleGame.java:789: cannot
find symbol
symbol : method updateLocalDictionary(java.lang.String)
location: class games2010rev3.scrabble.ScrabbleGame
    updateLocalDictionary(cmds[1].toUpperCase());
        ^
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
5 errors
gmake[1]: *** [ScrabbleBoard.class] Error 1
gmake[1]: Leaving directory `/export/home/faculty/parson/JavaLang/games2010rev3/scrabble'
gmake: *** [build] Error 2
-bash-3.00$
```

The failures have to do with the parts of the assignment that you must complete.

PART A: You must create class SpellingException inside subdirectory games2010rev3/scrabble, including Javadoc comments. It MUST BE A SUBCLASS of class GameException in package games2010.rev3. Use GameException as an example of how to write and document your function. One thing that you must be sure to document is that fact that the *String text* parameter of your SpellingException constructor takes a sequence of misspelled words on the current board in the form of a String with colons (":") separating the words; if there is only one misspelled word, then there is no colon. You will see how to build this word sequence in PART C.

When you have completed the assignment and "**gmake clean test**" works correctly from the games2010rev3/ directory, then from that directory enter **gmake turnitin**.

It will prompt you to hit Enter (Carriage Return) if you are sure, then it will bundle up the files and ship them to me. If you do not get an error message, then it worked. If you later make changes before the due date you can **gmake turnitin** again from the same directory, and it will over-write the prior submission. **Make sure that you turn this in by the end of March 18.**

PART B: You must complete code for STUDENT comments 1 through 4 in file games2010rev3/scrabble/ScrabbleGame.java. Below are some listings of code sections where you will add your code. Make sure to store all dictionary words IN UPPER CASE and to test all words from the board IN UPPER CASE as you did in assignment 1.

games2010rev3/scrabble/ScrabbleGame.java

```
1 /* ScrabbleGame.java
2 A Scrabble game engine that interprets player commands and runs a
3 game of Scrabble.
4 Demonstration of Java strings, characters, arrays, and textual file I/O.
5 Dr. Dale Parson, CSC 243, Fall, 2008
6 Updated Spring, 2009 with methods getRelationships(), getTiming(),
```

```

7  getValue(), and getCount() as added to interface TwoDimensionalBoardGame.
8  */

137 /***** SPELLING DICTIONARY FIELDS AND METHOD ADDED SPRING 2010
*****/
138
139 // Following spelling dictionaries added Spring, 2010.
140 // They are transient because we do not want to save them as part of
141 // a saved game. If both are left at null (no HashSet constructor for
142 // either), that means that spell checking is disabled. If one or two
143 // is initialized, then every word placed on a board or formed by
144 // a crossword must appear in either globalDictionary or localDictionary
145 // or both. These dictionaries are constructed by methods
146 // initGlobalDictionary(resourceName)
147 // initLocalDictionary(resourceName) respectively.
148 private transient HashSet<String> globalDictionary = null ;
149 private transient HashSet<String> localDictionary = null ;
150 // See updateLocalDictionary and initLocalDictionaryFile comments.
151 private transient PrintStream localDictionaryPrintStream = null ;
152
153 // STUDENT 1 (Spring 2010): You must write private void method
154 // initGlobalDictionary that takes one parameter, a String
155 // parameter named resourceName (that is its name, not its value),
156 // and performs the following work. See the completed definition
157 // of function initLocalDictionary below to see how to open
158 // resourceName within a Scanner object. Note that that method
159 // catches java.lang.Exception because there are several types
160 // of exceptions that can occur in opening a resource file.
161 // Make sure to invoke .close() on your Scanner object after reaching
162 // end-of-file.
163 //
164 // initGlobalDictionary attempts to open resourceName as a URL-based
165 // resource and read its lines. If it can open resourceName as
166 // a URL resource via getClass().getResource(), then it constructs
167 // a scanner, reads and stores all lines into field globalDictionary.
168 // initGlobalDictionary must construct globalDictionary upon its
169 // first SUCCESSFUL call with a valid resourceName; subsequent calls
170 // can be used to add additional contents into globalDictionary.
171 // If an attempt to open a resource at resourceName fails,
172 // initGlobalDictionary should print a warning to System.err but it
173 // should not throw an exception; if such a call is entered with
174 // globalDictionary == null, it should be left at null (do not construct
175 // a HashSet in that case). NOTE that any line in resourceName beginning
176 // with a '#' as its first character is considered a comment line;
177 // it should not be used as a valid spelling word.
178 // STUDENT 1 CODE GOES HERE.

```

```

179
180 // STUDENT 2 (Spring 2010): You must write private void method
181 // initLocalDictionary that takes one parameter, a String
182 // parameter named resourceName (that is its name, not its value),
183 // and performs the following work. See STUDENT 1 comments above
184 // for related instructions on exception handling and file closing.
185 //
186 // initLocalDictionary attempts to open resourceName as a URL-based
187 // resource and read its lines. If it can open resourceName as
188 // a URL resource via getClass().getResource(), then it constructs
189 // a scanner, reads and stores all lines into field localDictionary.
190 // Each line *MAY* contain an optional ":" separator, in which case
191 // initLocalDictionary should use the characters up to but not
192 // including the first ":" in the line as the spelling word.
193 // initLocalDictionary must construct localDictionary upon its
194 // first SUCCESSFUL call with a valid resourceName; subsequent calls
195 // can be used to add additional contents into localDictionary.
196 // If an attempt to open a resource at resourceName fails,
197 // initLocalDictionary should print a warning to System.err but it
198 // should not throw an exception; if such a call is entered with
199 // localDictionary == null, it should be left at null (do not construct
200 // a HashSet in that case). NOTE that any line in resourceName beginning
201 // with a '#' as its first character is considered a comment line;
202 // it should not be used as a valid spelling word.
203 // STUDENT 2 CODE GOES HERE.
204
205 // initLocalDictionaryFile attempts to open resourceName
206 // as a URL-based resource into a PrintStream in append
207 // mode for updating a local file-based copy of the
208 // local dictionary. It converts the resourceName into
209 // a URI-based File path and opens that File in append mode
210 // if possible; permissions or sandboxing may cause it to
211 // fail in its attempt to open.
212 // updateLocalDictionary() updates that PrintStream if it is open.
213 // initLocalDictionaryFile opens the file for APPEND only once,
214 // storing that object into localDictionaryPrintStream if and only if the
215 // latter is null. If the attempt to open localDictionaryPrintStream
216 // from filepath fails, initLocalDictionaryFile prints a
217 // warning to System.err without throwing an exception.
218 // In that case no new words will be added to filepath.
219 // (This code is complete as written; it is not a student assignment.)
220 private void initLocalDictionaryFile(String resourceName) {
221 if (localDictionaryPrintStream == null) {
222     try {
223         java.net.URL url = getClass().getResource(resourceName);
224         localDictionaryPrintStream = new PrintStream(

```

```

225         // Open the file in append mode (true argument below).
226         new FileOutputStream(new File(url.toURI()), true));
227     } catch (Exception ex) {
228         System.err.println("Warning: Cannot open local dictionary file "
229             + resourcename + " for writing: "
230             + ex.getMessage());
231     }
232 }
233 }
234
235 // STUDENT 3 (Spring 2010): You must write a method (return type void)
236 // with package-level protection named updateLocalDictionary() with
237 // one String parameter named colonSeparatedWords that satisfies
238 // the following requirements.
239 // It needs package-level protection so that the Scrabble-specific
240 // user interface classes can call it.
241 //
242 // updateLocalDictionary() adds words to the localDictionary
243 // HashSet *AND* an optional output file localDictionaryPrintStream
244 // if and only if that word is not already in localDictionary.
245 // If localDictionary is null then updateLocalDictionary does not change it
246 // and it does not print to localDictionaryPrintStream. It prints a
247 // warning to System.err when called with localDictionary == null.
248 // If localDictionaryPrintStream is null then updateLocalDictionary does
249 // not print to it.
250 // The current implementation of updateLocalDictionary() writes a
251 // new word to localDictionaryPrintStream with nothing else on the line.
252 // Parameter colonSeparatedWords is a list of one or more words
253 // to be saved as new words, separated by ":" characters;
254 // when there is no ":" character then there is only one new word.
255 // Each of these distinct words gets its own line in the output file.
256 // Package-level protection so Scrabble GUI can update upon a user request.
257 // STUDENT 3 CODE GOES HERE.
258
259 // Return true if localDictionary is non-null, meaning dictionary
260 // can be updated with new words. this function is already complete.
261 // Package-level protection so Scrabble GUI can update upon a user request.
262 boolean canUpdateLocalDictionary() {
263     return (localDictionary != null);
264 }
265
266 /***** SPELLING DICTIONARY FIELDS AND METHODS END HERE. *****/

```

```

372 // Load "ScrabbleGame.cfg" if it is available, and set defined
373 // configuration parameters.
374 public static final String CONFIGFILE = "ScrabbleGame.cfg";

```

```

375 public static final String GLOBALSPELL = "GLOBALSPELL";
376 public static final String LOCALSPELL = "LOCALSPELL";
377 private void loadConfigFile() {
378     HashSet<String> properties = new HashSet<String>(); // avoid repeats
379     Scanner cfgfile = null;
380     int linenum = 0;
381     try {
382         cfgfile = new Scanner(
383             getClass().getResource(CONFIGFILE).openStream());
384         // Catching java.lang.Exception instead of
385         // java.io.IOException thrown by openStream because
386         // getResource may return null, leading to NullPointerException.
387     } catch (java.lang.Exception iox) {
388         System.err.println("Warning: Cannot open configuration file "
389             + CONFIGFILE + ".");
390         return; // Do not abort but do not config any properties.
391     }
392     while (cfgfile.hasNextLine()) {
393         String line = cfgfile.nextLine();
394         linenum++;
395         if (line.charAt(0) == '#') {
396             continue; // Ignore comment lines.
397         }
398         String [] keyvalue = line.split(":");
399         if (keyvalue.length != 2) {
400             System.err.println("Warning: Configuration file "
401                 + CONFIGFILE + " has invalid line "
402                 + linenum + ": " + line + ".");
403             continue;
404         } else if (properties.contains(keyvalue[0])) {
405             System.err.println("Warning: Configuration file "
406                 + CONFIGFILE + " has duplicate property " + keyvalue[0]
407                 + " at line " + linenum + ": " + line + ".");
408             continue;
409         }
410         if (GLOBALSPELL.equals(keyvalue[0])) {
411             initGlobalDictionary(keyvalue[1]);
412         } else if (LOCALSPELL.equals(keyvalue[0])) {
413             initLocalDictionary(keyvalue[1]);
414             initLocalDictionaryFile(keyvalue[1]);
415         } else {
416             System.err.println("Warning: Configuration file "
417                 + CONFIGFILE + " has invalid property at line "
418                 + linenum + ": " + line + ".");
419             continue;
420         }

```

```

421     }
422     cfgfile.close();
423 }
424
717 if (cmds[0].equalsIgnoreCase("help")) {
718     outputStream.println(help());
719 } else if (cmds[0].equalsIgnoreCase("put")) {
720     if (cmds.length != 4) {
721         throw new GameException(
722             "usage: put WORD [A-O][1-15] ACROSS | DOWN");
723     }
724     String validword = board.validateWord(cmds[1], cmds[2], cmds[3]);
725     players[nextPlayer].validateLetters(validword);
726     ScrabbleBoard premove = (ScrabbleBoard) board.clone();
727     // Added Spring 2010 to undo effect of a spelling error.
728     try {
729         // board.putWord may throw SpellingException caught below.
730         int addscore = board.putWord(cmds[1], cmds[2], cmds[3],
731             globalDictionary, localDictionary);
732         int used = players[nextPlayer].deleteTiles(validword);
733         if (used == TILESPLAYER) {
734             addscore += BONUSFORALL ;
735         }
736         players[nextPlayer].addScore(addscore);
737         score.addEntry(players[nextPlayer].getName(),
738             addscore, players[nextPlayer].getScore());
739         players[nextPlayer].addTiles(drawFromSpareTiles(
740             TILESPLAYER - players[nextPlayer].tileCount()));
741         outputStream.println(SCORE);
742         movesmade++;
743         nextPlayer = (nextPlayer + 1) % players.length ;
744         // Get board state at PUT coords, added 9/2009
745         // for networked evts.
746         String [] boardstate = getRelationships(cmds[2]); // added 9/09
747         GameMoveEvent evt = new GameMoveEvent(this, movesmade,
748             cmds[1] + STRING_LOC_DIR_SPLITTER
749             + cmds[2] + STRING_LOC_DIR_SPLITTER + cmds[3], boardstate);
750         java.util.Iterator<GameMoveEventListener> iter
751             = moveevtlstnrs.iterator();
752         while (iter.hasNext()) {
753             try {
754                 iter.next().notifyMoveEvent(evt);
755             } catch (java.rmi.RemoteException ignoreme) {
756                 // Added 9/2009 to allow RMI distribution of events.
757             }

```

```

758     }
759     } catch (SpellingException badspell) {
760         board = premove ; // undo the spelling error
761         // STUDENT 4: Rethrow this exception on the next line.
762     }

```

PART C: You must complete code for STUDENT comments 5 through 7 in file games2010rev3/scrabble/ScrabbleBoard.java. Below are some listings of code sections where you will add your code. Make sure to store all dictionary words IN UPPER CASE and to test all words from the board IN UPPER CASE as you did in assignment 1.

games2010rev3/scrabble/ScrabbleBoard.java

```

85     ***** START OF THE SERIALIZABLE BOARD STATE VARIABLES
*****
86
87     private char [][] tiles
88         = new char [RowHeadings.length][ColumnHeadings.length];
89     private java.util.HashMap<Character, Integer> pointsmap = null ;
90     private boolean isfirstmove = true ;
91
92     // "history" keeps a history of successful putWords in FIFO order.
93     // history is used by getWordList().
94     private LinkedList<String> history = new LinkedList<String>() ;
95
96     // "timing" keeps a history of the timing of successful putWords
97     // in FIFO order.
98     // timing is used by getTiming().
99     private LinkedList<Long> timing = new LinkedList<Long>() ;
100    // lasttime records milliseconds from System.currentTimeMillis() at the
101    // time of the last move. It is initialized to System.currentTimeMillis()
102    // when this board is constructed.
103    private long lasttime = 0L ;
104
105    /**
106    * Return a cloned, depp copy of this ScrabbleBoard Object.
107    * @return a copy of the board with its own matrix of tiles and related
108    * state-bearing fields.
109    */
110    public Object clone() {
111        ScrabbleBoard result = new ScrabbleBoard(pointsmap);
112        for (int i = 0 ; i < RowHeadings.length ; i++) {
113            for (int j = 0 ; j < ColumnHeadings.length ; j++) {
114                result.tiles[i][j] = tiles[i][j];
115            }

```

```

116     }
117     result.isfirstmove = isfirstmove ;
118     result.history = (LinkedList<String>) history.clone();
119     result.timing = (LinkedList<Long>) timing.clone();
120     result.lasttime = lasttime ;
121     return result ;
122 }
123
124 /***** END OF THE SERIALIZABLE BOARD STATE VARIABLES
*****/

322 /**
323  * putWord() invokes validateWord(), then puts the word on the board and
324  * scores it.
325  *
326  * @param word is a word to place, in either lower or upper case,
327  * like this:
328  *     KUTzTOWN
329  * where the lower case 'z' signifies a blank being used as a Z.
330  * 'z' gains no points, and it must be used as a 'z' by any words
331  * that cross it. This goes for all lower case letters.
332  * Enhancement 11/17/200 now a " " blank tile can also be placed.
333
334  * @param location is a location of a letter 'A' through 'O'
335  * inclusive, signifying column 0 through 14, and then a numeric
336  * string '1' through '15' signifying row 0 through 14, where row
337  * '15' is at the top of the display. The first letter of the new word
338  * starts at this location.
339
340  * @param acrossORdown must be the word "ACROSS" or "DOWN" for the
341  * direction of the word of letters following the first letter.
342  *
343  * STUDENT 5: Replace these three lines with Javadoc parameter
344  * comments for globalDictionary and localDictionary. Running
345  * javadoc must report no errors.
346  *
347  * @return The score to add to the player's score resulting from this
348  * move, including any calculations of double-or-triple word-or-letter
349  * multipliers.
350  *
351  * @exception GameException If the proposed word is NOT valid at that
352  * location, this method throws a new GameException documenting
353  * the problem.
354  *
355  * STUDENT 6: Replace these three lines with Javadoc parameter
356  * comments for SpellingException. Running

```

```

357  * javadoc must report no errors.
358  *
359  **/
360 public int putWord(String word, String location, String acrossORdown,
361     Set<String> globalDictionary, Set<String> localDictionary)
362     throws GameException, SpellingException {
363     int rowdelta, coldelta ;
364     String usedword = validateWord(word, location, acrossORdown);
365     rowdelta = getRowdelta(acrossORdown);
366     coldelta = getColdelta(acrossORdown);
367     int startrow = -1, startcol = -1 ;
368     startcol = getStartCol(location);
369     startrow = getStartRow(location);
370     // Grow word as far as possible in its orientation.
371     StringBuilder bigword = new StringBuilder();
372     int prestartrow = startrow - rowdelta ;
373     int prestartcol = startcol - coldelta ;
374     int bigstartrow = startrow ;
375     int bigstartcol = startcol ;
376     while (prestartrow >= 0 && prestartrow < BOARDWIDTH
377         && prestartcol >= 0 && prestartcol < BOARDWIDTH
378         && tiles[prestartrow][prestartcol] != 0) {
379         bigstartrow = prestartrow ;
380         bigstartcol = prestartcol ;
381         bigword.insert(0,tiles[prestartrow][prestartcol]);
382         prestartrow -= rowdelta ;
383         prestartcol -= coldelta ;
384     }
385     bigword.append(word);
386     int poststartrow = bigstartrow + bigword.length() * rowdelta ;
387     int poststartcol = bigstartcol + bigword.length() * coldelta ;
388     while (poststartrow >= 0 && poststartrow < BOARDWIDTH
389         && poststartcol >= 0 && poststartcol < BOARDWIDTH
390         && tiles[poststartrow][poststartcol] != 0) {
391         bigword.append(tiles[poststartrow][poststartcol]);
392         poststartrow += rowdelta ;
393         poststartcol += coldelta ;
394     }
395     int score = wordscore(bigword, bigstartrow, bigstartcol,
396         rowdelta, coldelta);
397     // We have the extended word's score, now get the cross words.
398     int crossrowdelta = -1 * coldelta ;
399     int crosscoldelta = -1 * rowdelta ;
400     int nextouterrow = startrow, nextoutercol = startcol ;
401     for (int i = 0; i < usedword.length(); i++,
402         nextouterrow += rowdelta, nextoutercol += coldelta) {

```

```

403 char midchar = usedword.charAt(i);
404 if (midchar == ScrabbleGame.REUSED_TILE_CHAR) {
405     // If it is an old cross word, do not count it!
406     continue ;
407 }
408 StringBuilder crossword = new StringBuilder();
409 int startcrossrow = nextouterrow ;
410 int startcrosscol = nextoutercol ;
411 int testcrossrow = startcrossrow - crossrowdelta ;
412 int testcrosscol = startcrosscol - crosscoldelta ;
413 while (testcrossrow >= 0 && testcrossrow < BOARDWIDTH
414     && testcrosscol >= 0 && testcrosscol < BOARDWIDTH
415     && tiles[testcrossrow][testcrosscol] != 0) {
416     crossword.insert(0, tiles[testcrossrow][testcrosscol]);
417     startcrossrow = testcrossrow ;
418     startcrosscol = testcrosscol ;
419     testcrossrow -= crossrowdelta ;
420     testcrosscol -= crosscoldelta ;
421 }
422 crossword.append(midchar); // use the original letter
423 testcrossrow = nextouterrow + crossrowdelta ;
424 testcrosscol = nextoutercol + crosscoldelta ;
425 while (testcrossrow >= 0 && testcrossrow < BOARDWIDTH
426     && testcrosscol >= 0 && testcrosscol < BOARDWIDTH
427     && tiles[testcrossrow][testcrosscol] != 0) {
428     crossword.append(tiles[testcrossrow][testcrosscol]);
429     testcrossrow += crossrowdelta ;
430     testcrosscol += crosscoldelta ;
431 }
432 if (crossword.length() > 1) { // Only use it if it is a word.
433     score += wordscore(crossword, startcrossrow, startcrosscol,
434         crossrowdelta, crosscoldelta);
435 }
436 }
437 // Place the word's letters on the board.
438 int nextrow = startrow ;
439 int nextcol = startcol ;
440 for (int i = 0 ; i < word.length() ; i++,
441     nextrow += rowdelta, nextcol += coldelta) {
442     tiles[nextrow][nextcol] = word.charAt(i);
443 }
444 isFirstmove = false ;
445 history.addLast(word + STRING_LOC_DIR_SPLITTER
446     + location + STRING_LOC_DIR_SPLITTER + acrossORdown);
447 long now = System.currentTimeMillis() ;
448 timing.addLast(now - lasttime);

```

```

449 lasttime = now ;
450
451 // STUDENT 7: Here is pseudocode for the code you must write here:
452 /**** START OF PSEUDOCODE
453 if (globalDictionary is an object or localDictionary is an object) {
454     String [] wl = getWordList(location);
455     wl has an array of ":"-separated triplets.
456     Use the String split() function to get the first field
457     of each, which is the word on the board as in the last
458     assignment.
459
460     Create a StringBuffer object, initially empty, and then loop
461     through the words in wl[].
462     For each of these words {
463         if (globalDictionary is an object) {
464             if (it does not contain the upper case word) {
465                 if (localDictionary is null
466                     or if IT does not contain the upper case word) {
467                     NOTE that the word is misspelled
468                 }
469             }
470         } else if (localDictionary is an object and it
471             does not contain the upper case word) {
472             NOTE that the word is misspelled
473         }
474     }
475     If one of the above tests has NOTED a misspelled word {
476         Append that word to your StringBuffer object.
477         Every word after the first has a ":" delimiter
478         at its beginning.
479     }
480     If your StringBuffer object has a length > 0 {
481         raise a SpellingException with a String copy of
482         your StringBuffer object as the constructor parameter.
483     }
484 }
485 **** END OF PSEUDOCODE *****/
486
487 return score;
488 }

```

We will spend some time going over some of the steps I performed in order to prepare the game for this assignment. Among the more interesting is the undo-related processing of restoring board state after detection of a spelling error. Please attend.

Below is what the tests will look like once this is working. The “gmake testspell” test can be run alone. It uses files highlighted by the makefile command lines. Note that although gmake testing and the resulting .out, .ref and .dif files reside in directory games2010rev3/ (which is where you must run “gmake test” and “gmake turnitin”), your source files and the spelling configuration files reside in games2020rev3/scrabble/.

-bash-3.00\$ gmake testspell

```
cd scrabble && gmake build
gmake[1]: Entering directory `/export/home/faculty/parson/private/csc243/games2010rev3/scrabble'
gmake[1]: Nothing to be done for `build'.
gmake[1]: Leaving directory `/export/home/faculty/parson/private/csc243/games2010rev3/scrabble'
gmake ../games2010rev3.jar
gmake[1]: Entering directory `/export/home/faculty/parson/private/csc243/games2010rev3'
gmake[1]: `../games2010rev3.jar' is up to date.
gmake[1]: Leaving directory `/export/home/faculty/parson/private/csc243/games2010rev3'
cp scrabble/ScrabbleGame.cfg.bak scrabble/ScrabbleGame.cfg
cp scrabble/localenglish.0.txt.bak scrabble/localenglish.0.txt
java games2010rev3.GameMain games2010rev3.scrabble.ScrabbleGame
games2010rev3.GameTtyUI 2 2 < testcommandspell.txt > GameMainspell.out 2>&1
echo DICTIONARY: >> GameMainspell.out
cat scrabble/localenglish.0.txt >> GameMainspell.out
diff GameMainspell.out GameMainspell.ref > GameMainspell.dif
cp scrabble/localenglish.0.txt.bak scrabble/localenglish.0.txt
-bash-3.00$
```

Files:

scrabble/ScrabbleGame.cfg is a new configuration file for class ScrabbleGame.

scrabble/english.0.txt is the “global” file of “standard English” words.

scrabble/localenglish.0.txt is where our custom words are added.

GameMainspell.out, **GameMainspell.ref** and **GameMainspell.dif** are the test output, references and dif files created when “gmake testspell” runs.

USER NOTE: File games2010rev3/scrabble/ScrabbleGame.cfg contains default lines:

```
GLOBALSPELL:english.0.txt
LOCALSPELL:localenglish.0.txt
```

In directory games2010rev3/scrabble/ file english.0.txt holds the reference, immutable dictionary of words and localenglish.0.txt holds the local copy that players update when they add words. File localenglish.0.txt gets reinitialized during testing. So, if you play Scrabble and want to keep the words you add intact, change ScrabbleGame.cfg to point to a different local file when you play. Testing reinitializes ScrabbleGame.cfg to the above contents at the start of testing.