Project 1: Goblin

DUE: Sunday, September 15th at 11:59pm

Extra Credit Available for Early Submissions!

Basic Procedures

You must:

- Fill out a `readme.txt` file with your information (goes in your user folder, an example is provided)
- Have a style (indentation, good variable names, etc.)
- Comment your code well in JavaDoc style (no need to overdo it, just do it well)
- Have code that compiles with the command: `javac *.java` in your user directory
- Have code that runs with the command:

```
java GoblinGame [dictionary] [numLetters] [numGuesses] [debug]
```

You may:

- Add additional methods and variables, however these **must be private**.

You may NOT:

- Make your program part of a package.
- Add additional public methods or variables
- Use any built in Java Collections Framework classes anywhere in your program (e.g. no `ArrayList`, `LinkedList`, `HashSet`, etc.).
- Use any arrays anywhere in your program (except the data field provided in the `BetterArray` class)
- Alter any method signatures defined in this document of the template code. Note: "throws" is part of the method signature in Java, don't add/remove these.
- Alter provided classes/methods that are complete (`GoblinGame`, `BetterArray.toString()`, etc.).
- Add any additional import statements (or use the "fully qualified name" to get around adding import statements).
- Add any additional libraries/packages which require downloading from the internet.

Setup

- Download the `project1.zip` and unzip it. This will create a folder `section-yourGMUUserName-p1`
- Rename the folder replacing `section` with the `001`, `003`, `004` etc. based on the lecture section you are in
- Rename the folder replacing `yourGMUUserName` with the first part of your GMU email address
- After renaming, your folder should be named something like: `001-krusselc-p1`
- Complete the `readme.txt` file (an example file is included: `exampleReadmeFile.txt`)

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc.
- You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name the zip `section-username-p1.zip` (no other type of archive) following the same rules for `section` and `username` as described above.
  - The submitted file should look something like this:

```
001-krusselc-p1.zip --> 001-krusselc-p1 --> JavaFile1.java
                                            JavaFile2.java
                                            ...
```

- Submit to blackboard.

Grading Rubric

Due to the complexity of this assignment, an accompanying grading rubric pdf has been included with this assignment. Please refer to this document for a complete explanation of the grading.

Overview

Goblins are notorious for both their thieving skills and their riddle games. For example, some days when you came home there's a goblin stealing your ramen noodles. If you catch him at it, he always challenges you to a riddle game for your dinner. Unfortunately, goblins like to cheat! Here's a typical evening discussion:

Goblin: I'm thinking of a six letter word from this list: peanut, slower, packet, faster, parcel.
You can guess one letter at a time and I'll tell you if it's in the word and where.
You can only guess wrong twice.

<The goblin isn't really thinking of a word, he's going to change his answer to win the game.>

You: "l"
Goblin: Nope! One wrong guess left!

<The goblin looked at his list and decided there were more words without an "l". So he's going to use one of the following: peanut, packet, faster>

You: "p"
Goblin: Yeah, there's a "p" here: p - - - - -

<The goblin looked at his list of words and decided that there were more words with p in the first position. So he's going to use one of the following: peanut, packet>

You: "n"
Goblin: Nope! I win! Ramen is mine now.

<The goblin disappears in a puff of smoke with your dinner.>

You have so much fun playing this game that you decide to recreate the experience for others. This is the game we're going to be creating.

Differences between the scenario above and the game you are writing:

1. The virtual goblin will be using a much larger list (he'll be reading words in from a provided dictionary).
2. The virtual goblin will say very specific things during the game which you must make match the sample runs for full credit.
3. In "debug mode" you'll be able to see the goblin's current list of words, but usually you won't have that.
4. The number of letters in the word and the number of guesses will be set by a command line argument.
5. For simplicity, the goblin will not use words which contain duplicate letters.

Note: In the below examples, user input is shown in green highlight and underlined.

Sample Run

```
> java GoblinGame ..\dictionary-mini.txt 6 2
Goblin says "Guess a letter": l
Goblin says "No dice! 1 wrong guesses left..."
Goblin says "Guess a letter": p
Goblin says "Yeah, yeah, it's like this: p-----"
Goblin says "Guess a letter": n
Goblin says "No dice! 0 wrong guesses left..."
Goblin says "I win! I was thinking of the word 'packet'
Your stuff is all mine... I'll come back for more soon!"
```

Sample Run Debug Mode

```
> java GoblinGame ..\dictionary-mini.txt 6 2 debug
--------------------------------------------------
peanut, slower, packet, faster, parcel
--------------------------------------------------
Goblin says "Guess a letter": l
Goblin says "No dice! 1 wrong guesses left..."
--------------------------------------------------
peanut, packet, faster
--------------------------------------------------
Goblin says "Guess a letter": p
Goblin says "Yeah, yeah, it's like this: p-----"
--------------------------------------------------
peanut, packet
--------------------------------------------------
Goblin says "Guess a letter": n
Goblin says "No dice! 0 wrong guesses left..."
--------------------------------------------------
packet
--------------------------------------------------
Goblin says "I win! I was thinking of the word 'packet'
Your stuff is all mine... I'll come back for more soon!"
```

The Goblin Algorithm

Let's say the goblin has the following words in his dictionary (`dictionary-mini.txt`):

one	three	peanut	pepper	slower	packet	pagoda	faster	papers	parcel

And he plays a game where there are 6 letters and 2 guesses. He will choose the following highlighted words:

one	three	**peanut**	pepper	**slower**	**packet**	pagoda	**faster**	papers	**parcel**

He will not choose "one" or "three" because they are not of length 6. He will not choose "pepper", "pagoda", or "papers" because they each have duplicate letters.

This is why you see the following in Sample Run Debug Mode:

```
> java GoblinGame ..\dictionary-mini.txt 6 2 debug
--------------------------------------------------
peanut, slower, packet, faster, parcel
--------------------------------------------------
```

Let's say the user guesses "l":

```
Goblin says "Guess a letter": l
```

Now the Goblin wants to cheat. He's not really thinking of a word, he's thinking of winning the game! The goblin will partition his current list of words (peanut, slower, packet, faster, parcel) into seven categories (number of letters + 1).

Words without the letter "l"	Words with the letter "l" in position 1	Words with the letter "l" in position 2	Words with the letter "l" in position 3	Words with the letter "l" in position 4	Words with the letter "l" in position 5	Words with the letter "l" in position 6
peanut **packet** **faster**		**slower**				**parcel**

Now he will choose the biggest "partition", in this case 'Words without the letter "l"'. So he'll respond:

```
Goblin says "No dice! 1 wrong guesses left..."
```

But he can’t be caught cheating, so he can only use words from that partition later (peanut, packet, faster). So when the user guesses “p”:

```
Goblin says "Guess a letter": p
```

The goblin does this partition:

Words without the letter “p”	Words with the letter “p” in position 1	Words with the letter “p” in position 2	Words with the letter “p” in position 3	Words with the letter “p” in position 4	Words with the letter “p” in position 5	Words with the letter “p” in position 6
faster	**peanut** **packet**					

And now he picks partition ‘Words with the letter “l” in position 1’ and says:

```
Goblin says "Yeah, yeah, it's like this: p-----"
```

Now that you understand more about the game, let’s look at another sample run:

Sample Run Debug Mode 2

```
> java GoblinGame ..\dictionary-mini.txt 6 10 debug
Goblin says "Guess a letter": p
Goblin says "Yeah, yeah, it's like this: p-----"
Goblin says "Guess a letter": a
Goblin says "Yeah, yeah, it's like this: pa----"
Goblin says "Guess a letter": e
Goblin says "Yeah, yeah, it's like this: pa--e-"
Goblin says "Guess a letter": e
Goblin says "You already guessed: p, a, e
Guess a new letter": t
Goblin says "No dice! 9 wrong guesses left..."
Goblin says "Guess a letter": c
Goblin says "Yeah, yeah, it's like this: pa-ce-"
Goblin says "Guess a letter": t
Goblin says "You already guessed: p, a, e, t, c
Guess a new letter": banana
Goblin says "One letter! Guess a single letter": <enter>
Goblin says "One letter! Guess a single letter": l
Goblin says "Yeah, yeah, it's like this: pa-cel"
Guess a new letter": m
Goblin says "No dice! 8 wrong guesses left..."
Guess a new letter": 8
Goblin says "No dice! 7 wrong guesses left..."
Goblin says "Guess a letter": r
Goblin says "Yeah, yeah, it's like this: parcel"
Goblin says "You win... here's your stuff. But I'll get you next time!"
```

Current List: peanut, slower, packet, faster, parcel

Current List: peanut, packet, parcel

Current List: packet, parcel

Duplicate Guess

Current List: parcel

Duplicate Guess (!= wrong guess)

Invalid Guesses (!= wrong guess)
- more than one letter
- nothing (user just hits enter)

You may assume all single characters are valid guesses (even non-letters)

Notes:

- The goblin says different things for wrong, duplicate, and invalid guesses.
- Only wrong guesses (not duplicate or invalid guesses) will count down the number of remaining guesses.
- When the user picked “t” there were two even partitions (“no words” and “t at the end”), the goblin prefers to tell the user he’s wrong (and count down), or if there is a tie for letter position, he chooses the earliest letter position.
- If the Goblin wins, he always claims he was thinking of the first word in his remaining list.
- Words should always remain sorted in the order they appear in the dictionary file (not necessarily in alphabetical order). You should not need to really do anything to have this happen, but it will affect your debug output and the goblin’s final words if you mix them up somehow and you’ll lose points ☹

Requirements

An overview of the requirements are listed below, please see the grading rubric for more details.

- **Implementing the classes** - You will need to implement required classes and provided template files.
- **Big-O** - The template files provided to you contains instructions on the REQUIRED Big-O runtime for many methods. Your implementation of those methods should NOT have a higher Big-O.
- **Style** – You must follow the coding and conventions specified by the style checker.
- **JavaDocs** - You are required to write JavaDoc comments for all the required classes and methods.

JavaDocs?? Style Checker??

Yes, as you may often be asked to do in a professional setting, you'll need to document your code correctly and conform to a given code style (many companies have their own style requirements, but there are also some standard ones like Sun and Google). We'll be using `checkstyle` (https://checkstyle.org) which is tool to automate style checking (no one likes doing it manually). It has plugins for Eclipse, NetBeans, jGRASP, and many others (see their website), but there is also a command line interface (CLI) which has been provided to you. See the "Testing" and "Command Reference" section for more details.

Input Format(s)

Dictionary files will contain one word per line and only words with alphabetical characters (a-z) and only lower case letters. Two dictionaries have been provided (`dictionary.txt` is a Scrabble dictionary, `dictionary-mini.txt` is for use in debugging and for showing sample runs). You may assume the given dictionary is correctly formatted for this assignment.

Implementation/Classes

This project will be built using a number of classes representing the component pieces of the project. Here we provide a description of these classes. Template files are provided for each class in the project folder and these contain further comments and additional details.

BetterArray<T> (BetterArray.java): This class represents a generic collection of objects in an array whose capacity can grow and shrink as needed. It supports the usual operations of adding and removing objects including **`add()`** and **`delete()`**. Check the included template file for all required methods and their corresponding big-O requirements. The capacity of a `BetterArray` may change as items are added or removed and the following rules must be followed in your implementation:

- The default initial capacity is fixed to be 2 if not specified.
- When you need to add an item and there is not enough space, grow the array to double its capacity.
- When you delete an item and the size falls below 1/4 of the capacity, shrink the array to half its capacity.

Goblin (Goblin.java): This class implements the goblin game. It must use `BetterArray` objects for storage and you are not permitted to use any arrays anywhere in this file (the main method which interacts with args[] has already been implemented). You will be implementing a typical "game loop" structure:

- `init()` – this sets everything up for the game
- `step()` – this plays one round of the game (one guess by the user and goblin response)
    - `bestPartition()` – this is a helper method for `step()` which chooses subset of the goblin's current list based on the user's guess
- `finish()` – this presents the goblin's final choice

The class define multiple methods corresponding to those steps. Some methods include partial implementations to provide a starting point for you. Check the included template file for all required methods and their big-O requirements.

GoblinGame (GoblinGame.java): This class runs the actual game loop and is fully provided (do not edit).

How To Handle a Multi-Week Project

While this project is given to you to work on over two weeks, and you are unlikely to be able to complete this in one weekend if you don't plan ahead. We recommend the following schedule:

- Step 1 (Understand the Game): Before the first weekend
  - Read project specification, inspect the given template files and get yourself familiar with the goblin's game and how the goblin is "cheating".
  - You'll want to review using the `Scanner` class and various `String` methods, along with creating and using generic classes.
  - Make sure you remember how to compile and run `javadocs`, `checkstyle`, and `junit` tests.
- Step 2 (`BetterArray`): First weekend (9/6-9/8)
  - At this point you've learned about basic lists in class, so complete `BetterArray` if you haven't already and move on to the actual game.
  - For `Goblin`, start by planning what you need to store and how you want to store it (the accessors might give you a hint for this). Once you think you have that settled, start working on the class in this order: the four accessors, `init()`, `bestPartition()`, `step()`, and `finish()`.
  - If you're struggling... go to office hours as soon as you can!
- Step 3 (`Goblin`): Before the second weekend (9/9-9/12)
  - Finish up the rest of the game and play it through many times to test when you're done!

This schedule will leave you with the second weekend (9/13-9/15) to perform additional testing, get additional help, and otherwise handle the rest of life. ☺ Also, notice that if you get it done early in the week, you can get extra credit! Check our grading rubric PDF for details. Otherwise you'll have plenty of time to test and debug your implementation before the due date.

Testing

BetterArray and Goblin

The main methods in the provided in the template classes contain useful code to test your project as you work. You can use command like "`java BetterArray`" to run the testing defined in `main()`. You could also edit this `main()` to perform additional testing. JUnit test cases will not be provided for these classes, but feel free to create JUnit tests for yourself. A part of your grade *will* be based on automatic grading using test cases made from the specifications provided.

Goblin and GoblinGame

The best way to start testing the full game is to run it with the following command:

```
java GoblinGame [dictionary] [numLetters] [numGuesses] [debug]
```

- `dictionary` – the path to the dictionary file you want to use
- `numLetters` – the number of letters the goblin will choose
- `numGuesses` – the number of guesses the user will have
- `debug` – literally the word "debug" (this is optional)

For example, from your user directory you can run:

```
        java GoblinGame ..\dictionary-mini.txt 6 2
```

And play the sample game from earlier in this document.

Note: The default output from `GoblinGame` is to show the user-goblin interactions. To see the goblin's word list as you play, you can add the word "debug" as the last command line argument as is shown in the "Sample Run Debug Mode".

Exactly Matching the Output

A number of different sample runs have been given to you in the `sampleOuts` directory which you can use to check your spelling, spacing, newlines, etc.

Input File	Args	Correct Output
test1in.txt	..\dictionary-mini.txt 6 2	test1out-correct.txt
test1in.txt	..\dictionary-mini.txt 6 2 debug	test1out-debug-correct.txt
test2in.txt	..\dictionary-mini.txt 6 2	test2out-correct.txt
test2in.txt	..\dictionary-mini.txt 6 2 debug	test2out-debug-correct.txt
test3in.txt	..\dictionary.txt 5 100	test3out-correct.txt
test3in.txt	..\dictionary.txt 5 100 debug	test3out-debug-correct.txt

If you want to generate your own output into a file to compare you can use the following command:

```
java GoblinGame [args from above] < ..\sampleOuts\[input file from above] > [new output file]
```

For example, entering the following command from your user directory:

```
java GoblinGame ..\dictionary-mini.txt 6 2 < ..\sampleOuts\test1in.txt > out.txt
```

will generate a file called `out.txt` in your user directory. Comparing this file to `test1out-correct.txt` will show you what is correct and/or not correct.

Six unit tests have been provided to automate the above manual process (`GameTest.java`) since it is hard to eyeball differences in large text files. To run these tests, navigate to the directory *above* your user directory (from your user directory type `cd ..` to go up one directory) and do the following...

Compile and run the tests with the following in Windows:

```
javac -cp .;junit-4.11.jar;[userDirectory] GameTest.java
java -cp .;junit-4.11.jar;[userDirectory] GameTest
```

Or for Linux/Mac:

```
javac -cp .:junit-4.11.jar:[userDirectory] GameTest.java
java -cp .:junit-4.11.jar:[userDirectory] GameTest
```

Style Checking

The provided `cs310code.xml` checks for common coding convention mistakes (such as indentation and naming issues). The provided `cs310comments.xml` checks for JavaDoc issues and in-line comment indentation issues. You can use the following to check your style:

```
java -jar checkstyle.jar -c [style.xml file] [userDirectory]\*.java
```

For example, for a user directory 001-krusselc-p1 checking for JavaDoc mistakes I would use:

```
java -jar checkstyle.jar -c cs310comments.xml 001-krusselc-p1\*.java
```

Note: if you have a lot of messages from `checkstyle`, you can add the following to the end of the command to output it to a file called out.txt: `> out.txt`

Command Reference

All commands are from inside your user folder and assume you left the style items and dictionaries in an outer folder (as in you unzipped project1.zip into a single place as suggested).

From in your user directory:

Compile:	`javac *.java`
Run:	`java GoblinGame [dictionary] [numLetters] [numGuesses] [debug]`
Compile JavaDocs:	`javadoc -private -d ../docs`

From *above* your user directory:

Style Checker:	`java -jar checkstyle.jar -c cs310code.xml [userDirectory]\*.java`
Comments Checker:	`java -jar checkstyle.jar -c cs310comments.xml [userDirectory]\*.java`
Compile Unit Tests:	`javac -cp .;junit-4.11.jar;[userDirectory] GameTest.java`
Run Unit Tests:	`java -cp .;junit-4.11.jar;[userDirectory] GameTest`