Programming Assignment #2 (Java)

CS440

due 21 October 2021 11:59pm

1. Implement the class `HiLo`. 20 pts.
2. Implement the class `Liar`. 50 pts.
3. Complete the class `TextUI`. 30 pts.

The objective of this assignment is to implement a program that manages to guess a secret based on the user's answers to questions, even though the user is allowed to lie. The assignment illustrates the use of Java collections, including generics, interfaces, inner classes and simple I/O.

The package to be implemented, `guess`, includes an interface and three classes:

- Interface `Guesser` defines the public methods of a guessing object. A `Guesser` is used as follows:
  - the object is first initialized via the `initialize` method;
  - method `makeQuestion` produces a new question;
  - the user's answer to this question is indicated by using one of the the `yes` or `no` methods;
  - the cycle of questions and answers continues until method `hasSolved` returns true, at which point the secret can be retrieved using method `getSecret`;
  - during the guessing process, method `progress` indicates, with a value between 0 and 1, how close the object is to guessing the secret.
- Class `HiLo` is a first, very simple implementation of interface `Guesser`. It guesses a secret number by repeatedly asking whether this number is larger than a proposed number X. The strategy is straightforward: pick a number X in the middle of the range of possible secret numbers. Based on the user's answer, the possible range is halved with each question, until it contains exactly one number, which has to be the secret.
- Class `Liar` implements a more interesting guesser that allows the user to lie, as long as the number of lies is bounded. The algorithm to find the secret is explained in Paul Hartung's paper *Tell all the lies you want — Liars never win*, attached as an appendix. This algorithm produces the secret object *and* the number of times the user lied during the process.
- Class `TextUI` implements a simple text-based terminal interface. It displays messages and questions on a `java.io.Writer` and reads the user's yes/no answers from a `java.io.Reader`. The class also contains a program that creates a `HiLo` or a `Liar` instance based on command-line options and starts a game using `System.in` and `System.out`.

The `Guesser` interface is generic: it is parameterized by the type of secrets (i.e., the return type of method `getSecret`). Class `HiLo` produces a number and implements `Guesser<Integer>`. Class `Liar` is itself generic, based on a type `T` of secrets. So that the number of lies can be returned, it does not implement `Guesser<T>` but `Guesser<Liar.Secret<T>>`. `Liar.Secret` is a static member class or `Liar` and is itself generic. It simply represents a pair: a `T` value (the secret) and an `int` (the number of lies).

Classes `HiLo` and `Liar` must implement a method `Progress` that continuously increases as the guessing process progresses. For `HiLo`, it can be based on the length of the range of possible secrets (which starts as the full possible range of values and ends as a single point); for `Liar`, one can use the number of possible objects that remain, combined with what columns they are in (i.e., the number of times they can still be lied about and be eligible as the secret object). It is not specified how exactly the value returned increases

but it must be equal to 1 when the secret is found, it must be equal to 0 initially unless the secret is found, and it must increase after *each* question is answered.

Class `TextUI` implements a text-based user interface. It is created with a `Guesser` object and I/O objects to write questions and to read answers. It offers a public method `play`, which starts the interaction with the user and allows him to play several games. It is blocking (i.e., it does not return until the user is done playing) and it returns the number of games played.

The `TextUI` class also contains a command-line program, which can be used as follows:

```
java guess.TextUI -hilo min max
java guess.TextUI -liar #lies name1 name2 name3 ...
```

The first form starts a `HiLo` game with the specified range of possible numbers. The second form starts a `Liar` game with the specified upper bound on the number of lies and the given list of object names.

Notes:

- Classes to be implemented are described at:

  `http://www.cs.earlham.edu/~sofialemons/cs440/hw2/`

- The names and types of public (and protected) attributes **cannot** be modified.
- The names and signatures of public (and protected) methods **cannot** be modified. In particular, you cannot add/remove a parameter, add/remove a thrown exception, or change the return type.
- You are free to have all the private and package-private methods, attributes and classes you want, but they must be documented. You cannot add or remove public or protected members.
- Data structures from `java.util` can be used, but generic data structures must be used with their generic parameters set properly. In particular, compilation with the option `-Xlint` shouldn't produce any warning.
- Methods from the `Guesser` interface are expected to be called in a specific order (i.e., `initialize` first, then `makeQuestion`, then `yes` or `no`, etc.). When a method is called at the wrong time, it should throw an `IllegalStateException` specifically.
- In order to automate testing of `TextUI`, it is important that the `play()` method follows the given protocol precisely. Specifically, it should ask a yes/no question to the user after each guesser-generated question is displayed. It should also ask a yes/no question at the end of a game, to which a positive answer means to keep playing (e.g., the interface can ask: "*Do you want to play another game?*", not "*Are you done playing?*"). The interface should ask no other question.
- Be careful with precision loss when using `double` values. In particular, `progress() == 1.0` should be *true* when a secret is found.
- Class `HiLo` must return the secret as an `Integer` object because of the requirements of the `Guesser` interface. Internally, it should rely on primitive types for its calculations. Be careful with overflow.
- The easiest way to handle I/O in this assignment is to rely on `java.util.Scanner` and `java.io.PrintWriter`, but you are free to use other approaches. Observe, however, that `TextUI.play` does *not* throw `java.io.IOException`.
- Code for the `Guesser` interface is is provided in the file

  `Guesser.java`

- A possible implementation of `TextUI.main` is provided in the file

  `TextUI.java`

  It relies on two helper methods, `makeHiLoGuesser` and `makeLiarGuesser`, the first of which is implemented and the other left to be written as part of the assignment.

- Sample tests will be provided in the file

  `SampleTests.java`

  To run the provided sample tests, you need `JUnit (https://junit.org/junit4/)`. It is installed on `bowie` and included in the default class path. On other computers, you need to download `JUnit` and make sure it is in your class path. You can compile the tests using `javac` and run them using the command `java org.junit.runner.JUnitCore guess.tests.SampleTests`. You may need to set up a directory structure with your code in a `guess` directory and the sample tests in a `tests` subdirectory.

- Below is a sample `HiLo` interaction with the text-based interface:

```
> java guess.TextUI -hilo 1 10

Pick a number between 1 and 10
Is your number larger than 5? n
I'm 30% finished
Is your number larger than 3? n
I'm 52% finished
Is your number larger than 2? y
I'm 100% finished
The secret is: 3

Play again? y

Pick a number between 1 and 10
Is your number larger than 5? y
I'm 30% finished
Is your number larger than 8? n
I'm 52% finished
Is your number larger than 7? n
I'm 70% finished
Is your number larger than 6? n
I'm 100% finished
The secret is: 6

Play again? n
(2 games played)
```

- Below is a sample `Liar` interaction with the text-based interface:

```
> java guess.TextUI -liar 2 blue green yellow red black white

Pick one among green red blue yellow black white
Is the secret string among green red blue ? y
I'm 17% finished
Is the secret string among green blue yellow ? n
I'm 33% finished
```

```
Is the secret string among white yellow red ? n
I'm 50% finished
Is the secret string among white red black ? y
I'm 61% finished
Is the secret string among blue black white ? y
I'm 72% finished
Is the secret string among red black ? y
I'm 83% finished
Is the secret string red ? n
I'm 100% finished
The secret is: black (with 1 lie)

Play again? n
(1 game played)
```

J. RECREATIONAL MATHEMATICS, Vol. 12(3), 1979-80

TELL ALL THE LIES YOU WANT – LIARS NEVER WIN

PAUL HARTUNG
Bloomsburg State College
Bloomsburg, Pennsylvania

Place eight objects on the table. Secretly select one, and I, by asking several questions, will tell you what the object is. You need only answer "yes" or "no" in response to the questions. Simple, you say? But wait, I forgot to mention that you can tell the truth or you can lie in response to each question as many times as you wish. The system for selecting the object is a strictly mechanical procedure. Even though you can tell as many lies as you wish (you may tell 80 or 500 lies if so desired), you must tell me the *most* number of lies you will use (although you may use less than this number if desired). For simplicity in exhibiting this technique, let us say that you can tell at most two lies, that is, you may tell either zero, one, or two lies. The procedure is clearly valid for any number of lies. Let it be that the eight objects are written here as a, b, c, d, e, f, g, h. Let the secretly chosen object be e. Place the objects in an upper pile and a lower pile (see diagram).

No lies	1 lie	2 lies	dump
a b c d			
e f g h			

The following procedure will be the same each time:

Procedure

1. I ask the question: "Is it in the top half?"
2. If the response is "yes," I move the *bottom* objects one place to the right. If the response is "no," I move the *top* objects one place to the right.

3. Keeping all objects in their same "lie" column, evenly distribute them on top and bottom (or in the case of 5 objects, place 3 on top and 2 on the bottom).

Suppose you answer "yes" the first time. The objects then look like this:

No lies	1 lie	2 lies	dump
a b c d			
	e f g h		

Note, *one lie* has been told and the chosen object "e" is in the "one lie" column. I now evenly distribute them.

No lies	1 lie	2 lies	dump
a b	e f		
c d	g h		

Suppose the procedure is applied again and the response is "yes" this time. I move the bottom objects to the right and distribute them evenly, Note that only one lie has been told so far and "e" is still in the "one lie" column.

No lies	1 lie	2 lies	dump
a	e f	g	
b	c d	h	

We suppose the next response to the same question is "yes." The objects look like:

No lies	1 lie	2 lies	dump
a	e f	g	
	b	c d	h

Observe that only one lie has been told so far (on the first response) and "e" is still in the "one lie" column. "h" has been dumped (or is in the "three lie" column). If one follows h from the beginning, you will see that three lies have been "put on h." Since the maximum number of lies was two, h is not eligible as the secretly picked object. Let the next response to the question be "no." The objects look like:

No lies	1 lie	2 lies	dump
	a	e f	g
	b	c d	h

Two lies have now been told on "e" and "e" is in the "two lie" column. Continuing with a "yes" response,

No lies	1 lie	2 lies	dump
	a	e f	g
		b	c d h

and another "yes" response, and then separating,

No lies	1 lie	2 lies	dump
	a	e	g
		f	b c d h

and another "yes" response yields:

No lies	1 lie	2 lies	dump
	a	e	g
			f b c d h

Separating, we have:

No lies	1 lie	2 lies	dump
	a		g
		e	f b c d h

A "no" response gives us:

No lies	1 lie	2 lies	dump
		a	g
		e	f b c d h

and finally, a "no" response gives us:

No lies	1 lie	2 lies	dump
			a g
		e	f b c d h

We now have "e" sitting alone in the "two lie" column. It follows that "e" is the object and two lies have been told. If the procedure had been carried out and we ended up with:

No lies	1 lie	2 lies	dump
	b		a c e
			f g h d

then "b" is the object and one lie has been told.