Object Oriented Programming - Exercise I - Flow Control and working with Objects

March 3, 2014

1 Objectives

This exercise's purpose is to get you a little more comfortable with basic programming in Java. It focuses on the flow principles taught in class (such as loops and conditions). You will also need to *use* supplied objects. Hopefully you would get the notion of how intuitive and natural it is to integrate objects in your code (code examples are provided).

2 Exercise definition

Mastermind or Bulls and cows (Bul Pgiaa in Hebrew) is a code breaking game for two players. Your task in this exercise is to write a Java program that conducts a simplified version of the game. The computer will always play the side of the *codemaker* while the user will play the *codebreaker*. Specifically, you would have to implement the logic of the game, as described below. Your implementation should reside entirely in the *main* function of a class called Mastermind which you will submit.

The game you will implement has three user-specified parameters:

1. The length of the code, which must be a positive integer.
2. The number of possible values in each position of the code. This value must also be a positive number. If the given value of this parameter is n then the values for each position in the code will be a number between 0 and $n - 1$.
3. The maximum number of guesses the codebreaker has to discover the code. This number also has to be positive.

The program you will write should:

1. Play the side of the codemaker.
2. Interact with the user; and
3. Keep track of the user's performance:

(a) The number of games the user has played.

(b) The number of games the user has won.

(c) The user's win rate. (Calculated as the number of wins divided by number of games.)

(d) The average win length. (Can be calculated as total number of turns in wins divided by number of wins.)

All the interaction with the user, both output and input, are carried out using the *MastermindUI* class. In addition we also provide you with the *Code* class that represents a code. More about using the given objects in the next section.

3 Working with the given objects

As mentioned before, a large part of the tasks which are needed for this exercised are already implemented and ready for your use in the form of objects. The main object you will use is called *MastermindUI*. Using it you would be able to read information from the user, and respond accordingly. To get a copy (an instance) of this object just add the following line:

```
MastermindUI ui = MastermindUIFactory.newMastermindUI();
```

Now we can use it for the different input/ouput tasks required for the implementation of this exercise. For instance,

```
int num = ui.askNumber("Please enter a number:");
```

will read a number from the user, and store it in the variable *num*. Another example -

```
boolean answer = ui.askYesNo("Would you want to proceed?");
```

will prompt the user with the questions "Would you want to proceed?" and wait for it's response (either "yes" or "no"). The boolean variable *answer* will hold *true* for "yes" and *false* otherwise. You really should read *MastermindUI*'s full documentation which can be found here. Another useful object supplied to you is the *Code* class. This class represents a code either created by the codemaker or guessed by the codebreaker. If we want to generate a new code with 3 characters, each of them may have one of 5 values, we can use the following code:

```
Code code = codeGenerator.newCode(3,5); // Each position of the code will have a value in
    [0..4]
```

Similarly, if we would like to ask the user for a new code guess, we can use the *MasterMindUI* previously described -

```
Code c = ui.askGuess("Enter guess:", 3); // This will ask the user to enter a code of size 3.
```

Another very useful method of the *Code* object you might find useful is *getValue(int pos)* which returns the value at a specific a position in the code. As with *MastermindUI* you should really read the whole *Code* documentation which can be found here.
The flow of the Mastermind game you will write is outlined below. All user interaction must be performed through the **MastermindUI** class (UI is a common acronym for User Interface):

1. A new MastermindUI object is created as described in the previous section.

2. (a) The user is asked for the the game parameters (see: *MastermindUI.askNumber()*). In the following order

i. the length of the code,
ii. the number of possible values in the code,
iii. the maximal number of guesses per game.

Each of these values must be positive and if a nonpositive value is received, the program should display an error message (choose a informative message as you see right, however make sure to use *MastermindUI.displayErrorMessage()* to prompt it) and ask again for the value of that parameter.

(b) The *MasterMindUI* object should be reset using the parameter values provided by the user in step (a) above s(see:*MastermindUI.reset()*. An informative message will be prompted to the user).

(c) All game statistics are set to their initial values. Note the *MastermindUI* doesn't keep track of the user actions. You should keep track of the *codebreaker*'s guesses, and success rates on your own. This would be required for the calculation of the statistics as will be described later on.

3. A new random code (represented by Code object) is created as explained in the previous section.

4. One or more turns take place, until either the user correctly guesses the code or the maximal number of guesses were used. In each turn, the user enters a guess and in response is presented with the result of this guess. (See: *MastermindUI.askGuess()*) and the *MastermindUI.showGuessResult()*). The result of the guess is the number of values guessed correctly (called "bulls"), and the number of misplaced values ("cows"). You should calculate those two values by your own. To read values from the *Code* object you should use *Code.getValue()*; For example, if the hidden code is [1,2,3,4] and the *codebreaker* guessed [1,3,2,4] then this guess result is two "cows" and two "bulls".

5. The game outcome should be shown (use MastermindUI.displayMessage()), and the user performance statistics should be updated and shown. (The number of games the user has played, the number of games the user has won, the user's win rate, and the average win length. See: *MastermindUI.showStats()*)

6. The user is asked whether he or she wishes to play another game. If the user declines, then the UI is closed (*MastermindUI.close()* should be called and the program should complete. If the user chooses to play another game, then he or she is asked whether he would to like to change the game parameters (see: *MastermindUI.askYesNo*). If the *codebreaker* chooses to change them, then execution continues from step 2 above; otherwise the game board is cleared and execution continues from step 3 above.

3.1 Importing the given classes into the workspace

To be able to use the given classes you should perform few short steps that will allow your code to link with the supplied code. Users of Eclipse IDE should follow these steps:

- Download mastermind.jar from the moodle.
- Right click on your project in the "Package Explorer" package.

- Go to "Java Build Path" option, and choose the *Libraries* tab.
- Choose "Add external Jars" and point to the *jar* file you have just downloaded.
- Add the following line to the top of your Mastermind.java file:
  import il.ac.huji.cs.oop.mastermind.*;
  Now you can start working with the supplied objects.

4 Suggested guidelines

1. Start early!
2. Read the description of the game (Mastermind or Bulls and cows) to remind you how it works.
3. Next, run the school solution to see what your program should look like to the user.
4. Acquaint yourself with the provided APIs.
5. Write the code for a single turn: asking the user for a guess, and responding to the provided guess. Debug.
6. Extend the above code to a full single game. Debug.
7. Extend the above code to a sequence of games. Debug.
8. Add code for gathering and reporting performance statistics. Debug.
9. Run testers. You guessed it: debug!
10. Enjoy the rest of your week!

5 School Solution

An executable version of the school solution can be invoked from the lab computers by typing:
$\sim$ *oop/bin/ex1/mastermind*
on the shell command line.

6 Grading

The grade for this exercise is based on both automatic testing and on code review (equivalent weight). 90% of the tests are given to you and are presented in the next section. The rest of the tests are hidden and will tackle more subtle aspects of your code. Thus, you are encouraged to test your code on more elaborated scenarios.

7 Testing and submission

- In the following days you'd be supplied with the majority of the automatic testers. Those testers will also be executed automatically on the moment of submission.
- You should create the following files:
  1. Mastermind.java
  2. README (as explained here)
- Create a JAR file named ex1.jar containing only these files by invoking the shell command:
  jar cvf ex1.jar Mastermind.java README
  The JAR file should not contain any other files.
- It is recommended to check your JAR file by copying it to an empty directory, and running the automatic testers by executing the command:
  $\sim$ $oop/students\_testers/ex1/ex1LocalTests.py$ $ex1.jar$ and verifying that all the tests pass successfully.

Good Luck !