ITI 1121. Introduction to Computing II
Winter 2016

Assignment 1
(Last modified on January 20, 2016)

Deadline: February 5th, 2016, 11:59 pm

[PDF]

Learning objectives

- Applying basic object oriented programming concepts
- Using arrays to store information
- Editing, compiling and running Java programs
- Raising awareness concerning the university policies for academic fraud

Background information

The Monty Hall problem has a long and interesting history. The version we are going to use is as follows: we have three doors. Behind one of these doors is a prize, and there is nothing behind the other two doors. The player selects one of the doors and walks up to it. At this point, the game host, who knows where the prize is, stops the player and opens one of two doors that were **not** chosen by the player. The prize is not behind the open door. We now have two closed doors: the one selected by the player and one other one. The game host gives the player the opportunity to chose the second closed door instead of the door that was initially selected.

The question is the following: **Should the player switch door?**

It should be noted that the game host's routine is part of the show, it is done **every time** regardless of the initial choice of door by the player. In addition, the host knows which door has the prize, and thus can always safely open another (empty) door. In other words, the host routine is not an indication that the player chose the good or the bad door.

Problem

For this assignment, you will simulate playing the Monty Hall game. Your simulation will be a fair representation of the actual game, with the probabilities of each situation respected. You will simulate both strategies: player changing door and player sticking to the original choice. By simulating a large number of games, you will see which of the two strategies is more interesting.

Implementation

The assignment has three classes, **Door**, **MontyHall** and **Statistics**. The class **MontyHall** will use the class **Door** and simulate a game. It will use the class **Statistics** to keep track of the results as it iterates through several games.

Q1: Simulating one game (25 marks)

As a first step, you will simply simulate one game when executing your program. You only need the classes **Door** and **MontyHall** for now.

One object of the class **Door** stores the following information about one of the door:

- Does it have the prize behind it?
- Is it open or closed?
- Was it selected by the player?

A detailed description of the class can be found here:

- JavaDoc Documentation
- Door.java

The class **MontyHall** implements the game's logic. It has a **main** method which simulates one Monty Hall game and prints the result.

Here are two sample runs:

```
> java MontyHall

The prize was behind door C
The player selected door C
The host opened door B
Switching strategy would have lost

> java MontyHall

The prize was behind door A
The player selected door C
The host opened door B
Switching strategy would have won
```

A detailed description of the class can be found here:

- JavaDoc Documentation
- MontyHall.java

Implement both classes and make sure that your program works correctly.

Q2: Simulating a series of games (25 marks)

Simulating one game at a time does not tell much about the best strategy. You are going to change the class **MontyHall** to simulate a large number of games. The results are going to be accumulated in an object of a class **Statistics** and displayed at the end of the run. The actual number of games to simulate is going to be a runtime input parameter.

An object of the class **Statistics** accumulates information about the results of each game. In addition to track the successes and failures of each strategy, it also keeps track of several other information (how often a door has the prize behind it, how often it is selected by the player etc.). These additional information are not important for the main question of which strategy is the best, but it will provide some reassurance that your program works properly.

A detailed description of the class can be found here:

- JavaDoc Documentation
- Statistics.java

The class **MontyHall** must be updated to take as input the number of games to simulate, and to run these games while accumulating statistics. The input can by provided from the command line, in which case the output will be displayed on the standard output. If no command line input is provided, then the user will be prompted for one (using **JOptionPane.showInputDialog**, and the output will also be provided in a dialog box (using **JOptionPane.showMessageDialog**).

An updated description of the class can be found here:

- JavaDoc Documentation
- MontyHall.java

Here is a sample run, with the input provided from the command line.

```
> java MontyHall 5

Number of games played: 5
Staying strategy won 2 games (40%)
Switching strategy won 3 games (60%)
Selected doors:
  door 1: 1 (20%)
  door 2: 3 (60%)
  door 3: 1 (20%)
Winning doors:
  door 1: 1 (20%)
  door 2: 1 (20%)
  door 3: 3 (60%)
Open doors:
  door 1: 2 (40%)
  door 2: 2 (40%)
  door 3: 1 (20%)
```

Implement the classes and test your code. Simulate an increasing number of games and see if you can see a trend. Which strategy seems to work best? Can you understand why now?

Check that all of the statistics seem to make sense, and correct your program if you see anything wrong.

Q3: Generating a CSV output to use in a spreadsheet (5 marks)

In order to visualize your statistics, you can generate a comma-separated values output, which you can then import into a spreadsheet such as Microsoft Excel or Google doc. Once in a spreadsheet, you can use its built-in tools to obtain various charts to visualize the statistics.

You will add a method **toCSV** to the class **Statistics** that returns a **String** object which has the following information (each **val** is replaced by the corresponding data):

```
Number of games,val
Number of doors,3
,Win,Loss
Staying strategy,val,val
Switching strategy,val,val
,Selected doors,Winning doors,Open doors
Door 1,val,val,val
Door 2,val,val,val
Door 3,val,val,val
```

The first line provides the number of games played. The second line the number of doors used (at this point, always 3). The third line is a simple header. The fourth line provides the number of wins and the number of losses for the strategy that doesn't switch door. The fifth line provides the number of wins and the number of losses for the strategy that does switch door. The sixth line is a simple header. The sevenths line provides the number of times the first door was selected by the player, the number of times it had the prize behind it and the number of times it was open by the host. Lines eight and nine provide the same information for the second and the third door.

A slightly updated description of the class can be found here:

- JavaDoc Documentation
- Statistics.java

The class **MontyHall** must be updated to call the new **toCSV** method. You will call this method when the number of games to run is provided through the command line. Here is a sample run:

```
> java MontyHall 5
```

```
Number of games,5
Number of doors,3
,Win,Loss
Staying strategy,3,2
Switching strategy,2,3
,Selected doors,Winning doors,Open doors
Door 1,3,1,1
Door 2,0,1,2
Door 3,2,3,2
```

An example of an Excel spreadsheet with the corresponding charts can be found here. Import your CSV output in the **Data** sheet, and the **Results** sheet will automatically graph out the results. Alternatively you can copy this Google document, or create your own solution.

Q4: Generalizing the game (40 marks)

Although the original game only uses three doors, it can be easily generalized to **n** doors. In this version, the player still selects one door at random, but the hosts then opens **n-2** (empty) doors before offering to player to switch door.

In this new version of your program, the user provides 2 inputs: the number of games to simulate and the number of doors to use. Previously, we had a fixed number of doors, so we could use 3 variables, one per door. This approach was not particularly great[1], but it was simple and possible. Now that we have a number of doors that changes between each run, we cannot have a set number of individual variables anymore.

The solution we are going to choose is to use an **array** of **Door** objects.

You need to update the class **MontyHall** to accept 2 values as input, and use a reference variable to an array of **Door** objects instead of the 3 instance variables. Several methods of the class must be updated accordingly. An updated description of the class can be found here (Warning: the method **openOtherDoor** has now been renamed **openOtherDoors**!):

- JavaDoc Documentation
- MontyHall.java

The class **Statistics** must be updated as well. Statistics must be kept for an array of Door objects. The method **updateStatistics** now receives a reference to the array as input. The methods **toString** and **toCSV** must also be updated to provide the statistics for all the doors. An updated description of the class can be found here:

- JavaDoc Documentation
- Statistics.java

You have to implement this new feature and experiment with the results. How are both strategies faring as the number of doors increases?

Bonus (10 marks)

There is a little trick in this assignment: when the show host opens the door, if the strategy followed is something like walking from left to right and open the first available door(s), then, in the 3 doors version a smart player could for example choose door B and observe which door the host opens. If door C is opened, then necessarily the prize is behind door A, otherwise the host would have open door A (on the other hand, if door A is open, then we do not learn anything unexpected). The same problem occurs with the **n** doors version.

You can have up to 10 bonus marks if you avoid this problem in both the 3 doors and the **n** doors versions.

Rules and regulation (5 marks)

Follow all the directives available on the assignment directives web page, and submit your assignment through the on-line submission system Blackboard Learn.

You must preferably do the assignment in teams of two, but you can also do the assignment individually. Pay attention to the directives and answer all the following questions.

You must use the provided template classes below.

[1] As you probably have seen when implementing methods doing the same thing on all the doors, or when trying to select randomly a door to open.

Academic fraud

This part of the assignment is meant to raise awareness concerning plagiarism and academic fraud. Please read the following documents.

- http://www.uottawa.ca/administration-and-governance/academic-regulation-14-other-important-information
- http://web5.uottawa.ca/mcs-smc/academicintegrity/home.php

Cases of plagiarism will be dealt with according to the university regulations.

By submitting this assignment, you acknowledge:

1. **having read the academic regulations regarding academic fraud, and**
2. **understanding the consequences of plagiarism**

Files

You must hand in a zip file containing the following files.

- A text file README.txt which contains the names of the two partners for the assignments, their student ids, section, and a short description of the assignment (one or two lines).
- For each question, in a separated directory called **Q1**, **Q2** etc.:
  - The source code of **all** your classes. Each directory must be self contained and have all the files.
  - The corresponding JavaDoc doc directory.
  - StudentInfo.java, properly completed and properly called from your main.

Suggestion

You can create an **experiment** which iterates the same (large) number of games with an increasing number of doors (say from 3 to 50 doors, or even better, use a range provided as a new input to your program) and map out the evolution of the success and failure rate of both strategies as the number of doors increases. You can create a CSV output with the values, e.g. a series of lines like this:

```
# of Doors,# of wins switch strategy,# of losses switch strategy, # of wins stay \\
                                                  strategy,# of losses stay strategy
```

that you can then input to a spreadsheet and graph out.

A Frequently Asked Questions (FAQ)

1. **"Are we allowed to have other methods than those outlined in each question?"**

   The rule is simple, you cannot add **public** methods or variables; this would affect the interface of the class. However, you can add **private** methods to help you solve a problem. In fact, usually you are encouraged to create such methods to improve the structure of your programs. In the case of this assignment, it may not be necessary (although it is still fine).

2. **"Should we handle the case where the input provided is incorrect?"**

   You can add some **sanity checks** to the **main** method of the class MontyHall if you want to. We will really handle error situations once we have seen the concept of exceptions.

3. **"Can you suggest resources to brush up my Java skills?"**

   Oracle has a number of Java tutorials.

   - Trail: Learning the Java Language

- The “Hello World!” Application

Here is a giant index for various topics:

- The Really Big Index

Last Modified: January 20, 2016