

“Monty Hall” curriculum segment

Run the *Monty Hall* Game Simulation

Launch your **BlueJ** programming system and open the Monty Hall project from [here](#).

- You should see three main classes: **Monty**, **Player**, and **Display**. A fourth class, **Simulation**, will be used later.
- The three main classes should roughly correspond to the class structure that you and your group worked on previously. But, more on that later. Let's get into the code!

We'll be focusing on the **Monty** class for today. This is the main entry point for the Monty Hall game. To run the game, you should use the **play()** method inside the **Monty** class. Let's do that before we look at the code.

Make a **Monty** instance in the usual way: right click (ctrl-click on a Macintosh) on the box representing the **Monty** class. From the context menu that appears, select **new Monty()**, and call it whatever you like. This will put an instance down at the bottom of the BlueJ frame, as usual. Right-click this instance to invoke the **play()** method.

Now try playing a round of the game.
Did you win?

Run the program several more times. Try to determine from your runs whether switching choices or staying with your initial choice gives you the best chance of winning the car.

Tally your results. Did you generally

- switch and win
- switch and lose
- stay and win
- stay and lose?

Post your results in this brainstorm like usual (clicking "Save your Response" to create a new post.) After your submitted, compare your results with everyone else.

Now that you've seen what happened when you and a bunch of other people tried switching and not switching, do you think that switching is a better strategy than not switching, or vice versa? If you were on the game show and had the chance to win a car, what would you do? Briefly explain your answer.

You might have noticed that when you did win Monty would congratulate you. Find the method where your Monty instance congratulates the player when they win. (*Hint: it is inside the Monty class. Double-click on the Monty box in BlueJ to look at the source code for that class*).

Select your answer from the list below:

- playResult
- placeCar
- revealGoat
- getPlayersFinalChoice
- describeResults

Modify the "Congratulations" Message

When the contestant wins the car, Monty should deliver a few words of congratulations.

Find the current text string for Monty's congratulations message. Then modify the statement to add some humor or panache.

And don't forget: **Test it!** Iterative testing is an essential part of good programming skills: when you make a change, even a small one, you need to check to see whether your change was successful.

Run the program several times until the contestant wins the car. Does Monty deliver your newly revised congratulations message?

As you know, one way to test your program is by running the methods by themselves; that is, without running the whole program. It is easier to see exactly what that method is doing, and isolate any problems.

Try to run the `describeResult` method without running the whole program. From your `Monty` instance, right-click and look for the method `describeResults()`.

Why can't you find it? Why is the `play()` method available, but not the `describeResults` method?

The important difference between the two methods is in their *method headers*, which describe, among other things, who gets to run the particular method. The method `describeResult` is set to `private`: it can only be called from within the class.

Change the `describeResult` method to be a `public` method. We'll run it by itself in the next activity.

You should have changed the `describeResult` method to be a public method.

Now call `describeResult` directly (instead of by executing Monty's `play` method). Does it do what you expect? Are some of the parts of the display incorrect? Explain what you think is happening, and compare with other students in the class.

When you play the game using the normal `play` method in the `Monty` class, the method `chooseInitialDoor` is called. This is a method of the `Player` class. Instead of just looking at what this method does, try to figure out how it gets called. On the self quiz below, check which method you think calls the `initialDoor` method.

Which `Monty` method calls the `Player initialDoor` method?

Select your answer from the list below:

- `playResult`
- `getPlayersInitialChoice`
- `placeCar`
- `describeResults`

Now that we know when the `chooseInitialDoor` method is called, we will investigate more about the flow of the program. Now try to figure out which of the `Monty` class methods are called immediately after `chooseInitialDoor` returns. There might be a few ways to figure this out. Let's see if you figured it out in the same way as other people in your class. In this brainstorm, explain how you figured which out which `Monty` class methods is called immediately after `chooseInitialDoor` returns.

Create a FIRST post that includes the following:

1. Compare the methods of the `Monty` class with your earlier UML work.
2. List methods you created that have no corresponding `Monty` methods
3. List any `Monty` methods that did not appear in your design.

Then create a SECOND post that examines another student's organization.

1. Discuss how this student's method names differ from your own.
2. Describe which method organization you think is better, and why. Your feedback here should be polite and constructive.

Examine the `otherDoor` method

Find the `otherDoor` method in the `Monty` class.

The `Monty.otherDoor` method is very short and cryptic, which makes it hard to understand. It does an important bit of work, however: depending on which doors have been chosen, it returns the "other door".

In the next couple of activities, we'll look into how it works.









Fill in the table for the otherDoor method

SchemeHandler

Fill in the table for the otherDoor method.

Supply an appropriate value in each box, so that the table represents all cases for the otherDoor method.

door1	door2	otherDoor		Correct?
1	2	3		
1	3	<input type="text"/>		
2	1	<input type="text"/>		
<input type="text"/>	<input type="text"/>	<input type="text"/>		
<input type="text"/>	<input type="text"/>	<input type="text"/>		
<input type="text"/>	<input type="text"/>	<input type="text"/>		

If you miss any of these, put an entry in your Extra Brain explaining the reason you missed it.

Take a look at the `otherDoor` method code. Once you figure out how it works, explain in your own words why the value returned by `otherDoor` is correct.

Help the player cheat

If you were a player, wouldn't it be nice to get the car everytime? Since we have control over the simulation, lets do it!

Monty keeps the location of the car in a instance variable. Right now, that variable is private. Monty is pretty smart: it probably isn't a good idea to let just anybody know that value. But, we can make it accessible to the `Player`.

1. Make the `doorWithCar` instance variable public instead of private. Poor Monty.
2. Modify the `Player` class to take advantage of this change. That is, so that the player always chooses the door with the car.

Now, play the game and test your change. Does it work? You won't get to choose the door the first time because of your change. But, it should be pretty obvious whether to switch or stay...

When making an instance variable or method, your choice of `public` or `private` can have repercussions on how your code is used!

Quiz

1.

Suppose the `chooseRandomly` method were recoded to take only a single argument, namely the door *not to be returned*. Thus `chooseRandomly (1)` should return either 2 or 3, `chooseRandomly (2)` should return 1 or 3, and `chooseRandomly (3)` should return 1 or 2. Supply a version of `chooseRandomly` that works as just described. Use as much of the existing code as possible.

2.

Modify all calls to `chooseRandomly` to use the one-argument version of `chooseRandomly` rather than the two-argument version. Your modified method(s) should contain as few statements as possible.