HW4: Agents and Spaces
CSE1102 Spring 2015
Jeffrey A. Meunier
University of Connecticut

1. Introduction
This project will have you create a small object-oriented application in Java. The project consists of just a few Java classes that work closely together.

This application is a kind of spatial simulator. You will be able to create *spaces* (or rooms) and connect the spaces together with *portals* (or doors). There will be a single *agent* (or person) that is able to move between spaces through the portals.

If a *model* is a representation of a system, then a *simulation* is the use of a model to study that which is being modeled. And if the simulation happens to be fun to use, we call it a game. You can call this project a game if you like, but it won't be much fun just yet.

This type of simulation is the basis for any program that does interactive fiction. Interactive fiction is the technical term for an adventure game. See https://en.wikipedia.org/wiki/Interactive_fiction.

Note that there are no graphics in this project. You'll add that in a subsequent project.

[NERD STUFF — NOT ON EXAM: The coolest part, and this is the part that I really don't want to tell you, is that this is actually a *finite state machine*. Read about it if you have time: https://en.wikipedia.org/wiki/Finite_state_machine. That means that you can use this simulator as a programming language interpreter, or even a compiler, or a program that controls a physical process like you'd find in a factory, or in an embedded computer in a car.]

2. Value
This program is worth a maximum of 20 points. See the grading rubric at the end for a breakdown of the values of different parts of this project.

3. Due Date
This project is due by 11:59PM on Sunday, Feb 22, 2015. A penalty of 20% per day will be deducted from your grade, starting at 12:00am. (i.e., at 12:00am the maximum grade you can get is 16 out of 20). See this link for additional information.

4. Objectives

The purpose of this assignment is to give you experience working with classes, methods, and instance member variables. The classes you create, and the methods in those classes, will interact closely with each other.

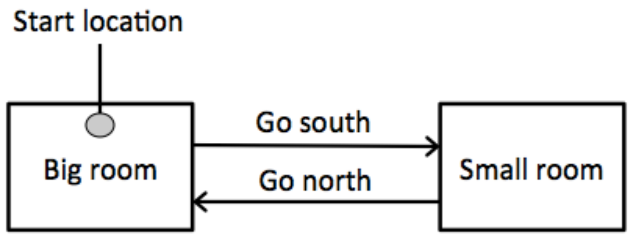
5. Background

Here are some Java constructs you will need to use:

- **do/while** loop (I didn't teach this to you, but it's in the book; this will be the only time this semester that you need to use a **do/while** loop)
- **this** (I'll spend time in lecture going over it, but it may help to start reading about it)
- instance variables and methods
- everything else you've learned so far, but probably not **for** loops
- no **static** member variables or methods, aside from the **main** method; they'll all be instance variables and methods

Description of the problem:

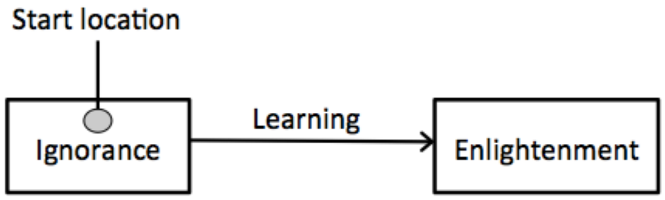
I have drawn a diagram to represent one kind of space I am interested in being able to simulate. There are two rooms connected by a door. It looks like this:



From this diagram you can see that:

- You start in the big room.
- From the big room it's possible to go south to the small room. Note that these descriptions and directions are conceptual, not necessarily literal.
- From the small room it's possible to go north to the big room.

I have drawn two arrows to represent the doorway, making movement in each direction explicit. At first I thought to have only one arrow going both directions, because most doorways let you go through them in two directions. However, I didn't want to restrict this program to just real physical spaces. For instance, this is a possible relationship:



The path from ignorance to enlightenment is through learning, and it's usually one-way. Perhaps there can be a return to ignorance, but it's not by following the same path that you followed from ignorance to enlightenment.

I have identified the following things that need to be modeled in the Java program:

- **Space**: This represents (is a model of) either a real or abstract space. Every space will contain a **Portal** that leads out of the **Space** to another **Space**.
- **Portal**: This models connections between spaces and allows a person to move between spaces.
- **Agent**: This models a person that can move between spaces. The **Space** that the **Agent** is in will be stored inside the **Agent** itself, and not the other way around.
- **Start location**: This is where the agent starts when the application begins running. You won't actually make a class for this. Instead, it will be the first space that you store inside the **Agent**.
- **Simulation**: This is the entire application that contains the spaces, portals, agents, and start location. This won't be an actual class, either, but you'll make something close to it.

6. Assignment

Read through all of the subsections in this section before you start to make sure you understand how you must proceed with the assignment.

Remember

- In this assignment please do make all member variables **private**, and **begin each member variable name with an underscore**. You don't have to like it, but you have to do it. This is the only time I will require it.
- Make all methods **public**.

6.1. Create a new Eclipse project

In Eclipse create a new project for HW4. Name it the same way you have been naming the other projects.

6.2. Space class

Summary

The core of this simulation is the spaces that make up the world of the simulation. In a subsequent project we will add items that can be placed in the spaces and picked up and carried by the agent.

Do this
Create a class called **Space**. It has three private member variables:

- **_name**: This should be a **String**.
- **_description**: This is also a **String**.
- **_portal**: This should be a **Portal**. You will create the **Portal** class after you finish the **Space** class, but if you want to make an empty **Portal** class right now, go ahead. It will keep Eclipse from complaining that the **Portal** class doesn't exist.

Notice that each space contains (rather, contains a reference to) only one **Portal**, and since a **Portal** is one-way and every **Portal** contains (a reference to) another **Space**, then this means that each **Space** has only one exit, and each exit leads to another **Space**. In a later project we'll change the **Space** class so that it can contain any number of **Portals**.

Write three getters (accessors), one for each of the three member variables, and write three setters (mutators).

Also add two more methods:

- **toString**: This method has no parameters and it returns a **String** . This method returns the string representation of a **Space** instance. There is no standard rule for what the string representation of any instance should be. In this method, just return the instance's name. Remember that a **toString** method must never display anything to the screen. It doesn't *display* a string, it *returns* a string. [See note below]
- **toStringLong**: This is like the **toString** method, but it returns a long description of the **Space** instance. To do this, build a string that contains the name and the description. Mine works like this: "classroom: a large lecture hall". Note that the name is "classroom" and the description is "a large lecture hall". So this method will combine both strings and insert a colon between them, then return the new combined string.

[**Note**: Yes, it's true. The **toString** method returns the same thing that the **getName** method returns. It's important to have both methods in this class. One is used when you want to know what the name of a **Space** is. The other is used when you want to (or more to the point, when Java wants to) see what the string representation of a **Space** is. It just so happens that both methods return the same information.]

Testing
Create a **Main** class with a **main** method. Test with these statements:

```
Space classroom = new Space();
classroom.setName("classroom");
classroom.setDescription("a large lecture hall");
System.out.println(classroom.toStringLong());;
```

My output looks like this:

```
classroom: a large lecture hall
```

6.3. Portal class

Summary

The portal represents a means to allow the agent to move from one space to another. Right now it will simply connect spaces together and contain some textual information that can be displayed. Later you will add a method that moves the agent.

Do this

Create this class with these private member variables:

- **_name**: a **String**
- **_direction**: a **String**
- **_destination**: a **Space**

Create getters and setters for all three variables.

Also write a **toString** method that returns the portal's name and direction. Say the portal's name is "door" and direction is "outside", then this method returns the string "door that goes outside". You have to insert the string: **" that goes "**. Note the spaces inside the string.

Write a **toStringLong** method that returns the same string as **toString** plus the string " to ", then the short description of the destination. Note that all these strings must be combined into one string, and that one final string must be returned from this method. Thus, if the destination's name is "sidewalk", then this method returns "door that goes outside to sidewalk". The destination's **getName** method and **toString** method return the same string, so you may use either one.

Modify Space class

Go back and change the **toStringLong** method in the **Space** class: If the space's portal is not equal to **null**, then add the string **" with a "** to the string, then add the result of calling the portal's **toStringLong** method. It may be simpler to see the example below.

Testing

Statements for the **main** method:

```
  Space classroom = new Space();
  classroom.setName("classroom");
  classroom.setDescription("a large lecture hall");
  Space sidewalk = new Space();
  sidewalk.setName("sidewalk");
  sidewalk.setDescription("a plain concrete sidewalk with weeds growing through the
cracks");
  Portal door = new Portal();
  door.setName("door");
  door.setDirection("outside");
  door.setDestination(sidewalk);
  classroom.setPortal(door);
  System.out.println(classroom.toStringLong());
```

Output:

```
  classroom: a large lecture hall with a door that goes outside to sidewalk
```

6.4. Agent class

Summary

An agent represents a person or other being who is interacting with the world created by the simulation. The user (you) can be an agent, or an agent can be a character in the world that is controlled by a program, sort of an artificial intelligence (we could in theory create an agent that makes simple decisions, but note that AI is *waaaay* beyond the scope of this course).

In this project there will be only one agent and it will be controlled by you, by entering commands at the console.

Do this

Create a public class called **Agent** with these member variables:

- **_location:** This is the current location of the agent. [See note below]
- **_name:** We would like to give the agent a name.

I'll let you determine what the correct types are for those variables.

Create getters and setters for both variables.

toString method: returns just the agent's name.

toStringLong method: returns the agent's name, followed by the string **" is in "**, followed by the agent's location. Use the short description of the location (meaning, call the **toString** method on the location and not the **toStringLong** method).

[**Note**: I understand that having the **Agent** class contain a reference to (I'll eventually stop saying "a reference to" and just leave it as "contains") a **Space** may seem confusing. In the real world, wouldn't a **Space** contain an **Agent** instead? Yes, it would, and when I first wrote this program, that's how I did it. However, there is always only one **Agent** in this program, so doing it this way the **Agent** always knows what space it's in. Otherwise, if the **Space** were to contain the **Agent** instead, how would you find out where the **Agent** is? You'd have to search through all the **Spaces** until you found the one that contained the **Agent**. Make sense? Great!]

Testing
The main method:

```
  Space classroom = new Space();
  classroom.setName("classroom");
  classroom.setDescription("a large lecture hall");
  Space sidewalk = new Space();
  sidewalk.setName("sidewalk");
  sidewalk.setDescription("a plain concrete sidewalk with weeds growing through the
cracks");
  Portal door = new Portal();
  door.setName("door");
  door.setDirection("outside");
  door.setDestination(sidewalk);
  classroom.setPortal(door);
  Agent student = new Agent();
  student.setName("Harry Potter");
  student.setLocation(classroom);
  System.out.println(student.toStringLong());
```

And the output:

```
  Harry Potter is in classroom
```

6.5. Portal.transport method

Summary

Now that we have the ability to create agents and we have spaces and portals, we need a way to move an agent from one space to another space through a portal. This gives the portal the responsibility of moving an agent. You place an agent in a portal, and the portal places the agent in the new space.

This method works sort of like this: *Hey, Space! You have a portal, right? Well, here's an agent. Send this agent through the portal.* And since the portal contains a space, the portal knows where to send the agent. Conceptually, this should be easy to grasp.

An agent is moved by changing the **Space** instance that it refers to in its **location** member variable to a new **Space** instance. In other words, the agent itself does not move. Instead, the space that it refers to changes.

Recall:

- A portal has a destination that is an instance of **Space**.
- An agent has a **setLocation** method that has a **Space** parameter. You just wrote this method in the previous section.

Thus:

- You should be able to move an agent to a new location by setting its location to be the portal's destination. Make sure that this makes sense to you.
- If we were to take the portal's **destination** (which is an instance of **Space**) and set the agent's **location** to be this destination using the **setLocation** method, it effectively moves the agent to this new location.

Do this

Create a **transport** method in the **Portal** class that has an **Agent** parameter. This method sets the agent's **location** to the portal's **destination**. This method does not return anything.

[Note: Even though the title of this section is **Portal.transport**, this must not be a static method. Using ***ClassName.methodName*** is a common way to refer to a method in a class (in a document, not in a program), even if it's an instance method. In fact, none of the methods in this project are static except for the **main** method.]

This is a very short method.

Note that this is the general idea in object-oriented programming: Here the **Portal** instance has the responsibility of moving an **Agent** instance through it to the destination space.

Testing

Add these two statements to the bottom of the **main** method:

```
door.transport(student);
System.out.println(student.toStringLong());
```

This is the output:

```
Harry Potter is in classroom
Harry Potter is in sidewalk
```

Yeah, the English usage is bad. We're not going to worry about that.

6.6. Agent.usePortal method

Summary

It would be nice if there were a single method in the **Agent** class to tell the agent to move through the portal in whatever space it's in. This gives the agent the responsibility of moving itself by using the **transport** method you just created in the **Portal** class.

This method works sort of like this: *Hey, Agent! You're in a space, right? Well, go use that space's portal.*

Do this

Add a method **usePortal** to the **Agent** class that has no parameters and a **void** return type.

This method gets the agent's location's portal, and checks to see if it's **null**.

- If the portal is not **null**, have the portal transport the agent using the portal's **transport** method. You will need to use the word **this** to refer to the agent that needs to be transported.
- If the portal is **null**, then don't do anything.

Now we're moving the transportation of an agent higher up: it's now the **Agent**'s responsibility to use a **Portal** to cause itself to move to another **Space**.

Testing

In the **main** method change this method call:

```
door.transport(student);
```

to this:

```
student.usePortal();
```

The output looks the same as before:

```
Harry Potter is in classroom
Harry Potter is in sidewalk
```

6.7. Create a command loop

Summary

Now you will create a class that will allow a user to interact with this simulation. The user will be able to enter simple commands to move the **Agent**, show information about the current **Space**, or quit the simulation.

Do this
Create a new class called **CommandInterpreter**.

Create a **public static void run** method that has an **Agent** parameter. The **run** method will use this agent as the main player of the game, err, I mean simulation.

This method will be a big loop that gets a command from the keyboard and does different things depending on what the user enters. For example, if the user enters **look**, a description of the space will be shown. If the user enters **quit**, the command loop exits. You'll add these commands in the next section. First you need to get the loop working.

This method has two local variables:

1. A **Scanner** instance so that the user can enter commands.
2. A **boolean** variable that will be used as the loop control variable, just like in the calculator project.

Now create a **do/while** loop that uses the loop control variable.

Inside the loop, display a simple prompt, like **==>** , and then get a string from the keyboard calling the **next()** method (not **nextLine()**) on the scanner.

If the string is equal to **quit**, set the loop control variable to **false** (see pages 62 – 63 for information on how to compare strings for equality). That will cause the loop to exit and the program to end. Otherwise, display some kind of error message. See my test run below for an example.

Change the **main** method to call the **CommandInterpreter.run()** method:

```
  public static void main(String[] args)
  {
    Space classroom = new Space();
    classroom.setName("classroom");
    classroom.setDescription("a large lecture hall");
    Space sidewalk = new Space();
    sidewalk.setName("sidewalk");
    sidewalk.setDescription("a plain concrete sidewalk with weeds growing through the
cracks");
    Portal door = new Portal();
    door.setName("door");
    door.setDirection("outside");
    door.setDestination(sidewalk);
    classroom.setPortal(door);
    Agent student = new Agent();
    student.setName("Harry Potter");
    student.setLocation(classroom);

    CommandInterpreter.run(student);
  }
```

Test it

The output should look like this:

```
==> help
Sorry, I don't understand 'help'
==> kwyjibo
Sorry, I don't understand 'kwyjibo'
==> quit
```

6.8. Fill in the command loop

Summary

Right now the command loop doesn't do much. It just lets the user enter **quit**. Here you will add a few more commands:

- **go**: Moves the agent through the space's portal.
- **help**: Displays a list of all the commands that can be used.
- **look**: Displays the long description of the agent's current location.
- **where**: Displays the short description of the agent's current location.

Do this

To code these command options, you will need to use a big **if/else if** ladder inside the **do/while** loop. All the code for each of these commands will be written right here in the **if/else if** statements. You do not need to create any more methods.

- **"help"**: The help message does not need to be fancy. Just display a string indicating what commands are available: go, help, look, quit, and where.
- **"where"**: This displays the agent's location's **toString** string. You know what agent to use, and you know how to get the agent's location.
- **"look"**: This displays the agent's location's **toStringLong** string.
- **"go"**: The agent must be moved through the current space's portal. All you need to do is call the **usePortal** method on the agent.

Resist the urge to use a **switch** statement (if you know what that is) in place of the **if/else** ladder. It won't work, owing to the way that Java compares strings. It may work in a newer version of Java, but don't assume that your TAs will be using a new version of Java. Use the **if/else** ladder. We'll discuss this at length later.

Test it

After adding these commands you should be able to move around and see where you are.

```
  ==> help
  go, help, look, quit, where
  ==> where
  classroom
  ==> look
  classroom: a large lecture hall with a door that goes outside to sidewalk
  ==> go
  ==> where
  sidewalk
  ==> look
  sidewalk: a plain concrete sidewalk with weeds growing through the cracks
  ==> quit
```

6.9. Add narration

Summary
Whenever an agent moves from one space to another using the **go** command, it happens silently. After the agent moves, the user must enter the command **where** or **look** in order to see where the agent is. It would be nice to have some information displayed on the screen automatically whenever the agent moves through a portal.

Do this
Modify the **Agent.usePortal** method so that it describes the transportation that is taking place. When the program runs it will look like this:

```
==> go
Harry Potter is moving from classroom to sidewalk
==>
```

A single **System.out.println** statement is sufficient.

6.10. Display initial location

Summary
One last change is to let the user know where the agent is right when the program starts running.

Do this
In the **CommandInterpreter.run** method, before the loop starts, display the agent's name and location.

```
Harry Potter is in classroom
==>
```

You must display:

- The agent's name
- The string **" is in "**
- The agent's location

6.11. Design your own spaces

Change the agent, spaces, and portals from what I have shown you to something of your own design. Use at least three unique spaces and two portals connecting them. It must be possible for the agent to visit all the portals.

You can model any of the following:

- A real place like your home or your dorm room.
- An fictional place like Hobbiton and Mordor (does one simply walk into Mordor?), or Hogwarts or the Death Star.
- An imaginary place that you create.
- A physiological process like going to sleep and waking up.
- An abstract or physical process like one you read about on the FSM web page.
- Anything else

Whatever you use, make at least three spaces and two portals.

Note that it is possible to create a loop of spaces. To do this, place a portal in the last space and set its destination to be any one of the other spaces.

You must draw a diagram of your spaces and portals to include in your document. See the next section.

7. Report

Create a Microsoft Word or OpenOffice Word file (or some other format that your TA agrees on -- ask him or her if you are not sure). Save the file with a .doc or .docx format.

At the beginning of the document include this information:

Project title goes here
CSE1102 Project *project-number*, *semester*
Your name goes here
The current date goes here
TA: *Your TA's name goes here*
Section: *Your section number goes here*

Instructor: Jeffrey A. Meunier

Be sure to replace the parts that are underlined above.

Now create the following three sections in your document.

1. **Introduction**
   In this section copy & paste the text from the introduction section of this assignment. (It's not plagiarism if you have permission to copy something. I give you permission.)
2. **Space diagram**
   Draw a sketch of your spaces and portals similar to the one I showed in the Background section of this document, and include that sketch here. I don't care if it's drawn using the word processor, or some other drawing program (Google Docs has a good drawing program, then take a screen shot or download it), or if it's a hand-drawn picture and you take a photo of it.
3. **Output**
   Run your program and copy & paste the output from the command window here. You do not need to write anything. Show at least the results of using each of the commands that the command interpreter handles.
4. **Source code**
   Copy & paste the contents of your Java file(s) here. You do not need to write anything.

8. Submission

Submit the following things things on HuskyCT:

1. Your exported Eclipse project.
2. The report document.

If for some reason you are not able to submit your files on HuskyCT, email your TA before the deadline. Attach your files to the email.

9. Grading Rubric

Your TA will grade your assignment based on these criteria:

- (2 points) The comment block at the beginning of the Main class file is correct.
- (2 points) You created unique spaces, portals, and agent.
- (10 points) The program generates the correct output, and does it in the correct way.
- (2 points) The program is formatted neatly; member variables begin with underscores.
- (4 points) The document contains all the correct information (including the diagram you drew) and is formatted neatly.

10. Getting help

Start your project early, because you will probably not be able to get help in the last few hours before the project is due.

- If you need help, send e-mail to your TA immediately.
- Go to the office hours for (preferably) your TA or any other TA. I suggest you seeing your own TA first because your TA will know you better. However, don't let that stop you from seeing any TA for help.
- Send e-mail to Jeff.
- Go to Jeff's office hours.