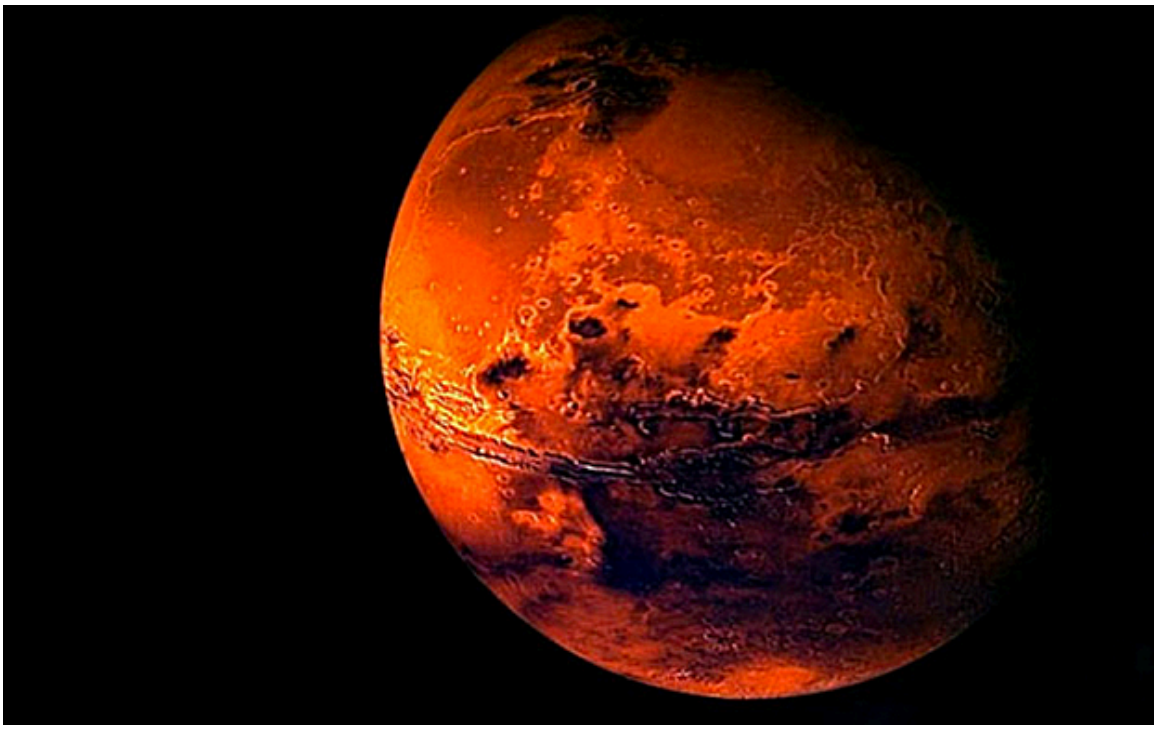
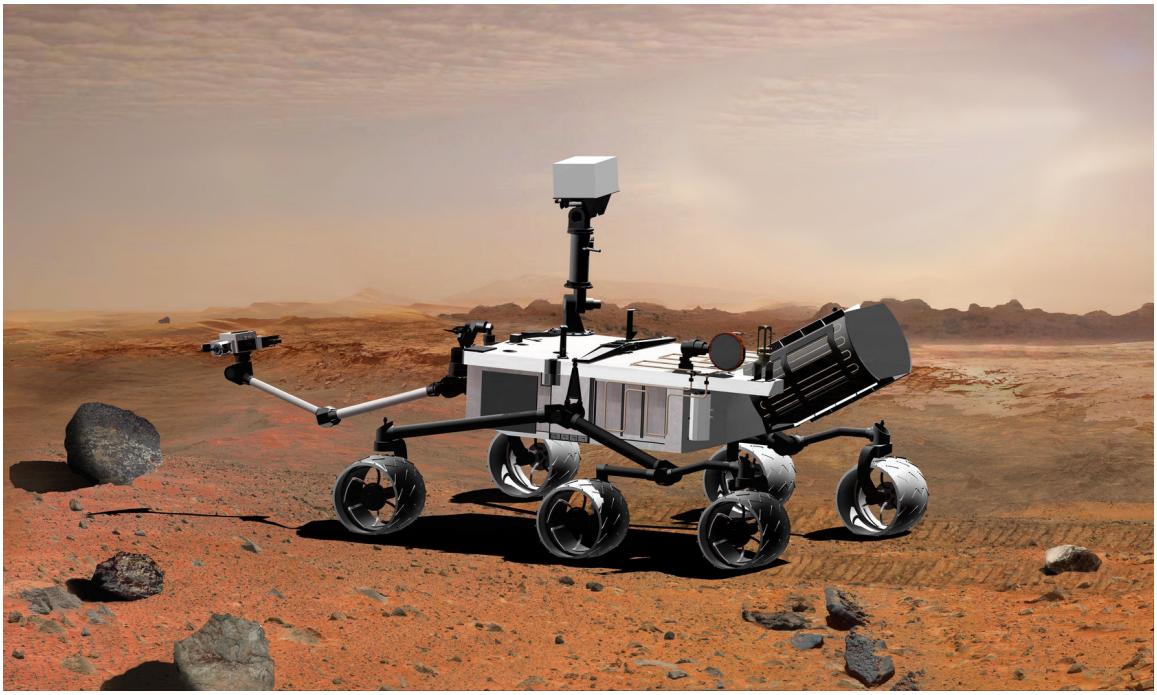
CS 46B
Lab 2: September 13& 16, 2016





In this lab you will work with subclasses and superclasses, and will get experience with the Eclipse debugger.

Work in pairs. One of you will be the “driver”, who types into Eclipse. The other is the “scribe”. Alternate jobs, week by week.

You will both submit lab reports.

Driver: At the top of your lab report write

CS 46B Lab Report
Your name
I was the driver, XxxxYyyy was the scribe.

Scribe: At the top of your lab report write

CS 46B Lab Report
Your name
I was the scribe, XxxxYyyy was the driver.

As you work through the lab assignment, instructions/questions highlighted in yellow should be discussed by both of you and addressed by the driver in the driver’s report; instructions/questions highlighted in blue should be discussed by both of you and addressed by the scribe in the scribe’s report.

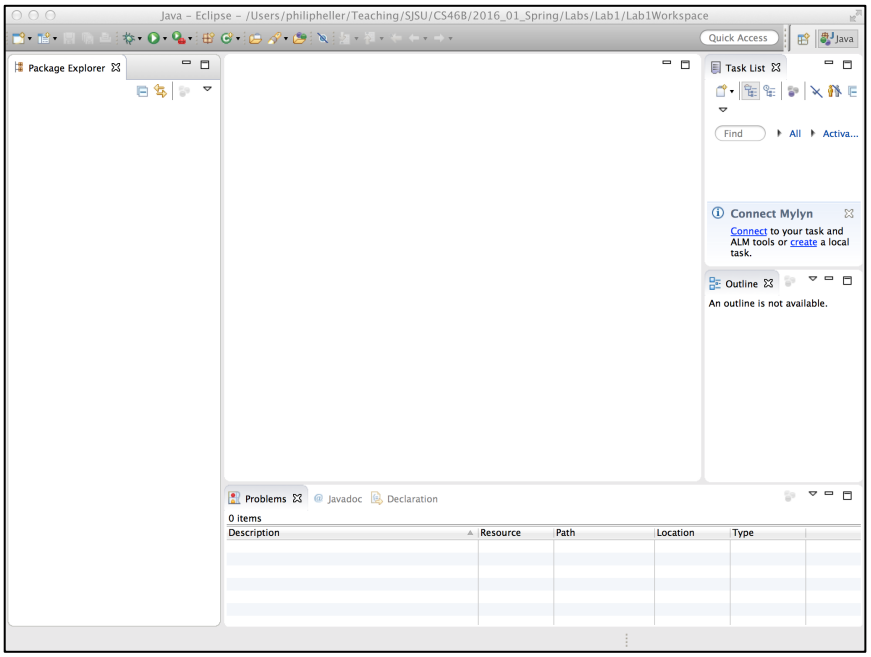
This week’s assignment is to build and play with a 4-level class hierarchy: DamagedRover, which extends MarsRover, which extends UnmannedVehicle, which extends Vehicle.

STEP 1: Create an Eclipse workspace directory anywhere you want on the driver’s computer. Call it “lab2workspace”.

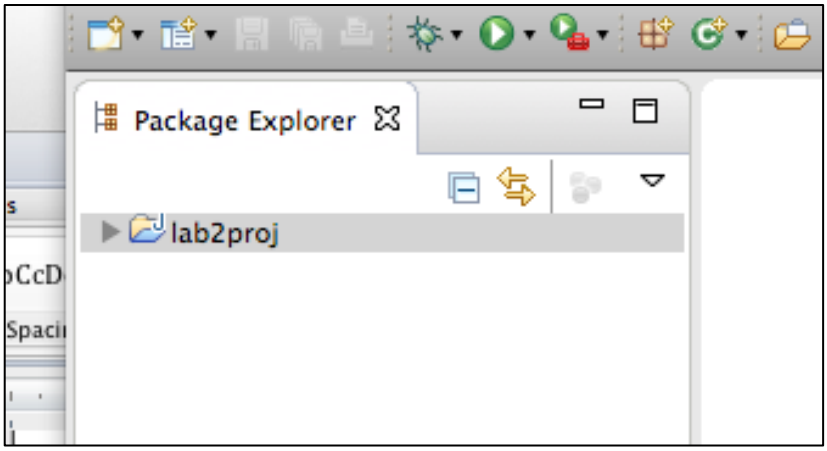
STEP 2: Start Eclipse. If it prompts you to “Select a Workspace”, browse to lab1workspace. Otherwise, it might come up in the last workspace you were in; in this case select File -> Switch Workspace and browse to the "lab2workspace" you created.

If you see the Welcome screen below, click the arrow circled in red.

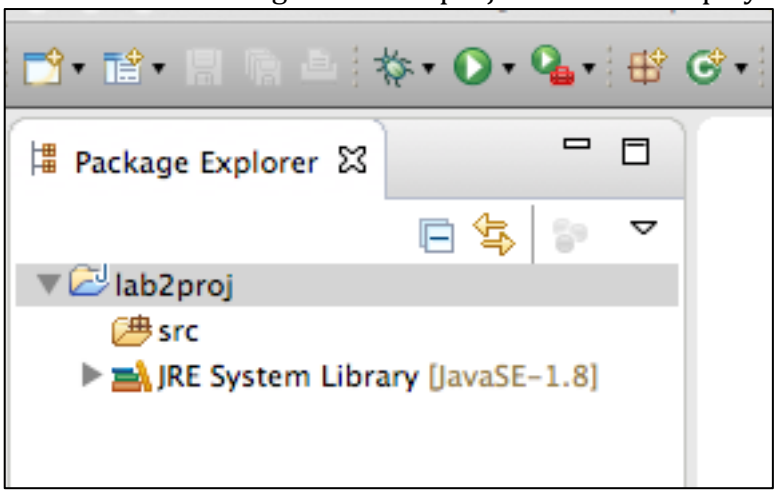
The Workbench screen (see below) is your main work area. The screenshot is from a Mac; appearance on other machines will be different. The workbench contains lots of panels that you don't need. If you find that they just get in your way, you can get rid of everything except the Package Explorer on the left and the big empty space in the center that will contain source file editors.



STEP 3: Create an Eclipse project called lab2proj. In the File menu, select New -> Java Project. A wizard will pop up. Type in the project name, leave all other settings as they are, and select Finish. The Package Explorer will display your new project:



Click the small triangle near the project name to display the project's contents:



If you see something like the figure above:

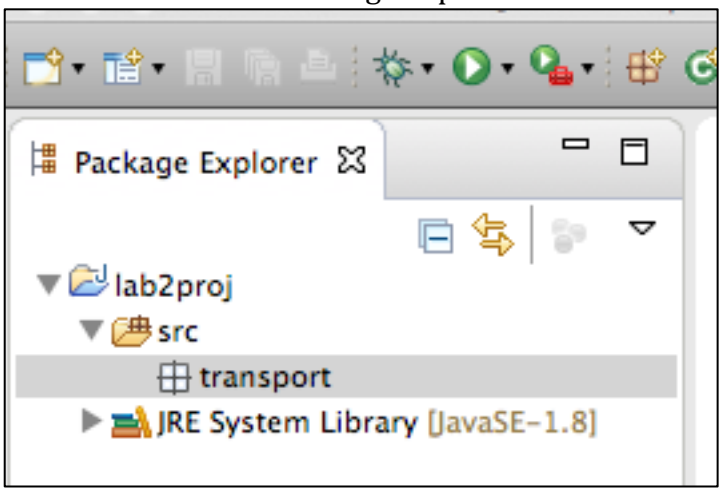
Scribe, write "Opened project".

If you don't:

Scribe, write "Tried to open project" and describe what you see.

Driver, write "Tried to open project" and insert a screenshot of the top of the Package Explorer panel.

STEP 4: Create a package called transport. Right-click on the small icon next to "src" in the Package Explorer, and select New -> Package in the popup menu. In the wizard that appears, "Source folder" should be "lab2proj/src". Type "transport" into the "Name" field and click Finish. The Package Explorer should now look like this:



If you see something like the figure above:

Scribe: write "Created package".

If you don't:

Scribe: write "Tried to create package" and describe what you see.

Driver: write "Tried to create package" and insert a screenshot of the top of the Package Explorer panel.

Both: fix it!

STEP 5: Create class "Vehicle" in the transport package. Right-click the icon next to "transport" and select New -> class in the popup menu. In the wizard that appears, type in the class name, leave all other settings alone, and click "Finish". A source code template will appear in the large central editor area of Eclipse.

Scribe: what source code elements did Eclipse automatically create?

Add the following main method, which obviously won't compile:

```
public static void main(String[] args) { xxx }
```

Scribe:describe in words how Eclipse says that you have a compiler error.

Driver: insert a screenshot of the error indication.

Hover your mouse cursor over the error indication and wait for popup information.

Scribe: what pops up?

Delete the main method.

Scribe: There is no constructor code in Vehicle.java. If you added the following to Vehicle, would you get a compiler error? Why or why not? (Don't actually type in the code, just think & discuss).

```
public static void main(String[] args) { Vehicle v = new Vehicle(); }
```

Now type in the main method above. Were you right about getting a compiler error? If you were right:

Scribe, write "We were right about the compiler error."

If you were wrong:

Scribe, write "We were wrong about the compiler error." Discuss, figure out why, and briefly explain.

STEP 6: Delete your main method and replace it with this:

```
public static void main(String[] args) {
  int i = 10;
  System.out.println(i);
  i = i + 100;
  System.out.println(i);
  i = i + 1000;
  System.out.println(i);
}
```

To execute this, click on the little Run button at the top of Eclipse. It's a white triangle in a green circle. Notice where the output appears.

STEP 7: If Eclipse isn't displaying source code line numbers, right-click in the margin to the left of the code and select "Show Line Numbers". Scribe, write "Showing line numbers.

Create a breakpoint at the line "i = i + 100;". Right click on the line number for that line, and select "Toggle breakpoint". What do you see? Scribe, write a sentence.Driver, insert a screenshot.

Execute up to the breakpoint. Instead of clicking the Run button, click the Debug button, which looks like a green bug and is near the Run button. You will probably get a "Confirm Perspective Switch" dialog. Select "Remember my decision" and then click "Yes". Your Eclipse will rearrange itself. The JVM has executed your code up to the breakpoint that you set. The current line is marked in the main source code window. Hover your mouse cursor over any occurrence of the variable i in that window. Scribe, what happens?

Click the single-step button:  Again, hover your mouse cursor over any occurrence of the variable i in the source window. Scribe, now what is i?

Run the app to completion by clicking this button: 

Change from the Debug Perspective to the Java Perspective by clicking this button, which is probably in the upper-right corner: 

Delete the main method.

You now have basic Eclipse survival skills. During the semester, it is your responsibility to get familiar with more features of Eclipse. When you discover something cool, post it on Piazza.

Scribe: write "I understand that it is my responsibility to get familiar with more features of Eclipse."
Driver: write "I understand that it is my responsibility to get familiar with more features of Eclipse."

STEP 8: Create 2 more classes in the transport package. UnmannedVehicle is a subclass of Vehicle. MarsRover is a subclass of UnmannedVehicle. To specify the superclasses, you can either type "extends …" in the source code, or you can configure the "New Java Class" wizard. I find that typing "extends …" is easier. Scribe, write "Created 2 more classes".

Add a private int nWheels to Vehicle. Add a Vehicle constructor that takes one int arg called nWheels. The constructor should store its arg in the instance variable nWheels.

Look at the code for UnmannedVehicle. It now has an error. Hover the mouse cursor over the error. Scribe, type "UnmannedVehicle error = " and the error message.

If you comment out the Vehicle constructor, what happens to the error message in UnmannedVehicle? Do you understand what causes the error? Discuss. Scribe, write a simple explanation.

STEP 9: With the Vehicle constructor in place (and not commented out), add a no-args constructor to UnmannedVehicle that explicitly calls the Vehicle constructor, with an argument of 4 to specify 4 wheels. Do you have any compiler errors now? Scribe, write "Explicit UnmannedVehicle constructor => compiler errors" or "Explicit UnmannedVehicle constructor => no compiler errors"

Add a no-args constructor to MarsRover that prints out "MarsRoverctor". ("Ctor" is an abbreviation of "Constructor"). Add a line to the UnmannedVehiclector that prints out "UnmannedVehiclector". Can that line be the first line of the UnmannedVehiclector? If yes: Scribe, write "UnmannedVehiclector: 1st line ok"; if no: Scribe, write "UnmannedVehiclector: 1st line not ok". Add a line to the Vehicle ctor that prints out "Vehicle ctor".

Now every ctor prints out an identifying line. If a MarsRover is constructed, in what order will the lines be printed out. Discuss. Scribe, write "The order will be:" and the three lines in expected order.

STEP 10: Add a main method to MarsRover with just this line:
MarsRover mr = new MarsRover();
Execute this main. Driver, paste the output into your report. If it's not what you expected, figure out why.

STEP 11: Congratulations! You got a job with NASA, writing Java software to simulate a rover in the hostile environment of the Martian surface.

On your first day at work, you learn that a huge Martian dust storm has blown the rover to the top of a mountain and damaged its motor and steering mechanism. The rover is safe for now, but if it travels 1000 meters in any direction it will fall of a cliff and be destroyed. It can only travel forward or backward, in little steps of exactly 1, 2, 3, or 4 meters. The distance and direction of each step is random. The rover's battery has enough energy for 10 km of travel.

It seems like it's just a matter of time before poor little Mars Rover falls off a cliff. NASA wants to know how far the rover will travel back and forth before that happens. You can't know this precisely because the rover is controlled by random processes, but you can simulate those processes, run the simulation lots of times, and compute the average travel distance.

In package "transport" create a class called "DamagedRover" which extends MarsRover. At the beginning of the class, define 2 constants:

```
private final static int  MAX_TRAVEL_METERS_BEFORE_EMPTY_BATTERY  = ????;
private final static int  METERS_FROM_START_TO_CLIFF              = ????;
```

Set the values of these ints. Also add the following line:

```
private final static int  N_SIMULATIONS                           =  5000;
```

Follow the Java convention for constants by putting these lines right after the open-curly that starts the class definition. Then add these variable definitions:

```
private double            position;
private double            metersTraveled;
private boolean           fell;
```

Paste and complete the following method:

```
//
// Simulates travel under damage conditions. In each turn, travels forward or
// backward either 1, 2, 3, or 4 meters. Continues until there's no more power
// in the battery, or we fall off a cliff. Cliffs are at position = 1000 or
// position = -1000.
//
public void simulateStormDamageTravel()
{
  position = 0;
  while (metersTraveled< MAX_TRAVEL_METERS_BEFORE_EMPTY_BATTERY)
  {
    doubledistanceNextTurn = (int)(1 + 4*Math.random());
    booleanforwardNotBack = (Math.random() > 0.5);
    // Adjust position and metersTraveled.

    // Check for falling off cliff.
  }
}
```

Add accessor methods `public double getMetersTraveled()` and `public boolean getFell()`.

Add a `main()` method that constructs an instance of DamagedRover, and calls `simulateStormDamageTravel()`. If the rover fell off the cliff, print "Fell" and the number of meters the rover traveled before falling. If the rover ran out of power without falling, print "Out of power".

Run your `main()` method. Driver: paste the output.

STEP 12: Now rewrite the `main()` method so that it runs the simulation 5000 times. Count the number of times the rover falls off the cliff, and keep a total of the distance traveled before falling. If the output messages from the constructors are irritating, you can delete them. After all 5000 simulations, print out the average distance traveled before falling. Driver: paste the output.

STEP 13: If you saw "NaN" in the output from Step 12, ask your lab instructor what it means. Set an eclipse breakpoint just before you compute the average distance traveled, and run using the "debug" button. When Eclipse hits the breakpoint and displays the Debug Perspective, you'll see a panel called "Variables". The variable values will give a clue as to why the output was "NaN".

Driver: paste a screenshot of the "Variables" panel.

Scribe: Why did your output say "NaN"? Is it due to a bug in your code? Recall that NASA wants to know how far the rover will travel on average before it falls off the cliff. What do you tell NASA?