JOURNEY TO MARS

There and Back Again

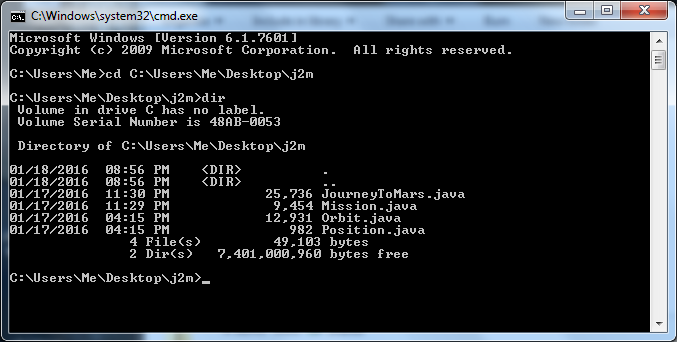
This assignment will introduce you to how software projects are (often) run in the *real* world. Specifically, you will leverage your current programming skills to understand, troubleshoot, and modify an existing piece of software (ah, yes…the joys of working with someone *else’s* code!). Over the course of this assignment, you will be given a series of tasks. Many of these tasks will originate from requests by the *end user*. As you will soon see, the *end user* is mostly concerned with “the what”, and less so with “the how” (or “the why”). Hence, many of these tasks will lack sufficient technical detail, and it will be up to you to fill in the gaps. Take your time, *think* before you code, and don’t be afraid to ask questions. Good luck!

**Level 1**

Congratulations! Fresh out of college, computer science degree in hand, you have landed your dream job at the NASA Jet Propulsion Laboratory (JPL) in *sunny* Pasadena, CA. Not that you would know much about the *sun*, though. Since your arrival two weeks ago, you have been working in a dark, window-less, basement-level “cube farm”. Signs of life are scarce so far, but they did place you close enough to the restrooms to occasionally hear the toilets flush. You know there must be *someone* out there…

Your suspicions are confirmed one morning when your boss suddenly peers above your cubicle wall, throws over a stack of hand-written notes and a floppy disk (yes, a floppy!), and makes a hasty retreat (back to a bright, cheery corner office, no doubt!). You hear him in the distance, like a passing train, bellow, “Looks like we’re sending humans to Mars…again! Review this orbit calculator from 2004 and have an SDD on my desk by COB today!”

SDD? COB? **SMH!!!!** A quick Google search reveals that an SDD is a “Software Design Document” and COB means “Close of Business”. Uh-oh, there’s not much time! You’d better get crackin’! Not surprisingly, your workstation does *not* have a floppy drive; but you find an old clunker in the abandoned cube next to yours. You brush the cobwebs aside, boot up the old dinosaur, and *feed* it the floppy. This is what you see:



**Exercise 1A:** Review the four Java files shown in the previous figure. In the space below, list each associated class (e.g. *JourneyToMars*, *Mission*, etc.) and include one sentence that describes what function that class performs. *HINT: Don’t waste too much time trying to understand everything about a given class. Look at the comments included in each Java file to get a general idea of what the class does.*

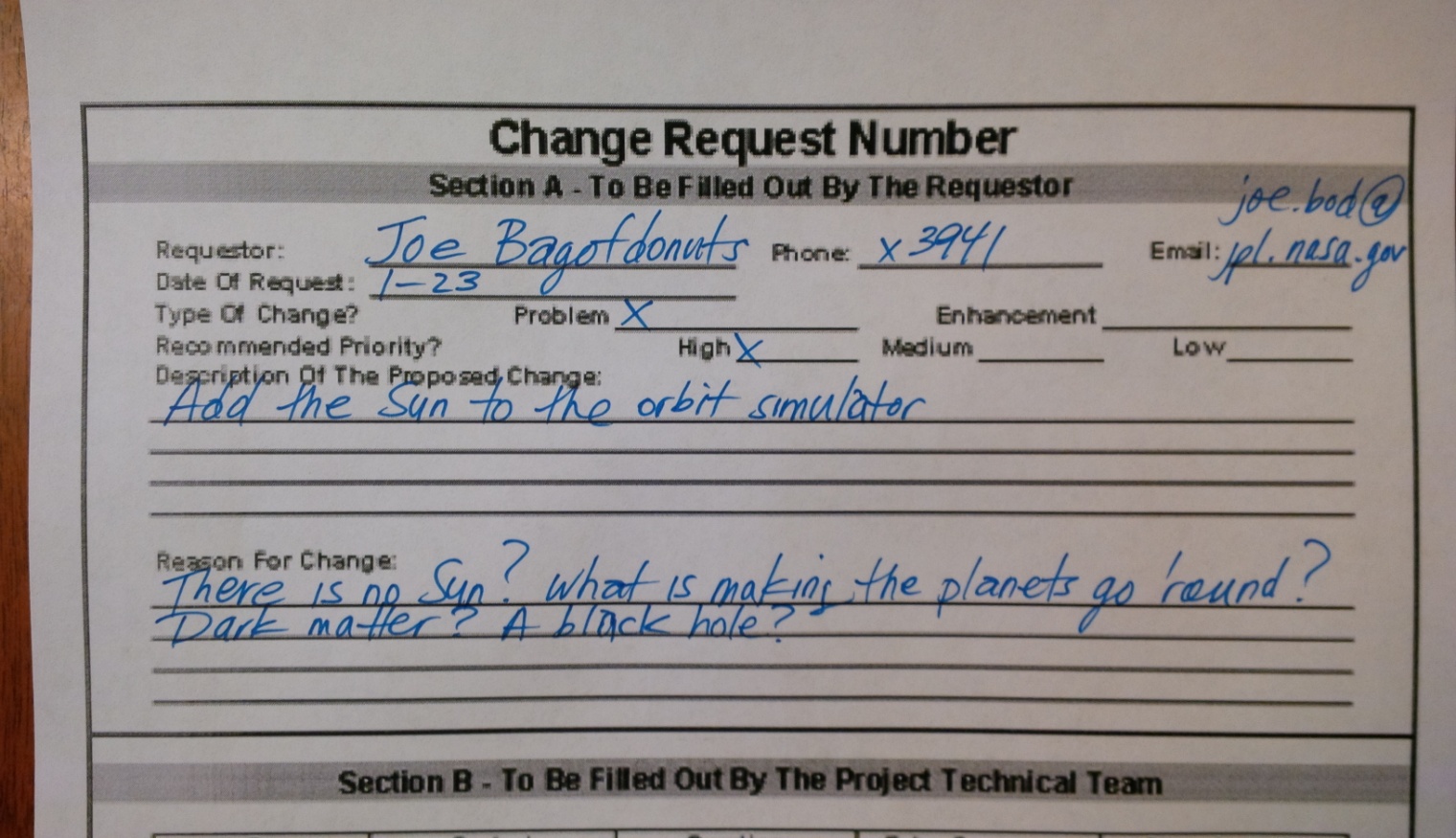
**Exercise 1B**: Draw a simple diagram that illustrates how the four classes are related. For instance if Class A creates an instance of Class B (e.g., *ClassB classb = new ClassB()*), draw blocks that represent the instances of Class A and Class B. Label each block with the name of the given instance (e.g., *classb*) and draw a line between them. The resulting *class diagram* will resemble a family tree. *HINT: Use the occurrences of the “new” command in the four Java files to find connections between the four classes.*

**Exercise 1C**: Compile and run the *Journey To Mars* program. Give the program a *whirl* to get a feel for what it does. While much of it might look like *Greek* to you now, try to relate some of the program’s functions to your analysis in the previous two exercises. *HINT: The right and left arrow keys make the worlds go ‘round.*

**Level 2**

Just as the clock strikes quittin’ time, you place the SDD on your boss’s desk and breathe a sigh of relief. With a quick glance, he reviews your work, and then looks up with a smile. “Nice job!” he says. “You seem to have a good understanding of how this program works.” He points to a stack of papers on his desk and adds, “As you can see, I’ve got a bunch of whiney astrodynamicists on the third floor who won’t stop bugging me with change requests. To tell you the truth, I don’t have time for this. In fact, I could give a rat’s aaa…” Pausing, he looks at the pile of papers, then turns back to you with a grin, “…ssso, I am appointing ***you*** team lead on this project! I need the final version of the software delivered to the third floor engineering department no later than one week from today.” You stand dumbfounded in his office, and by the time you recover your senses enough to speak, your boss has left for the evening.

Over the next few days, you wait patiently in your cube for your *team* to arrive, but hear only the sounds of the ghostly toilets flushing. It’s quickly becoming clear that your “team” is a team of ***one***. If this project is going to get done, it’s going to be all up to you! You glance at the pile of software change requests (now) on your desk and grab the one on the top…



**Exercise 2A:** Read this change request carefully (try your best to ignore the smart-aleck comments of the astrodynamicists who submitted it). Compile and run the *Journey To Mars* program and see if you can form a preliminary understanding of how this request applies. Proceed to Exercise 2B.

**Exercise 2B:** Add “the Sun” to the center of the orbit animation window (top right) by updating the *drawSun* function of the *Mission* class. Use the *fillOval* (java.awt.Graphics)function. Be creative, but choose a color scheme and an overall size that is appropriate (you want to ensure that the end user *actually* recognizes it as “the Sun”). *HINT: The overall pixel dimensions of the orbit animation window are given by the Mission class variables: width and height.*

**Level 3**

Storyline goes here…

The *day* variableis a member of the *Mission* class. This variable determines the current positions of all orbiting objects (Earth, Mars, and spaceship).

**Exercise 3A:** When the user clicks the “Reset Day” button, the current *day* should be set to 0. Update the *resetDay* function of the *JourneyToMars* class to enable this functionality. *HINT: Use the setDay function of the Mission class to set the value of day.*

**Exercise 3B:** Update the *incrementDay* function of the *Mission* class to increment the *day* variable by the number of days specified by the *dDay* parameter. Ensure that *day* can never be less than zero. *HINT: Use the setDay function of the Mission class to set the value of day.*

**Exercise 3C:** When the user clicks the “+” button, the current *day* should increase by 1. Update the *addDay* function of the *JourneyToMars* class to enable this functionality. *HINT: Use the incrementDay function you created in 3B.*

**Exercise 3C:** When the user clicks the “-” button, the current *day* should decrease by 1. Update the *minusDay* function of the *JourneyToMars* class to enable this functionality. *HINT: Use the incrementDay function you created 3B.*

**Level 4**

Storyline here…

**Exercise 4A:** Draw the orbits of Earth and Mars by updating the *drawOrbit* function of the *Orbit* class*.* In the *Orbit* class, the xy-coordinates of the given orbit are contained in the class variable *positions*. This variable is an array of type *Position*. The x and y coordinates of a given *Position* can be accessed using the *getX* and *getY* functions, respectively.

In this exercise, use the *drawPolyline* (java.awt.Graphics)function to draw the orbit. To use this function, you will need to build two integer arrays, *x* and *y*, using the xy-coordinates contained in the *positions* array. In addition, the xy-coordinates in *positions* will need to be converted to **pixel** values and centered in the animation window. This conversion is shown below:

x = xpositions \* *scale* + *xOffset*

y = ypositions \* *scale* + *yOffset*

where *scale*, *xOffset*, and *yOffset* are parameters passed to the *drawOrbit* function.

**Level 5**

Storyline here…

The leftmost window in the *Journey To Mars* orbit calculator provides useful information to the mission engineer, including orbit parameters (semi-major axis, eccentricity, period, etc.), velocities, energy requirements (called *delta V*), and trajectory times. Two values of particular interest when calculating an orbit connecting two solar system bodies (e.g. Earth and Mars) are their angular (dAng) and linear (dD) separations. These parameters are illustrated below:



**Exercise 5A:** Update the *getEarthMarsDistance* function of the *Mission* class to calculate and return the current distance between the two planets. *HINT: The current day of the mission can be obtained from the getDay function of the Mission class. The current position of a given planet can be obtained using the getPositionAt function of the Orbit class. The distance between two Positions can be calculated by the getDistanceTo function of the Position class.*

**Exercise 5B:** Update the *getEarthMarsAngle* function of the *Mission* class to calculate and return the angular distance (in degrees) between the two planets. This function should return the **absolute value** of the angular distance in the range from **0 to 180 degrees**. *HINT: The current day of the mission can be obtained from the getDay function of the Mission class. The current position of a given planet can be obtained using the getPositionAt function of the Orbit class.*