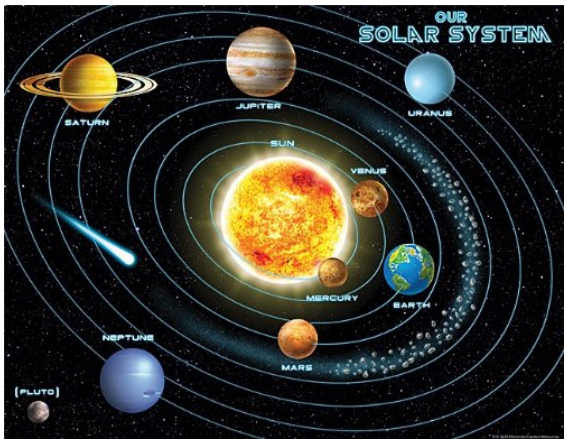
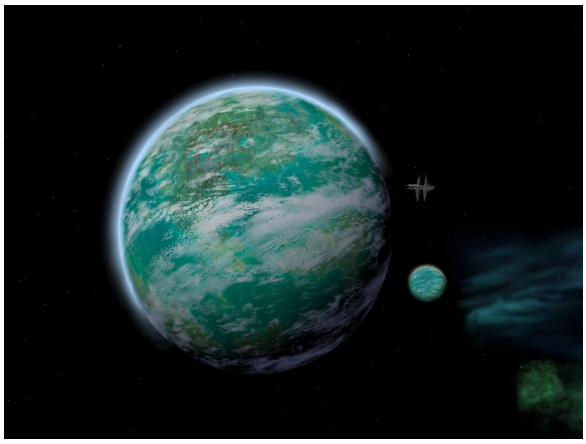
CS 46B
Lab 10
Space Missions





Congratulations! Your hard work this semester has reached the attention of the highest levels of government, and you and your lab partner have been appointed Chief Java Programmers of the United Nations Space Administration (UNSA). UNSA plans space exploration and diplomacy missions. Your first assignment is to create a custom data structure that will collect space missions and iterate them in an order that will be useful to your bosses.

As always: 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, Xxxx Yyyy was the scribe.

Scribe: At the top of your lab report write

CS 46B Lab Report
Your name
I was the scribe, Xxxx Yyyy 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.

Setup

Create an Eclipse workspace containing a Java project. In the project's src, create package "space". Copy/import the 2 starter files Mission.java and MissionOrganizer.java into the space package. Complete these classes as directed below.

Mission.java

This simple class contains the destination and cost of a space mission.

Complete 3 methods in this class:

- compareTo() – Compare by cost. If costs are equal, compare alphabetically by mission.
- equals() – Implement this just a few lines, 1 of which calls compareTo().
- hashCode() – Do this as you saw in lecture, by creating an ArrayList<Object>. Add both instance variables to the ArrayList, and return the hashCode of the ArrayList.

These Mission methods are simple, and it's tempting to move on without testing them if they compile correctly. If you think you should test them now, write a main() method that creates some instances of Mission, compares them, calls equals() on them, and computes

their hash codes. Scribe: Did you test this class or did you move on to the next part of the lab?

MissionOrganizer.java

This class will only ever contain Missions, so it doesn't need to be generic.

The class should be iterable, so that you can iterate over all its members in correct order like this:

```
for (Mission m: myMissionOrganizer)
     System.out.println(m);
```

Notice that the class' declaration says "implements Iterable<Mission>". The class will need a method `iterator()` that returns an iterator over the class' Missions. You saw something similar with the Roster class that we covered in lecture.

UNSA has a huge budget. Any mission that costs less than 1 trillion dollars (1.0 x 10^12) is considered cheap; any mission that costs more than that is expensive. Cheap missions should always appear first, in the order they were added to the MissionOrganizer. Expensive missions should always appear after all cheap missions, in increasing order by cost; if 2 expensive missions have the same cost, they should appear in alphabetical order by destination (note this is the natural sorting order of the Mission class).

The MissionOrganizer class should use an appropriate collection type to store its cheap missions and a different appropriate collection type to store its expensive missions. Scribe: considering the requirements for ordering of cheap missions, what is a good collection type for storing them? Considering the requirements for ordering of expensive missions, what is a good collection type for storing them? Declare these 2 instance variables, and initialize them in the ctor.

Complete the 3 unfinished methods of MissionOrganizer (contains, add, and iterator) as instructed by the comments in the starter file.

Below the iterator() method is an array of test missions, in chaotic order as you would expect from a large international organization. In main(), a TreeMissionOrganizer is constructed and populated from the test array, and then the organizer's missions are printed out in an enhanced for-loop. Scribe: Why is this possible? (Hint: Because class _____ implements interface _____). The correct output appears in a comment below main(). The costs are slightly different from the costs you see in the TEST_MISSIONS array because of float rounding. Don't worry about that.

Now add 4 missions to the end of TEST_MISSIONS: 2 cheap missions and 2 expensive ones. Get creative with the destinations. If you aren't feeling creative, Google "stars near earth". Driver: paste the new TEST_MISSIONS into your lab report. Before you rerun the app: what output do you expect? Driver: put your expected output into your report. Now run the app. Scribe: does the

output match what you expect? If not, was your expectation wrong or was your code wrong? If your output was wrong, fix your code.