Homework 2: Mutation
Due: 5:00 PM, Feb 14, 2020

Contents

1 How to Hand In 1
2 Testing 2
3 The Mystery Continues... 2
4 Tickets 2
5 Lookup 3
6 Mutation 3
7 Deletion 3
8 Copying 4
9 Reflection 4

Objectives

By the end of this homework, you will:

- Know when and how to mutate objects in Java!

1 How to Hand In

Begin by copying the stencil code from the course directory to your own personal directory. That is, copy the following files from /course/cs0180/sol/hw02/sol/*.java to ~/course/cs0180/workspace/javaproject/sol/hw02/sol:

- Ticket.java containing public class Ticket
- EmptyList.java containing public class EmptyList
- AbsList.java containing public abstract class AbsList
- IList.java containing public interface IList
• NodeList.java containing public class NodeList
• ListTest.java containing public class ListTest

After completing this assignment, you should be ready to turn in the following solution files:

• Ticket.java containing public class Ticket
• EmptyList.java containing public class EmptyList
• AbsList.java containing public abstract class AbsList
• IList.java containing public interface IList
• NodeList.java containing public class NodeList
• ListTest.java containing public class ListTest
• TicketMemoryLayout.jpg
• MutationQuestions.txt
• ShallowCopyLayout.jpg
• DeepCopyLayout.jpg

To hand in your files, submit them to Gradescope. Please make sure that MutationQuestions.txt is a plain text file, not a rich text file (you can check this by opening your submission in Gradescope and verifying that MutationQuestions.txt is readable). Once you have handed in your homework, you should receive an email, more or less immediately, confirming that fact. If you don’t receive this email, try handing in again, or ask the TAs what went wrong.

Note: If your code does not work with the autograder, you will be penalized. There is no compatibility script for this assignment, as we provide all of the method signatures for you. If Gradescope says that your code doesn’t work with our autograder, please contact the TAs via hours or Campuswire.

2 Testing

Please make sure to test your code! Testing and good code design make up a lot of your grade for this homework. Put your testing in the file ListTest.java.

3 The Mystery Continues...

It’s almost afternoon by the time you get out of class, and the CIT is absolutely buzzing — the CIT Planning Committee is throwing a Roaring Twenties CIT Formal to welcome the new decade and ticket sales opened on their website this morning! Even you, the resident CIT detective, are excited for the dance next week — so excited, in fact, that you’re considering buying a coveted Platinum-tier ticket.
You’re still mulling it over when CS 18 TA Lena, the head of the planning committee, bursts into
your office. She frantically explains that despite extensive testing beforehand, ticket purchases are
mysteriously disappearing. Could this be something more sinister? Or maybe they just didn’t test
as well as they thought . . .

Either way, you agree to help her — on the condition that you get a Platinum ticket at the end.
With that settled, you and Lena get to work on saving the CIT Formal!

4 Tickets

You remember that Prof. Fisler was just talking about mutation being able to cause problems
like this, so understanding how it works seems like a good starting point. Lena helps you get the
NodeList implementation Prof. Fisler used in class. A list of ints isn’t very helpful, though, so
the first thing to do is modify that. Lena also gives you the Ticket class, which contains two fields:
name and tier. There are three possible tiers: Platinum, Diamond, and Gold.

Task: Modify the IList, AbsList, EmptyList, and NodeList files so their elements are Ticket
objects instead of ints.

Task: Now, in the testLists method of ListTest, add more Ticket objects to list1 and list2
and draw a diagram representing the layout of these lists in memory. Note that you should only
have one diagram containing both lists. Upload a photo or scan of your diagram to Gradescope in a
file called TicketMemoryLayout.jpg.

Hint: Notice that both list1 and list2 already contain the Ticket lena.

5 Lookup

Now that you understand more about how lists and objects are laid out in memory, you and Lena
are ready to add more tools to successfully keep up with ticket orders. Lena asks you to modify the
NodeList and EmptyList classes with a variety of features to make the attendance system both
more effective and more difficult to sabotage.

Lena wants to be able to search the list for individual names, so she can easily deal with requests
from specific ticket buyers.

Task: Fill in findTicketByName. This method should take in the name of the ticket holder we
want to find and return the Ticket object associated with that name. If it doesn’t find a Ticket
that matches the input name, it should return throw a RuntimeException.

Note: You do not need to test thrown RuntimeExceptions for this homework.

6 Mutation

People often change their minds, so Lena also wants to give attendees the ability to change their
ticket tier using their name.

Task: Fill in updateTierOfTicket. This method should take in a name and a tier and update all
the Tickets with a given name to the new tier. For example, if you had bought two tickets and
used this method to update your tier, it would change both tickets in the list.

Task: Update lena’s tier to Diamond using the setTier method in the Ticket class and write test cases to check the tier value in both of the lists. Be sure to include these test cases in your ListTest .java file! Then, change the tier of lena back to Platinum by using updateTierofTicket on list2 and again write tests to check lena’s tier value in both of the lists and the object itself.

Task: What changes, if any, did you observe and why did they occur (or not occur)? Did changing one object affect the others? Why or why not? Put your answer to these questions, as well as the rest of the written questions labeled “Task” on this HW, in a plain text file calledMutationQuestions.txt and upload it to Gradescope along with the rest of your submission.

7 Deletion

Lena plans to use the attendee list to keep track of guests once they’ve checked into the Formal. Since different Ticket tiers arrive at different times, she wants to be able to remove entire tiers at once.

Task: Fill in removeTier. It should take in a tier and return a list that is the same as the original, but with all Ticket objects of the given tier removed.

Task: Write tests that use removeTier to remove all Platinum Tickets in list1. What happens to lena in list2? Write a sentence or two explaining this behavior.

Sometimes, people show up at the wrong time, or simply no longer want to attend the formal. As such, Lena realizes that removing entire tiers is not enough; she also wants to be able to remove individual tickets.

Task: Fill in equals in the Ticket class so that it returns true if the input object is a Ticket with the same name and tier as the Ticket that equals is called on.

Task: Using the new equals method that you wrote, fill in removeTicket so that it returns a list with all ticket objects equal to the input Ticket removed.

8 Copying

The Formal organizers are excited about your new ticket system, but before they roll it out to students they want to make sure that it works. However, given the volume of complaints they’ve already received about incorrectly deleted names, they want you to test a copy of the ticket list instead of the ticket list itself, so that they don’t accidentally end up breaking it even more.

You will be writing two methods to “clone” the list. One should be a shallow copy, in which you make a copy of the list but do not make copies of the Ticket objects in it; the other should be a deep copy, meaning that you make a copy of every individual Ticket object.

Task: Fill in shallowCopy so that it returns a new list with the same Ticket objects as the original. Next, fill in deepCopy so that it returns a new list with objects that are copies of the Ticket objects in the original.

Task: Draw a picture of the memory layout after calling shallowCopy on list1. Next, draw a picture of the memory layout after calling deepCopy on list1.
Upload photos or scans of your diagrams to Gradescope in files called ShallowCopyLayout.jpg and DeepCopyLayout.jpg, respectively.

9 Reflection

Task: Write responses to the following questions in MutationQuestions.txt:

1. You’ve now implemented a functional version of lists in both Java and one of the functional programming languages that you encountered in an earlier intro course. How does Java’s object mutation affect this homework’s implementation, as opposed to the implementations of lists that you have done in your earlier intro course?

2. What problems might this list implementation create for users used to list implementations written in programming languages that do not support mutation, and what principles did you learn in this homework that help you mitigate those problems?

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS18 document by filling out the anonymous feedback form: https://cs.brown.edu/courses/cs018/feedback