Introduction to Java

CS9053, Summer 2024

Assignment 5

Thursday 6:00 PM – 8:30 PM

Prof. Dean Christakos

June 22nd, 2024

Due: June 28th, 2024 11:59 PM

Part I: Exceptions

1. In the class ReadPersonellFile, there is a file called personnel.txt. I’ve opened it on line 21. This has a list of personnel for a university. You’re going to read each line for each shape and call createPerson, which will create one of the available Person objects, Faculty, Staff, Contractor, UndegraduateStudent, GraduateStudent, or SpecialStudent and returns a Person, which you will add to the ArrayList called personList.

Each entry is comma-separated and provides the arguments to the constructor for that class. For Staff managers and Graduate Student advisors, this is the name of an existing Person object that you should have already created, and you should be able to find that object you already created and pass it to the constructor. If an entry is blank, treat it as null (one of the Staff entries has no manager). Read up on String methods to figure out how to extract all the values from a comma-separated entry (hint: use the split method)

The file has some unavailable personnel. If the shape is unavailable, createPerson should throw a PersonException. You should catch a PersonException and continue reading the file.

By the end, the side of personList should be 20.

Summary:

* Create a PersonException class
* Implement createPerson to return the appropriate shape depending on the string shapeName
* createPerson should throw a PersonException if it is not a Faculty, Staff, Contractor, UndegraduateStudent, GraduateStudent, or SpecialStudent
* A loop should read in the personnel.txt file line-by-line
* If the file cannot be read, you should break out of the loop
* If you get a PersonException, you should continue reading the file

1. Take the following code, ListOfNumbers.java:

import java.io.\*;

import java.util.List;

import java.util.ArrayList;

public class ListOfNumbers {

private List list;

private String inFile;

public ListOfNumbers () {

// create an ArrayList of RDFTriples of Integers

}

public List getList() {

return this.list;

}

public void createList() {

for (int i = 0 ; i< 100 ; i++) {

Integer number1 = (int) (Math.*random*()\*10000);

Integer number2 = (int) (Math.*random*()\*10000);

Integer number3 = (int) (Math.*random*()\*10000);

// fill the existing list with RDFTriple objects

// of three numbers.

}

}

public ListOfNumbers (String inFile) {

this();

this.inFile = inFile;

}

public void readList() {

}

public void writeList() {

PrintWriter out = null;

try {

System.*out*.println("Entering try statement");

out = new PrintWriter(new FileWriter("outFile.txt"));

for (int i = 0; i < list.size(); i++)

out.println(list.get(i).getSubj() + " " + list.get(i).getPred() + “ “ + list.get(i).getObj());

} catch (IndexOutOfBoundsException e) {

System.*err*.println("Caught IndexOutOfBoundsException: " +

e.getMessage());

} catch (IOException e) {

System.*err*.println("Caught IOException: " + e.getMessage());

} finally {

if (out != null) {

System.*out*.println("Closing PrintWriter");

out.close();

} else {

System.*out*.println("PrintWriter not open");

}

}

}

}

You’re going to do a couple of things:

1. You can see the class “RDFTriple”. Now, this takes three Objects, a subject, a predicate, and an object. Like ArrayList, it’s parameterized. So you can have an RDFTriple with a subject of a String, a predicate of Integer, and an Object of a Circle, like RDFTriple<String, Integer, Circle>, or an RDFTriple of integers where the subject, predictate, and object are Integers, such as RDFTriple <Integer, Integer, Integer>. You would access each item of the RDF triple with getSubj(), getPred(), and getObj().

For example, I could create an RDFTriple of 5, 6, and 7 like so:

RDFTriple<Integer, Integer, Integer> t = new RDFTriple<Integer, Integer, Integer>(5,6, 7);

Here, t.getSubj() would be 5, t.getPred() would be 6, and t.getObj() would be 7

What you’re going to do first is have the field rdfTripleList be an ArrayList of RDFTriple objects, properly parameterized (there should be no warnings associated with ArrayList in the code).

Next, you’re going to implement createList. Currently in createList, you can see that it generates three random integers between 0 and 9999. You’re going to take each triple of integers and put them in an RDFTriple object, and then add that RDFTriple object to the ArrayList called rdfTripleList.

So at this point rdfTripleList should have 100 RDFTriple objects, where each object contains a Key and a Value of random integers. Once you’ve done this, the method writeList should compile correctly without errors (you shouldn’t have to modify that code directly for the errors to go away).

1. Add a readList method to ListOfNumbers.java. This method should re-initialize the rdfTripleList field with a new, empty ArrayList, read in int values from a file, print each value, put the triple of numbers in each line in a RDFTriple object, and append them to the end of rdfTripleList. You should catch all appropriate errors. You will read from the text file numberfile.txt.

Once again, you will have to read in a line and split up the data.

You will convert the number Strings to Integers and use those integers in the constructor to your RDFTriple object, and add the RDFTriple object to the ArrayList.

The writeList method writes out the contents of the ArrayList to rdfoutput.txt.