Introduction to Java

CS9053 Section I2

Wednesday 6:30 PM – 9:00 PM

Prof. Dean Christakos

February 23rd, 2023

Due: March 3rd, 2023 11:55 PM

Part I: Exceptions

1. Go back to the Vehicle Hierarchy code from Assignment 4. As long as you got the basics right from that assignment, you should be all set with the code you’ve already written. I’ve put in the basic employee hierarchy, but you should paste in your own code. Looking at the code, a few simple rules should be obvious:
   * Cargo space can never be negative
   * Cars cannot have fewer than 2 doors

But there’s also other rules from the previous problem set that specifies other rules.

Given this, if one tries to create an illegal Vehicle object (or its subclasses), then we should raise a VehicleException.

1. Create a VehicleException class
2. Ensure that the constructor accepts a String argument which is passed up to the Exception superclass
3. Throw a VehicleException when any of the three “illegal” conditions are raised.
4. The VehicleException should have a message explaining the error. (eg, “cargoSpace cannot be negative”)
5. 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 Car, like RDFTriple<String, Integer, Car>, or an RDFTriple of integers where they 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.

There’s a trick when reading in data that you want to split up. If you read in a line, it will contain two numbers separated by a space, and you will have a String that looks like “5 6 7”. Call it line, which is a String object. If you execute the method line.split(), it will return an array of Strings such that if you have String[] nums = line.split(), then nums[0] will be the String “5”, nums[1] will be the String “6”, and nums[2] will be the String “7”. Convert those 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 outFile.txt.

1. In the assignment project is a file called vehicles.txt. This has a list of vehicle types. Use the default constructors.

You’re going to read each line for each employee and call createVehicle, which will create one of the available vehicles, Car, Motorcycle, Bicycle, or CargoCycle and returns a Vehicle, which you will add to the ArrayList called vehicleList.

The file has some unavailable vehicle types. If the vehicle is unavailable, createVehicle should throw a VehicleException. You should catch a VehicleException and continue reading the file.

By the end, the size of vehicleList should be 20

Summary:

* Implement createVehicle to return the appropriate employee depending on the input string
* createVehicle should throw an VehicleException if it is not a Car, Bicycle, Motorcycle, or CargoCycle
* A loop should read in the vehicles.txt file line-by-line
* If the file cannot be read, you should break out of the loop
* If you get a VehicleException, you should continue reading the file