

Introduction to Java
CS9053 Section I
Thursday 6:00 PM – 8:30 PM
Prof. Dean Christakos
Feb 22nd, 2024
Due: Feb 29th, 2024 11:59 PM

Part I: Exceptions

1. In the class `ReadHeroFile`, there is a file called `heroes.txt`. This has a list of Heroes and their attributes. You're going to `read each line for each shape` and `call createHero`, which will create a `Hero` like you did in Assignment 3. However, you should be able to set attributes like `level` and `experience` to arbitrary values to accommodate having to set them based on the values in `heroes.txt`.

The `heroes.txt` file has comma separated fields of the format:

`Name,Role,Level,Experience`

You will want to use the `split` method for `String` objects to turn a line from the `heroes.txt` file into an `Array` of `String` objects with the attributes of a `Hero`.

The file has some unavailable Roles which were not listed in the `ROLES` array. If the role is unavailable, `setRole` should throw a `HeroException` which is caught and thrown by `createHero`. Furthermore, if the level and experience values do not match (ie, if the amount of experience is insufficient to match the level listed or if the level is smaller than would be expected for the amount of experience listed), the you should throw a `HeroException` (the mechanism through which you do this is up to you. It could be adding a new `Hero` constructor, or it could be something else. Up to you). In the main method, you should catch a `HeroException` and continue reading the file.

Next, you are going to `modify the Party class` (you can use your own implementation or the one provided) to add the `Hero` objects to the `Party`. Instead of accommodating just 3 Heroes, it should accommodate an arbitrary number. (hint: `use ArrayList`) Furthermore, the other methods in `Party` should be modified to work with this new data structure.

Summary:

- Create a `HeroException` class
- Implement `createHero` to return a `Hero` based on the input from the file
- `setRole` should throw a `HeroException` which is caught and then thrown by `createHero` if it is not an allowed `ROLE`
- A loop should read in the `heroes.txt` file line-by-line

- If the file cannot be read, you should break out of the loop
- If you get a `HeroException`, you should continue reading the file

2. 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:

- a) 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, predicate, 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 Subject, Predicate, and Object 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).

- b) 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`.