Homework 8

1 Introduction

Hello there! Welcome to the world of Pokemon! My name is Oak! People call me the Pokemon Prof! This world is inhabited by creatures called Pokemon. For some people, Pokemon are pets. Others use them for fights. Myself I study Pokemon as a profession. And you! From today, you will be my apprentice you will help me gather information on all the Pokemon there are out there! Now my lab aide Honey will give you the details of your adventure

2 Problem Description

The goal of this homework is to get you familiar with throwing, catching, and creating exceptions, as well as file I/O. You will be provided with

- Pokemon.txt A text file with a list of all 151 Pokemon in the Kanto region
- Driver.java the driver for your Pokemon adventure. You will run the driver passing in the name of the Pokedex text file as a command line argument (i.e. "java Driver Pokemon.txt").

3 Solution Description

You will notice that in the Driver file, there are a couple of different things that can happen on this adventure of yours. You can choose between walking around to encounter Pokemon, viewing your current party Pokemon, reporting to Professor Oak on the progress of your Pokedex, and ending your adventure.

When the driver creates an instance of an Adventure, the Adventure constructor should store all the Pokemon in the given text file in an Array of type String these are the possible Pokemon that you can encounter on your adventure. If no file is selected or passed in, an UnknownRegionException should occur, signifying we couldnt tell which region of the Pokemon world you are from. The constructor should also create a new text file which will serve as your Pokedex, which will be populated when you end your adventure.

Adventure.java should have the following public methods:

- walkAround(): a method to continue walking while on your adventure
- viewParty(): a method to view your current party Pokemon (see Method Implementations for what a party is if you dont know)
- reportToProfessorOak() a method to report to Prof. Oak how many Pokemon you have encountered and how many steps you have taken
- endAdventure(): a method to end the current adventure

However, it will also have some private methods:

- addToParty(String name): a method to add an encountered Pokemon to your party
- addToPokedex(String name, int index): a method to add a Pokemon to the Adventures Pokedex
- encounter(): a method to encounter a Pokemon

The method implementations below will help you think about which private methods should be called by which public methods.

3.1 Method Implementations

- walkAround(): If the option to continue walking is chosen, this method should be called. It should randomly decide how many steps to take, between 1 and 10. Every time you walk on your adventure, the chance to encounter a wild Pokemon is 90%. You should keep track of the total number of steps you take on your adventure. If you dont encounter a Pokemon, print "No wild Pokemon encountered!"
- encounter(): To determine which Pokemon is encountered, access a random index of the array of Pokemon names. To achieve this randomness, use the appropriate class (hint, youve used it before). There is more than one correct range of numbers to randomly generate! Whatever implementation you decide, make sure it allows you to randomly access each index of the array. If you encounter a wild Pokemon, it gets added to your Pokedex (which method should you call to add the Pokemon?). You should keep track of the total number of Pokemon you encounter. If you encounter a wild Pokemon, you should print out: "A wild [insert poke name] appeared!" and add it to your Pokedex. 70% of the time when you encounter a wild Pokemon, you should have the opportunity to add it to your party, which the user can accept or decline.

- addToPokedex(String name, int index): For the duration of your adventure, you will keep track of the Pokemon you have encountered in an array, which is of the same type and length as the array you made from Pokemon.txt. When you add to your Pokedex, you should check if the spot in your Pokedex array is empty or not, and proceed from there. If trying to add a Pokemon that is already in your Pokedex, a PokemonAlreadyExistsException should occur. You should print "A new Pokedex entry was made for [insert poke name]" if it is successfully added to the Pokedex. If trying to add a Pokemon that is already in your Pokedex, a PokemonAlreadyExistsException should occur.

- addToParty(String name): Your party consists to up to six Pokemon that you carry with you. They must be Pokemon that you have already encountered you cannot have a Pokemon in your party if you have never seen it! If your party has six Pokemon and another one is trying to be added, a PartyIsFullException should occur. If a Pokemon is trying to be added that is already in your party, a PokemonAlreadyExistsException should occur. If the Pokemon is successfully added, print: "[insert poke name] was added to your party!"

- reportToProfessorOak(): This method will report the progress of your adventure to Oak! You should tell him how far youve come (how many steps you've taken) and how many Pokemon are in your Pokedex.

- viewParty(): This method should list the current Pokemon in your party.

- endAdventure(): This method will bring your adventure to an end. When endAdventure() is called, you are to transfer the contents of your Pokedex array to a new file, called Pokedex.txt. This text file should contain all the Pokemon that you encountered on your adventure, in the correct order (each Pokemon has a unique number).

3.2 Exceptions

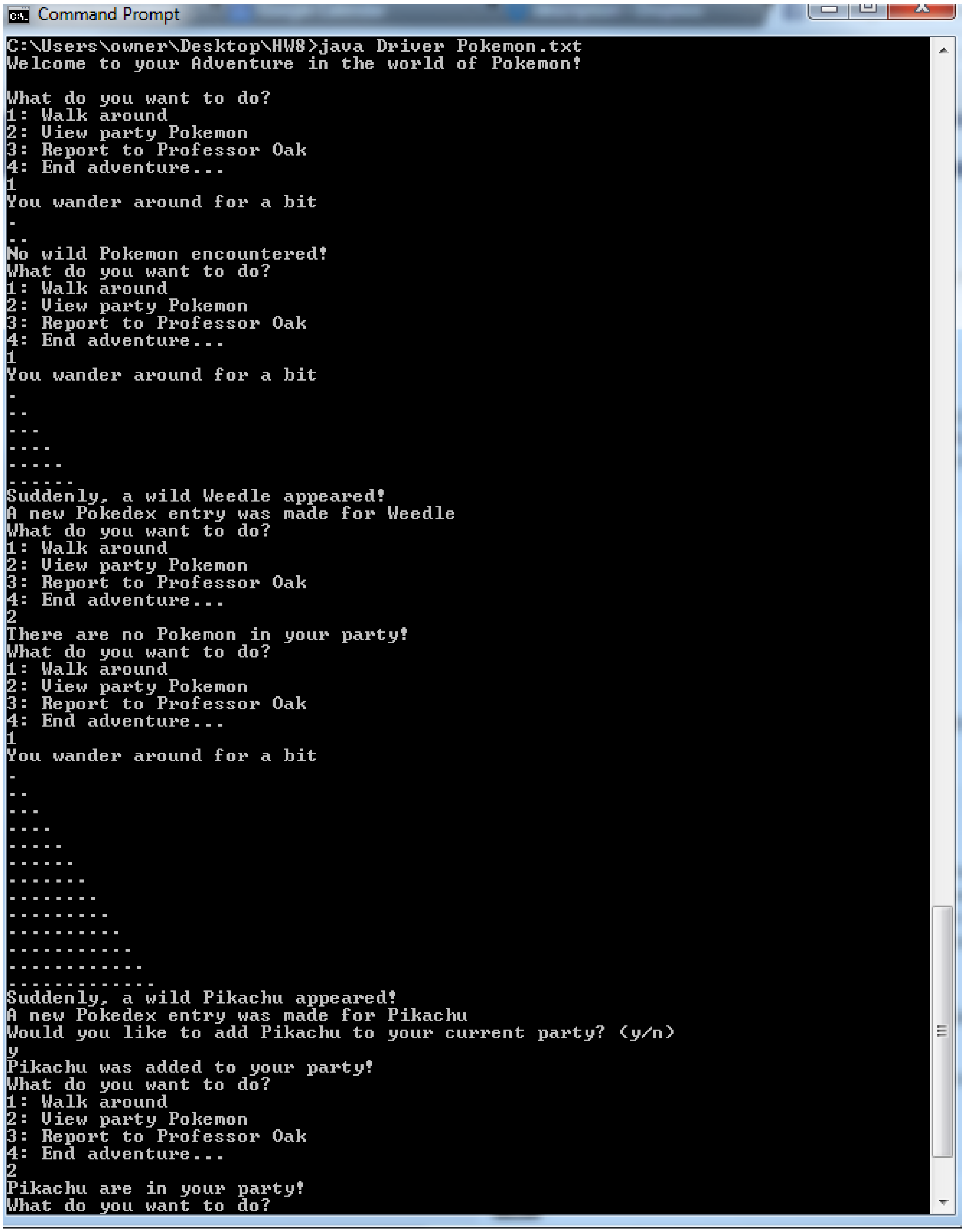
You will be creating three exceptions as detailed below. Recall from class how we make exceptions and the differences between checked and unchecked exceptions.

- PokemonAlreadyExistsException: Should include the no-arg constructor, as well as one that takes in a String message. This should be an unchecked exception.

- PartyIsFullException: Should include the no-arg constructor, as well as one that takes in a String message. This should be an unchecked exception.

- UnknownRegionException: Should include the no-arg constructor that passes along the message Could not find your region! to its super constructor. This exception signifies that we couldnt tell which region of the Pokemon world you are from. This should extend FileNotFoundException, and should be a checked Exception.

3.3 Sample Output



```
C:\Users\owner\Desktop\HW8>java Driver Pokemon.txt
Welcome to your Adventure in the world of Pokemon!

What do you want to do?
1: Walk around
2: View party Pokemon
3: Report to Professor Oak
4: End adventure...
1
You wander around for a bit
.
..
No wild Pokemon encountered!
What do you want to do?
1: Walk around
2: View party Pokemon
3: Report to Professor Oak
4: End adventure...
1
You wander around for a bit
.
..
...
....
.....
......
Suddenly, a wild Weedle appeared!
A new Pokedex entry was made for Weedle
What do you want to do?
1: Walk around
2: View party Pokemon
3: Report to Professor Oak
4: End adventure...
2
There are no Pokemon in your party!
What do you want to do?
1: Walk around
2: View party Pokemon
3: Report to Professor Oak
4: End adventure...
1
You wander around for a bit
.
..
...
....
.....
......
.......
........
.........
..........
...........
............
.............
Suddenly, a wild Pikachu appeared!
A new Pokedex entry was made for Pikachu
Would you like to add Pikachu to your current party? (y/n)
y
Pikachu was added to your party!
What do you want to do?
1: Walk around
2: View party Pokemon
3: Report to Professor Oak
4: End adventure...
2
Pikachu are in your party!
What do you want to do?
```

4 Tips

- Ask the lab aides (the TAs) for help if you get stuck!
- The only time you read from a file should be at the beginning of your adventure, when you take in the list of Pokemon in the region.
- The only time you write to a file is when your adventure ends, when you transfer the contents of your Pokedex array to a file. Dont forget to close the PrintWriter when youre done!
- Although the instance variables and their visibility modifiers werent explicitly stated, use the clues in the method implementations to figure out what variables you need.
- Although the return types of the methods werent explicitly stated, look at the Driver and see what it expects back. Dont overthink or overcomplicate things.
- Keep the idea of using an integer (array index) to refer to non-integer typed things (Pokemon name) in mind when we talk about Hashing later!

5 Javadocs

For this assignment you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to have are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```java
import java.util.Scanner;

/**
 * This class represents a Dog object.
 * @author George P. Burdell
 * @version 13.31
 */
public class Dog {

   /**
    * Creates an awesome dog (NOT a dawg!)
    */
   public Dog(){
      ...
   }

   /**
    * This method takes in two ints and returns their sum
    * @param a first number
    * @param b second number
    * @return sum of a and b
    */
   public int add(int a, int b){
      ...
   }
}
```

Take note of a few things:

1. Javadocs are begin with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@version` tag included. The comments for a class start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

5.1 Javadoc and Checkstyle

Be sure you download the updated Checkstyle file (6.2.1) from T-Square along with this assignment. It will be used to verify both your Checkstyle and Javadocs. Javadocs will count towards your grade on this assignment.

You can use the Checkstyle jar mentioned in the following section to test your javadocs for completeness. Simply add -j to the checkstyle command, like this:

```
$ java -jar checkstyle-6.2.1.jar -j *.java
Audit done. Errors (potential points off):
0
```

6 Checkstyle

Checkstyle counts for this homework. You may be deducted up to 100 points for having Checkstyle errors in your assignment. Each error found by Checkstyle is worth one point. This cap will be raised next assignment. Again, the full style guide for this course that you must adhere to can be found by clicking **here**.

Come to us in office hours or post on Piazza if you have specific questions about what Checkstyle is looking for and how to fix Checkstyle errors.

First, make sure you download the `checkstyle-6.2.1-all.jar` from the T-Square assignment page. Then, make sure you put this file in the same directory (folder) as the `.java` files you want to run Checkstyle on. Finally, to run Checkstyle, type the first line into your terminal while in the directory of your Java files and press enter.

```
$ java -jar checkstyle-6.2.1.jar *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count Checkstyle errors by piping the output of Checkstyle through `wc -l` and subtracting 2 for the two non-error lines printed above (which is how we will deduct points). For example:

```
$ java -jar checkstyle-6.2.1.jar *.java | wc -l
     2
```

Alternatively, if you are on Windows, you can use the following instead:

```
C:\> java -jar checkstyle-6.2.1.jar *.java | findstr /v "Starting audit..." | findstr /v "Audit
    done" | find /c /v "hashcode()"
0
```

7 Turn-in Procedure

Submit all `.java` files you wrote/changed on T-Square as an attachment. Do not submit any compiled bytecode (`.class` files), the Checkstyle jar file, or the `cs1331-checkstyle.xml` file. When you're ready, double-check that you have submitted and not just saved a draft.

8 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.

2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.

3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.

4. Recompile and test those exact files.

5. This helps guard against a few things.

   (a) It helps insure that you turn in the correct files.

   (b) It helps you realize if you omit a file or files.[1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

   (c) Helps find last minute causes of files not compiling and/or running.

[1]Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is 8PM Thursday. Do not wait until the last minute!