Homework 12 - Pokemon (The Sequel)

Authors: Vince Li, Jack Kelly, Shishir Bhat

Problem Description

Having now solved the Pokemon health crisis, word of your programming ability quickly spreads throughout the land like a cool seed in The Game of Life. One day, you receive an email:

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.

You soon learn that Oak is a professor at "Georgia Tech University", which, he assures you, is a well known school. He wants you to help him write software to catalog and organize Pokemon so he can better research them. At first you're skeptical, but once he promises you meal swipes, you promptly get to programming.

Solution Description

For this assignment, create the following files:

- `PokemonSpecies.java`
- `AbstractDex.java`
- `Pokedex.java`

PokemonSpecies Class:

In the world of Pokemon - all species of Pokemon (the equivalent of animals in this world) have been assigned a number for ease of reference and sorting. This representation of a PokemonSpecies, therefore, should implement the Comparable interface. A PokemonSpecies is "greater" when it has a higher speciesNumber; a PokemonSpecies is "lesser" when it has a lower speciesNumber.

Variables:

- `speciesNumber`, an int that represents the specific species of a given PokemonSpecies. Once set, this should not be changeable outside this class.
- `speciesName`, a String holding the name of the species. Once set, this should not be changeable outside this class.
- `speciesNickname` a String holding the nickname of the PokemonSpecies
- `description` a String holding a description of the PokemonSpecies.

Constructors:

- `PokemonSpecies(String speciesName, int speciesNumber, String speciesNickname, String description)`
  - Sets speciesName, speciesNumber, speciesNickname, and description
- `PokemonSpecies(String speciesName, int speciesNumber)`
  - Sets speciesName and speciesNumber. speciesNickname and description should be set to null.

Methods:

- `int compareTo(PokemonSpecies other)`
- `boolean equals(Object other)` PokemonSpecies are considered equal if they have the same speciesName and speciesNumber.
- `int hashCode()` Write a proper hashCode for PokemonSpecies that considers the proper fields (Hint: Look at equals)

- `String toString()` Should return "[speciesNumber]: [speciesName], the [speciesNickname] Pokemon. [speciesDescription]."
- Getters for all fields.
- Setters for speciesNickname, description.

AbstractDex Class:

This class should be generic. The generic type should be bounded such that you can perform searching and sorting operations on a List of any generic type that can be sorted. We will refer to the generic as T but you can name it what you want as long as you follow the proper conventions. Because this simply serves as a base of reusable code for any type of dex, it should be abstract.

Instance Variables:

- `ArrayList<T> entries` An ArrayList of objects that are comparable to each other.
- `boolean sorted` A boolean value storing whether or not entries have already been sorted by the dex.

Constructors:

- `AbstractDex()` Initializes entries to an empty ArrayList

Methods:

- `void insertionSort()` sorts entries using the insertionSort() algorithm taught in class. Sets sorted to true
- `void selectionSort()` sorts entries using the selectionSort() algorithm taught in class. Sets sorted to true.
- `int binarySearch(T element)` searches entries for the passed in element using the binary search algorithm taught in class, returning its index if found or -1 if not found or sorted is false. DO NOT USE JAVA'S BUILT IN indexOf() METHOD.
- `void addToDex(T newElement)` If there is not already an equivalent element in entries, as determined by compareTo(), add newElement to the end of entries and set sorted to false.
- `boolean equals(Object other)` Write a proper equals method for AbstractDex. Two AbstractDex objects are considered equal if they have the same entries. For comparison, you can use ArrayList's equals() method.
- `int hashCode()` Write a proper hashCode for a AbstractDex, using only the proper fields. (Hint: look at equals). You can use ArrayList's hashCode() method.
- `String toString()` Should loop through each entry and call it's toString(). Separate the outputs with newline characters. An example of what this should look like can be found in the sample tests.
- Getters for entries and sorted.
- Setter for entries.

Pokedex Class:

Is a concrete subclass of AbstractDex which holds PokemonSpecies only (Hint: how can we extend AbstractDex to only allow PokemonSpecies)

Constructors:

- `Pokedex()` initializes entries to an empty ArrayList
- `Pokedex(ArrayList<PokemonSpecies> initialEntries)` initializes entries with the elements in the passed in List.

Testing your code:

If you want to test out your own code before submitting we recommend creating a separate java file and implementing a main method there. You can then create new animals and test your methods. Below is some sample testing code. **These tests are not comprehensive!**

```
Pokedex myPokedex = new Pokedex();
PokemonSpecies carl = new PokemonSpecies("Carl", 1, "Head TA",
  "Its Mixtapes are said to be too hot for the average listener");
System.out.println(carl);
// 1: Carl, the Head TA Pokemon. Its Mixtapes are said to be too hot for the average listener.
PokemonSpecies karl = new PokemonSpecies("Karl", 2, "Spooky TA",
  "It is said Karl can be found lurking in Klaus during spooktober");
PokemonSpecies qarl = new PokemonSpecies("Qarl", 3, "Checkstyle TA",
  "Qarl likes to remind students to checkstyle their work");
System.out.println(qarl.compareTo(karl));
// positive number
myPokedex.addToDex(karl);
myPokedex.addToDex(qarl);
myPokedex.addToDex(carl);
myPokedex.insertionSort();
System.out.println(myPokedex.binarySearch(carl));
// 0
System.out.println(myPokedex.binarySearch(karl));
// 1
System.out.println(myPokedex.binarySearch(qarl));
// 2
Pokedex secondPokedex = new Pokedex();
secondPokedex.addToDex(qarl);
secondPokedex.addToDex(carl);
secondPokedex.addToDex(karl);
secondPokedex.selectionSort();
System.out.println(myPokedex.binarySearch(carl));
// 0
System.out.println(myPokedex.binarySearch(karl));
// 1
System.out.println(myPokedex.binarySearch(qarl));
// 2
System.out.println(myPokedex.equals(secondPokedex));
// true
System.out.println(myPokedex);
/*
1: Carl, the Head TA Pokemon. Its Mixtapes are said to be too hot for the average listener.
2: Karl, the Spooky TA Pokemon. It is said Karl can be found lurking in Klaus during spooktober.
3: Qarl, the Checkstyle TA Pokemon. Qarl likes to remind students to checkstyle their work.
 */
```

Feel free to create your own tests in the `main` method! Try to write tests for the remaining methods in the file!

Rubric

- [35] PokemonSpecies
  - [5] Correct Fields

  - [5] Correct Constructors
  - [5] `equals(Object)`
  - [5] `compareTo(PokemonSpecies)`
  - [5] `hashCode()`
  - [5] `toString()`
  - [5] Getters and Setters
- [60] AbstractDex
  - [5] Generic with correct type bounds
  - [4] Correct Constructor
  - [1] Correct Field
  - [10] `insertionSort()`
  - [10] `selectionSort()`
  - [10] `binarySearch(T element)`
  - [5] `addToDex(T element)`
  - [5] `equals(Object)`
  - [5] `hashCode()`
  - [5] `toString()`
- [5] Pokedex
  - [5] Correct Constructors

Allowed Imports

- To prevent trivialization of the assignment, you are only allowed to import the following class:
  - java.util.ArrayList

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. For more information on what Javadocs are and why they are awesome, the online overview for them is extremely detailed and helpful.

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 include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
* @author George P. Burdell
* @version 1.0
*/

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) {
    ...
    }
}
```

A more thorough tutorial for Javadocs can be found here. Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@verion` tag included. The comments for a class should 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.

Checkstyle can check for Javadocs using the -a flag, as described in the next section.

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded here. To run Checkstyle, put the `jar` file in the same folder as your homework files and run

```
java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **100 points**. This means that up to 100 points can be lost from Checkstyle errors.

Collaboration Statement

To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment AT THE TOP of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Recall that comments are special lines in Java that begin with `//`.

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `PokemonSpecies.java`
- `AbstractDex.java`
- `Pokedex.java`

Make sure you see the message stating "HW12 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.