# Homework 11 - Pokemon

**Authors: Emma Barron, Peter Dong, Manny Sonubi, Tushna Eduljee, Ryan Miles, Shishir Bhat**

## Problem Description

Struck by sudden nostalgia, you quit your job and start playing games from your childhood, like Pokemon! However, life seems determined to make you a Software Engineer. One day, as you sit huddled underneath a blanket snacking on saltine crackers, your DS suddenly glows bright blue. You shield your eyes, and when you open them you find yourself in the world of Pokemon. You immediately walk to the nearest Pokemon Center that you know is 108.15 yards to your right. Nurse Joy is very kind to you, but after watching her treat the injured Pokemon, you realize she is overworked and extremely disorganized. This calls for some programming!

## Solution Description

For this assignment, create the following files:

- `Pokemon.java`
- `Pikachu.java`
- `Charmander.java`
- `PokemonCenter.java`

### Pokemon Class:

Implements the Comparable interface. A pokemon is "greater" when it has more health; a pokemon is "lesser" when it has less health. Should be abstract.

*Instance Variables*:

- `medicalHistory`, an ArrayList of Integers that tracks changes to the pokemon's health. For example, if a pokemon loses 5 health, -5 should be appended to medicalHistory. As another example, if a pokemon gains 7 health, 7 should be appended to medicalHistory. This instance variable should have private access.
- `trainerName`, a String which is the pokemon's trainer's name. It should have private access.
- `health`, an int that is the pokemon's health. It should have private access.
- `maxHealth`, an int that is the pokemon's maximum health. It should have private access.

*Constructors*:

- `Pokemon(String trainerName, int health, int maxHealth)`
  - Sets trainerName, health and maxHealth properly.
  - Initializes an empty ArrayList of Integers for medicalHistory.
- `Pokemon(Pokemon p)` copy-constructor, provides a deep copy of the object.

*Methods*:

- `void heal(int amount)` increase health by specified amount. However, health cannot be greater than maxHealth. In cases where health plus amount is greater than maxHealth, set health to maxHealth. Because health has changed, make sure to update medicalHistory. Only update with actual (not expected) change to health. For example, if a pokemon has a health of 6, maxHealth of 7, and is healed by amount 5, only append 1 to medicalHistory. As another example, if a pokemon has health 8, maxHealth 8, and is healed by amount 2, append 0 to medicalHistory.

- `void damage(int amount)` decrease health by the specified amount. However, health cannot go below 0. In cases where decreasing health by the specified amount makes health negative, set health to 0 and update medicalHistory with only the amount of health lost; for example, if a pokemon has 3 health and takes 4 damage, you would append -3 to its medicalHistory.
- `int compareTo(Pokemon other)` A pokemon is "greater" when it has more health; a pokemon is "lesser" when it has less health.
- `boolean equals(Object other)` Pokemon are considered equal if they have the same trainerName, health, maxHealth, and medicalHistory.
- `int hashCode()` Write a proper hashCode for Pokemon that contains the proper fields. Only use the maxHealth field to compute the hash code.
- Getters for trainerName, health, and maxHealth.
- Setter for trainerName.

**Pikachu Class:**

Is a subclass of Pokemon

*Instance Variables*:

- `int friendshipLevel` the level of friendship the pikachu has with its trainer. It should have private access.

*Constructors*:

- `Pikachu(String trainerName, int health, int maxHealth, int friendshipLevel)` sets trainerName, health, maxHealth, and friendshipLevel properly.
- `Pikachu(Pikachu p)` copy-constructor, provides a deep copy of the object.

*Methods*:

- `void heal(int amount)` increase health by twice the amount because pikachus heal quickly. However, health cannot be greater than maxHealth. In cases where health plus amount is greater than maxHealth, set health to maxHealth. Because health has changed, make sure to update medicalHistory. Only update with actual (not expected) change to health. Reuse code if possible!
- `boolean equals(Object other)` Pikachus are considered equal if they have the same trainer, health, maxHealth, medicalHistory, and friendshipLevel. Reuse code if possible!
- `int hashCode()` Write a proper hashCode for Pikachu that contains the proper fields. Use the fields maxHealth and friendshipLevel for computing the hash code.

**Charmander Class:**

Is a subclass of Pokemon

*Instance Variables*:

- `int flameLevel` represents how much flame the charmander has. It should have private access.

*Constructors*:

- `Charmander(String trainerName, int health, int maxHealth, int flameLevel)` sets trainerName, health, maxHealth, and flameLevel properly.
- `Charmander(Charmander c)` copy-constructor, provides a deep copy of the object.

*Methods*:

- `boolean equals(Object other)` Charmanders are considered equal if they have the same trainer, health, maxHealth, medicalHistory, and flameLevel. Reuse code if possible!
- `int hashCode()` Write a proper hashCode for Charmander that contains the proper fields. Use the fields maxHealth and flameLevel for computing the hash code.

- `void damage(int amount, boolean isWaterDamage)` decrease health by amount if it is not water damage. if it is water damage, decrease health by twice the amount. Health cannot go below 0. In cases where health falls below 0, set health to 0 and update medicalHistory with only the amount of health lost; for example, if a pokemon has 3 health and takes 4 damage, you would append -3 to its medicalHistory. Make sure to update medicalHistory accordingly and reuse code wherever possible!

**PokemonCenter Class:**

*Instance Variables*

- `injured`, an ArrayList of pokemon that holds the injured pokemon in line to be treated. The list should be ordered by how recently a pokemon was seen. Most recently seen pokemon are always at the end of the list. It should have private access.
- `nurses`, an int that is the number of nurses at the Pokemon Center. It should have private access.

*Constructors*:

- `PokemonCenter(ArrayList<Pokemon> injured, int nurses)` sets the two instance variables to the provided values.
- `PokemonCenter()` sets the two instance variables to their default values; empty ArrayList of injured pokemon and one nurse.

*Methods*:

- `boolean add(Pokemon pokemon)` adds a pokemon to the list of injured pokemon, regardless of whether or not it is injured. Always add to the end of the `injured` list. Just make sure that a pokemon isn't already on the list! This method returns true if a pokemon was added, else false.
- `void heal(int amount)`. For each nurse in the center, remove a pokemon from the list and increase its health by the specified amount. If there are more nurses than pokemon in the list, simply heal each pokemon once. If (after healing) a pokemon is not fully healed, add it to the list of injured Pokemon. Remember, most recently seen pokemon go at the end!
- `void healMostInjured(int amount)`. Sort the list, putting the most injured pokemon in the front and the least injured pokemon in the back. (HINT: Check out the Collections class) Then, for each nurse in the center, remove a pokemon from the front of the list and increase its health by the specified amount. If there are more nurses than pokemon in the list, simply heal each pokemon once. If the pokemon isn't fully healed, add it back to the list. Make sure that after this method finishes executing, `injured` is still ordered by how recently a pokemon has been seen. (most recently seen pokemon go at the end!)
- `String toString()` which returns "This Pokemon Center has [number of nurses] nurses and [number of pokemon in injured list] pokemon waiting to be seen."

## 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!**

```
Pikachu pikachu = new Pikachu("Ryan", 20, 25, 10);
Pikachu pikachu2 = new Pikachu(pikachu);
Charmander charmander = new Charmander("Emma", 20, 30, 10);
Charmander charmander2 = new Charmander("Emma", 10, 30, 10);

System.out.println(pikachu.equals(pikachu2)); // True
pikachu.heal(10);
System.out.println(pikachu.equals(pikachu2)); // False
```

```
PokemonCenter pokemonCenter = new PokemonCenter();
pokemonCenter.add(charmander2);
pokemonCenter.add(charmander);

pokemonCenter.healMostInjured(); //charmander2 health = 17 now
```

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

## Rubric

- [35] Pokemon
  - [5] `heal(int)`
  - [5] `damage(amount)`
  - [5] `equals(Object)`
  - [5] `compareTo(Pokemon)`
  - [5] `hashCode()`
  - [5] Copy-constructor assigns properly
  - [2.5] Constructor with four parameters assigns properly
  - [2.5] Specified getters and setters
- [15] Pikachu
  - [2.5] `heal(int)` reuses code
  - [5] `equals(Object)` reuses code
  - [2.5] `hashCode()`
  - [2.5] Copy-constructor assigns properly
  - [2.5] Constructor with four parameters assigns properly
- [15] Charmander
  - [2.5] `damage(int, boolean)` reuses code
  - [5] `equals(Object)` reuses code
  - [2.5] `hashCode()`
  - [2.5] Copy-constructor assigns properly
  - [2.5] Constructor with four parameters assigns properly
- [35] PokemonCenter
  - [5] `add(Pokemon)`
  - [10] `heal()`
  - [10] `healMostInjured()`
  - [5] `toString()`
  - [2.5] Constructor with two parameters assigns properly
  - [2.5] Constructor with zero parameters assigns properly

## Allowed Imports

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

## 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:

- `Pokemon.java`
- `Pikachu.java`
- `Charmander.java`
- `PokemonCenter.java`

Make sure you see the message stating "HW11 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.