# Homework 2

**Due Tuesday, July 5th, at 11:59pm on Classes for 20 points.**

## Motivation

- Demonstrate your ability to apply **polymorphism** in the Java programming language.

## Tasks

Take a look at the following code:

```
// Create two Pikachu objects
Pikachu puka   = new Pikachu("Puka", 5, "Electric");
Pikachu sparky = new Pikachu("Sparky", 5, "Electric");

// Level them both up a completely random, probably different number of levels
for (int i = 0; i < (int) (Math.random() * 100); i++) { puka.levelUp();   }
for (int j = 0; j < (int) (Math.random() * 100); j++) { sparky.levelUp(); }

// Print the name and level of the winner
System.out.printf(
    "The Pikachu with a higher level is: %s, with a level of %d!\n",
    puka == sparky ?          "Both" : (puka.compareTo(sparky) > 0 ? puka.getNickname() : spar
    puka == sparky ? puka.getLevel() : (puka.compareTo(sparky) > 0 ? puka.getLevel()    : spar
);
```

**Code Block 1**: Two `Pikachu` objects being initialised, levelled up (through `levelUp()`), and compared in order to print out the nickname and final level of the stronger one.

For this problem, `Pikachu` objects are **compared only using their *level values* alone, and nothing else**.

`Pikachu` objects are created using three parameters: their nickname (a `String`), their current level (and `int`), and their type (another `String`). Here, you will find that both `Pikachu` objects `puka` and `sparky` are being compared for both equality (e.g. `puka == sparky`) and for inequality (e.g. `puka.compareTo(sparky) > 0`). Your job is to make sure that, if we ran the code in code block 5, we got a results similar to the ones below:

- *Example 1*:

```
The Pikachu with a higher level is: Puka, with a level of 20!
```

- *Example 2*:

```
The Pikachu with a higher level is: Sparky, with a level of 23!
```

- *Example 3*:

```
The Pikachu with a higher level is: Sparky, with a level of 18!
```

- *Example 4*:

```
The Pikachu with a higher level is: Puka, with a level of 9!
```

The way to do this is to modify the `Pikachu` class is similar to the way we prepared our `Country` classes to be sortable and searchable. Here's what it looks like right now:

```
import java.util.concurrent.atomic.AtomicInteger;

public class Pikachu {
    private final String nickname;
    private final AtomicInteger level;
    private final String type;

    public Pikachu(String nickname, int level, String type) {
        this.nickname = nickname;
        this.level = new AtomicInteger(level);
        this.type = type;
    }

    public void levelUp() {
        level.incrementAndGet();
    }

    public String getNickname() {
        return nickname;
    }

    public int getLevel() {
        return level.get();
    }

    public String getType() {
        return type;
    }
}
```

**Code Block 2**: Our starting point.

In the **actual file**, you will see the code contained in code block 1 contained in the `Pikachu` class's commented-out `main()` method. Once you have added the necessary code to make `Pikachu` objects comparable, you can uncomment and run the code to check if it works.

A few of things to keep in mind:

- There may be some Java types that you may not recognise. **That's totally fine**. Part of this assignment is to be able to add to somebody else's class. All you need to worry about is making sure that `Pikachu` objects are **comparable** to each other.
- Use `getLevel()` to access the value of `level`.
- You only need to add a few things to the `Pikachu` class in order to finish this assignment, so make sure to ask me questions if you need to!
- Although it is not part of this problem, `Pikachu` objects must also be sortable and searchable in `Pikachu` arrays.

## Bonus Points

- Create an array of random `Pikachu` objects and use `Arrays.sort()` to sort it.

## Implementation

- Ensure your code is correct by compiling and testing it.
- A portion of your grade will be based upon readability and organization of your code.
  - Follow the naming guidelines of lecture.
  - Break large functions (if you have any) into multiple functions based on logical organizations.

## How to Submit

When submitting to Classes, make sure to do so by placing all of your `java` files into one folder, zipping it, and then uploading it. The zipped folder's structure must be as follows:

```
[lastName-firstName-assignment01]
    |
    |----> Pikachu.java
```