# Programming Exercise 03 – Math, Random, Enums

*Authors: Ananay, Owen, Chloe, Ruchi*

## Problem Description

Water. Earth. Fire. Air. While the Avatar cycle has seemingly ended, the 4 nations are preparing for war and it is up to you, fellow 1331 student, to help determine who will win! As each nation gathers their armies, the fire nation decides to attack the 3 other nations, forcing the remaining 3 to ally with each other. As all the nations summon heroes to help them defeat the enemy, and you will have to write a program to determine which army has the strongest support from their heroes.

Your assignment will consist of 2 main parts:

- Determining how many heroes are summoned, which nation they belong to and what statistics they have.
- Printing out the hero information and averaging it out for the hero's kingdom to see who wins.

***Some notes for all variables in the assignment:***

I. When asked to generate a random variable, you **must** use `java.util.Random`

II. When asked to generate a floating-point value, **round the value to 2 decimal places**. You can only use `Math.round()` to round the value, but remember that you have operators like ***** , **–** , and **/** to help you achieve proper rounding.

## Solution Description

1. Create a public `enum` called `BendingType`.
    a. This special "class" should be defined in a separate file.
2. Populate the `enum` with the values `AIR, WATER, FIRE, EARTH, NON_BENDER`.
3. Create a public class called `BendingBattlefield`
4. Create the `main` method
5. Inside, create the following variables:
    a. An instance of **Random** named `rand`.
    b. An **int** called `numHeroes` and assign to it a random value in the range [50, 100].
    c. A **double** called `boomerangBoost` and assign to it a random value in the range [3, 7).

    **Note the interval notation:**

    `[a, b]` means a number x such that **`a <= x <= b`**.
    `(a, b]` means a number x such that **`a <  x <= b`**.
    `[a, b)` means a number x such that **`a <= x <  b`**.
    `(a, b)` means a number x such that **`a <  x <  b`**.

6. Create four **double** variables: `averageFirePower, averageFireHealth, averageAlliancePower`, and `averageAllianceHealth`.
7. Print `numHeroes` with the following format:
    `"Selecting <numHeroes> heroes."`
8. Create an **int** named `numFire` that will keep a running total of the number of heroes belonging to the `FIRE` nation.
9. Create an **int** named `numAlliance` that will keep a running total of the number of heroes belonging to the `AIR`, `WATER`, `EARTH`, or `NON_BENDER` nations.
10. Create a loop that terminates after `numHeroes` iterations. **Perform steps 11 to 17 inside the loop**.

11. Create a local `BendingType` variable called `heroType` and assign a random `BendingType` to it.
    a. You can do this using this line:
       `BendingType heroType = BendingType.values()[rand.nextInt(5)];`
12. Create a double variable called `health` and assign a random value in the range [50, 150).
    a. Make sure all your floating-point values are rounded to **2 decimal places** using `Math.round()`.
13. Create a double variable called `power` and assign a random value in the range (40, 120] to it.
    a. Make sure all your floating-point values are rounded to **2 decimal places** using `Math.round()`.
14. If the `heroType` is `FIRE`, then `health` and `power` will be added to `averageFireHealth` and `averageFirePower`, respectively. Increment `numFire` appropriately.
15. If `heroType` is `NON_BENDER`, multiply `power` by `boomerangBoost.`
16. If the `heroType` is `NON_BENDER`, `AIR`, `WATER`, or `EARTH`, then `health` and `power` will be added to `averageAllianceHealth` and `averageAlliancePower`, respectively. Increment `numAlliance` appropriately.
17. Print the status of the hero. You can follow the following format:
    `"<heroType> hero has been summoned by his army, adding <power> power and <health> health to the army."`
18. Once the loop has completed, divide each of the averages, `averageFirePower`, `averageFireHealth`, `averageAlliancePower`, and `averageAllianceHealth` by their total number of respective heroes, `numFire` or `numAlliance`, to get the correct average.
19. Print out the following statistics:
    a. `"The Fire Nation has an average of <averageFirePower> power and <averageFireHealth> health."`
    b. `"The Alliance has an average of <averageAlliancePower> power and <averageAllianceHealth> health."`
20. Calculate whether the Fire nation or the Alliance won by comparing `2 * averagePower + 3 * averageHealth` of each side and checking which one is larger. If they are equal, then the Alliance wins.
    a. If the Fire Nation won, print
       `"The Fire Nation won!"`
    b. else if the Alliance won, print
       `"The Alliance won!"`

### *Example Outputs*

Please refer to the PE clarification thread on Ed for examples.

***HINT: To help debug your code, try inserting print statements in places where variables are changed.***

## Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

## Import Restrictions

You can import only the following classes:

- `java.util.Random`

## Checkstyle

You must run Checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of Canvas.) **The Checkstyle cap for this assignment is 0 points. This means there is a maximum point deduction of 0.** If you don't have Checkstyle yet, download it from Canvas -> Modules -> CheckStyle Resources -> checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java

Starting audit...

Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the Checkstyle cap mentioned in the Rubric section). In future PEs and Homeworks, **we will be increasing this cap**, so get into the habit of fixing these style errors early!

For additional help with Checkstyle see the CS 1331 Style Guide.

## Collaboration

Only discussion of the PE at a conceptual high level is allowed. You can discuss course concepts and HW assignments broadly, that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going **too far**. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs, and this exchange should be private.

## Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- It is expected that everyone will follow the Student-Faculty Expectations document, and the Student Code of Conduct. The professor expects a positive, respectful, and engaged academic environment inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment, deemed by the professor, to contain inappropriate, offensive language or threats will get a zero**. You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.

## Turn-In Procedure

### *Submission*

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

- `BendingBattlefield.java`
- `BendingType.java`

Make sure you see the message stating "PE3 submitted successfully". 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 so be sure to **submit every file each time you resubmit**.

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder test are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via Piazza for clarification.

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 latest submission. **Be sure to submit every file each time you resubmit**.

## *Gradescope Autograder*

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond 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:

- Prevent upload mistakes (e.g. non-compiling code)
- 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.