INF1-OP Mock Programming Exam 2018

1. Note that all questions are compulsory.
2. Remember that a file that does not compile, does not pass the simple JUnit tests provided, or uses Java packages will get no marks.
3. This is an Open Book exam. You may bring in your own material on paper. No electronic devices are permitted.
4. CALCULATORS MAY NOT BE USED.

1 Boss Monster

In this question you will implement the class `Boss` which represents the boss monster in a video game. It is a specific monster type which you have to defeat at certain points in the game.

Regular monsters in the game have a value for *health* and a value for *attack power*. They can *take damage* from the player which reduces their current health. Once the health reaches zero, the monster is defeated. They can also *deal damage* to the player which depends on a constant value for attack power.

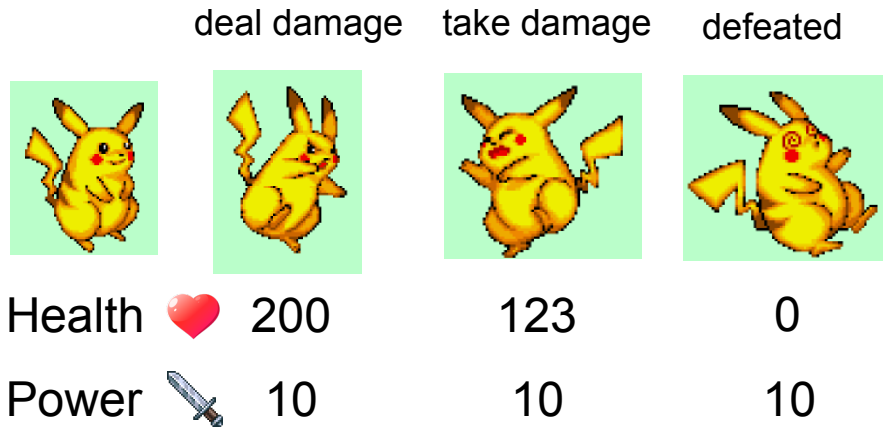

Figure 1: Monsters can deal damage and take damage. Once its health reaches zero it is defeated.

The boss monster is a specific monster which goes through three different *stages* of behaviour while fighting the player. It changes its attack power depending on its current health. The lower the boss monster's health, the stronger its attack power.

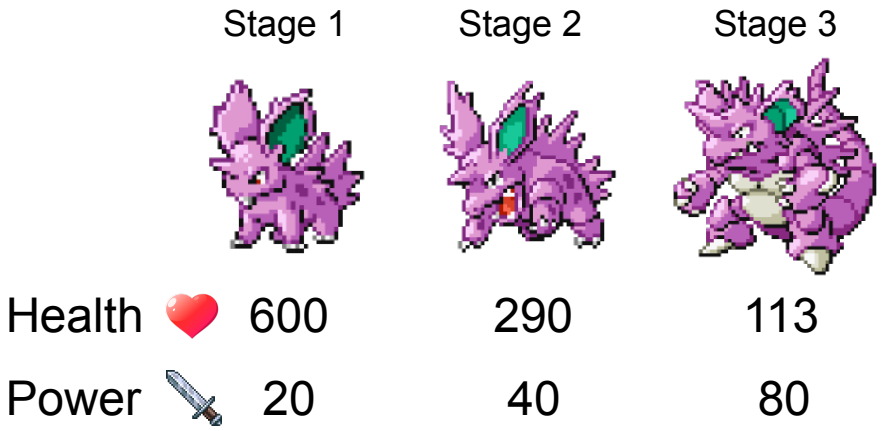

Figure 2: Boss monster going through three stages while loosing health. Every stage doubles the boss monster's attack power.

For your implementation you will be provided with the class `Monster` which is an implementation of a basic monster. You must not modify this class but should have a look at it to understand its functionality.

From the exam template directory, please use all of the following files to answer this question (if you are using Eclipse, make sure you import ALL of them):

- Monster.java
- BossBasicTest.java

Please execute the following steps for your implementation of `Boss`:

1. Define the class `Boss` as a subclass of `Monster`. It should have two instance variables:
   - a private variable `stage` of type `int` representing the current boss stage
   - a private `final` variable `initialHealth` of type `int` representing the initial health value the boss had before taking any damage

   [5 marks]

2. Write a public constructor for `Boss`, which gets the following parameters in that order: an integer `health`, which is the total health of the boss and an integer `power` which is the boss' initial attack power. In this constructor, invoke the `Monster` constructor with total health and initial attack power. Also, initialise the `Boss`' instance members by setting `initialHealth` to the provided `health` parameter and `stage` to one.

   You do not need to check if the given parameters are larger than zero. This is already done in the `Monster` class.

   [5 marks]

3. Override `Monster`'s instance method `toString`. In your implementation use `Monster`'s `toString` method to get a `String` representation of the `Monster` base class and append a `String` for the boss' current stage. For the boss in stage one in the Figure 2, the returned `String` should look **exactly** like the following (only one space between colon and number):

   ```
   Health: 600
   Power: 20
   Stage: 1
   ```

   [5 marks]

4. Override `Monster`'s instance method `takeDamage`, which reduces the health by the provided `damage` value. In your implementation, use `Monster`'s `takeDamage` method to process the `damage` parameter.

   Then check the resulting `health` and adapt the `stage` and `damage` values as follows: If the boss is not yet defeated, and the inflicted damage brought the current health below `50%` or `20%`, increase its current stage by one and double its attack power.

   As an example, consider the scenarios in Figure 3:

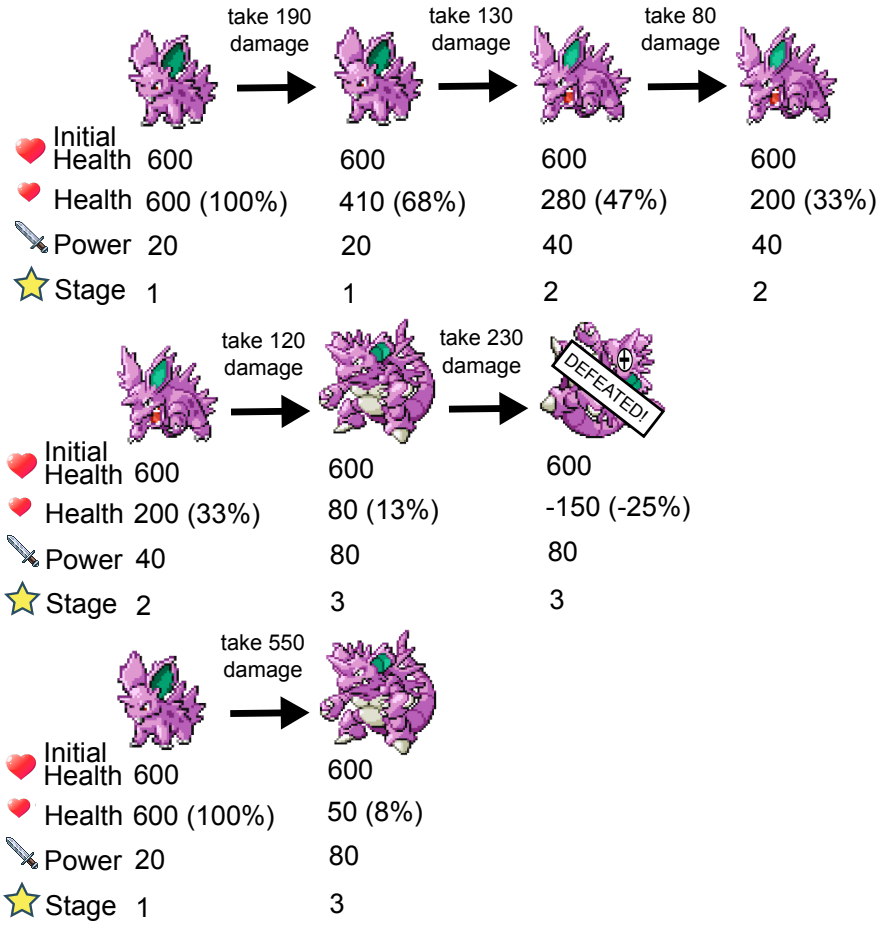

Figure 3: Sample scenarios for boss stage transitions.

[10 marks]

The file you must submit for this question is `Boss.java`. Before you submit, check that it compiles, passes the basic JUnit tests provided and uses no packages, otherwise it will get 0 marks.

2 Rock-Paper-Scissors

Game In Rock-Paper-Scissors, two players compete with each other using three different symbols, namely, Rock, Paper and Scissors. For a single round each player secretly decides which symbol to use in this round. On a predetermined signal both players show their selected symbol to each other. The following rules then decide which player won the match:

- Rock beats Scissors
- Scissors beat Paper
- Paper beats Rock

Should both players show the same symbol, the game ends in a draw and no one wins.

Matchup For your program you are given a class `Matchup` which represents the matchup of two players for a single round of the game. You must not change this class but you will have to use it for your implementations.

Rock-Paper-Scissors Skeleton For this question you are provided with a `RockPaperScissors` skeleton implementation in file `RockPaperScissors.java`. The skeleton has method stumps for your implementation where you can fill in your solutions. The skeleton also comes with a main function which is already filled with an example execution of the game and some helper functions you can use for your solution. You are not marked on the contents of the main function, so feel free to alter it for testing your code. However, you must make sure that your final solution has *no compiler errors!*

You can execute the `RockPaperScissors` class' main function as you are used to from the labs. The required main method arguments are space separated blocks of characters. For example: `RP PP PS SP SS`

JDK Library Classes In this question you will use the collection classes `ArrayList` and `Hashtable` which are familiar from lectures and labs. You may need to look at the JDK documentation in case you need information about specific methods. Note that the types of your methods will involve `Map` and `List`, thus hiding, from clients, which implementation of the `Map` or `List` interface is used; your code is expected to create, concretely, `Hashtable`s and `ArrayList`s.

Parameter Assumptions You do not have to check if any of the parameters to your methods are `null`.

From the exam template directory, please use all of the following files to answer this question (if you are using Eclipse, make sure you import ALL of them):

- RockPaperScissors.java
- Matchup.java
- RockPaperScissorsBasicTest.java

For your solution, implement the following two game aspects:

1. Implement the method `parseMatchups` as given in the skeleton file. This method parses an array of `String`s where each `String` represents a single round of the game. For each entry in the array, your implementation should create a `Matchup` object and set the corresponding symbols for each player. All `Matchup`s generated this way should be put into an `ArrayList` in the same order as they are given in the `String` array. This list must then be returned.

   A single `String` is represented by two characters; the first one for player one and the second one for player two. The character 'R' stands for Rock, 'P' for Paper and 'S' for Scissors.

   Ignore `Strings` in the array which don't have exactly two characters. Also, ignore `Strings` which have other characters then the three mentioned before. You can use the method `isValidSymbol` to verify if a symbol is valid or not.

   As an example, for the input `String` array `["PP", "EO", "PS", "SP", "R"]`, the returned `ArrayList` should contain `["PP", "PS", "SP"]`.

   [10 marks]

2. Implement the method `countOutcomes` as given in the skeleton file. This method gets a `List` of matchups, counts the outcomes of each `Matchup` in the `List` and stores the counts in a `Hashtable`. The `Hashtable` maps a possible outcome to its number of occurrences in the `List`. An outcome is either the winning symbol of a `Matchup` or the `String` `DRAW` in case of a draw. You can use the method `decideOutcome` to determine the outcome of a given `Matchup`.

   Additionally, your implementation should determine the outcome which happened the most and print it to the console. In case of a tie between outcomes, you can choose any. If the provided `List` of matches is empty, you should return an empty `Hashtable` and print nothing.

   Consider the following examples (Only one space between printed words):

   - Input: `["PP", "PS", "SP", "RP", "RR"]`,
     Returned `Hashtable`: {`S=2, DRAW=2, P=1`},
     Output: `Most outcomes: S`
   - Input: `["PP", "SS", "RS", "PR", "SP", "RR"]`,
     Returned `Hashtable`: {`S=1, R=1, DRAW=3, P=1`},
     Output: `Most outcomes: DRAW`
   - Input: `[]`,
     Returned `Hashtable`: {},
     Output:

   [15 marks]

The file you must submit for this question is `RockPaperScissors.java`. Before you submit, check that it compiles, passes the basic JUnit tests provided and uses no packages, otherwise it will get 0 marks.