CS 101 - Algorithms & Programming I

Fall 2022 - Lab 8

Due: Week of December 5, 2022

*Remember the **honor code** for your programming assignments.*
*For all labs, your solutions must conform to the CS101 style **guidelines**!*
All data and results should be stored in variables (or constants where appropriate) with meaningful names.

The objective of this lab is to learn how to define custom classes and understand object references. As you know an object-oriented software is essentially a collection of interacting objects (class instances) that know each other through these references. Remember that analyzing your problems and designing them on a piece of paper *before* starting implementation/coding is always a best practice. Specifically for this lab, you are to both organize your data and methods working on them.

0. Setup Workspace

Start VSC and open the previously created workspace named `labs_ws`. Now, under the `labs` folder, create a new folder named `lab8`.

In this lab, you are to have four Java classes/files (under `labs/lab8` folder) as described below. We expect you to submit your modified/completed `GameManager` class and any revision files, **without compressing them**. Do ***not*** upload other/previous lab solutions in your submission. Outputs of sample runs are shown as **brown** whereas the user inputs are shown in blue color.

Important Note: All of the data members are to have **private** access modifiers, and should be obtained or modified with accessor or mutator methods, respectively.

1. Escaping the dungeon

In this lab, you are going to implement a simple game, where the player (you) will start from a corner of a dungeon (room) shaped as an n x n grid and try to reach the opposite corner while avoiding your enemies (a monster and a bug).

For this purpose you will use three classes named `Player`, `Enemy` and `GameManager`. `Player` and `Enemy` are the classes for representing the player and the enemies in your game, respectively, both provided to you with an implementation in full. On the other hand, even though a template of the `GameManager` is already available, you will complete the rest (specifically the missing implementation of most methods marked with "`// TODO`").

Below are sample runs with which your output must match *exactly*.

Sample run:

```
---------------------
| P |   |   |   |   |
---------------------
|   |   |   |   |   |
---------------------
|   |   | B |   |   |
---------------------
|   |   |   |   |   |
---------------------
|   |   |   |   | M |
---------------------
Health: **********
```

```
---------------------
|   | P |   |   |   |
---------------------
|   |   | B |   |   |
---------------------
|   |   |   |   |   |
---------------------
|   |   |   |   | M |
---------------------
|   |   |   |   |   |
---------------------
Health: **********
```

```
w: up, x: down, d: right, a: left, s: no
change, q: quit: d
```

```
w: up, x: down, d: right, a: left, s: no
change, q: quit: d
```

```
--------------------
|   |   | PB|   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   | M |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
Health: ********

w: up, x: down, d: right, a: left, s: no
change, q: quit: x
```

```
--------------------
|   |   |   | B |   |
--------------------
|   |   | P |   | M |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
Health: ********

w: up, x: down, d: right, a: left, s: no
change, q: quit: x
```

```
--------------------
|   |   | B |   |   |
--------------------
|   |   |   |   | M |
--------------------
|   |   | P |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
Health: ********

w: up, x: down, d: right, a: left, s: no
change, q: quit: s
```

```
--------------------
|   |   | B |   |   |
--------------------
|   |   |   | M |   |
--------------------
|   |   | P |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
Health: ********

w: up, x: down, d: right, a: left, s: no
change, q: quit: d
```

```
--------------------
|   |   | B | M |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   | P |   |
--------------------
|   |   |   |   |   |
--------------------
Health: ********

w: up, x: down, d: right, a: left, s: no
change, q: quit: x
```

```
--------------------
|   |   |   | M |   |
--------------------
|   |   | B |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   | P |   |
--------------------
Health: ********

w: up, x: down, d: right, a: left, s: no
change, q: quit: d
```

```
--------------------
|   |   |   |   |   |
--------------------
|   |   |   | M |   |
--------------------
|   |   | B |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   | P |
--------------------
Health: ********

Player wins!!!
```

Sample run:

```
--------------------
| P |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   | B |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   | M |
--------------------
Health: **********

w: up, x: down, d: right, a: left, s: no
change, q: quit: d
```

```
--------------------
|   | P |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   | B |   |
--------------------
|   |   |   |   | M |
--------------------
|   |   |   |   |   |
--------------------
Health: **********

w: up, x: down, d: right, a: left, s: no
change, q: quit: s
```

```
--------------------
|   |   | P |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |M B|
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
Health: **********

w: up, x: down, d: right, a: left, s: no
change, q: quit: x
```

```
--------------------
|   |   |   |   |   |
--------------------
|   |   |   | P |   |
--------------------
|   |   |   | B | M |
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
Health: **********

w: up, x: down, d: right, a: left, s: no
change, q: quit: d
```

```
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   | P |
--------------------
|   |   | B |   |   |
--------------------
|   |   |   |   | M |
--------------------
|   |   |   |   |   |
--------------------
Health: **********

w: up, x: down, d: right, a: left, s: no
change, q: quit: x
```

```
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
|   | B |   |   | P |
--------------------
|   |   |   | M |   |
--------------------
|   |   |   |   |   |
--------------------
Health: **********

w: up, x: down, d: right, a: left, s: no
change, q: quit: s
```

```
--------------------
|   |   |   |   |   |
--------------------
|   |   |   |   |   |
--------------------
| B |   |   |   |   |
--------------------
|   |   |   |   |MP |
--------------------
|   |   |   |   |   |
--------------------
Health:

Player loses!!!
```

2. Game manager

The `GameManager` class is where all the pieces come together. This class' `main` method will create an instance, which in turn will set up the game (`setupGame()` method) by creating an instance of the `Player` and two instances of the `Enemy` classes (one for a monster and other for a bug), positioning these objects in predefined places (player goes to the upper left corner of the grid representing the room, the monster goes to the lower right, where the target is, and the bug is in the middle of the grid).

Then the game loop starts, where the following are performed until the game is over or the player decides to quit:

- A character is input by the user which determines whether they would like to quit or continue by typing a valid character indicating the new direction ('w': up, 'x': down, 'd': right, 'a': left) of the player (the user might prefer to not change the direction by typing an 's').
- Then the player's direction is set accordingly.
- Then all game objects are moved. Here the enemies might change direction in a random manner (a monster is twice as likely to change direction than a bug).
- Next any collisions of game objects are handled. Here the collision of the player with a bug is going to result in loss of some minimal health but collision of the player with a monster is fatal, resulting in loss of all health and hence the game. Enemies colliding with each other do not require any action. Notice however that `displayBoard()` method should be sophisticated enough to show multiple game objects in the same grid location as exemplified in the sample run.
- Before moving into the next iteration the latest state of the board (grid) is displayed.

When the loop finishes, we check whether or not the reason we are out of the loop is the desire to quit or the game ended by winning or losing. Then display an appropriate message for all cases.

Notice here that using `GameManager.random.int(n)`, one may generate a random integer between 0 and $n - 1$.

Also notice that in your implementation you should refrain from "hardcoding" values such as number of rows and columns, initial and target locations and damages upon collisions. In other words, changing such values (e.g. enlarge the grid) from a single location in the code and re-running should work nicely.