Assignment 4

Consider the Plants vs. Zombies example we referred to in our Analysis Case study. For this assignment, ignore the actual game, but follow the assumptions below closely.

1. The plants we need to consider are:

   a) Peashooters. They shoot bullets which hit the zombies from the front.

2. The zombies we need to consider are Regular Zombies, and their variations that carry (only) one "accessory" with them. The "accessory" can be one of three options: a bucket, a cone, or a screen-door.

3. You need to write a simple demo game play function that can demonstrate the process of a Peashooter damage an array of Zombies. Each time the plant attacks, the Zombies are damaged by a certain amount. For now, we assume there is no "projectile" involved, and the plant damages the Zombie directly, and thus there's no need for a "plant" class either. Also, we are not concerned about the Zombie damaging the plant either, so we are ignoring that feature. Details of the demo play function are specified below.
4. For simplicity purpose, we assume the Peashooter's damage is 25 each time it attacks. We also assume the **total health** of the 4 types of Zombies above are: 50, 75, 150, 75 respectively. Every time a Zombie gets damaged, its health is reduced by the value of the damage. Once the health of the Zombie is <= 0, the Zombie "dies".
5. Also, if the Zombie has an "accessory", the damage from the Peashooter will be applied to the "accessory" of the zombie first. Once the "accessory part" is "destroyed", the accessory will fall and the "Zombie with an accessory" should become a "Regular Zombie".

Read the assumptions carefully and complete your tasks on the next pages.

Notes: If you have questions/confusions regarding the assumptions, send me an email for clarification.

More importantly: Absolutely no plagiarism allowed, just as all other assignments and submissions. There would be zero tolerance in case of copy/pasting code that is not written by yourself. You may discuss with your classmates about general ideas, but make sure you don't give or receive specific details to or from each other.

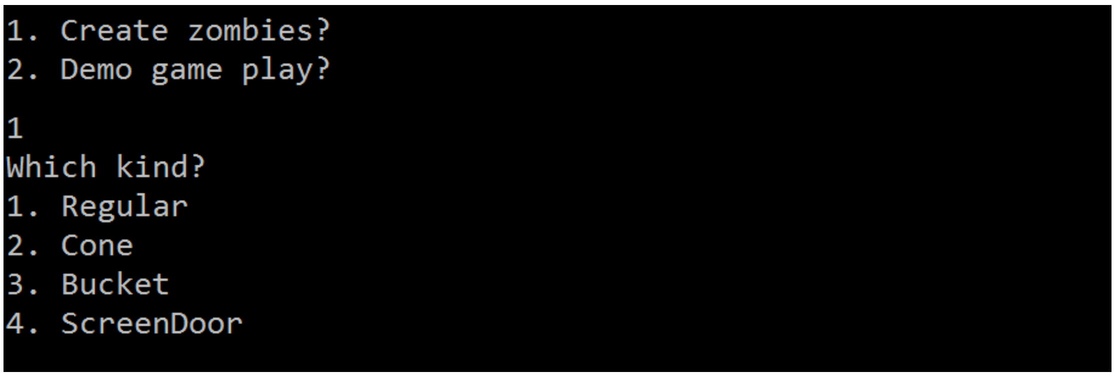
1. **[10pts]** Choose a Creational pattern we have covered (Factory/Abstract Factory/Builder) to create the 4 types of "Zombies". This time, follow Composite pattern to model the "Zombie"-related classes (please note this is not referring to the "array of Zombies" that the game will use to store all zombies created; this "array" is not the composite that we are interested in).

   Draw a class diagram for your solution using StarUML (or other UML tools). The diagram should connect the Creational pattern part, and the Composite part together. Also, include necessary attributes and operations in your diagram. Hint: the "Zombie"-related classes MUST have takeDamage(int d) and die() operations.

2. **[10pts]** Next to your diagram, list the match-ups between the classes in your diagram, and the ones shown in the pattern structure. For instance, "ZombieFactory" – Creator; "Zombie" – AbstractProduct; etc. This would be helpful for you to apply the patterns properly. To get full credits, you need to include the match-ups for **both your choice of Creational pattern (Factory/Abstract Factory/Builder) and the Composite pattern**.

   Remember that when applying the design patterns, not only should the classes match up, **the relationships (arrows between different classes), and the attributes/operations of the classes** should match up too. Hint: Compare your answer with the slides, see if all the arrows are the right shape/direction, for instance.

3. **[40pts]** Write an executable demo program that follows your design above. The program should provide a simple command line interface of several commands. A sample command line interface may look like this:

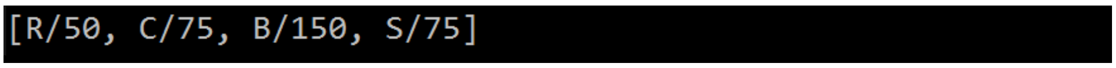


```
1. Create zombies?
2. Demo game play?

1
Which kind?
1. Regular
2. Cone
3. Bucket
4. ScreenDoor
```

   a) **[15pts]** One option in the commands should allow the user to create different types of Zombies. For instance, when selecting "Create zombies", the program should prompt the user which kind of zombie to create. The user may create several different zombies before stopping. All the zombies created should be saved in an array/vector.

      After the user is done creating zombies, the user shall return to the main menu. The program shall display the array of Zombies that the user just created, with their type and health specified.



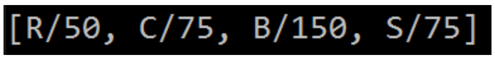
```
[R/50, C/75, B/150, S/75]
```

   b) **[25pts]** Next, the user can simulate a plant (Peashooter) attacking the zombie by selecting the "Demo game play" option. For this assignment, we assume there is only one Peashooter.

The zombies are attacked one at a time from left to right. For instance, in the array above, the Regular Zombie will be attacked first until it dies, then the Cone Zombie will be attacked, and so forth.

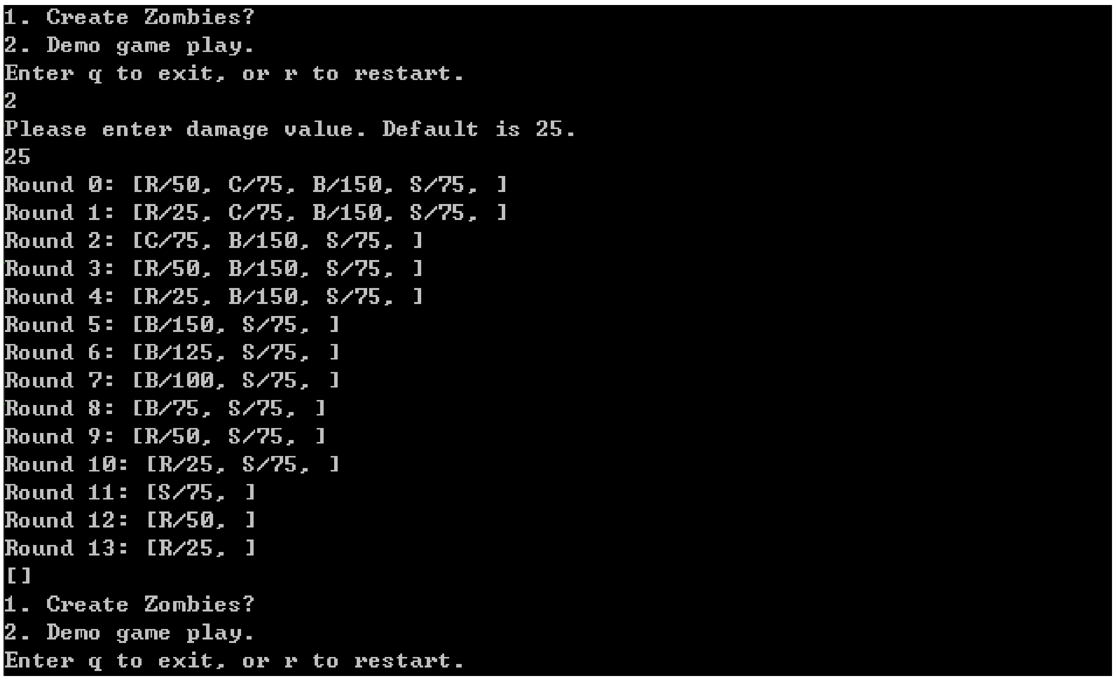
Use the command line output to simulate the attack process: each time the plant attacks, print out an updated array of the Zombies and their remaining health values. Remember, the Zombies with an "accessory" should change type after the accessory is destroyed/removed, i.e. turn from a Cone Zombie to a Regular Zombie once the Cone's health is gone, and this change should be reflected in your array display.

For example, these zombies:



```
[R/50, C/75, B/150, S/75]
```

After taking 25 damage three times, should be displayed as: [R/50, B/150, S/75], with the original first zombie dead and removed, and the second ConeZombie becoming a RegularZombie. See the screenshot below:



```
1. Create Zombies?
2. Demo game play.
Enter q to exit, or r to restart.
2
Please enter damage value. Default is 25.
25
Round 0: [R/50, C/75, B/150, S/75, ]
Round 1: [R/25, C/75, B/150, S/75, ]
Round 2: [C/75, B/150, S/75, ]
Round 3: [R/50, B/150, S/75, ]
Round 4: [R/25, B/150, S/75, ]
Round 5: [B/150, S/75, ]
Round 6: [B/125, S/75, ]
Round 7: [B/100, S/75, ]
Round 8: [B/75, S/75, ]
Round 9: [R/50, S/75, ]
Round 10: [R/25, S/75, ]
Round 11: [S/75, ]
Round 12: [R/50, ]
Round 13: [R/25, ]
[]
1. Create Zombies?
2. Demo game play.
Enter q to exit, or r to restart.
```

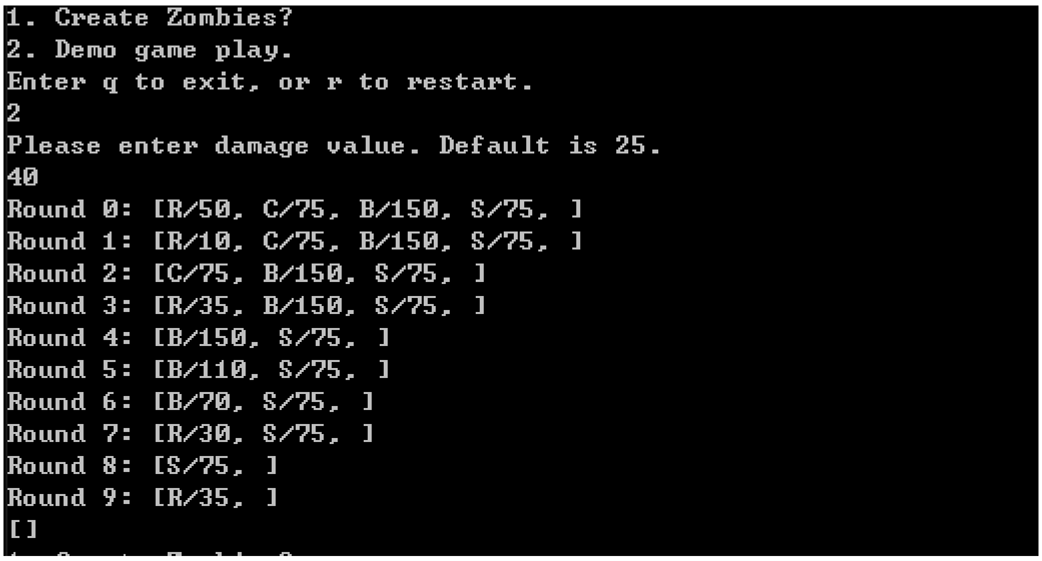
The demo should automatically continue until all the zombies are dead.
Hint: do not include a "Peashooter" class as there is no need. To simulate the attack, simply invoke the "takeDamage(int d)" function in a loop on your Zombies.

4. **[20pts]** What would change if we increased the damage of Peashooter to 40, instead of 25? Does this change impact your code and/or game logic at all? Write a short answer to this question, then accommodate this change into your program, allow the user to choose a

different damage value by entering an integer **after** selecting “Demo game play” and **before** the game begins.

Then, implement “leftover damage” in your program by taking advantage of the Composite pattern: consider a Cone Zombie with full health 75; after taking a damage of 40, it should become a Regular Zombie with health 35 – NOT a full health Regular Zombie with health 50.

See the screenshots below as an example when damage value is set to 40.



```
1. Create Zombies?
2. Demo game play.
Enter q to exit, or r to restart.
2
Please enter damage value. Default is 25.
40
Round 0: [R/50, C/75, B/150, S/75, ]
Round 1: [R/10, C/75, B/150, S/75, ]
Round 2: [C/75, B/150, S/75, ]
Round 3: [R/35, B/150, S/75, ]
Round 4: [B/150, S/75, ]
Round 5: [B/110, S/75, ]
Round 6: [B/70, S/75, ]
Round 7: [R/30, S/75, ]
Round 8: [S/75, ]
Round 9: [R/35, ]
[]
```

5. **[20pts]** Say we want to introduce a new plant: Watermelon.

The way a Watermelon attacks is that it catapults a watermelon above and hit the Zombies from the top. Therefore, for Regular, Cone and Bucket Zombies, the Watermelon would function the same way as the Peashooter. However, for the Screen-Door Zombie, the Watermelon “goes above” the screen-door it’s holding, and hit the Zombie directly.

What does the new feature change about your program? Does the Composite pattern still work? DO NOT change your code for this question. However, write up a short explanation on how this new change would impact your design and implementation. What kind of modification would be necessary?

- **Submission Details**
  1. Submit your class diagram and short answers in a well-formatted PDF file, with your name and student ID included. The diagram must be drawn with StarUML and show correct usage of the UML notations, as well as sufficient details for specified requirements.

2. For the coding part, write your program in either C# or Java **only**. Other languages are not allowed. Include a Readme.txt file explains how to run and test your program. You might assume I/TA will not test on illegal inputs. The key here is to test on your understanding on the design patterns.
3. Your code need to also correspond to your design of class diagram.
4. Zip both the PDF and your entire coding project (include all source codes and files necessary, and the Readme.txt), name it "ID-FirstnameLastname-487Assignment4.zip", and submit it on to the Canvas dropbox before the due date.
5. Your program should be executable on a Windows machine with Visual Studio 2019 or Eclipse properly installed. If your program does not execute on the TA's machine for any reasons, that might impact your grades. If you are packaging a Visual Studio project, make sure it can run as a fresh project by testing it yourself first.
6. Finally, and once again, do not plagiarize.