School of Science, Computing and Engineering Technologies

Object Oriented Programming

Hurdle Task 2: Semester Test Resit

Overview

Note: This hurdle task is a time-bound test. You have a 48 hour window during week 12 to complete it.

- **If you receive a Pass grade**, you have passed the hurdle and can include the test as evidence of that in your portfolio.
- **If you receive a Fail grade**, you have not passed the hurdle. The maximum mark you can receive for this unit is 45.

In this unit, you have been using object-oriented programming to implement all of your programs. For this task, you will need to show your understanding of the OO principles.

Purpose:	Demonstrate your understanding of object-oriented programming and the core concepts of object-oriented design.
Task:	You must complete two tasks. The first is a coding task, to be submitted as C# source code files and a screenshot showing your program's output. The second task asks for a written response, to be submitted as a PDF.
Deadline:	This task should be completed during week 12 — see Canvas for the assignment window.

Submission Details

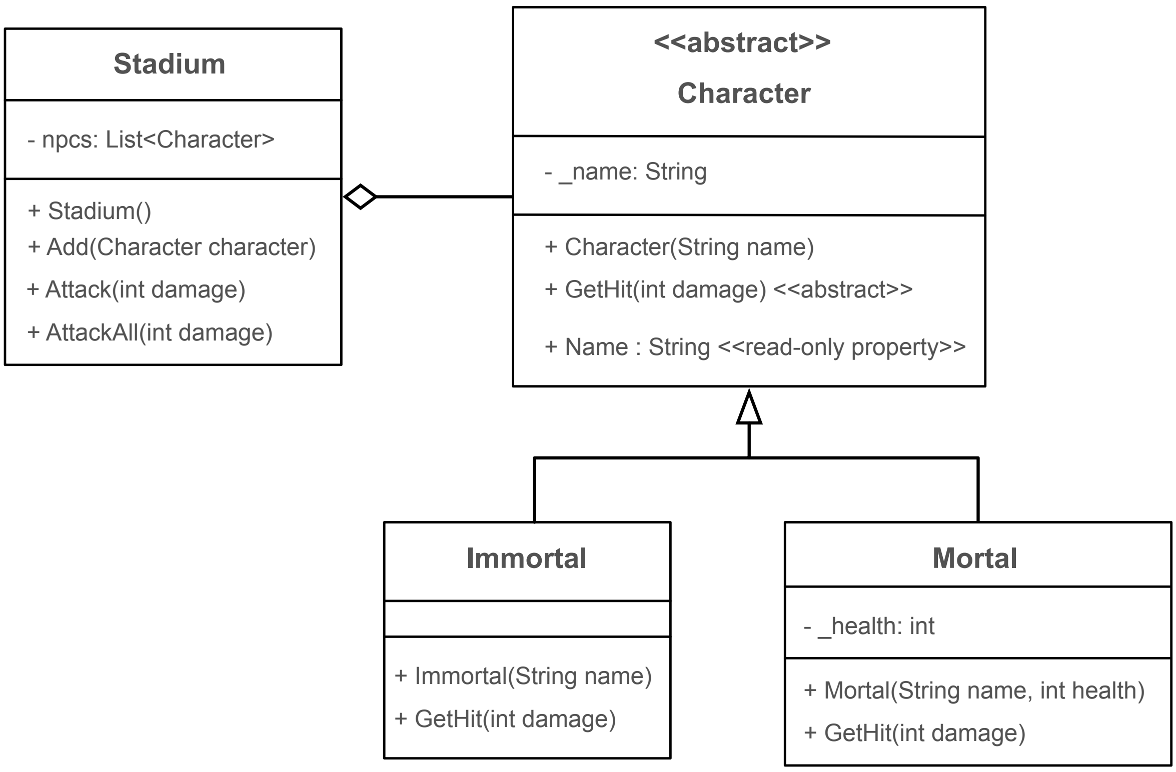
Please print your solution to PDF and combine it with the screenshots taken for this test.

- Task 1:
  - C# code files of the classes created
  - A screenshot of the program output
- Task 2:
  - A PDF document containing your written answers.

Make sure that you submit code that is readable, follows the universal task requirements, and is appropriately documented.

Task 1

Consider the following program design:



Class *Stadium* maintains a list of non-player characters and processes any attacks. Initially, the list ***_npcs*** is empty. Non-player characters can be added by using the ***Add*** method of class *Stadium*.

When you tell an object of class *Stadium* to ***Attack***, one of two things will happen:

- If the list of non-player characters is not empty, then
  - The message “Bring it on!” is sent to the console.
  - The first character in the list is told to ***GetHit*** using the specified amount of ***damage***.
- Else
  - The message “Not very effective…” is sent to the console.

When you tell an object of class *Stadium* to ***AttackAll***, one of two things will happen:

- If the list of non-player characters is not empty, then
  - The message “Charge!” is sent to the console.
  - Every character in ***_npcs*** is told to ***GetHit*** using the specified amount of ***damage***.
- Else
  - The message “”Why?” is sent to the console.

The abstract class *Character* defines the common features of non-player characters. There are two classes of non-player characters: *Mortal* and *Immortal*.

Class *Mortal* is a kind of *Character*. Objects of class Mortal also define a field ***_health***. When creating new objects of class Mortal, use 15 as initial value for health. When an object of class *Mortal* is told to ***GetHit***, then one of two thing will happen:

- If the *Mortal*'s health is greater than zero, then
    - The message "Odysseus: Ow!" is sent to the console, where "Odysseus" is the ***_name*** of the *Mortal*.
    - The field ***_health*** is reduced by the value ***damage***.
- Else
    - The message "Odysseus: You already got me!" is sent to the console, where "Odysseus" is the ***_name*** of the *Mortal*.

Class Immortal is a kind of Character. When an object of class *Immortal* is told to ***GetHit***, then

- The message "Zeus: Ha, Nice try." is sent to the console, where "Zeus" is the ***_name*** of the *Immortal*.

Your task is to:

1. Implement the program described above. You must write code for all classes, methods, fields, and constructors.
2. Write a ***Main*** method to test your implementation:
    a. Create a *Stadium* object.
    b. Tell the *Stadium* object to ***Attack*** with 10 ***damage***.
    c. Tell the *Stadium* object to ***AttackAll*** with 5 ***damage***.
    d. Add three *Mortal* objects to the *Stadium* object.
    e. Add one *Immortal* object to the *Stadium* object.
    f. Tell the *Stadium* object to ***AttackAll*** with 8 ***damage***.
    g. Tell the *Stadium* object to ***Attack*** with 10 ***damage***.
    h. Tell the *Stadium* object to ***AttackAll*** with 9 ***damage***.

Task 2

1. Explain four fundamental principles of object-oriented programming. For each principle, refer to some task that you have completed for COS20007 and elaborate how the principle is used to achieve the objective of the task.

   You are not allowed to merely reuse previous work. The responses in this task must be original. Repeating previous work verbatim is not permissible.

Note: Write your answers in your own words. You can use as many reference materials as you see fit, but you not copy text from anywhere or just ask a chat bot.

Tip: Be concise. Explain each principle in a single paragraph.

Assessment Criteria

Outcome	Requirements
Pass	Code compiles and runs, and mostly matches the provided design. At least 3 of the 4 core concepts are explained correctly with relevant and correct examples from the unit tasks.
Fail	Pass criteria not met. OR The submission was not in the correct format.