School of Science, Computing and Engineering Technologies

Object Oriented Programming

Hurdle Task 1: Semester Test

Overview

Note: This hurdle task is a time-bound test. You have a 48 hour window during week 8 to complete it.

- **If you receive a Complete grade**, you have passed the hurdle and can include the test as evidence of that in your portfolio.
- **If you receive a Fix grade**, you must correct all issues and get your test signed off as Complete to meet the hurdle requirements. If you do not get it marked as Complete during the teaching period, you must include a copy of your corrections in your portfolio to demonstrate that you have addressed the issues raised. Failure to do this will result in an overall fail grade for the unit.

If you receive a Redo grade, you must pass a re-sit test that will occur in week 12 in order to meet the hurdle requirements.

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, and the second is a series of short answer questions.
Deadline:	This task should be completed during week 8 — see Canvas for the assignment window.

Submission Details

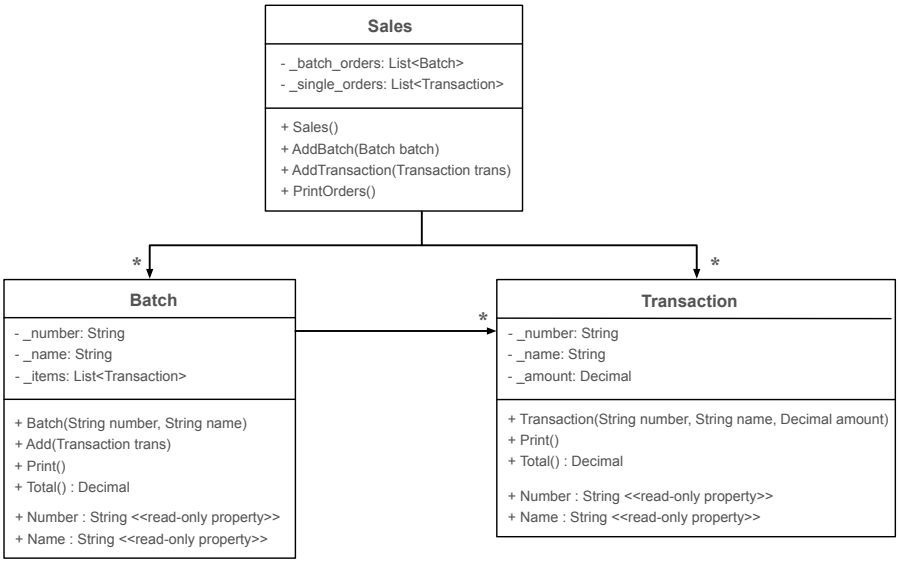
Please print your solution to PDF and combine it with the screenshots taken for this test.

- Task 1:
  - C# code files of the classes created
  - An image of your modified UML diagram
  - 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 UML diagram:



Sales is a class that contains knowledge of purchase orders that a business has received. There are two types of purchase orders: single transaction orders and batch orders, which the class *Sales* records separately in two dedicated lists. Batch orders are instances of class *Batch* and can be added to its corresponding list, ***_batch_orders***, using method ***AddBatch***. Single transaction orders, on the other hand, are instances of class *Transaction* and can be added to its corresponding list, ***_single_orders***, using method ***AddTransaction***. All orders can be printed to the console via method ***PrintOrders***.

Class *Batch* defines three member variables:

- ***_number***, a string that identifies this order,
- ***_name***, a string that captures the purpose of this order, and
- ***_items***, a list of single orders attached to the bulk order.

In addition, class *Batch* defines the methods

- ***Add*** to add a single order,
- ***Print*** to print this order, and
- ***Total*** to return the total sum of this order.

Finally, class *Batch* defines two read-only getters to obtain the ***Number*** and ***Name*** of the order.

Class *Transaction* is similar. Its member variables are:

- ***_number***, a string that identifies this order,
- ***_name***, a string that describes the product in this order, and
- ***_amount***, a decimal value to store the total of the single transaction.

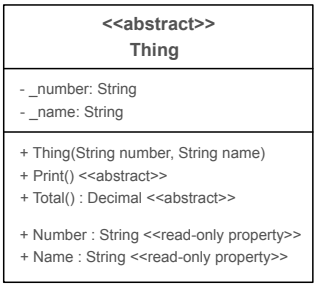
Class Transaction defines a ***Print*** and a ***Total*** method, which print the order and return the amount of this order, respectively. The read-only getters return the ***Number*** and ***Name*** of a single transaction order.

A sample output of an application implementing the above UML diagram may look like the following (result of telling a *Sales* object to ***PrintOrders***):

```
Sales:
Batch sale: #2024x00001, CompSci Books
#1, Deep Learning in Python, $67.90
#2, C# in Action, $54.10
#3, Design Patterns, $129.75
Total: $251.75
Batch sale: #2024x00002, Fantasy Books
Empty order.
#00-0001, Compilers, $134.60
#10-0003, Hunger Games 1-3, $45.00
#15-0020, Learning Blender, $56.90
Sales total: $488.25
```

You may have noticed a peculiar feature duplication in the design. In addition, batch orders are limited to containing single transaction orders. This is an arbitrary limitation. Batch orders should be able to contain nested batch orders. Consequently, class *Batch* should allow for both, objects of class *Batch* and objects of class *Transaction*, to be stored (i.e., added to ***_items***).

To achieve this, you need to redesign the system. Start with the abstract class *Thing* as shown in the following UML diagram:



Restructure the current solution as follows:

1. Implement the abstract class *Thing*.
2. Change class Batch so that it stores instances of class *Thing* rather than class *Transaction*.
3. Make class *Batch* and class *Transaction* subclasses of class *Thing*. Apply the necessary refactoring steps.
4. Change class Sales to maintain only a single list ***_orders*** of *Thing* objects.
5. Replace the methods ***AddBatch*** and ***AddTransaction*** with a single method ***Add*** in class *Sales*.
6. Revise the implementation of ***PrintOrders*** in class Sales.
7. Write a simple ***Main*** program to demonstrate that your new design works. The ***Main*** program must
    a. Create a *Sales* object.
    b. Add batch orders to the *Sales* object.
    c. Add single transaction orders to the Sales object.
    d. Must create one nested batch order.

e. Add an empty batch order.

f. Tell ***PrintOrders*** to the Sales object.

You are required to:

a) Provide a new UML class diagram for your updated design (it can be drawn by hand).

b) Write the code for all classes, including the abstract class Thing, and all fields, constructors, methods, and properties.

c) Write a simple ***Main*** method as described above.

d) Take a screenshot of the output of your program.

Task 2

1. Describe the principle of polymorphism and how and where it is being used in Task 1.
2. What is wrong with the class name *Thing*? Suggest a better name and explain the reasoning behind your answer.
3. What is abstraction and how and where it is being used in Task 1.
4. Can you think of a scenario or system design that resembles Task 1? Look a the classes and their interaction in Task 1 and identify a real-world system or approach that uses a similar relationship.

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.

Assessment Criteria

Outcome	Requirements
Pass	All parts of the submission are correct.
Fix and Resubmit	The submission clearly demonstrates that the author understands the key OO concepts and how to apply them in code, but there are some issues.
Redo	The submission does not clearly demonstrate that the author understands the key OO concepts and how to apply them in code. There are likely clear mismatches between the provided design and the code submitted, and one or more of the written answers are incorrect. UML may not match the code submitted. OR The submission was not in the correct format.