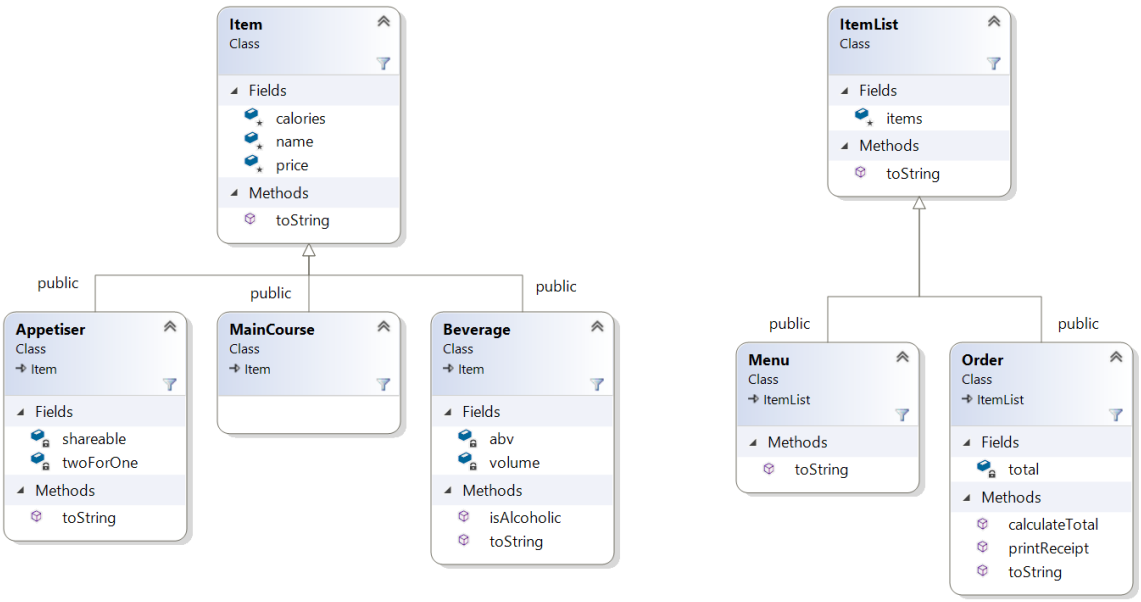
Module Code & Title: CMP2801M Advanced Programming RESIT

Contribution to Final Module Mark: 100%

Description of Assessment Task and Purpose:

Your task is to implement and critically evaluate a simplified take-away food ordering system in C++. The purpose of this assessment is to demonstrate your understanding of advanced programming techniques such as low-level memory management, input/output, and object-oriented design principles. Users will interact with your application via a command line interface (CLI), rather than a graphical interface.

Once implemented, users should be able to use the system to view a menu, create orders, add/remove items from their order, and get a receipt for once they "checkout". Your implementation will be marked based on the completeness of functionality (as per the brief), the overall program structure, and your demonstration of advanced object-oriented design principles. You are also asked to produce a short report detailing the functionality you've managed to implement (i.e., whether it deviates from the brief), the overall structure of your program, and the tests you've performed to verify its correctness.



The inheritance hierarchy of the application you must implement is shown above. You are provided with a skeleton program called **`Takeaway.cpp`**, and a file containing food and drink items for sale in **`menu.csv`**. You may edit the content of **`Takeaway.cpp`**, but the general structure should be left unchanged. Do NOT edit **`menu.csv`**. You may need to implement additional functions to those shown in order to produce a working implementation.

The **`menu.csv`** file continues one row per item available for purchase, with the following columns; *type, name, price, calories, shareable, 2-4-1, volume (ml), alcohol by volume (abv).*

To get started, implement an **`Item`** class which stores the general information (**`name`**, **`calories`**, **`price`**) about the food and drink items available to purchase. You should then implement the three derived classes - **`Appetiser`**, **`MainCourse`** and **`Beverage`**. The **`Appetiser`** class contains two attributes; **`shareable`** and **`twoForOne`**. The **`Beverage`** class also contains two attributes; **`abv`** and **`volume`**. The **`MainCourse`** class has no additional attributes. You should consult the **`menu.csv`** file to identify appropriate data types for each attribute.

Next is the **ItemList** abstract class, which provides an interface for storing and manipulating food and drink items. Instances of the **Item** class should be stored by reference as a vector of pointers. The sole member function **toString** should be pure virtual.

The derived **Menu** class should contain a function to 'load' a menu object from a file path. This should be provided as an argument to the constructor (in your main code, this will be **menu.csv**). Each row corresponds to a different item to be created and added to a vector of pointers. You should ensure that the correct object type is created based on the character in the first column – **'a'** is an appetiser, **'m'** is a main course, and **'b'** is a beverage. Implement a **toString** function to display the whole menu in attractive way, organised by item type.

The derived **Order** class should contain functions to **add** and **remove** items from an order based on their numerical position in the respective item list (starting at one, rather than zero). Each time an item is added or removed, inform the user of the result and recalculate the overall total of the items in their order. The **calculateTotal** function should consider the number of items in the order that are eligible for a "2-4-1" discount, i.e., if a user adds two appetisers to their order where **twoForOne == true**, one of them should be free of charge. You may assume that all such items are the same price.

Implement a **toString** function to display the items in the order, along with the total price and the savings made. The **printReceipt** function should write the user's order to a text file called **receipt.txt**, using the output from the **toString** function. In the output, the total and savings should be displayed in the following way:

```
Savings: £x.xx
Total: £x.xx
```

The remaining program logic should be implemented in **Takeaway.cpp**, which should initialise a **Menu** object from **menu.csv** and an (initially empty) **Order** object. You should modify the code to accept input commands with the following syntax:

`menu` - display the menu to the user
`add [INDEX]` - add an item to the order by numeric index in the menu (starting at 1)
`remove [INDEX]` - remove item from order by numeric index in the order (starting at 1)
`checkout` - display the items in the user's order, the price, and discount savings
`help` - display a help menu for the user with the available options
`exit` - terminate the program gracefully

Upon checkout, give the user the option to either complete their order (output their receipt and terminate gracefully) or go back and modify it. Additional marks may be awarded for the use of STL/contemporary C++ components in your solution, as well as attempts at the following stretch tasks:

- Allow users to add and remove multiple items at once, e.g., `add 2 5 7`
- Modify the **calculateTotal** function so that 2-4-1 discounts cause only the cheapest eligible item(s) to be free. You may add a row to **menu.csv** in order to test this behaviour, but do not modify the other data in the file.
- Add commands to sort the menu by item price in ascending/or descending order, using some combination of operator overloading/functional programming in your implementation.

Sample execution

1. User selects the **`menu`** option, producing output such as shown below:

```
---------------Appetisers---------------
(1) Nachos: £4.99, 600 cal (shareable)
(2) Buffalo wings: £3.99, 450 cal (2-4-1)
(3) Garlic bread: £3.99, 500 cal (2-4-1)
---------------Main dishes---------------
(4) Burger: £9.99, 950 cal
(5) Mac & cheese: £7.99, 850 cal
(6) Fish & chips: £8.99, 1000 cal
(7) Chicken tikka masala: £6.99, 700 cal
---------------Beverages---------------
(8) Lager: £3.50, 200 cal (568ml, 4.5% abv)
(9) White wine: £4.00, 150 cal (175ml, 11.5% abv)
(10) Red wine: £4.00, 170 cal (175ml, 12.5% abv)
(11) Coke: £2.50, 140 cal (330ml)
(12) Water: £1.50, 0 cal (330ml)
```

2. User adds an appetiser, e.g., **`add 2`**
   The corresponding pointer is retrieved from the **`Menu`** object
   The pointer is then added to the **`Order`** object
   Output: `Buffalo wings added to order!`

3. User adds a second appetiser, e.g., **`add 3`**
   The corresponding pointer is retrieved from the **`Menu`** object
   The pointer is then added to the **`Order`** object
   Output: `Garlic bread added to order!`

4. User adds a main course, e.g., **`add 4`**
   The corresponding pointer is retrieved from the **`Menu`** object
   The pointer is then added to the **`Order`** object
   Output: `Burger added to order!`

5. User selects **`checkout`**, producing output like the following:

```
===== Checkout =====
(1) Buffalo wings: £3.99, 450 cal (2-4-1)
(2) Garlic bread: £3.99, 500 cal (2-4-1)
(3) Burger: £9.99, 950 cal
-------------
2-4-1 discount applied! Savings: £3.99
Total: £13.98

Do you want to place your order?
Type 'y' to confirm, or 'n' to go back and modify it.
```

6. User confirms their order (**`y`**), and the string shown above is written to a file called **`receipt.txt`**. This may overwrite any existing file that may be present.

General considerations

- You must follow the guidelines in this brief as closely as possible, including the names of files, classes, functions and commands. Your code will undergo automated testing, which could fail if components are named incorrectly.

- While you may discuss your solution with your peers, please note that this an individual assessment and therefore all code and report content must be your own. Both components of your assessment will undergo checking for plagiarism.

Learning Outcomes Assessed:

[LO 1] Apply concepts of advanced software development and programming methods to computational problems;
[LO 2] Use advanced object-oriented principles and programming techniques in software development;
[LO 3] Apply advanced logical and mathematical techniques in the development of software solutions.

Knowledge & Skills Assessed:

- Input/output streams and C-strings
- Low-level memory management/pointers
- Evaluation strategies (call-by-value/call-by-reference)
- Effective use of encapsulation and access modifiers
- Inheritance, virtual functions and abstract classes
- Operator overloading and functional programming

Assessment Submission Instructions:

You should submit your work as a single ".ZIP" file to the "*Assessment Item 1 RESIT Supporting Documentation Upload"* section. The written report should be submitted separately on Blackboard to the "*Assessment Item 1 Upload"*.

a) The ZIP file which is uploaded to *Assessment 1 RESIT Supporting Documentation Upload* should only contain your code and menu.csv. Do not include your report here.
b) The pdf report to be uploaded to *Assessment Item 1 RESIT Upload* should contain:
    a. A contents page (not included in the two-page limit)
    b. A basic design for the application (1.5 pages) including:
        i. A written description of the application.
        ii. Your class structure, program design and algorithmic choices
        iii. Which of the additional tasks you attempted, and how you solved them
    c. A description of your evaluation of the application (1/2 page):
        i. A description of what tests you did to assure that your program runs.
        ii. An evaluation of the time complexity of your solution.

DO NOT include this briefing document with your submission.

Date for Return of Feedback:

As this is a resit assessment no feedback will be returned. Your grade will be submitted to the school admin team and made available to you after the resit board completes. Please contact your personal tutor for more information on resits.

Format for Assessment:

Your submission should consist of the following:

1) A zipped directory containing ONLY the header (.h) and source (.cpp) files for your application, and the menu.csv file. Do not submit any other files (e.g., build files, VS project files). Other compressed formats (tar.gz, rar, etc.) will not be accepted.
2) A report in PDF format detailing your program structure and solution to the problem. A template for this report is given in a separate document.

Your zipped directory should be compressed using Windows' *Send to → Compressed (zipped) folder* option, or in UNIX-based operating systems: `zip -r compressed.zip project_folder/`

Your report must be NO MORE than 2 pages long using the supplied template and should adhere to the headings and structure set out in the report template document. There will be a penalty for reports longer than 2 pages.

Feedback Format:

Prior to submission, oral feedback may be provided by academics during online workshops and office hours, and from demonstrators via the online helpdesk. Written feedback on your final submission will be provided via Blackboard, where commentary will be provided on:

- Overall program structure and adherence to C++ conventions
- Missing or incomplete functionality, and areas where it may be improved
- Effective use of object-oriented design principles in accordance with the brief
- Any attempts at the additional tasks and use of STL/contemporary C++ components
- How well comments were used to provide explanations of the program's logic
- The quality of your technical writing and critical evaluation of your implementation

Additional Information for Completion of Assessment:

Please make sure you have a clear understanding of the grading principles for this component as detailed in the accompanying Criterion Reference Grid. If you are stuck on any component of the assessment, please consult the recommended reading lists, lecture materials (slides, recorded lectures) and workshop tasks in the first instance. Please be advised that the delivery team is not here to debug/fix your code, but to give general advice and further explanation of concepts you may be finding difficult.

Assessment Support Information:

Resits are completed independently over the summer period.

Important Information on Dishonesty & Plagiarism:

University of Lincoln Regulations define plagiarism as 'the passing off of another person's thoughts, ideas, writings or images as one's own...Examples of plagiarism include the unacknowledged use of another person's material whether in original or summary form. Plagiarism also includes the copying of another student's work'.

Collusion is defined as when a student submits work for assessment done in collaboration with another person as entirely their own work or collaborates with another student to complete work which is submitted as that other student's work. Collusion does not apply in the case of the submission of group projects, or assessments that are intended to be produced collaboratively.

Plagiarism and collusion is a serious offence and is treated by the University as a form of academic dishonesty. Students are directed to the University Regulations for details of the procedures and penalties involved.

For further information, see www.plagiarism.org