UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program Design: Introduction

Practical 6 Specifications: Polymorphism

Release date: 02-10-2023 at 06:00
Due date: 06-10-2023 at 23:59
Total Marks: 390

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually; no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of C++'s built-in data structures. Doing so will result in a mark of zero. You may only make use of 1-dimensional native arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile, you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experiences a runtime error, you will be awarded a mark of zero. Runtime errors are considered unsafe programming.
- Read the entire specification before you start coding.
- **Ensure your code compiles with C++98**

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers to avoid any misunderstanding.** Also note that the OOP principle of code reuse does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

On completion of this practical, you will have gained experience with the following:

- Polymorphism
- Virtual and Pure virtual functions
- Abstract classes

4 Introduction

Polymorphism is an important feature of OOP languages, which allows users to create base classes, and then children classes, which inherit from the base class. This allows users to re-use the code they already wrote, without needing to copy and paste or call the functions, if implemented properly.

C++ also allows pure virtual functions, which are functions that have a signature in the base class but are not implemented. Thus all children's classes must implement these functions. Because the base class does not implement all of the functions, meaning that the base class is abstract. Thus the C++ compiler won't allow you to create objects of this class, but you can use pointers of this class, and save the children object inside these.

5 Tasks

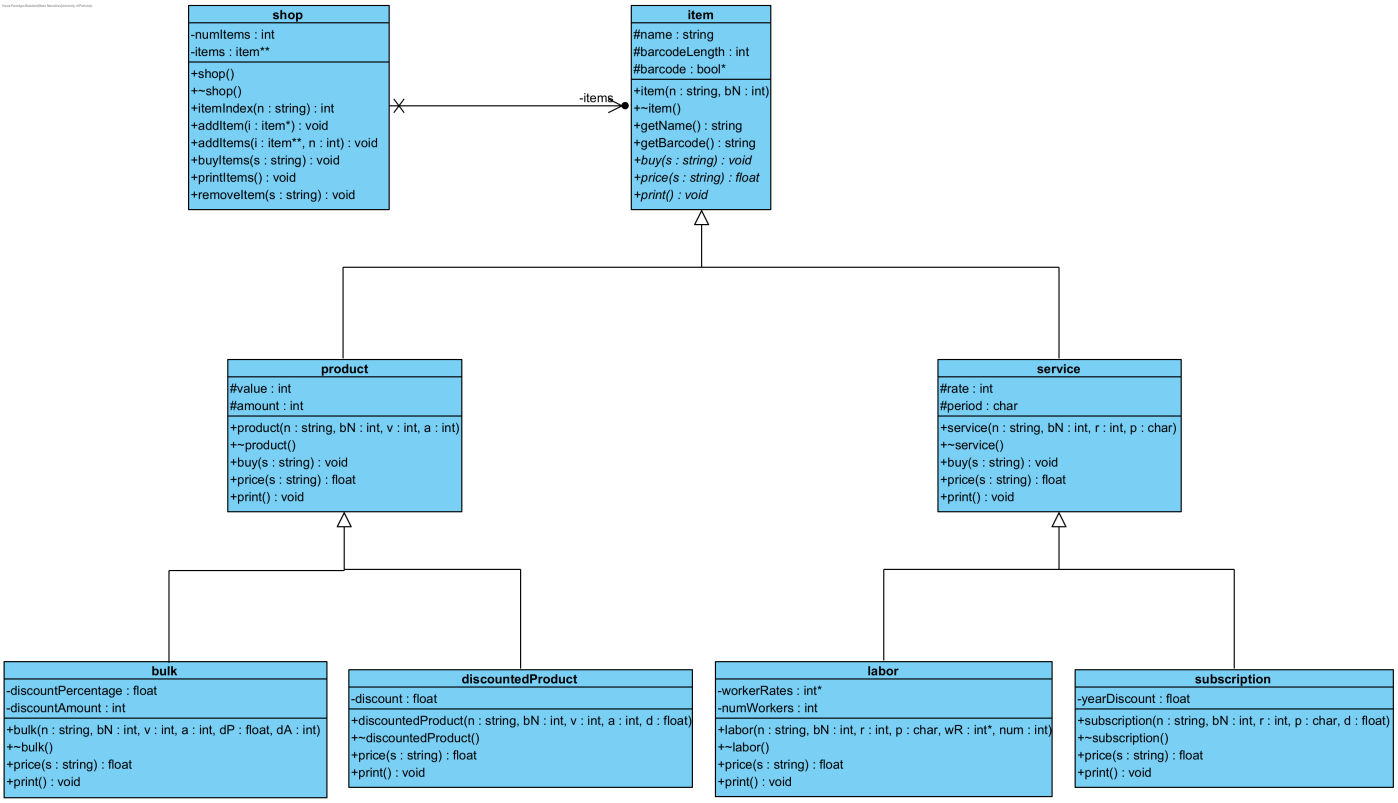

Figure 1: UML

All polymorphism shown in the diagram is **public** inheritance.

Note : The software used to create this does not indicate virtual functions. These will be mentioned in their section, but you may assume that all functions that are overwritten by child classes are virtual. Additionally, the functions in *itallics* are pure virtual.

5.1 item

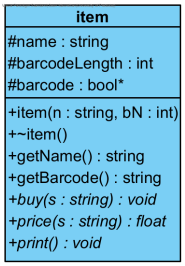

Figure 2: item UML

- Members
  - name: string
    - This is the name of the item.
  - barcodeLength: int
    - This is the length of the barcode.
    - The length of the barcode array is always stored inside this.
  - barcode : bool*
    - This is a dynamic boolean array used to store the barcode.
    - This array has length barcodeLength.
- Functions
  - item(n: string, bN: int)
    - This is the item class constructor.
    - Set the name equal to n.
    - The passed-in integer should be converted to binary.
    - The barcodeLength should be set to the minimum number of bits needed to represent the passed-in number in its binary form.
    - The barcode array should then represent the passed-in number in its binary form.
    - Example: If the number 13 is passed in the barcode should be:

```
[True, True, False, True]
```

  - ~item()
    - This is the item class destructor.
  - getName(): string
    - Return the corresponding member variable.

- getBarcode(): string
  - This returns the string representation of the barcode.
  - The format is as follows:
    - The barcode should be printed, inside square brackets.
    - A space represents False, and a "|" represents True.
    - After the closing square bracket, print the original barcode number inside brackets.
  - Example: If the barcode number is 13, then this function should return:

```
[||  |](13)
```

- buy(s: string): void
  - This is a pure virtual function.
- price(s: string): float
  - This is a pure virtual function.
- print(): void
  - This is a pure virtual function.

5.2 product

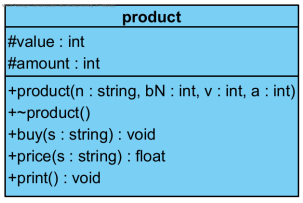

Figure 3: product UML

- Members
  - value: int
    - This is the price for one item.
  - amount: int
    - This is the number of items currently available.
- Functions
  - product(n: string, bN: int, v: int, a: int)
    - This is the product class constructor.
    - n and bN should be used for the parent constructor.
    - v is the value, and a is the amount for this object.
  - ~product()
    - This is the product class destructor.

- buy(s: string): void
  - This is a virtual function.
  - The passed-in parameter should be converted to an integer.
  - If the passed-in number is larger than the amount that is left over, then only buy the amount that is left.
  - After buying that many products, the amount should be updated to reflect this change.
  - See the provided example for the format.
- price(s: string): float
  - This is a virtual function.
  - This calculates the price for a purchase.
  - The passed-in parameter is the number of items that a customer wants to buy.
  - You may assume that the passed-in parameter is a valid integer and that it is non-negative.
  - The amount of items left should be ignored.
  - **Do not** change the amount of items left.
- print(): void
  - This is a virtual function.
  - This will print out the information regarding the product.
  - See the provided example for the format.

5.3 service

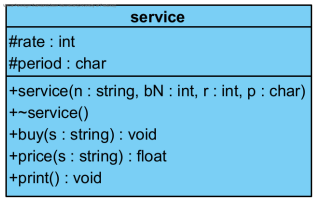

Figure 4: service UML

- Members
  - rate: int
    - This is the rate per unit of time given by the period for this service.
  - period: char
    - This is the time unit that the rate is given in.
    - Note that it can be any character. If the character is not in `[h,d,m,w,y]`, you may assume that it is per year.
    - **Note :** For this practical you **must** assume the following:
      - A day has 24 hours.
      - A week as 7 days.
      - A month has 4 weeks.
      - A year has 12 months.
- Functions
  - service(n: string, bN: int, r:int , p:char)
    - This is the service class constructor
    - *n* and *bN* should be used for the parent constructor.
    - The *rate* and *period* should be set using the passed-in parameters.
  - ∼service()
    - This is the service class destructor.
  - buy(s: string): void
    - This is a virtual function.
    - The passed-in string will have the following format. Note that values inside curly braces should be extracted and that spaces are replaced by _'s. Time refers to how many periods you are buying.

      ```
      {time}_{period}
      ```

      For example:

      ```
      10 h
      ```

    - The period is represented using the lowercase of the first letter of the word. If the character is not in "hdwm", then you may assume that the period is in years.
    - See the provided example for the format.

- price(s: string): float
  - This is a virtual function.
  - The passed-in string is the same format as buy().
  - Use the price per unit of time to calculate the price of the service.
- print(): void
  - This is a virtual function.
  - This will print out the information about the service.
  - See the provided example for the format.

5.4 bulk

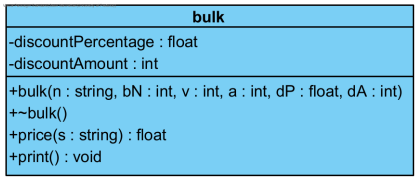

Figure 5: bulk UML

- Members
  - discountPercentage: float
    - This is the discount that is given for bulk purchases.
    - This float will always be between 0 and 1.
    - A 10% discount will be represented by the value 0.10.
  - discountAmount: int
    - This is the amount of items that need to be bought for a discount to apply.
- Functions
  - bulk(n: string, bN: int, v: int, a: int, dP: float, dA: int)
    - This is the bulk class constructor.
    - dP is the discountPercentage, and dA is the discountAmount.
    - The other parameters are used in the parent constructor.
  - ∼bulk();
    - This is the bulk class destructor.

- price(s: string): float
  - The passed-in string should be converted to an integer. You may assume the string is a valid non-negative integer.
  - The discount should be applied in batches. Thus for every group of size discountAmount a discount is applied to that group.
  - The amount of items left should be ignored.
  - Example:
    - If you have a bulk object with discountPercentage = 0.1, and discountAmount = 8, and price = 10.
    - If you buy 20 items of this object, the total is calculated is follows:

```
Group 1: 8 items at 10% discount = R 72.00
Group 2: 8 items at 10% discount = R 72.00
Group 3: 4 items at  0% discount = R 40.00
                           Total = R184.00
```

- print(): void
  - This will print out the information about the bulk product.
  - See the provided example for the format.

5.5 discountedProduct

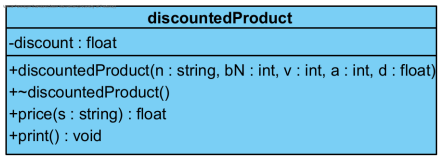

Figure 6: discountedProduct UML

- Members
  - discount: float
    - This is the percentage discount that is applied to every purchase of this item.
    - The discount is a float value between 0 and 1.
- Functions
  - discountedProduct(n: string, bN: int, v: int, a: int, d:float)
    - This is the discountedProduct class constructor.
    - d is the discount.
    - The rest of the parameters should be used for the parent constructor.
  - ~discountedProduct
    - This is the discountedProduct class destructor.
  - price(s: string): float
    - The amount of items left should be ignored.
    - The passed-in parameter is a non-negative valid integer.
    - The discount should be applied on all items bought for this classs.
  - print(): void
    - This will print out the information regarding the discounted product.
    - See the provided example for the format.

5.6 labor

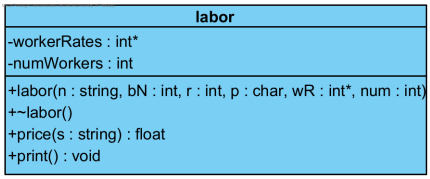

Figure 7: labor UML

- Members
  - workerRates: int*
    - This is an integer array that stores the worker's wages.
    - The size of this array is numWorkers.
  - numWorkers: int
    - This is the size of the workerRates array.
- Functions
  - labor(n: string, bN: int, r: int, p: char, wR: int*, num: int)
    - This is the labor class constructor.
    - *num* is the number of workers.
    - *wR* is the wages of the workers, a deep copy should be made.
    - The rest of the parameters are used in the parent class.
  - ~labor()
    - This is the labor class destructor.

- price(s: string): float
  - The passed-in parameter follows the same rules as the parent class' price function.
  - The workerRates array indicates by how much the base rate (the base rate is the rate in the parent class) is multiplied by for each worker.
  - Example

```
If we have a labor object with rate R10/h, and a workerRates array =
    [1,2,3].
If we buy 5 hours of labor its calculated as:
Worker 1: R50
Worker 2: R100
Worker 3: R150
Total   : R300
```

  - print(): void
    - This will print out the information regarding the labor.
    - See the provided example for the format.

5.7 subscription

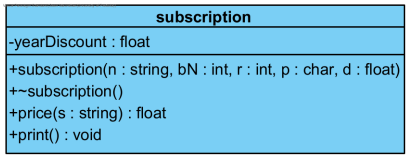

Figure 8: Subscription UML

- Members
  - yearDiscount: float
    - This is the discount that is applied if yearly subscriptions are bought.
- Functions
  - subscription(n: string, bN: int, r:int, p:char, d:float)
    - This is the subscription class constructor.
    - d is the yearlyDiscount.
    - The rest of the parameters are used in the parent constructor.
  - ~subscription
    - This is the subscription class destructor.

- price(s: string): float
  - The passed-in parameter has the same format as the price function in service.
  - The discount only applies if the passed-in period is treated as years.
  - Example

```
2 y: Discount applied
24 m: Discount NOT applied
```

- print(): void
  - This will print out the information regarding the subscription.
  - See the provided example for the format.

5.8 shop

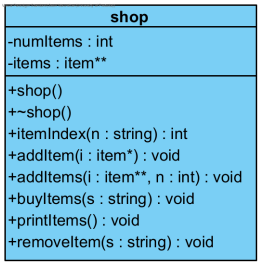

Figure 9: shop UML

- Members
  - items: item**
    - This is a 1D array of dynamic term objects.
    - This array will be of size numItems, and there will be no NULL values inside the array.
  - numItems: int
    - This is the number of items in the shop.
- Functions
  - shop()
    - This is the shop class constructor.
    - Set the members to show that there are 0 items in the shop.
  - ~shop()
    - This is the shop class destructor.
  - itemIndex(n: string): int
    - If there is an item in the shop with the same name as the passed-in parameter, return the index of that item, otherwise return -1.
  - addItem(i: item*): void
    - If there is already an item in the shop with the same name as the passed-in parameter, then print out the following on its own line:

```
Item already in shop
```

    * If not, then resize the array and add the passed-in item at the back using a **shallow copy**.

- addItems(i: item**, n: int): void
    * n is the size of the i array.
    * Insert every item in i, into the shop.
    * For every item in the passed-in array that is already in the shop print

```
Item already in shop
```

- buyItems(s: string): void
    * The passed-in string is a list of purchase requests.
    * The name of the item will be listed first, followed by a colon and after that is the string which should be passed to the object. This is then terminated by a "|". The next object will automatically start after this bar. You may assume the last object in the list also ends with a bar.
    * If you encounter a name which is not found inside the shop then print the following on its own line

```
Couldn't_find_{itemName}
```

    * After calling buy on every item, tally the prices and print this on its own line at the end. _'s need to be replaced by spaces and the price should be printed to 2 decimal points. Do not print out the curly braces.

```
Total:_R_{total price}
```

- printItems(): void
    * This function calls the print function of every item in the shop.
- removeItem(s: string): void
    * If there is an item in the shop that has the same name as the passed-in parameter, then delete the object. Also resize the array and shift the elements down to fill the gap.
    * If the item was not found, then print the following on its own line:

```
Item not found
```

6 Memory Management

As memory management is a core part of COS110 and C++, each task on FitchFork will allocate approximately 10% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocate *all* the memory that was allocated.

7 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the main.cpp file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [1] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j bulk discountedProduct item labor product service shop
   subscription
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

and we will scale this ration based on class size.
The mark you will receive for the testing coverage task is determined using table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the function stipulated in this specification will be considered to determine your mark. Remember that your main will be testing the Instructor provided code and as such it can only be assumed that the functions outlined in this specification are defined and implemented.

[1] For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

8 Upload checklist

The following files should be in the root of your archive

- main.cpp
- bulk.cpp
- discountedProduct.cpp
- item.cpp
- labor.cpp
- product.cpp
- service.cpp
- shop.cpp
- subscription.cpp

9 Allowed libraries

- cmath
- iomanip
- iostream
- sstream
- string

10 Submission

You need to submit your source files, only the .cpp files, on the FitchFork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. There is no need to include any other files or .h files in your submission. Your code must be able to be compiled with the C++98 standard

For this practical, you will have 10 upload opportunities and your best mark will be your final mark. Upload your archive to the appropriate slot on the FitchFork website well before the deadline. **No late submissions will be accepted!**