Department of Computer Science

Faculty of Engineering, Built Environment & IT

University of Pretoria

COS110 - Program Design: Introduction

Practical 8 Specifications

Release Date: 07-10-2024 at 06:00

Due Date: 11-10-2024 at 23:59

Total Marks: 115

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.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Unless otherwise stated, the usage of c++11 or additional libraries outside of those indicated in the assignment, will **not** be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use c++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.
- Note, UML notation for variable and function signatures are used in this assignment.

2 Overview

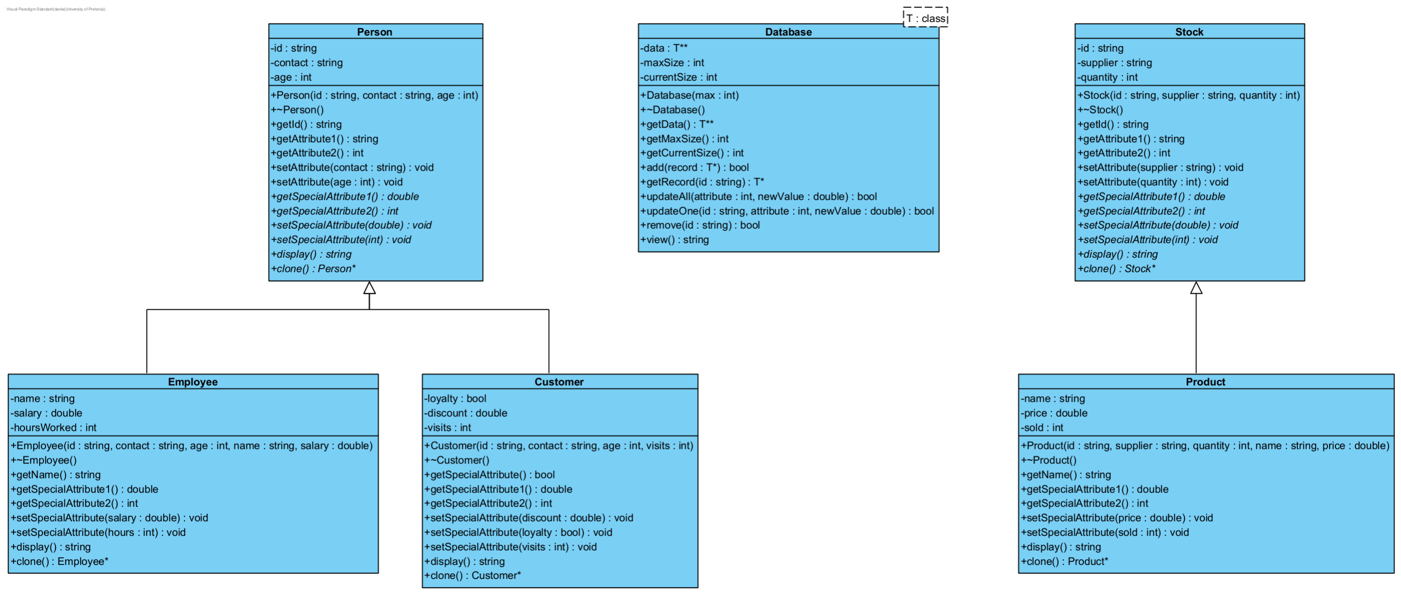
Templates are a feature in C++ that enable a function or a class to accept generic parameters. Rather than coding a function to accept a specific kind of input parameter type, it can be coded to accept a generic type that can then be implemented to respond to different kinds of inputs. This can be extended for classes which enable them to be parameterised with different types. Templates can be applied to both functions and classes in order to create a generic method/class once, and have it apply to many different types at compile time. By specifying a type in a template parameter, the class is compiled by using that specific datatype in the way specified. While multiple template parameters can be used (more than one type can be put in a template parameter list) for this practical, you will only use single-parameter templates.

3 Background

In this practical you will simulate a simple database for a small business. It can store information about their employees, customers and the products they sell. The operations performed on the data are the standard CRUD operations (Create, Read, Update and Delete).

4 Your Task:

You are required to implement the following class diagram illustrated in Figure 1. Pay close attention to the function signatures as the `h` files will be overwritten, thus failure to comply with the UML, will result in a mark of 0.



Visual Paradigm Standard(danie(University of Pretoria))
Person
-id : string
-contact : string
-age : int
+Person(id : string, contact : string, age : int)
+~Person()
+getId() : string
+getAttribute1() : string
+getAttribute2() : int
+setAttribute(contact : string) : void
+setAttribute(age : int) : void
+getSpecialAttribute1() : double
+getSpecialAttribute2() : int
+setSpecialAttribute(double) : void
+setSpecialAttribute(int) : void
+display() : string
*+clone() : Person**
T : class
Database
-data : T**
-maxSize : int
-currentSize : int
+Database(max : int)
+~Database()
+getData() : T**
+getMaxSize() : int
+getCurrentSize() : int
+add(record : T*) : bool
+getRecord(id : string) : T*
+updateAll(attribute : int, newValue : double) : bool
+updateOne(id : string, attribute : int, newValue : double) : bool
+remove(id : string) : bool
+view() : string
Stock
-id : string
-supplier : string
-quantity : int
+Stock(id : string, supplier : string, quantity : int)
+~Stock()
+getId() : string
+getAttribute1() : string
+getAttribute2() : int
+setAttribute(supplier : string) : void
+setAttribute(quantity : int) : void
+getSpecialAttribute1() : double
+getSpecialAttribute2() : int
+setSpecialAttribute(double) : void
+setSpecialAttribute(int) : void
+display() : string
*+clone() : Stock**
Employee
-name : string
-salary : double
-hoursWorked : int
+Employee(id : string, contact : string, age : int, name : string, salary : double)
+~Employee()
+getName() : string
+getSpecialAttribute1() : double
+getSpecialAttribute2() : int
+setSpecialAttribute(salary : double) : void
+setSpecialAttribute(hours : int) : void
+display() : string
+clone() : Employee*
Customer
-loyalty : bool
-discount : double
-visits : int
+Customer(id : string, contact : string, age : int, visits : int)
+~Customer()
+getSpecialAttribute() : bool
+getSpecialAttribute1() : double
+getSpecialAttribute2() : int
+setSpecialAttribute(discount : double) : void
+setSpecialAttribute(loyalty : bool) : void
+setSpecialAttribute(visits : int) : void
+display() : string
+clone() : Customer*
Product
-name : string
-price : double
-sold : int
+Product(id : string, supplier : string, quantity : int, name : string, price : double)
+~Product()
+getName() : string
+getSpecialAttribute1() : double
+getSpecialAttribute2() : int
+setSpecialAttribute(price : double) : void
+setSpecialAttribute(sold : int) : void
+display() : string
+clone() : Product*

Figure 1: Class diagrams

Note, the importance of the arrows between the classes are not important for COS 110 and will be expanded on in COS 214.

4.1 Person

This is the base class of the Employee and Customer classes and is an abstract class.

4.1.1 Members

- - id: string
  - This member variable represents the id of the person.
- - contact: string
  - This member variable represents the contact details of the person.
- - age: int
  - This member variable represents the age of the person.

4.1.2 Functions

- + Person(id: string, contact: string, age: int)
  - This is the constructor for the Person class.
  - Initialise the member variables to the corresponding parameters.
- + ∼Person()
  - This is the destructor for the Person class.
  - There is no dynamic memory to deallocate, therefore it can be left blank.
- + getId(): string
  - This is an accessor method for the id member variable.
  - Return the id member variable.
- + getAttribute1(): string
  - This is an accessor method for the contact member variable.
  - Return the contact member variable.
- + getAttribute2(): int
  - This is an accessor method for the age member variable.
  - Return the age member variable.
- + setAttribute(contact: string): void
  - This method should set the contact member variable to the passed in parameter.
- + setAttribute(age: int): void

  - This method should set the age member variable to the passed in parameter.

- The following are pure virtual functions that will be described in the derived classes:
  - *+ getSpecialAttribute1(): double*
  - *+ getSpecialAttribute2(): int*
  - *+ setSpecialAttribute(double): void*
  - *+ setSpecialAttribute(int): void*
  - *+ display(): string*
  - *+ clone(): Person**

4.2 Employee

The Employee class is one of the Person's derived classes. It inherits publicly from Person. It has three additional member variables.

4.2.1 Members

- - name: string
  - This member variable represents the name of the Employee.
- - salary: double
  - This member variable represents the salary of the Employee.
- - hoursWorked: int
  - This member variable represents the total hours an Employee worked.

4.2.2 Functions

- + Employee(id: string, contact: string, age: int, name: string, salary: double)
  - This is the constructor for the Employee Class.
  - Set the member variables to the corresponding parameters.
  - Initialise hoursWorked to 0.
  - Remember to call the base class's constructor.
- + ∼Employee()
  - This is the destructor for the Employee class.
  - There is no dynamic memory to deallocate, therefore it can be left blank.
- + getName(): string
  - This is an accessor method for the name member variable.

  - Return the name member variable.

- + getSpecialAttribute1(): double
  - This is an accessor method for the salary member variable.
  - Return the salary member variable.
- + getSpecialAttribute2(): int
  - This is an accessor method for the hoursWorked member variable.
  - Return the hoursWorked member variable.
- + setSpecialAttribute(salary: double): void
  - This method should set the salary member variable to passed in parameter.
- + setSpecialAttribute(hours: int): void
  - This method should set the hoursWorked member variable to passed in parameter.
- + display(): string
  - This method returns a string with all the information of the employee.
  - The string should be formatted as follows:

```
ID: id | Contact: contact | Age: age | Name: name | Salary:
    salary | Hours: hoursWorked |
```

- + clone: Employee*
  - This method returns a copy of the current employee.
  - This method is given.

4.3 Customer

The Customer class is one of the Person's derived classes. It inherits publicly from Person. It has three additional member variables.

4.3.1 Members

- - loyalty: bool
  - This member variable represents if a customer is a loyalty customer.
- - discount: double
  - This member variable represents the % discount the customer get.
- - visits: int
  - This member variable represents the number of times the customer visited the business.

4.3.2 Functions

- + Customer(id: string, contact: string, age: int, visits: int)
  - This is the constructor for the Customer Class.
  - Set the member variables to the corresponding parameters.
  - Initialise loyalty to false and discount to 0.0.
  - Remember to call the base class's constructor.
- + ∼Customer()
  - This is the destructor for the Customer class.
  - There is no dynamic memory to deallocate, therefore it can be left blank.
- + getSpecialAttribute(): bool
  - This is an accessor method for the loyalty member variable.
  - Return the loyalty member variable.
- + getSpecialAttribute1(): double
  - This is an accessor method for the discount member variable.
  - Return the discount member variable.
- + getSpecialAttribute2(): int
  - This is an accessor method for the visits member variable.
  - Return the visits member variable.
- + setSpecialAttribute(loyalty: bool): void
  - This method should set the loyalty member variable to passed in parameter.
- + setSpecialAttribute(discount: double): void
  - This method should set the discount member variable to passed in parameter.
- + setSpecialAttribute(visits: int): void
  - This method should set the visits member variable to passed in parameter.
- + display(): string
  - This method returns a string with all the information of the customer.
  - The string should be formatted as follows:

```
ID: id | Contact: contact | Age: age | Loyalty: loyalty
   Discount: discount | Visits: visits |
```

- + clone: Customer*
  - This method returns a copy of the current customer.
  - This method is given.

4.4 Stock

This is the base class of the Product class and is an abstract class.

4.4.1 Members

- - id: string
  - This member variable represents the id of the item in stock.
- - supplier: string
  - This member variable represents the name of the supplier of the item.
- - quantity: int
  - This member variable represents the amount of that item in stock.

4.4.2 Functions

- + Stock(id: string, supplier: string, quantity: int)
  - This is the constructor for the Stock class.
  - Initialise the member variables to the corresponding parameters.
- + ∼Stock()
  - This is the destructor for the Stock class.
  - There is no dynamic memory to deallocate, therefore it can be left blank.
- + getId(): string
  - This is an accessor method for the id member variable.
  - Return the id member variable.
- + getAttribute1(): string
  - This is an accessor method for the supplier member variable.
  - Return the supplier member variable.
- + getAttribute2(): int
  - This is an accessor method for the quantity member variable.
  - Return the quantity member variable.

- + setAttribute(supplier: string): void
  - This method should set the supplier member variable to the passed in parameter.
- + setAttribute(quantity: int): void
  - This method should set the quantity member variable to the passed in parameter.
- The following are pure virtual functions that will be described in the derived class:
  - *+ getSpecialAttribute1(): double*
  - *+ getSpecialAttribute2(): int*
  - *+ setSpecialAttribute(double): void*
  - *+ setSpecialAttribute(int): void*
  - *+ display(): string*
  - *+ clone(): Stock**

4.5 Product

The Product class is one of the Stock's derived classes. It inherits publicly from Stock. It has three additional member variables.

4.5.1 Members

- - name: string
  - This member variable represents the name of the Product.
- - price: double
  - This member variable represents the price of the product.
- - sold: int
  - This member variable represents the number of units sold.

4.5.2 Functions

- + Product(id: string, supplier: string, quantity: int, name: string, price: double)
  - This is the constructor for the Product Class.
  - Set the member variables to the corresponding parameters.
  - Initialise sold to 0.
  - Remember to call the base class's constructor.
- + ~Product()
  - This is the destructor for the Employee class.

  - There is no dynamic memory to deallocate, therefore it can be left blank.

- + getName(): string
  - This is an accessor method for the name member variable.
  - Return the name member variable.
- + getSpecialAttribute1(): double
  - This is an accessor method for the price member variable.
  - Return the price member variable.
- + getSpecialAttribute2(): int
  - This is an accessor method for the sold member variable.
  - Return the sold member variable.
- + setSpecialAttribute(price: double): void
  - This method should set the price member variable to passed in parameter.
- + setSpecialAttribute(sold: int): void
  - This method should set the sold member variable to passed in parameter.
- + display(): string
  - This method returns a string with all the information of the product.
  - The string should be formatted as follows:

```
ID: id | Supplier: supplier | Quantity: quantity | Name:
    name | Price: price | Units Sold: sold |
```

- + clone: Product*
  - This method returns a copy of the current product.
  - This method is given.

4.6 Database<T>

This is a template class that can store people or stock.

4.6.1 Members

- - data: T**
  - This member variable represents the data stored in the database.
  - Each index in the array correspond to a record in the database.
  - It is a one-dimensional array of dynamic objects.

- - maxSize: int
  - This member variable represents the maximum number of objects the database can store.
- - currentSize: int
  - This member variable represents current number of objects stored.

4.6.2 Functions

- + Database(max: int)
  - This is the constructor for the Database class.
  - Set the maxSize member variable to the passed in parameter.
  - Set the current size to 0.
  - Initialise the data array to have a size of maxSize.
  - Initialise all the indexes of the array to NULL.
- + ~Database()
  - This is the destructor for the Database class.
  - It should deallocate the data array.
- + getData(): T**
  - This is an accessor method for the data member variable.
  - Return the data member variable.
- + getMaxSize(): int
  - This is an accessor method for the maxSize member variable.
  - Return the maxSize member variable.
- + getCurrentSize(): int
  - This is an accessor method for the currentSize member variable.
  - Return the currentSize member variable.
- + add(record: T*): bool
  - This method adds a record to the database.
  - If the database is full or if the id of the record to be added is already in the database it should return false.
  - If a record can be added it should append the record to the data array by making use of the clone method. Return true.
  - Increment the current size.

- + getRecord(id: string): T*
  - It should search the data array and return the record with the passed in id.
  - If there is no record with the specified id, return NULL.
- + updateAll(attribute: int, newValue: double): bool
  - This method updates all the records in the database.
  - If the passed in attribute is equal to 1:
    - If the first special attribute is equal to 0, set it to the $newValue/100$.
    - Else set the first special attribute of the record to the result of the following: $result = val + val * \frac{newVal}{100}$. It simulates an increase in salary or price.
  - If the passed in attribute is equal to 2:
    - Set the second special attribute of the record to the result of the following: $result = val + newval$. It simulates an increase in the hoursWorked or number of units sold. Remember to cast the double parameter to an integer.
  - If the passed in attribute is any other value, return false.
  - If all the records was updated successfully, return true. In any other case, return false.
  - **Specialisation**
    - The update all method should be specialised for Customer.
    - This method updates all the records in the database.
    - If the passed in attribute is equal to 1:
      - If the customer is a loyalty customer, increase the passed in newValue with 2.5.
      - If the first special attribute is equal to 0, set it to the $newValue/100$.
      - Set the first special attribute of the record to the result of the following: $result = val + val * \frac{newVal}{100}$. It simulates an increase in discount.
    - If the passed in attribute is equal to 2:
      - Set the second special attribute of the record to the result of the following: $result = val + newval$. It simulates an increase in the number of visits. Remember to cast the double parameter to an integer.
      - If the number of visits is greater than 15, set the loyalty member variable to true.
    - If the passed in attribute is any other value, return false.
    - If all the records was updated successfully, return true. In any other case, return false.
- + updateOne(id: string, attribute: int, double: newValue): bool
  - This method updates a single record in the database.
  - Search the database for the record with an id equal to the passed in id.
  - If a record is found, follow the same steps as for updateAll but update only the found record and return true.

  - If a record is not found, return false.
  - This method should also be specialised for Customer. Follow the same steps as above and as the *updateAll* specialisation.

- + remove(id: string): bool
  - This method removes the record from the database with the passed in id.
  - If the record is found, delete the record from the data array and shift the rest of the records one down.
  - If the record is not found, return false. Otherwise return true.

- + view(): string
  - This method returns a string representing all the records in the database.
  - To build the string call the display method for each item in data and add a newline to the end of each record.

5 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 ${files}
```

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

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

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%

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

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 functions stipulated in this specification will be considered to determine your testing 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.

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

6 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **c++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using `namespace std` in any of the files.
- You may only use the following libraries:
  - string
  - sstream
  - iostream
- You are supplied with a makefile, and the basic outline of all the required header files.
- The compilation strategy followed is the same as the preferred approach in the lectures.
- If you decide to not use the provided makefile ensure that:
  - In database.h, include database.cpp at the bottom.
  - In database.cpp, include database.h at the top and ensure the entire file is in a #ifndef #define #endif guard.
  - Do not compile database.cpp.

7 Upload Checklist

The following c++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- person.h
- person.cpp
- employee.h
- employee.cpp
- customer.h
- customer.cpp
- stock.h
- stock.cpp
- product.h
- product.cpp
- database.h
- database.cpp
- `main.cpp`
- `testingFramework.h` and `testingFramework.cpp` if you used these files.

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

8 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your `h` file will be overwritten, so ensure you do not alter the provided `h` files.
You have 10 submissions and your best mark will be your final mark. Upload your archive to the Practical 2 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**

9 Accessibility

1 Show the class diagram of the six classes to be implemented in this practical:

- Person
- Employee
- Customer
- Stock
- Product
- Database

The member variables and functions are discussed in Section 4. The following relationships exists between the classes:

- Employee and Customer inherit from Person.
- Product inherit from Stock.