|  |
| --- |
| **Computer Engineering Department - ITU** |
| **CE101L: Object Oriented Programming Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 21/04/2022** |
| **Teaching Assistant: Aqsa Khalid** | **Semester: Spring 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 7B. Use of Composition and Inheritance in Classes and Objects**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Roll number** | **Report**  **(out of 100)** | **Scaled to 10** | **Total**  **(out of 10)** |
| Muhammad Abubakar Saif | BSCE21017 |  |  |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Objective**

The objective of this lab is to observe the basic knowledge of programming classes in C++.

## **Equipment and Component**

|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **Conduct of Lab**

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

## **Theory and Background**

**Composition** is one of the fundamental concepts in object-oriented programming. It describes a class that references one or more objects of other classes in instance variables. This allows you to model a has-a association between objects. You can find such relationships quite regularly in the real world.

**Inheritance** is one of the core concepts of object-oriented programming (OOP) languages. It is a mechanism where you can to derive a class from another class for a hierarchy of classes that share a set of attributes and methods.

A **multi-level inheritance** is a concept in which a class can also be derived from one class, which is already derived from another class. In **multiple inheritance**, a class can also be derived from more than one base class.

Suppose the same function is defined in both the derived class and the based class. Now if we call this function using the object of the derived class, the function of the derived class is executed. This is known as **function** **overriding** in C++. The function in derived class overrides the function in base class.

It's a type of polymorphism in which an operator is overloaded to give it the user-defined meaning. C++ allows us to specify more than one definition for a function name or an operator in the same scope, which is called **function** overloading and **operator** overloading, respectively.

**Lab Task**

**Task A: Composition [Marks: 20]**

In this task, you are required to create two classes’ **A and B** with the following data members and member functions,

***Private Data Members of class A such as:***

num(int)

***Public Member Functions of class A such as:***

**void set\_value(int k) –** It will set value of private data member num.

**void show\_sum(int n) –** It will print sum n+num.

***Public Data Member of class B such as:***

classA\_obj(A)

***Public Member Functions of class B such as:***

**void print() –** It will call class A member function e.g. classA\_obj.show\_sum(int)

Create a **UML diagram.**

Do the following operations in main function:

1. Create object of class **B** and call member function **set\_value(int), show\_sum(int)** of class A.

2. Call member function of class B **print().**

|  |
| --- |
| **(ClassA.cpp):**  #include **"ClassA.h"** #include **<iostream>  using namespace** std;  **void** A::set\_value(**int** k) {  num = k;  **return**; }  **void** A::show\_sum(**int** n) {  cout << **"Sum: "** << n + num << endl;  **return**; }  **void** B::print() {  **int** n;  cout << **"Enter value of n in member function of class B [void print()]: "**;  cin >> n;  classA\_obj.show\_sum(n);  **return**; }  **(ClassA.h):**  **class** A { **private**:  **int** num; **public**:  **void** set\_value(**int** k); *//It will set value of private data member num* **void** show\_sum(**int** n); *//It will print sum n+num.* };  **class** B { **public**:  A classA\_obj;  **void** print(); *//It will call class A member function e.g. classA\_obj.show\_sum(int)* };  **(Main function):**  **int** main {  int val  B obj\_B;  cout << **"Enter Value of private Member of classA from classB: "**;  cin >> val;  obj\_B.classA\_obj.set\_value(val);  cout << **"Enter the value of n in MAIN Function: "**;  cin >> n;  obj\_B.classA\_obj.show\_sum(n);  cout << **"Calling member function of class 'B'.....\n"**;  obj\_B.print();  **return 0**; } |

Output:

Text

Description automatically generated

**UML:**

**Shape

Description automatically generated with medium confidence**

**Task B: Hierarchical Inheritance [Marks: 20]**

In this task, you are required to create Employee **(Parent class)**, and three child classes **Admin, IT, and HR** with the following data members and member functions,

***Public Data Members of class Employee such as:***

salary(int), name, designation(string)

***Public Member Functions of class Employee such as:***

**void enter\_record() –** It will take input name, designation, and salary.

**void display\_record() –** It will display name, designation, and salary.

***Private Data Members of class Admin such as:***

total\_service(int) – Service will be in years e.g. total\_service = 2 (two years)

***Public Member Functions of class Admin such as:***

**void enter\_record () –** It will call member function **enter\_record()** of class Employee and take input total\_service in years.

**void display\_record() –** It will call member function **display\_record()**  of class Employee and print total\_service.

***Private Data Members of class IT such as:***

total\_service(int) – Service will be in years e.g. total\_service = 2 (two years)

***Public Member Functions of class IT such as:***

**void enter\_record () –** It will call member function **enter\_record()** of class Employee and take input total\_service in years.

**void display\_record() –** It will call member function **display\_record()**  of class Employee and print total\_service.

***Private Data Members of class HR such as:***

total\_service(int) – Service will be in years e.g. total\_service = 2 (two years)

***Public Member Functions of class HR such as:***

**void enter\_record () –** It will call member function **enter\_record()** of class Employee and take input total\_service in years.

**void display\_record() –** It will call member function **display\_record()**  of class Employee and print total\_service.

Create a **UML diagram**.

Do the following operations in main function:

1. Create object of class ***Admin*** and call member functions such as **enter\_record() and display\_record().**

2. Create object of class ***IT*** and call member functions such as **enter\_record() and display\_record().**

3. Create object of class ***HR*** and call member functions such as **enter\_record() and display\_record().**

|  |
| --- |
| **(Organization.h):**  #include **<string>** #include **<iostream>  using namespace** std;  **class** Employee { **public**:  **int** salary;  string name, designation;  **void** enter\_record(){ *//It will take input name, designation, and salary.* cout<<**"Enter Name: "**;  cin>>name;  cout<<**"Enter Designation: "**;  cin>>designation;  }  **void** display\_record(){ *//It will display name, designation, and salary.* cout<<**"Name: "**<<name<<endl<<**"Designation: "**<<designation<<endl;  } };  **class** Admin: **public** Employee{ **private**:  **int** total\_service; *//Service will be in years e.g. total\_service = 2 (two years)* **public**:  **void** enter\_record () { *// It will call member function enter\_record() of class Employee and take input total\_service in years.* Employee::enter\_record();  cout<<**"Enter total service: (in years) "**;  cin>>total\_service;  }  **void** display\_record() { *// It will call member function display\_record() of class Employee and print total\_service.* Employee::display\_record();  cout<<**"Total Service: "**<<total\_service<<endl;  } };  **class** IT:**public** Employee{ **private**:  **int** total\_service; *// Service will be in years e.g. total\_service = 2 (two years)* **public**:  **void** enter\_record () { *// It will call member function enter\_record() of class Employee and take input total\_service in years.* Employee::enter\_record();  cout<<**"Enter total service: (in years) "**;  cin>>total\_service;  }  **void** display\_record() { *// It will call member function display\_record() of class Employee and print total\_service.* Employee::display\_record();  cout<<**"Total Service: "**<<total\_service<<endl;  } };  **class** HR: **public** Employee{ **private**:  **int** total\_service; *// Service will be in years e.g. total\_service = 2 (two years)* **public**:  **void** enter\_record () { *// It will call member function enter\_record() of class Employee and take input total\_service in years.* Employee::enter\_record();  cout<<**"Enter total service: (in years) "**;  cin>>total\_service;  }  **void** display\_record() { *// It will call member function display\_record() of class Employee and print total\_service.* Employee::display\_record();  cout<<**"Total Service: "**<<total\_service<<endl;  } };  **(Main Function):**  **int main** {  int opt;  string rep;  task:  cout << **"1. Invoke Admin Functions \n"**;  cout << **"2. Invoke IT Functions \n"**;  cout << **"3. Invoke HR Functions \n"**;  cout << **"Enter your choice: "**;  cin >> opt;  **switch** (opt) {  **case** 1: {  Admin a1;  a1.enter\_record();  a1.display\_record();  **break**;  }  **case** 2: {  IT tech;  tech.enter\_record();  tech.display\_record();  **break**;  }  **case** 3: {  HR uni;  uni.enter\_record();  uni.display\_record();  **break**;  }  }  cout << **"Do you want to use this task again? (Y/N): "**;  cin >> rep; *//takes input from user* **if** (rep == **"N" or** rep == **"n" or** rep == **"no" or** rep == **"NO" or** rep == **"No" or** rep == **"nO"**) {  **return** 0; *//returns the process with exit code 0* } **else goto** task; *//restart the program flow* } |

Output:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**UML:**

Diagram

Description automatically generated

#### **Assessment Rubric for Lab**

**Method for assessment:**

Lab reports and instructor observation during lab sessions. Outcome assessed:

a. Ability to conduct experiments, as well as to analyze and interpret data (P) b. Ability to function on multi-disciplinary teams (A)

c. Ability to use the techniques, skills, and modern engineering tools necessary for engineering practice (P)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Performance metric** | **Task** | **CLO** | **Description** | **Max marks** | **Exceeds expectation** | **Meets expectation** | **Does not meet expectation** | **Obtained marks** |
| 1. Realization of experiment (a) | 1 | 1 | Functionality | 40 | Executes without errors excellent user prompts, good use of symbols, spacing in output. Through testing has been completed (35-40) | Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed (20-34) | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non-existent. No testing has been completed (0-19) |  |
| 2. Teamwork (b) | 1 | 3 | Group Performance | 5 | Actively engages and cooperates with other group member(s) in effective manner (4-5) | Cooperates with other group member(s) in a reasonable manner but conduct can be improved (2-3) | Distracts or discourages other group members from conducting the experiment (0-1) |  |
| 3. Conducting experiment (a, c) | 1 | 1 | On Spot Changes | 10 | Able to make changes (8-10) | Partially able to make changes (5-7) | Unable to make changes (0-4) |  |
| 1 | 1 | Viva | 10 | Answered all questions (8-10) | Few incorrect answers (5-7) | Unable to answer all questions (0-4) |  |
| 4. Laboratory safety and disciplinary rules (a) | 1 | 3 | Code commenting | 5 | Comments are added and does help the reader to understand the code (4-5) | Comments are added and does not help the reader to understand the code (2-3) | Comments are not added (0-1) |  |
| 5. Data collection (c) | 1 | 3 | Code Structure | 5 | Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap (4-5) | Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables (2-3) | Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy (0-1) |  |
| 6. Data analysis (a, c) | 1 | 4 | Algorithm | 20 | Solution is efficient, easy to understand, and maintain (15-20) | A logical solution that is easy to follow but it is not the most efficient (6-14) | A difficult and inefficient solution (0-5) |  |
| 7. Computer use (c) | 1 | 2 | Documentation & GitHub Submissions | 5 | Timely (4-5) | Late (2-3) | Not done (0-1) |  |
|  | Max Marks (total): | | | 100 | Obtained Marks (total): | | |  |

Lab Engineer Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_