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

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 20/04/2022** |
| **Teaching Assistant: Aqsa Khalid** | **Semester: Spring 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 7A. Use of Association & Aggregation in Classes and Objects**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Roll number** | **Report**  **(out of 100)** | **Scaled to 10** | **Total**  **(out of 10)** |
| NIMRA MAQBOOL | BSCE21012 |  |  |  |

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**

**Association** depicts the relationship between two classes. Both of the classes are not dependent on each other. They can exist independently. If one is broken, it doesn’t mean the other one also has to be broken.

**Aggregation** in OOPS is defined as a relation that exists between two or more two objects which individually have their own individual life cycle along with the ownership.

**Lab Task**

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

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

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

tName (string)

tDesignation(string)

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

**ParameterizedConstructor(string,string) –** It will print “Creating Teacher” and initialize tName and tDesignation.

**void showInfo() –** It will print tName and tDesignation.

***Public Static Member Functions of class Student such as:***

**void teach(Teacher &tr) –** It will call member function **showInfo()** of Teacher class.

Create a **UML diagram.**

Do the following operations in main function:

1. Create two objects of class Teacher with parameters.

2. Create two objects of class Student and use these objects to call member function teach(pass object of class Teacher).

|  |
| --- |
| **Function.h:**  class Teacher { //making a class of teacher private:  string name; //creating private members  string designation; //name and designation public:  Teacher(string name1,string designation1); //creating a constructor to initialize the values of name and designation  void showInfo(); //function to show the info (designation and name) }; class student{ //creating a class of students public:  static void teach(Teacher &tr){ //creating a static function having the obj of teacher as parameter by reference  tr.showInfo(); //calling the show function by the obj of teacher  } };  **function.cpp:**  Teacher::Teacher(string name1, string designation1){ //constructor of Teacher class  cout << "CREATING TEACHER" << endl; //displaying  cout << "ENTER NAME : "; //taking name  cin >> name1;  cout << "ENTER DESIGNATION : "; //and designation from the user  cin >> designation1;  cout<<endl;  name = name1; //copying the value  designation = designation1; //copying }  void Teacher::showInfo(){ //a function to display name and designation  cout << "NAME : " << name << endl;  cout << "DESIGNATION : " << designation << endl; //displaying name and designation }  **Main.cpp:**  int opt; do{  cout<<"WHICH TASK DO YOU WANT TO PERFORM?"<<endl;  cout<<"1.TASK 1"<<endl;  cout<<"2.TASK 2"<<endl;  cout<<"3.EXIT"<<endl;  cin>>opt;  if(opt==1){  string name1; //declaring the name and designation  string designation1;  Teacher Teach(name1, designation1); //creating the object  Teacher Teach1(name1, designation1);  student st; //declaring  student st1;  st.teach(Teach);  st1.teach(Teach1); //calling  }  **output:**  **Text  Description automatically generated**  **Uml:**  **Diagram  Description automatically generated** |

**Task B: Aggregation [Marks: 20]**

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

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

mModel(int)

mName(string)

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

**ParameterizedConstructor(string, int) –** It will initialize mModel and mName and print “Creating Car”.

**void printCarInfo() –** It will print mModel and mName.

**Destructor() –** It will print “Destroying Car”.

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

\*cptr(Car)

***Public Member Function of class Person such as:***

**ParameterizedConstructor(Car \*pcar) –** It will initialize cptr and print “Creating Person”.

**Destructor() –** It will print “Destroying Person”.

Create a **UML diagram**.

Do the following operations in main function:

1. Create object of Car class and pass car name and model no.

2. Create pointer object of person class.

3. Delete pointer object of person class.

4. Call **printCarInfo()** using object of Car class.

|  |
| --- |
| **Function.h:**  class car{ //creating a class of car private:  int mModel; //it has private members of model and the name in it  string mName; public: //the public members are car(string name,int model ); //making a constructor having name and model as parameter in it void printCarInfo(); //making a function to display the info of car ~ car(); //making a destructor }; class person{ //creating a person class private:  car \*cptr; //creating a pointer in private public:  person(car \*pcar){ //creating a function having pointer in parameter  cout<<"CREATING PERSON"<<endl; //displaying  this->cptr; //setting the pointer  }  ~person(){ //making a destructor  cout<<"DESTROYING PERSON"<<endl;  } };  **function.cpp:**  car::car(string name,int model ){ //a constructor to set the values of name and model  cout<<endl;  cout<<"CREATING CAR"<<endl;  cout<<"ENTER NAME : "; //taking name  cin>>name;  cout<<"ENTER MODEL : "; //and model from the user  cin>>model;  cout<<endl;  mModel=model; //copying the name and model  mName=name; } void car::printCarInfo(){ //a function to display the car info  cout<<"NAME : "<<mName<<endl; //displaying the name and model  cout<<"MODEL : "<<mModel<<endl; } car::~ car(){  cout<<"DESTROYING CAR"<<endl; //displaying }  **main.cpp:**  int main() {  int opt;  do{  cout<<"WHICH TASK DO YOU WANT TO PERFORM?"<<endl;  cout<<"1.TASK 1"<<endl;  cout<<"2.TASK 2"<<endl;  cout<<"3.EXIT"<<endl;  cin>>opt;  if(opt==1){  string name1; //declaring the name and designation  string designation1;  Teacher Teach(name1, designation1); //creating the object  Teacher Teach1(name1, designation1);  student st; //declaring  student st1;  st.teach(Teach);  st1.teach(Teach1); //calling  }  if(opt==2){  string name;  int model; //declaring  car c(name, model); //creating an obj  person \*ptr;  person p(&c); //calling  c.printCarInfo(); //calling  }  if(opt==3){  cout<<"YOU CHOOSE TO EXIT .."<<endl;  exit(3);  }  }while(opt>=1 && opt<=3);   return 0; }  **output:**  **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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_