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

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 28/04/2022** |
| **Teaching Assistant: Aqsa Khalid** | **Semester: Spring 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 8B. Use of Static Variable and Singleton 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**

In object-oriented programming, there is also the concept of a **static** member variable, which is a "class variable" of a statically defined class, i.e., a member variable of a given class which is shared across all instances (objects), and is accessible as a member variable of these objects.

In object-oriented programming, a **singleton** class is a class that can have only one object (an instance of the class) at a time. After the first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created.

**Lab Task**

**Task A: Static Variable inside of Class [Marks: 20]**

In this task, you are required to create a class **NumberIncrement** with the following data members and member functions,

***Private Static Data Members of class NumberIncrement such as:***

count (static int)

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

**void increment\_count() –** It will increment static data member.

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

**static void print() –** It will print static data member.

Create a **UML diagram.**

**UML Diagram:**

Table

Description automatically generated

Initialize static data member

Do the following operations in main function:

1. Create object of class and call member function **void increment\_count().**

2. Access static function directly with class name.

|  |
| --- |
| (Header File):  #include **<iostream>   class** NumberIncrement { **private**:  **static int** count; *//static int variable* **public**:  **void** increment\_count() { *//It will increment static data member.* count++; *//increments the statin int variable* **return**;  }  **static void** print() { *// It will print static data member.* std::cout << **"Count: "** << count << std::endl; *//prinys the variable* **return**;  } };  (Main Function):  **int** NumberIncrement::count = 0; *//initialize the static variable to ZERO*  **int** main {  NumberIncrement num; *//declares the instance of class* num.increment\_count(); *//calls method of the class* NumberIncrement::print(); *//directly calls the static function from main function via Classname* **return 0**; *//breaks the casee* } |

Output:

Graphical user interface, text, application

Description automatically generated

**Task B: Singleton Design Pattern [Marks: 20]**

In this task, you are required to create a class Singleton with the following data members and member functions,

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

data(int)

\*instance(static Singleton)

***Private Member Functions of class Singleton such as:***

**PrivateConstructor() –** It will initialize data to zero.

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

**static Singleton \*getInstance() {**

if (!instance)

instance = new Singleton;

return instance;

**}**

**int get() –** It will return data. e.g. return this->data;

**void set() –** It will set data. e.g. this->data = data;

Create a **UML diagram**.

Initialize pointer to zero so that it can be initialized in first call to getInstance

**e.g. Singleton \*Singleton::instance = 0;**

Do the following operations in main function:

1. Create pointer \*s of class ***Singleton*** that points to getInstance() method e.g. Singleton \*s = s->getInstance();

2. Call get() method with pointer and display the result.

3. Set value of data using set(int)

4. Call get() method again with pointer and display the result.

UML Diagram:

Table

Description automatically generated

Code:

|  |
| --- |
| (Header File):  **class** Singleton { **private**:  **int** data; *//declares variable of int-type* **static** Singleton \*instance; *//declares static pointer of manually-declared type* Singleton() { *//private constructor so that it can't be accessed on every call* data = 0; *//sets the value of data to ZERO* } **public**: *//public methods* **static** Singleton \*getInstance() { *//assigns a new memory to the member pointer* **if** (!instance) *//if instance already exists, it will not be re-declared* instance = **new** Singleton; *//allocates new memory to the pointer* **return** instance; *//returns the pointer* }  **void** releasesMem(){  **delete**[] instance; *//releases the memory which is dynamically allocated* }  **void** set(**int** data){ *//sets the value of private member* **this**->data = data;  **return**;  }  **int** get(){ *//get the value of private member through this function* **return this**->data; *//return value of 'data' to invoker function* } };  (Driver Function):  Singleton \*Singleton::instance = 0; *//sets the pointer to ZERO so that pointer will be directed to new allocated memory in first call*  **int** main {  Singleton \*pnt = (\*pnt).getInstance(); *//declares the pointer of manually-declared type and directs it to existing instance*  **int** val; *//declares variables of int-type*  val = pnt->get(); *//gets the value of private member of class* cout << **"data: "** << val << endl; *//prints the value* cout << **"Enter the value you want to set for 'DATA' (private) variable: "**;  cin >> val; *//takes input from the user* pnt->set(val); *//sets the value by calling method of class through pointer* cout << **"New Value of 'DATA' (private) variable: "** << pnt->get() << endl; *//get and prints the value* **return 0**; } |

Output:

Text, letter

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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_