|  |
| --- |
| **Software Engineering Department - ITU** |
| **SE102BL: Object Oriented Programming Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 19/03/2024** |
| **Teaching Assistant: Zain ul Abidin, Muhammad Abdullah** | **Semester: Spring 2024** |
| **Lab Engineer: Muhammad Saqib Baig** | **Batch: BSSE2023(Section B)** |

# **Lab 8B. Solving Problems by using Singleton’s**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Roll number** | **Report**  **(out of 35)** | **Scaled to 10** |
|  |  |  |  |

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

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

## **Objective**

The objective of this lab is to practice on how to use a singleton in a program.

## **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**

**Singleton:**

Singleton refers to a design concept where a class ensures that only one instance of itself exists in the entire application and provides a global point of access to that instance.

**Purpose:**

* Ensures that a class has only one instance throughout the application’s lifetime.
* Provides a global access point for that instance, allowing easy access to shared resources or managing global states.

**Implementation:**

* Achieved by making the constructor of the class private to prevent external instantiation.
* Utilizes a static member variable to hold the single instance of the class.
* Offers a static member function (often named **‘getInstance’**) to access the single instance, creating it if necessary.

**Key Components:**

* Private Constructor
* Static Member Variable
* Static Member Function (**‘getInstance’**)

**Advantages:**

* Guarantees single instance existence, maintaining consistency across the application.
* Provides a global access point for easy integration and resource sharing.
* Supports lazy initialization, conserving resources until the instance is needed.

class Singleton {

public:

static Singleton\* getInstance( ) { if(instance == NULL)

instance = new Singleton;

return instance; }

private:

static Singleton \* instance;

Singleton() {} // Private constructor to prevent external instantiation

};

Singleton\* Singleton::instance = 0;

**Lab Tasks**

**Task 1**

**Library Management System**

**Problem Statement:**

A Library Management System is required to manage books, users, and library operations efficiently by using Singleton’s Class. The system should allow users to borrow, return, add, and remove books from the library. Additionally, administrators should have privileges to perform administrative tasks such as adding and removing books from the library.

**Classes:**

**1. Book:**

**Attributes:**

* **‘string title’ (private):** Title of the book.
* **‘bool available’** **(private):** Boolean flag indicating whether the book is available for borrowing.

**Constructors:**

* **‘Book( )’:** Default constructor.
* **‘Book(const string& t)’:** Parameterized constructor to set the title of the book.

**Methods:**

* **‘const string& getTitle( ) const`:** This method returns the title of the book.
* ‘**bool isAvailable( ) const’:** This method checks whether the book is available for borrowing or not.
* **‘void borrowBook( )`:** This method changes the availability status of the book to "borrowed".
* **‘void returnBook( )’:** This method changes the availability status of the book to "available".
* **‘friend ostream& operator<<(ostream& os, const Book& book)’:** Overloaded output stream operator to display book information.
* **‘friend istream& operator>>(istream& is, Book& book)’:** Overloaded input stream operator to input book information.

**2. Library:**

**Attributes:**

* **‘instance’ (private, static):** Singleton instance of the Library class.
* **‘books’ (private):** Array of books stored in the library.Use const int Max\_BOOKS is equal to 10;
* **‘int numBooks’ (private):** Number of books currently stored in the library.

**Constructors:**

* **‘Library( )’:** Private constructor.

**Methods:**

* **‘static Library\* getInstance( )’:** This static method is used to get the singleton instance of the Library class. If an instance doesn't exist, it creates one; otherwise, it returns the existing instance.
* **‘void addBook(const Book& book)’:** This method adds a book to the library if there's available space.
* **‘void removeBook(const Book& book)’:** This method removes a book from the library if it exists.
* **‘void displayBooks( ) const’:** This method displays all the books currently available in the library.
* **‘Book\* findBook(const string& title)’:** This method searches for a book in the library based on its title and returns a pointer to it if found.

**3. Admin:**

**Attributes:**

* None

**Constructors:**

* None

**Methods:**

* **‘static void addBookToLibrary(const Book& book)’:** This static method allows an admin to add a book to the library by calling the **‘addBook’** method of the **‘Library’** class.
* **‘static void removeBookFromLibrary(const string& title)’:** This static method allows an admin to remove a book from the library by calling the **‘removeBook’** method of the **‘Library’** class.

**4. User:**

**Attributes:**

* **‘string name’ (private):** Name of the user.

**Constructors:**

* **‘User(const string& n)’:** Constructor to initialize the user with a name.

**Methods:**

* **‘User(const string& n)’:** This constructor initializes a **‘User’** object with a provided name.
* **‘void borrowBook(Book& book)’:** This method allows a user to borrow a book by calling the **‘borrowBook’** method of the **‘Book’** class.
* **‘void returnBook(Book& book)’:** This method allows a user to return a book by calling the **‘returnBook’** method of the **‘Book’** class.
* **‘const string& getName( ) const’:** This method returns the name of the user.

**Usage of `getInstance`:**

Singleton\* obj1 = Singleton::getInstance( );

**Console Application (main.cpp):**

The program provides a menu-driven console interface to the user. Options include:

Library System Menu:

1. Display Library Books

2. Borrow a Book

3. Return a Book

4. Add a Book (Admin)

5. Remove a Book (Admin)

6. Exit

Enter your choice (1-6):

1. Adding a Book to the Library:

* Select option 4 from the main menu.
* Enter the title of the new book when prompted.
* The book will be added to the library if there is available space.

2. Removing a Book from the Library:

* Select option 5 from the main menu.
* Enter the title of the book to be removed when prompted.
* If the book is found, it will be removed from the library.

3. Borrowing a Book:

* Select option 2 from the main menu.
* Enter the title of the book to borrow when prompted.
* If the book is available, it will be borrowed by the user.

4. Returning a Book:

* Select option 3 from the main menu.
* Enter the title of the book to return when prompted.
* If the book is borrowed by the user, it will be returned to the library.

5. Displaying Library Books:

* Select option 1 from the main menu.
* All books currently available in the library will be displayed.

6. Exiting the Program:

* Select option 6 from the main menu to exit the program.

**Note:**

Ensure that the program files (‘Admin.h’, ‘Book.h’, ‘Library.h’, ‘User.h’, ‘Admin.cpp’, ‘Book.cpp’, ‘Library.cpp’, ‘User.cpp’, ‘main.cpp’, and ‘test.cpp’) are included and compiled together to run the Library System.

* All data members should be private unless there is a specific reason to make them public.
* Use appropriate access specifiers (public, private) for class members.
* Make a menu driven program in main.cpp.

Assessment Rubric for Lab

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Performance metric** | **CLO** | **Able to complete the task over 80% (4-5)** | **Able to complete the task 50-80% (2-3)** | **Able to complete the task below 50% (0-1)** | **Marks** |
| 1. Realization of experiment | 1 | Executes without errors excellent user prompts, good use of symbols, spacing in output. Through testing has been completed . | Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed . | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non- existent. No testing has been completed. |  |
| 1. Conducting experiment | 1 | Able to make changes and answered all questions. | Partially able to make changes and few incorrect answers. | Unable to make changes and answer all questions. |  |
| 1. Computer use | 2 | Document submission timely. | Document submission late. | Document submission not done. |  |
| 1. Teamwork | 3 | Actively engages and cooperates with other group member(s) in effective manner. | Cooperates with other group member(s) in a reasonable manner but conduct can be  improved. | Distracts or discourages other group members from conducting the experiment |  |
| 1. Laboratory safety and disciplinary rules | 3 | Code comments are added and does help the reader to understand the code. | Code comments are added and does not help the reader to understand the code. | Code comments are not added. |  |
| 1. Data collection | 3 | Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap. | Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables. | Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy. |  |
| 1. Data analysis | 4 | Solution is efficient, easy to understand, and maintain . | A logical solution that is easy to follow but it is not the most efficient. | A difficult and inefficient solution . |  |
| **Total (out of 35):** | | | | |  |