CC-211L

Object Oriented Programming

Laboratory 02

Introduction to Classes: Constructors and other Member Functions - II

Version: 1.0.0

Release Date: 31-1-2024

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

Learning Objectives:

- Concept of Encapsulation and Abstraction
- Getter/Setter Member Functions
- Public private access specifiers
- Constructors
- Overloaded Constructor
- Default Constructor
- Destructor

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Teacher Assistants	Syed M.Jafar Ali Naqvi	bitf21m032@pucit.edu.pk
	M. Usman Munir	bitf19m020@pucit.edu.pk
	Aqdas Shabir	bitf21m022@pucit.edu.pk
	Ibad Nazir	bitf21m024@pucit.edu.pk
	Umer Farooq	bitf21m010@pucit.edu.pk

Background and Overview:

Abstraction:

Abstraction in C++ is the process of hiding the implementation details from the user. It allows the user to focus on the functionality of an object without worrying about its internal implementation. Abstraction is achieved in C++ through the use of classes and objects, which can be used to define data and functions that can be used to manipulate the data. By hiding the implementation details, the user is able to use the object without having to worry about its internal working. This helps make code easier to understand and maintain.

Encapsulation:

Encapsulation in C++ is the process of combining data and functions into a single unit, called a class. This process allows data and functions to be accessed and used together while keeping the data hidden from outside users. Encapsulation is a powerful concept that helps keep code organized, secure, and manageable. By using encapsulation, developers can define the interactions between different parts of a program in a clear and concise way. This helps to reduce potential errors and maintain the program's functionality over time.

Class:

A class is an important concept in Object Oriented Programming (OOP). It is a blueprint that is used to create objects, which can hold data and perform functions. A class is made up of two parts: attributes and methods. Attributes are the properties of the class that describe its characteristics, such as its name, size, and color. Methods are the functions that the class can perform, such as adding, deleting, or editing data. Classes are reusable and allow for code to be written more efficiently. They are also a key tool for creating complex applications.

Object:

Objects are instances of classes, which are templates for creating objects. Objects have properties and methods that define their state and behavior, respectively. It is a self-contained entity that contains both data and instructions for manipulating that data. Objects are used in object-oriented programming (OOP) to represent real-world entities such as people, places, and things. Objects contain data and methods that describe the object's characteristics and behavior. The data is stored in the form of properties, while the methods are functions that can access and modify the object's data.

Comparison Between class and object:

Objects are often referred to as the "nouns" of programming, while classes are the "verbs". Objects are tangible, physical objects in the real world, while classes are abstract concepts that define how objects should behave.

Activities:

Pre-Lab Activities:

Getter/Setter Member Functions:

Getter/Setter member functions are used to control access to class data members in C++. Getters are used to retrieve data from the class, and setters are used to assign values to the class. They allow for the encapsulation of data within a class, which makes it easier to manage class data and ensures that data is used correctly.

Example:

For example, if a class has a data member called "age", it is best to create a getter and setter function for that data member. The getter function would return the age of the class object, and the setter function would allow us to set the age of the class object. This way, we can ensure that the data member is being properly used and make sure it is not being passed invalid data.

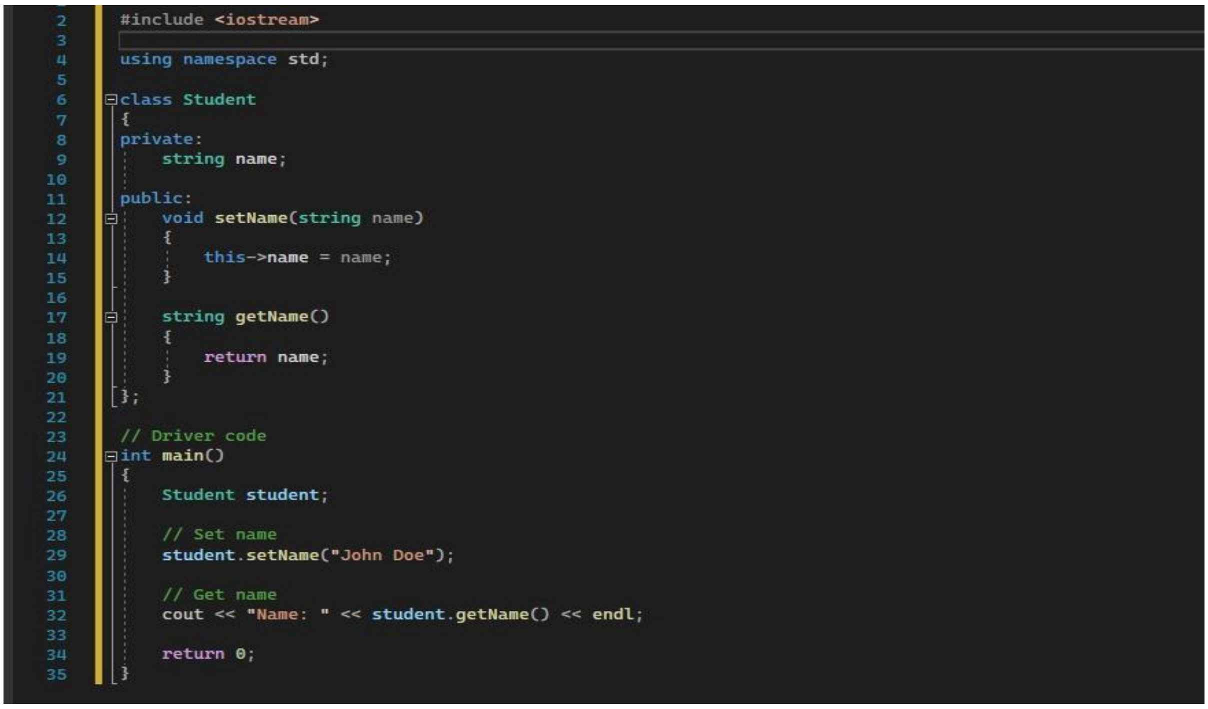
Syntax of Getters and Setters in C++:
Getter:
The syntax of Getter method is as follows

```
Type ClassName::getPropertyName(){
   return propertyName;
}
```

Setter:
The syntax of Setter method is as follows

```
void ClassName::setPropertyName(Type propertyName){
   this->propertyName = propertyName;
}
```

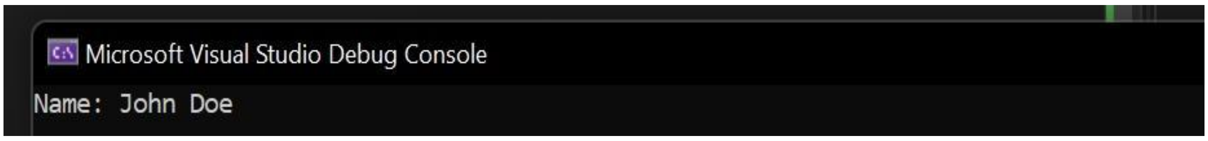
Example Code:



```
#include <iostream>

using namespace std;

class Student
{
private:
    string name;

public:
    void setName(string name)
    {
        this->name = name;
    }

    string getName()
    {
        return name;
    }
};

// Driver code
int main()
{
    Student student;

    // Set name
    student.setName("John Doe");

    // Get name
    cout << "Name: " << student.getName() << endl;

    return 0;
}
```

Fig. 01 (Getter and Setter Demonstration)

Output:



```
Name: John Doe
```

Fig. 02 (Output with respect to figure 1)

Some Key points about getter and setter methods:

- Getters and setters are used to access private member variables of a class from outside the class.
- Getters and setters should be used to ensure data integrity in the class by validating the input data and restricting access to certain areas of the class.
- Getters and setters should not be used to perform business logic or complex operations on private member variables.
- Getters should always return a copy of the private member variable and not the variable itself.
- Setters should always be used to modify the private member variable and not perform any other operations on the variable.
- Getters and setters should have meaningful names that clearly describe the purpose of the methods.
- Getters and setters should be declared as public methods of the class.

When to Use Getters and Setters in C++:

Getters and setters should be used when you want to maintain control over how users access and set the data members of a class. This allows you to control how the data is accessed and can be used to validate user input before setting a value or to perform calculations based on user input before setting a value. Getters and setters also make it easier to maintain the code, as the logic for how the data is accessed and set is self-contained.

Public/Private Access Specifiers:

Access Specifiers:

Access specifiers in C++ can be used to control the access level of the data members of a class. The three access specifiers used in C++ are public, protected, and private. In this introduction, we will discuss the purpose of public and private access specifiers and how to use them in C++.

Public Access Specifiers:

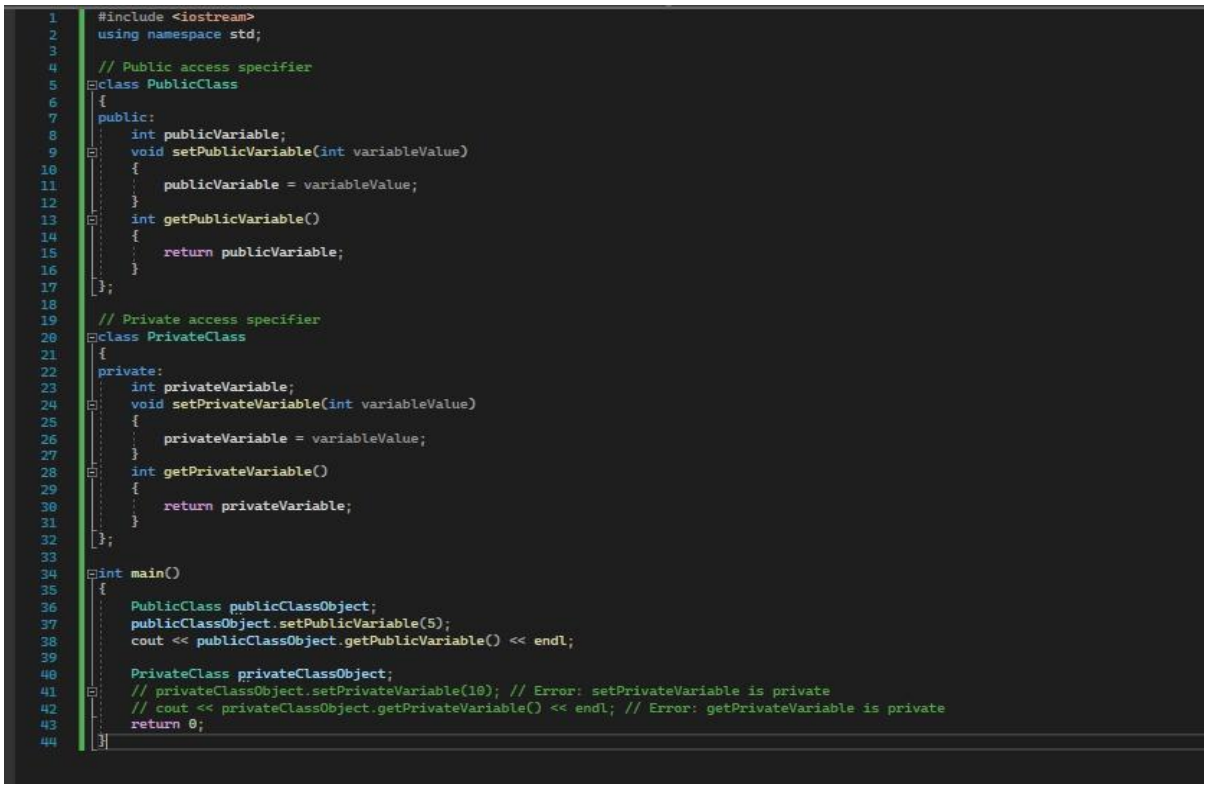
Public data members can be accessed and modified from anywhere within the program. To make a member of a class public, use the keyword “public” before the member declaration.

Private Access Specifiers:

Private data members can only be accessed and modified by the member functions of the class. They cannot be accessed or modified from outside the class. To make a member of a class private, use the keyword “private” before the member declaration.

How to Use Public and Private Access Specifiers:

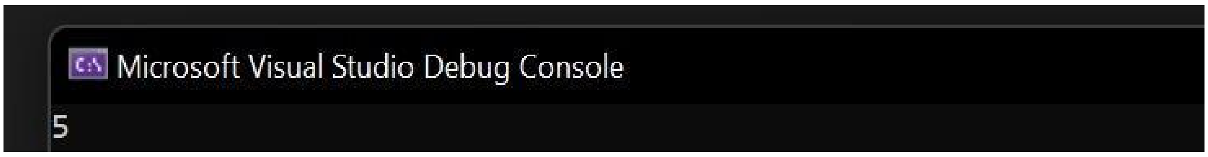
Example Code for Public and Private Access Specifiers:



```cpp
#include <iostream>
using namespace std;

// Public access specifier
class PublicClass
{
public:
    int publicVariable;
    void setPublicVariable(int variableValue)
    {
        publicVariable = variableValue;
    }
    int getPublicVariable()
    {
        return publicVariable;
    }
};

// Private access specifier
class PrivateClass
{
private:
    int privateVariable;
    void setPrivateVariable(int variableValue)
    {
        privateVariable = variableValue;
    }
    int getPrivateVariable()
    {
        return privateVariable;
    }
};

int main()
{
    PublicClass publicClassObject;
    publicClassObject.setPublicVariable(5);
    cout << publicClassObject.getPublicVariable() << endl;

    PrivateClass privateClassObject;
    // privateClassObject.setPrivateVariable(10); // Error: setPrivateVariable is private
    // cout << privateClassObject.getPrivateVariable() << endl; // Error: getPrivateVariable is private
    return 0;
}
```

Fig. 03 (Public and Private Access Specifiers)

Output:

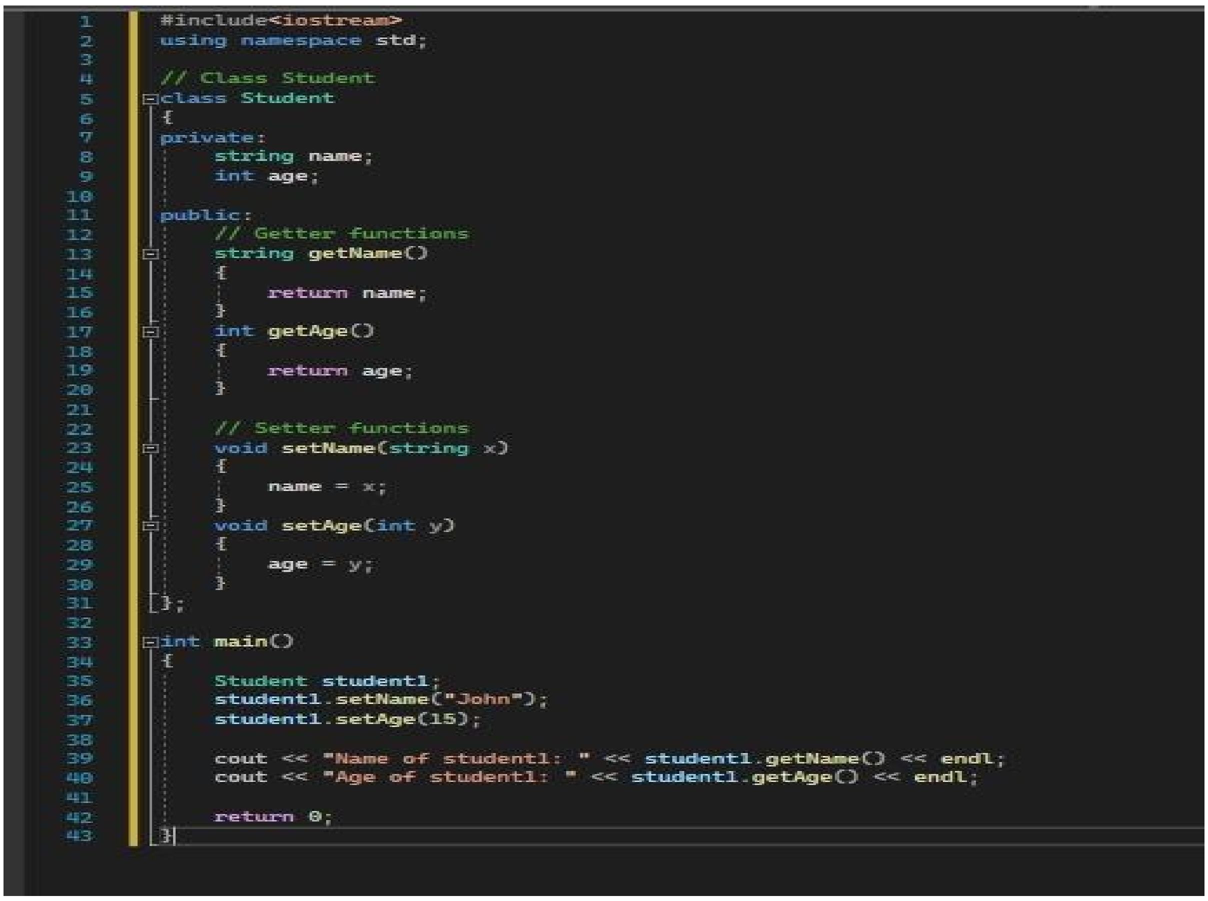


```
Microsoft Visual Studio Debug Console
5
```

Fig. 04 (Output for figure 3)

Demonstration of Encapsulation and Abstraction using Getter/Setter functions

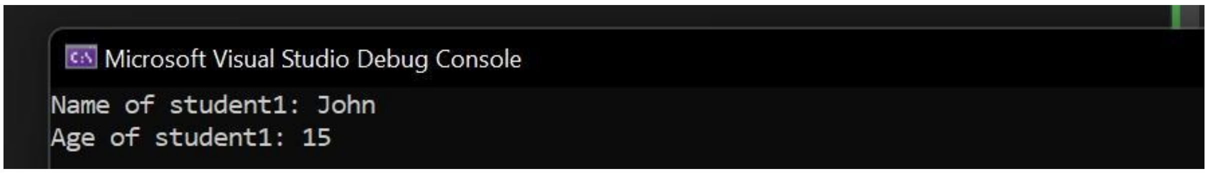
The demonstration of the encapsulation and abstraction using getter and setter functions is given in the code below:



```cpp
#include<iostream>
using namespace std;

// Class Student
class Student
{
private:
    string name;
    int age;

public:
    // Getter functions
    string getName()
    {
        return name;
    }
    int getAge()
    {
        return age;
    }

    // Setter functions
    void setName(string x)
    {
        name = x;
    }
    void setAge(int y)
    {
        age = y;
    }
};

int main()
{
    Student student1;
    student1.setName("John");
    student1.setAge(15);

    cout << "Name of student1: " << student1.getName() << endl;
    cout << "Age of student1: " << student1.getAge() << endl;

    return 0;
}
```

Fig. 05 (Demonstration of Encapsulation and Abstraction using Getter/Setter functions)

Output:



```
Microsoft Visual Studio Debug Console
Name of student1: John
Age of student1: 15
```

Fig. 06 (Output for figure 5)

Task 01: Little Humpty's Pharmacy **[Estimated 50 minutes / 40 marks]**

Little Humpty wants to start a new business. He has heard there is great scope in Pharmacy stores business. He wants to have Computerized system to handle all the medicines in a pharmacy. Being your friend he asks you for help and lucky for Humpty, you are a great programmer. Develop a system for Humpty's Pharmacy. He wants to have following info about medicines:

- Medicine id
- Medicine name
- Manufacturer
- Sail Person he bought medicine from.
- Price at which he bought
- Price to sell at
- Quantity

Humpty says he does not know how much medicine he will have but he will add them as we go along. He wants a nice little menu based system where he can add new medicine, watch the data for a medicine, watch the data of all the medicine, watch all the medicine he bought of a particular manufacturer, watch all the medicine he bought from a particular Sail Person, Total Quantity of each medicine he has at store. Humpty is a friend and not very smart, so provide him a nice user-friendly program, which will display user-friendly messages if Humpty makes any mistake. Advance Thanks from little Humpty!