# LAB SESSION 6

**Classes in C++**

**Objective:**

The goal of this lab is to understand the concept of classes in Object-Oriented Programming (OOP). By the end of this lab, you should be able to:

* Create objects (instances) of a class
* Understand and implement constructors
* Define and use instance variables and methods
* Understand and implement class variables and methods

**Introduction:**

**Classes**

In Object-Oriented Programming (OOP), a class is a blueprint for creating objects. A class encapsulates data for the object and methods to manipulate that data. Classes define the properties and behaviors of the objects created from them.

**Defining a Class**

Here is a basic example of defining a class in C++:

#include <iostream>

#include <string>

class Person {

public:

// Constructor

Person(std::string name, int age) : name(name), age(age) {}

// Instance method

std::string greet() {

return "Hello, my name is " + name + " and I am " + std::to\_string(age) + " years old.";

}

private:

std::string name;

int age;

};

int main() {

// Creating an instance of the Person class

Person person1("Alice", 30);

// Using the greet method

std::cout << person1.greet() << std::endl; // Output: Hello, my name is Alice and I am 30 years old.

return 0;

}

**Explanation:**

The Person class has a constructor that initializes the name and age attributes.

The greet method returns a greeting message using the instance attributes.

**Empty and Parameterized Constructor:**

In C++, constructors are special member functions of a class that are automatically called when an object of that class is created. Constructors can be of two types: empty constructors and parameterized constructors.

**Empty Constructor:** An empty constructor does not take any arguments. It is called when an object is created without providing any values. In the example provided, the empty constructor of the Student class is invoked when emptyStudent is created. This constructor simply prints a message indicating that an empty student object has been created.

**Parameterized Constructor:** A parameterized constructor takes one or more arguments. It is called when an object is created with specific values provided as arguments. In the example, the parameterized constructor of the Student class is invoked when parameterizedStudent is created with the arguments "John" and 20. This constructor initializes the name and age member variables of the Student object with the provided values and prints a message indicating the creation of a student object with the given name and age.

Both constructors are useful for initializing objects with appropriate values based on different scenarios. The choice between an empty constructor and a parameterized constructor depends on the requirements of the class and the desired behavior when creating objects.

#include <iostream>

#include <string>

class Student {

public:

// Empty constructor

Student() {

std::cout << "An empty student object is created." << std::endl;

}

// Parameterized constructor

Student(std::string name, int age) : name(name), age(age) {

std::cout << "A student object with name " << name << " and age " << age << " is created." << std::endl;

}

// Method to display student information

void displayInfo() {

std::cout << "Name: " << name << ", Age: " << age << std::endl;

}

private:

std::string name;

int age;

};

int main() {

// Creating objects using different constructors

Student emptyStudent; // Creating an empty student object

Student parameterizedStudent("John", 20); // Creating a parameterized student object

// Displaying student information

std::cout << "Details of the parameterized student:" << std::endl;

parameterizedStudent.displayInfo();

return 0;

}

**Public and Private Access Specifiers :**

Access specifiers define the accessibility of class members (variables and methods). The two most commonly used access specifiers are public and private.

public: Members declared as public are accessible from outside the class.

private: Members declared as private are only accessible within the class.

#include <iostream>

#include <string>

class Person {

public:

// Public members

std::string name;

// Constructor

Person(std::string name, int age) : name(name), age(age) {}

// Public method

void displayInfo() {

std::cout << "Name: " << name << ", Age: " << age << std::endl;

}

private:

// Private member

int age;

};

int main() {

Person person("Alice", 30);

// Accessing public member

std::cout << "Name: " << person.name << std::endl; // OK

// Trying to access private member (will cause a compilation error)

// std::cout << "Age: " << person.age << std::endl; // Error

// Accessing private member via public method

person.displayInfo(); // Output: Name: Alice, Age: 30

return 0;

}

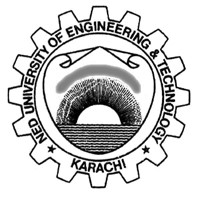
**Explanation:**

name is a public member and can be accessed directly from outside the class.

age is a private member and can only be accessed indirectly through a public method.

**Exercise:**

1. **Question:** Create a class **Rectangle** with attributes length and width. Implement methods to calculate the area and perimeter of the rectangle.
2. **Question:** Create a class **Circle** with attribute radius. Implement methods to calculate the area and circumference of the circle.
3. **Question:** Create a class **Employee** with attributes name and salary. Implement a method to display the details of the employee.
4. **Question:** Create a class **BankAccount** with attributes accountNumber, accountHolder, and balance. Implement methods to deposit and withdraw money from the account.
5. **Question**: Create a class **Car** with attributes brand, model, and year. Implement a method to display the details of the car.
6. **Question:** Create a class **Fraction** with attributes numerator and denominator. Implement a method to simplify the fraction.

**NED University of Engineering & Technology**

**Department of Software Engineering**

**Object Oriented Concepts and Programming**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **COGNITIVE DOMAIN ASSESSMENT RUBRIC LEVEL C3-PLO3** | | | | |
| **SKILL SETS** | **EXTENT OF ACHIEVEMENT** | | | |
| **CRITERIA** | **0-1** | **2-3** | **4-5** | **TOTAL** |
| **Understanding of Object-Oriented Concepts** | Poor Understanding of Object-Oriented Concepts | Fair  Understanding of Object-Oriented Concepts | Good  Understanding of Object-Oriented Concepts |  |
| **Design of Object-Oriented Solutions** | Poor Design of Object-Oriented Solutions | Fair Design of Object-Oriented Solutions | Good Design of Object-Oriented Solutions |  |
| **Implementation of Object-Oriented Solutions** | Poor Implementation of Object-Oriented Solutions | Fair Implementation of Object-Oriented Solutions | Good  Implementation of Object-Oriented Solutions |  |
| **Testing and Debugging** | Poor Testing and Debugging | Fair Testing and Debugging | Good Testing and Debugging |  |
| **Documentation and Comments** | Poor Documentation and Comments | Fair Documentation and Comments | Good Documentation and Comments | . |

**Laboratory Session No.** \_\_\_\_\_\_\_\_\_\_\_\_\_ **Date:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| **Weighted CLO (Psychomotor Score)** |  |
| **Remarks** |  |
| **Instructor’s Signature with Date:** |  |