Object-Oriented Programming

Lecture No. 4

Private access specifier, parametrized constructor, destructor, encapsulation, data hiding, abstraction, encapsulation, function/constructor overloading

Abstraction in OOP

- Abstraction is the concept of object-oriented programming that “shows” only essential attributes and “hides” unnecessary information. The main purpose of abstraction is hiding the unnecessary details from the users. e.g., students’ abstraction, teachers’ abstraction in a university management system.

Encapsulation

- Encapsulation is a process of combining data members and functions in a single unit called class. This is to prevent the access to the data directly, the access to them is provided through the functions of the class. It is one of the popular feature of Object Oriented Programming(OOP) that helps in data hiding.

Data hiding

- Data hiding is a technique of hiding internal object details, i.e., data members. It is an object-oriented programming technique. Data hiding ensures, or guarantees to restrict the data access to class members. It maintains data integrity.

Private access specifier

- Most restricted access specifier
- Members cannot be accessed (or viewed) from outside the class
- Accessible through public methods
- Default access specifier of class

Private access specifier - Example

```
class Student
{
        private:
                string name;
                int   rollNo;
                int   total;
                float perc;
        public:
                void getDetails();
                void putDetails();
};
int main()
{       Student std;
       //  cin>> std.name;
        std.getDetails();
        std.putDetails();
        return 0;
}
```

```
//member function definition, outside of the class
void Student::getDetails(){
        cout << "Enter name: " ;
        cin >> name;
        cout << "Enter roll number: ";
        cin >> rollNo;
        cout << "Enter total marks outof 500: ";
        cin >> total;
        perc= total/500*100;
}

//member function definition, outside of the class
void Student::putDetails(){
        cout << "Student details:\n";
        cout << "Name:"<< name << ",Roll Number:"
<< rollNo << ",Total:" << total << ",Percentage:" <<
perc;
}
```

Parameterized constructor

- It is used to initialize various data elements of different objects with different values when they are created.
- It is used to overload constructors.
- Default constructors are called when constructors are not defined for the classes.

Parameterized constructor and constructor overloading - Example

```
class Student
{
        private:
                string name;
                int   rollNo;
                int   total;
                float perc;
        public:
                void getDetails();
                void putDetails();
                Student() {   }
                Student(string name1, int rollNo1, int marks)
                {
                        name=name1;
                        rollNo=rollNo1;
                        total=marks;
                }
};

//member function definition, outside of the class
void Student::putDetails(){
        cout << "Student details:\n";
        cout << "Name:"<< name << ",Roll Number:" << rollNo <<
",Total:" << total << ",Percentage:" << perc;
}
```

```
//member function definition, outside of the class
void Student::getDetails(){
        cout << "Enter name: " ;
        cin >> name;
        cout << "Enter roll number: ";
        cin >> rollNo;
        cout << "Enter total marks outof 500: ";
        cin >> total;
        perc= (float)total/500*100;
}


int main()
{       Student std1;
        Student std2("Ali", 234, 450);
        std1.getDetails();
        std1.putDetails();
        std2.putDetails();
        return 0;
}
```

Constructor overloading

- We can have more than one constructor in a class with same name, as long as each has a different types or number of arguments. This concept is known as Constructor Overloading and is quite similar to function overloading.

Method/Function overloading

Based on:

- Signature – must be different
- Scope – must be same

Destructor

- Destructor is a special member function that is called when the lifetime of an object ends
- The purpose of the destructor is to free the resources that the object may have acquired during its lifetime
- It has same name as the class
- It is preceded by a tilde (~)
- Destructor does not take arguments, it can never be overloaded

Rules of destructor

- The name of the destructor must begin with the tilde character (~). You must not include any return type, not even void!
- A class can have no more than one destructor.
- The destructor cannot have any parameters.
- The class destructor is automatically invoked when the instance of that class goes out of scope.

Reference

- C++ How to Program
  By Deitel & Deitel

- The C++ Programming Language
  By Bjarne Stroustrup

- Object oriented programming using C++ by Tasleem Mustafa, Imran Saeed, Tariq Mehmood, Ahsan Raza

- https://www.tutorialspoint.com

- https://www.includehelp.com/cpp-programs/