Object Oriented Programming (CS1143)

Week 3

Department of Computer Science

Capital University of Science and Technology (CUST)

Outline

- Use of access specifiers (access modifiers)
- Concept of Constructor and Object Instantiation
- Importance of Destructor

Outline

- Use of access specifiers (access modifiers)
- Concept of Constructor and Object Instantiation
- Importance of Destructor

Access Modifiers

- The access modifier determines how a class can be accessed.
- The designer of a class can apply an access modifier to the declaration of a member (i.e. data member or member function) to control access to that member.
- C++ uses three modifiers
  - private
  - protected
  - public
- The declaration of data members and member functions in a class is by default **private**.
- When there is no access modifier for a member, it is private by default.

Access Modifiers Contd…

- When a member is **private**, it can only be accessed inside the class.
  - These members and member functions cannot be accessed directly for retrieving or changing.
  - They can be accessed only through member functions.
- When a member is **public**, it can be accessed from anywhere (inside the same class, inside the subclasses, and in the application).
- When a member is **protected**, it can be accessed inside the same class and inside the subclass but cannot be accessed from anywhere.
  - We will study subclasses later.

Modifier	Access from same class	Access from subclass	Access from anywhere
private	Yes	No	No
protected	Yes	Yes	No
public	Yes	Yes	Yes

Access Modifiers for Data Members

- The modifiers for data members are normally set to private for emphasis (although no modifier means private).
- This means that the data members are **not accessible directly**, they **must be accessed through the member functions**.
- We can also set them to public or protected if we want.

```
class Circle   // Header
{
   private:
         double radius;
```

Access Modifiers for Member Functions

- To operate on the data members, the application must use member functions, which means that the declaration of member functions usually must be set to public.
- Sometimes the modifier of a member function must be set to private, such as when a member function must help other member functions but is not allowed to be called by functions outside the class.

Example Private Data Member (P1)

```
1  #include <iostream>
2  using namespace std;
3
4  class Student
5  {
6      private:
7          string id;
8
9      public:
10         string name;
11         void setID(string n);
12
13 };
14
15 void Student :: setID (string stdID)
16 {
17     id=stdID;
18     cout<<"The ID has been set to: "<<id<<endl;
19 }
```

```
22 //*************************
23 //Application section
24 int main ( )
25 {
26     Student s1;
27     s1.setID("Student ID");
28     s1.id="Student ID";
29
30     return 0;
31 }
```

Example Public Data Member (P2)

```
1  #include <iostream>
2  using namespace std;
3  class Student
4  {
5      public:
6          string name;
7          void setName(string stdName);
8
9  };
10 void Student :: setName (string stdName)
11 {
12     name=stdName;
13     cout<<"The name has been set to: "<<stdName<<endl;
14 }
15 //************************************
16 //Application section
17 int main ( )
18 {
19     Student s1;
20     s1.setName("Ahmad");
21
22     s1.name="Ali";
23     cout<<s1.name;
24
25     return 0;
26 }
```

Example Protected Data Member (P3)

```
1  #include <iostream>
2  using namespace std;
3  class Student
4  {
5      protected:
6          string name;
7
8      public:
9          void setName(string stdName);
10
11 };
12 void Student :: setName (string stdName)
13 {
14     name=stdName;
15     cout<<"The name has been set to: "<<stdName<<endl;
16 }
17 //************************************
18 //Application section
19 int main ( )
20 {
21     Student s1;
22     s1.setName("Ahmad");
23
24     s1.name="Ali";
25     cout<<s1.name;
26
27     return 0;
28 }
```

Example Private Member Function (P4)

```
1  #include <iostream>
2  using namespace std;
3  class Student
4  {
5      private:
6          string name;
7          void setName(string stdName);
8
9  };
10 void Student :: setName (string stdName)
11 {
12     name=stdName;
13     cout<<"The name has been set to: "<<stdName<<endl;
14 }
15 //************************************
16 //Application section
17 int main ( )
18 {
19     Student s1;
20     s1.setName("Ahmad");
21
22     return 0;
23 }
```

Compiler (3) | Resources | Compile Log | Debug | Find Results | Close

Line	Col	File	Message
		D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1...	In function 'int main()':
10	6	D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-...	[Error] 'void Student::setName(std::string)' is private
20	20	D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-...	[Error] within this context

Group Modifier Access

- We have used only one keyword, private, and one keyword, public, in the whole class definition.
- This is referred to as group modification.
- A modifier is valid until we encounter a new one.

Outline

- Use of access specifiers (access modifiers)
- Concept of Constructor and Object Instantiation
- Importance of Destructor

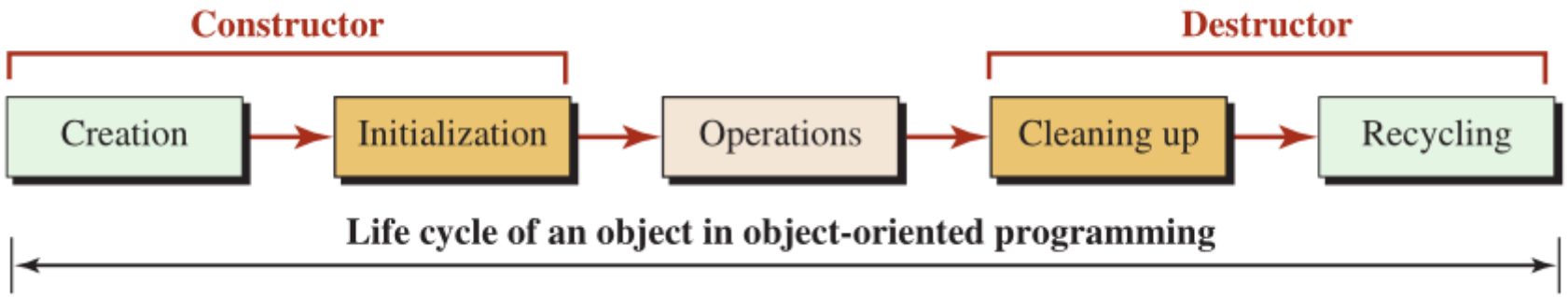
Constructor and Destructor

- We should first **create** the object and **initialize** its data members.
- Creation is done when a special member function named a **constructor** is called
- Initialization is when the body of a constructor is executed.

- When we do not need an object anymore, the object should be cleaned up and the memory occupied by the object should be recycled.
- Cleanup is automatically done when another special member function named a **destructor** is called.

A constructor is a special member function that creates and initializes an object.

A destructor is a special member function that cleans and destroys an object.


Constructor
Destructor
Creation
Initialization
Operations
Cleaning up
Recycling
Life cycle of an object in object-oriented programming

Constructor Declaration

- A constructor is a member function of the class, which means it must be declared in the class definition.
- A constructor has no return value, its name is the same as the name of the class, and it cannot have the const qualifier because the constructor initializes the value of the data Members
- All constructors of the class are normally public, so the application can call any of the constructors to initialize an object of the class.
- The following shows how we add the declaration of three constructors to our Circle class.

```
class Circle
{
   ...
   public:
        Circle (double radius);                 // Parameter Constructor
        Circle ();                              // Default Constructor
        Circle (const Circle& circle);          // Copy Constructor
        ...
}
```

Constructor Definition

- The main difference between the definition of a constructor and definition of other member functions is that a constructor can have an initialization list after the header to initialize the data members.
- The initialization list is put after the header and before the body of the constructor and it starts with a colon.
- If we need to initialize more than one data member, the initialization of each data member must be separated by a comma from other data members
- We can think of each initialization as an assignment statement that assigns the parameter to the data member,

  dataMember = parameter

```
: dataMemember (parameter), ... , dataMember (parameter)
```

Constructor Definition

```
// Definition of a default constructor
Circle :: Circle ()
: radius (1.0)  // Initialization list. If it is missing, radius is set to garbage values
{
   // Any other statements
}
```

Constructor Definition Contd..

- Another important point is that a constant data member of an object must be initialized when the object is created.
- C++ allows us to initialize it in the initialization section of a constructor.
- The body of a constructor can also be used for additional processing, such as validating a parameter, opening files if needed, or even printing a message to verify that the constructor was called.

Default Constructor

- The default constructor is a constructor with no parameters.
- It is used to create objects with each data member for all objects set to some literal values or default values.
- We must have at least one of **Default** or **Parameter** constructors in our class.
- If we write neither, the system provides a default constructor, referred to as a **synthetic default constructor**, that initializes each member to what is left over as garbage in the system.

Outline

- Use of access specifiers (access modifiers)
- Concept of Constructor and Object Instantiation
- Importance of Destructor

Destructor

- Like a constructor, a destructor has two special characteristics.
- First, the name of the destructor is the name of the class preceded by a tilde symbol (~)
- Second, like a constructor, a destructor cannot have a return value (not even void) because it returns nothing.
- A destructor is guaranteed to be automatically called and executed by the system when the object instantiated from the class goes out of scope.
- For example if we have instantiated five objects from the class, the destructor is automatically called five times to guarantee that all objects are cleaned up.
- A destructor can take no arguments.

A destructor is a special-purpose member function with no parameter and is designed to clean up and recycle an object.

Destructor Declaration

```
class Circle
{
   ...
   public:
        ...
        ~Circle ();                          // Destructor
}
```

Destruction Definition

- The definition of a destructor is similar to the definition of the other three member functions,
- But it must have a tilde (~) in front of the first name.
- A destructor should be public like all constructors.

```
// Definition of a destructor
Circle :: ~Circle ()
{
   // Any statements as needed
}
```

Creating and Destroying Objects

- Calling a constructor creates an object.
- When the constructor is executed, it initializes the data members.
- When a destructor is executed, the data members are cleaned up.
- A destructor is automatically called and executed by the system

```
// Instantiation of circle1
Circle circle1;
```

```cpp
#include <iostream>
using namespace std;
class Circle
{
    private:
        double radius;
    public:
        Circle (); // Default Constructor
        ~Circle(); //Destructor
};

// Definition of default constructor
Circle :: Circle ()
: radius (0.0)
{
    cout << "The default constructor was called. " << endl;
}

// Definition of destructor
Circle :: ~Circle ()
{
    cout << "The destructor was called for circle with radius "<<radius<<endl ;
}

int main ( )
{
    // Instantiation of circle1 and applying operations on it
    Circle circle1;

    return 0;
}
```