CS-1201 Object Oriented Programming

Friend Class and Copy Constructors

Arbish Akram

Department of Computer Science
Government College University

Friend Functions and Classes

- A friend is a function or class that is not a member of a class but has access to its private members.
- Private members are hidden from all parts of the program outside the class, requiring public member functions for access. However, a friend function can bypass this restriction.
- A friend function can be a stand-alone function or a member of another class.
- An entire class can be declared as a friend of another class.

Example 1: Friend Function

```
1  // C++ program to demonstrate the working of the friend function
2   class Distance {
3       private:
4           int meter;
5           // friend function
6           friend int addFive(Distance);
7       public:
8           Distance() : meter(0) {}
9   };
10  // friend function definition
11  int addFive(Distance d) {
12      //accessing private members from the friend function
13      d.meter += 5;
14      return d.meter;
15  }
16  int main() {
17      Distance D;
18      cout << "Distance: " << addFive(D);
19      return 0;
20  }
```

Example 2: Friend Function

```
1  // Add members of two different classes using friend functions
2   #include <iostream>
3   using namespace std;
4   // forward declaration
5   class ClassB;
6   class ClassA {
7       public:
8           // constructor to initialize numA to 12
9           ClassA() : numA(12) {}
10      private:
11          int numA;
12           // friend function declaration
13           friend int add(ClassA, ClassB);
14  };
15  class ClassB {
16      public:
17          // constructor to initialize numB to 1
18          ClassB() : numB(1) {}
19      private:
20          int numB;
21          // friend function declaration
22          friend int add(ClassA, ClassB);
23  };
```

Example 2: Friend Function

```
1  // access members of both classes
2  int add(ClassA objectA, ClassB objectB) {
3      return (objectA.numA + objectB.numB);
4  }
5
6  int main() {
7      ClassA objectA;
8      ClassB objectB;
9      cout << "Sum: " << add(objectA, objectB);
10     return 0;
11 }
```

- `ClassA` and `ClassB` have declared `add()` as a friend function, it can access the private data of both classes.
- The friend function inside `ClassA` may utilize `ClassB`, even if `ClassB` is not defined at that point.
- To resolve this, a forward declaration of `ClassB` is required in the program.

Friend Class

- A friend class can access the private and protected members of another class using the `friend` keyword.
- All member functions of `ClassB` become friend functions of `ClassA`.
- `ClassB` can access all members of `ClassA`.
- However, `ClassA` cannot access members of `ClassB`.

```
class ClassB; // Forward declaration
class ClassA {
    friend class ClassB; // ClassB is a friend class
    // ClassA members
};
class ClassB {
    // ClassB members
};
```

Example: Friend Class

```
1  // C++ program to demonstrate the working of friend class
2
3  #include <iostream>
4  using namespace std;
5
6  // forward declaration
7  class ClassB;
8
9  class ClassA {
10     private:
11         int numA;
12         // friend class declaration
13         friend class ClassB;
14
15     public:
16         // constructor to initialize numA to 12
17         ClassA() : numA(12) {}
18
19 };
```

Example: Friend Class

```
1  class ClassB {
2      private:
3          int numB;
4      public:
5          // constructor to initialize numB to 1
6          ClassB() : numB(1) {}
7      // member function to add numA
8      // from ClassA and numB from ClassB
9      int add() {
10         ClassA objectA;
11         return objectA.numA + numB;
12     }
13 };
14
15 int main() {
16     ClassB objectB;
17     cout << "Sum: " << objectB.add();
18     return 0;
19 }
```

Cascading Member Functions

- Cascading allows us to call multiple member functions for an object in a single statement.
- Let's say `sq` is an object of the class `Square`. To print the area and perimeter, we can call these methods like this:

```
sq.area();
sq.perimeter();
```

- Using cascading, we can achieve the same result in a single statement:

```
sq.area().perimeter();
```

Example

```
1  class Square
2  {
3    public:
4      int side;
5      Square area()
6      {
7        cout << "Area of the square is :" << side*side <<endl;
8        return *this;
9      }
10     Square perimeter()
11     {
12       cout << "Perimeter of the square is :" << 4*side <<endl;
13       return *this;
14     }
15 };
16
17 int main()
18 {
19   Square sq;
20   sq.side =3;
21   sq.area().perimeter(); //cascading function calls
22   return 0;
23 }
```

Destructor

- A destructor is a special member function that is executed when an object of a class is destroyed.
- It is used to free resources allocated to the object.
- The destructor has the same name as the class, preceded by a tilde (~):

```
~ClassName() {
    // Cleanup code
}
```

- Cannot be overloaded and does not take parameters or return values.

Example

```
1  class Demo {
2      public:
3          Demo() {
4              cout << "Constructor called." << endl;
5          }
6
7          ~Demo() {
8              cout << "Destructor called." << endl;
9          }
10 };
11
12 int main() {
13     Demo obj; // Constructor is called
14     // Destructor is called automatically when obj goes out of scope
15     return 0;
16 }
```

Types of Constructors

- Default Constructor:
  - A constructor with no parameters.
  - Automatically provided by the compiler if no constructors are defined.
- Parameterized Constructor:
  - A constructor that takes arguments to initialize an object with specific values.
- Copy Constructor:
  - A constructor that creates a new object as a copy of an existing object.

Copy Constructor

Two types of copy constructors

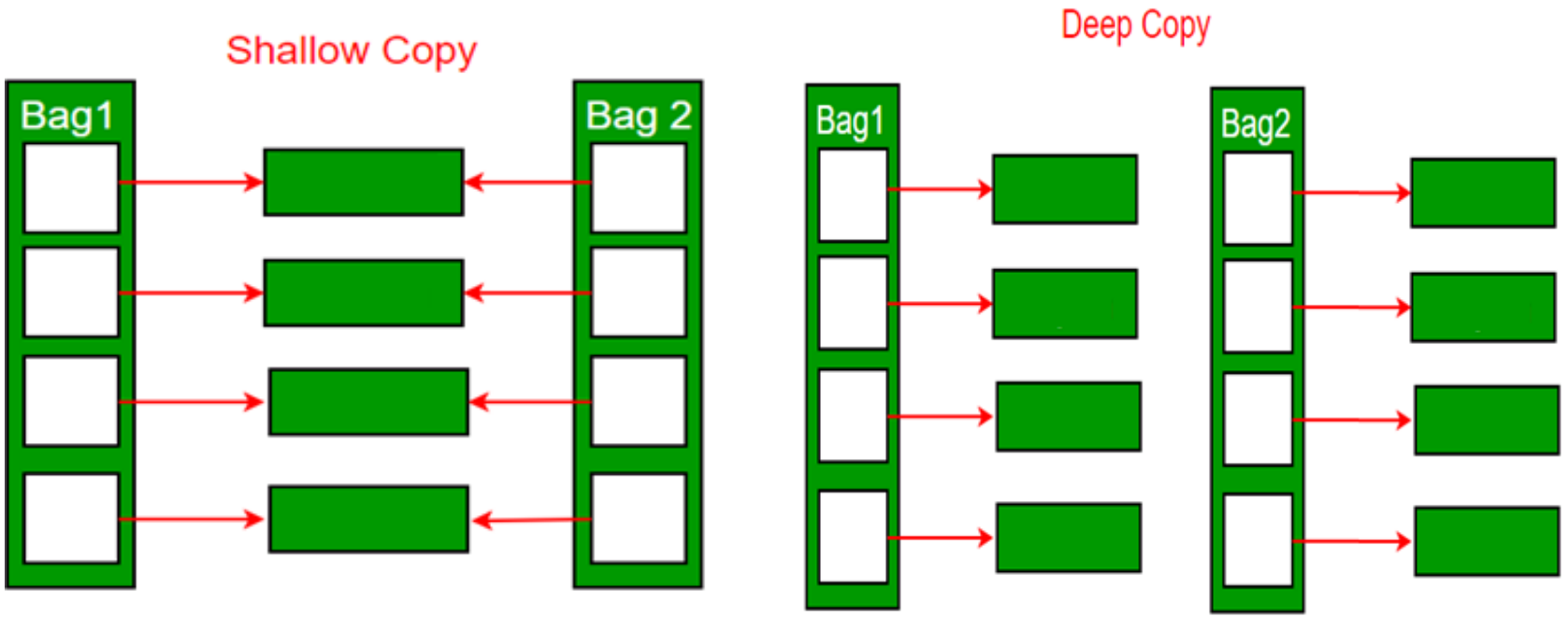
- Default copy constructor
- User-defined copy constructor

Default Copy Constructor

- Automatically provided by the compiler.
- Performs a shallow copy of the object's members.
- Used when an object is initialized with another object of the same type.

```
1  class MyClass {
2      public:
3          int value;
4  };
5
6  MyClass obj1;
7  obj1.value = 10;
8  MyClass obj2 = obj1; // Default copy constructor is called
```

Deep vs Shallow Copy



User-defined Copy Constructor

```
ClassName(const ClassName &other) {
    // Copy member variables from 'other'
}
```

- `ClassName`: The name of the class.
- `const ClassName &other`: A reference to another object of the same class, marked as `const` to prevent modification.
- `body`: Code to copy the relevant data members from `other` to the new object.

Example

```
1  // C++ program to illustrate the use of copy constructor
2   #include <iostream>
3   using namespace std;
4   class Student {
5       int rollNumber;
6       string Name;
7       public:
8           Student(int, string);
9           // Copy constructor
10          Student(const Student &t) {
11              rollNumber = t.rollNumber;
12              Name = t.Name;
13              cout << "Copy Constructor Called" << endl;
14          }
15          // Function to display student details
16          void display();
17  };
18
19  // Implementation of the parameterized constructor
20  Student::Student(int number, string name) {
21      rollNumber = number;
22      Name = name;
23  }
```

Example

```
1  // Implementation of the display function
2  void Student::display() {
3      cout << rollNumber << "\t" << Name << endl;
4  }
5
6  int main() {
7      // Create student object with parameterized constructor
8      Student s1(501, "Ali");
9      s1.display();
10
11     // Create another student object using the copy constructor
12     Student s2(s1);
13     s2.display();
14
15     return 0;
16 }
```