Programming with C and C++

CSC-101 (Lecture 36)

Dr. R. Balasubramanian
Professor
Department of Computer Science and Engineering
Mehta Family School of Data Science and Artificial Intelligence
Indian Institute of Technology Roorkee
Roorkee 247 667

bala@cs.iitr.ac.in
https://faculty.iitr.ac.in/cs/bala/

The private Members

- A **private** member variable or function cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members.

```
1  #include <iostream>
2  using namespace std;
3
4  class Box {
5     public:
6        double length;
7        void setWidth( double wid );
8        double getWidth( void );
9
10    private:
11       double width;
12 };
13
14 // Member functions definitions
15 double Box::getWidth(void) {
16    return width ;
17 }
18
```

https://ideone.com/pK2jzV

```
19  void Box::setWidth( double wid ) {
20      width = wid;
21  }
22
23  // Main function for the program
24  int main() {
25      Box box;
26      // set box length without member function
27      box.length = 10.0; // OK: because length is public
28      cout << "Length of box : " << box.length <<endl;
29
30      // set box width without member function
31      // box.width = 10.0; // Error: because width is private
32      box.setWidth(10.0);  // Use member function to set it.
33      cout << "Width of box : " << box.getWidth() <<endl;
34      return 0;
35  }
```

Output

stdout

```
Length of box : 10
Width of box : 10
```

The protected Members

- A protected member variable or function is very similar to a private member but it provides one additional benefit that they can be accessed in child classes which are called derived classes.

The protected Members

https://ideone.com/L2wch5

```
1  #include <iostream>
2  using namespace std;
3
4  class Box {
5     protected:
6        double width;
7  };
8
9  class SmallBox:Box { // SmallBox is the derived class.
10    public:
11       void setSmallWidth( double wid );
12       double getSmallWidth( void );
13 };
14
```

```
15  // Member functions of child class
16  double SmallBox::getSmallWidth(void) {
17     return width ;
18  }
19  void SmallBox::setSmallWidth( double wid ) {
20     width = wid;
21  }
22
23  // Main function for the program
24  int main() {
25     SmallBox box;
26
27     // set box width using member function
28     box.setSmallWidth(5.0);
29     cout << "Width of box : "<< box.getSmallWidth() << endl;
30
31     return 0;
32  }
```

stdout

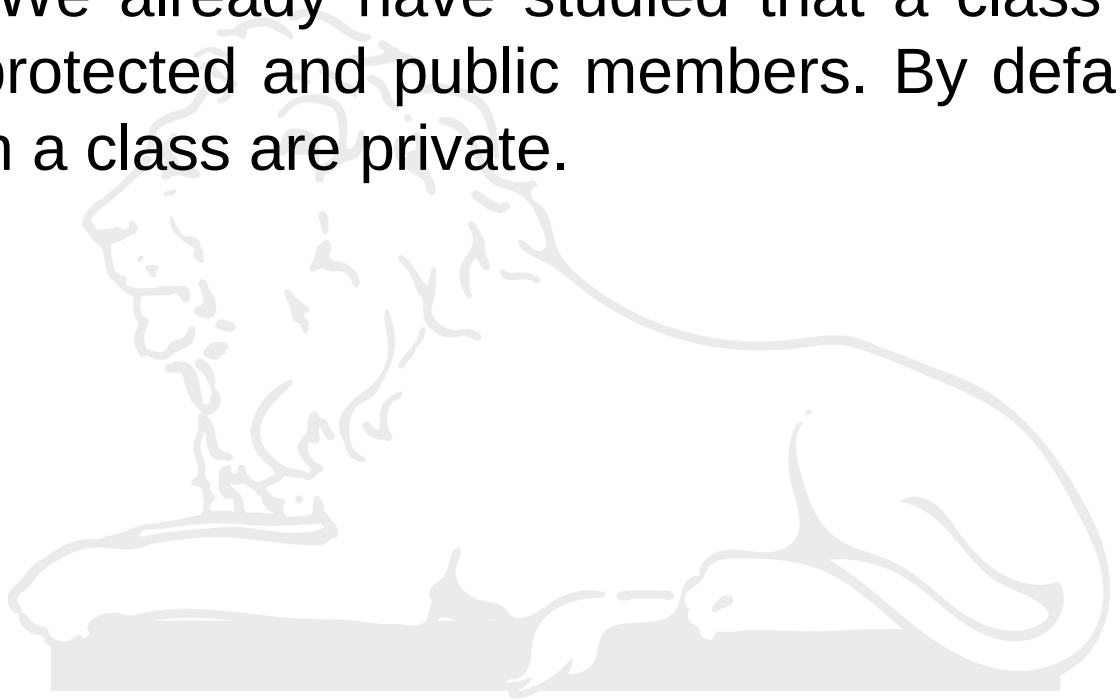
```
Width of box : 5
```

Encapsulation

- Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse.
- Data encapsulation led to the important OOP concept of **data hiding.**
- Data encapsulation is a mechanism of bundling the data, and the functions that use them and **data abstraction** is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

Encapsulation

- C++ supports the properties of encapsulation and data hiding through the creation of user-defined types, called classes. We already have studied that a class can contain private, protected and public members. By default, all items defined in a class are private.



Encapsulation

```
1   #include <iostream>
2   using namespace std;
3
4   class Adder {
5      public:
6         // constructor
7         Adder(int i = 0) {
8            total = i;
9         }
10
11        // interface to outside world
12        void addNum(int number) {
13           total += number;
14        }
15
```

https://ideone.com/OuLtIM

```
16          // interface to outside world
17          int getTotal() {
18              return total;
19          };
20
21      private:
22          // hidden data from outside world
23          int total;
24  };
25
26  int main() {
27      Adder a;
28
29      a.addNum(10);
30      a.addNum(20);
31      a.addNum(30);
32
33      cout << "Total " << a.getTotal() <<endl;
34      return 0;
35  }
```

stdout

```
Total 60
```

getter and setter functions

```
1   #include <iostream>
2   using namespace std;
3
4   class Circle {
5   private:
6       double radius;
7
8   public:
9       void setRadius(double r) {
10          if (r > 0) {
11              radius = r;
12          } else {
13              cout << "Please enter a valid radius." << endl;
14          }
15      }
16
```

https://ideone.com/VHT10o

```
17      double getRadius() {
18          return radius;
19      }
20
21      double calculateArea() {
22          return 3.14 * radius * radius;
23      }
24  };
25
26  int main() {
27      Circle c;
28      // Set the radius using the setter method
29      c.setRadius(5.0);
30      // Get the radius using the getter method
31      cout << "Radius of the circle: " << c.getRadius() << endl;
32      cout << "Area of the circle: " << c.calculateArea() << endl;
33      return 0;
34  }
```

stdout

```
Radius of the circle: 5
Area of the circle: 78.5
```

Data Abstraction

```
1   #include <iostream>
2   using namespace std;
3
4   class Adder {
5      public:
6         // constructor
7         Adder(int i = 0) {
8            total = i;
9         }
10
11        // interface to outside world
12        void addNum(int number) {
13           total += number;
14        }
15
```

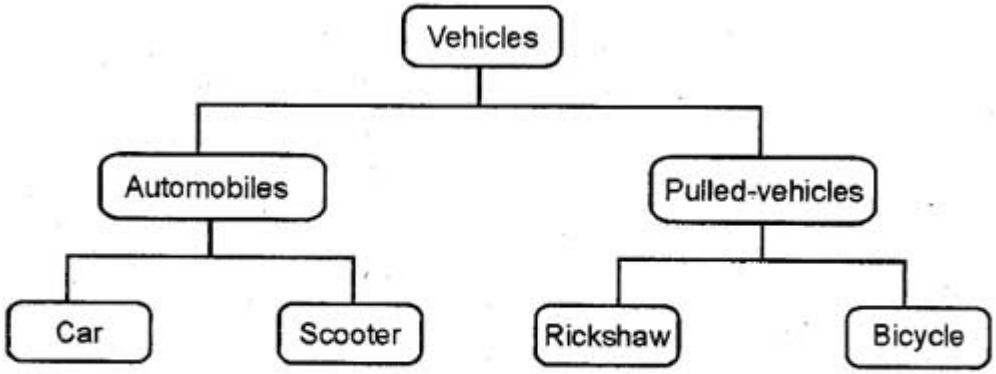
https://ideone.com/OuLtIM

```
16          // interface to outside world
17          int getTotal() {
18              return total;
19          };
20
21      private:
22          // hidden data from outside world
23          int total;
24  };
25
26  int main() {
27      Adder a;
28
29      a.addNum(10);
30      a.addNum(20);
31      a.addNum(30);
32
33      cout << "Total " << a.getTotal() <<endl;
34      return 0;
35  }
```

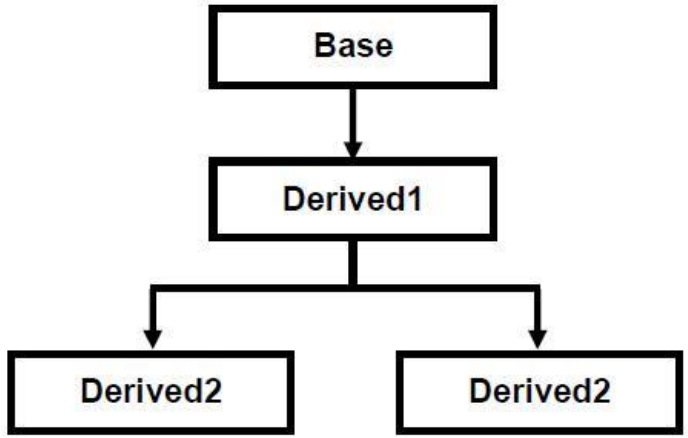
stdout

```
Total 60
```

Inheritance



Inheritance



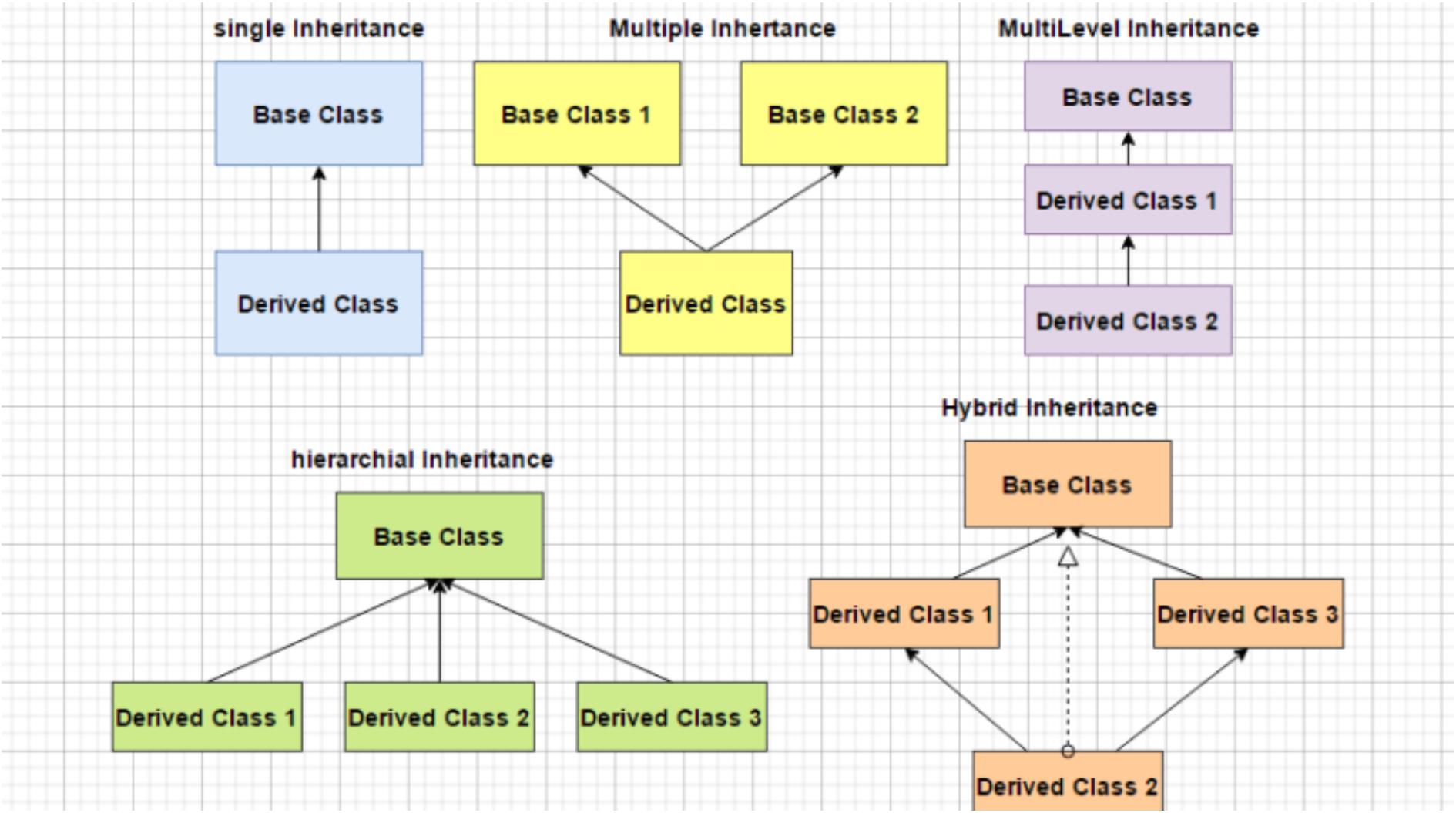
Inheritance

- Inheritance can be defined as the process where one class acquires the properties (functions and fields) of another.
- With the use of inheritance the information is made manageable in a hierarchical order.
- The class which inherits the properties of other is known as derived class (subclass, child class) and the class whose properties are inherited is known as base class (superclass, parent class).

Use of inheritance in C++

- For Function Overriding (runtime polymorphism can be achieved).
- For Code Reusability.

Types of Inheritance



Derived Classes

- The Syntax of Derived class:

```
class derived_class_name :: visibility-mode
base_class_name
{
    // body of the derived class.
}
```

- **derived_class_name:** It is the name of the derived class.
- **visibility mode:** The visibility mode specifies whether the features of the base class are publicly inherited or privately inherited. It can be public or private.
- **base_class_name:** It is the name of the base class.

Inheritance

- When the base class is privately inherited by the derived class, public members of the base class becomes the private members of the derived class.
- Therefore, the public members of the base class are not accessible by the objects of the derived class only by the member functions of the derived class.

- When the base class is publicly inherited by the derived class, public members of the base class also become the public members of the derived class.
- Therefore, the public members of the base class are accessible by the objects of the derived class as well as by the member functions of the base class.

Thank You!