Advanced Programming Lab
CS6150

18-August-2025

Anantha Padmanabha and Meghana Nasre

Inheritence

(Slides Courtesy : Rupesh Nasre)

Reuse

- In large software systems, it is not a good idea to start from scratch every time.
  - We should reuse existing functionality and build upon it.
- Reuse in procedural style is achieved using function libraries.
- OOP provides us with another interesting way to reuse the functionality of a class.
  - An MTech student is a student, and so is a BTech student.

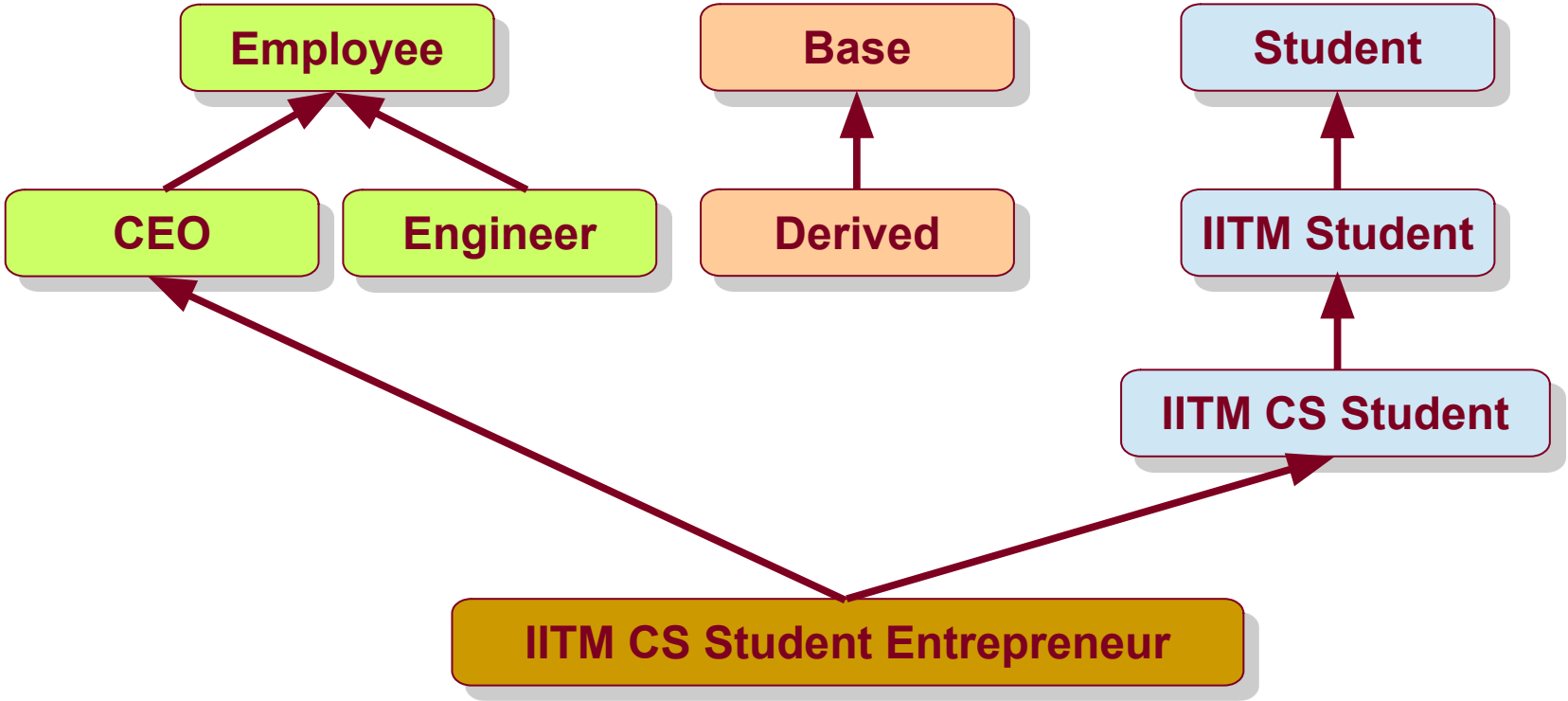
Inheritance

- **Base class**: Parent class with some functionality.

- **Derived class**: Child class which inherits properties of the parent class and defines its own.
  - It also would add other functionality.
  - Similar to how we inherit styles / behavior of our parents.

```cpp
#include <iostream>
using namespace std;

// Base class
class Animal {
public:
    void eat() {
        cout << "This animal eats food." << endl;
    }

    void sleep() {
        cout << "This animal is sleeping." << endl;
    }
};

// Derived class (inherits from Animal)
class Dog : public Animal {
public:
    void bark() {
        cout << "The dog barks." << endl;
    }
};

int main() {
    Dog myDog;

    // Calling methods from the base class
    myDog.eat();     // inherited from Animal
    myDog.sleep();   // inherited from Animal

    // Calling method from the derived class
    myDog.bark();

    return 0;
}
```

Derivation


Employee
Base
Student
CEO
Engineer
Derived
IITM Student
IITM CS Student
IITM CS Student Entrepreneur

What all is inherited?

- An object of a derived class has stored in it all the fields of the base type.
- An object of the derived type can use the methods of the base type.
- But
  - Derived class needs its own constructor(s)
  - Appropriate base constructor needs to be invoked explicitly (otherwise, default is executed if exists)
  - Need to respect the access permissions

Access Permissions

- A derived class method can access (irrespective of Inheritance type)
  - All public member functions and fields of base
  - All protected member functions and fields of base
  - All methods and fields of itself
- A derived class method cannot access (irrespective of Inheritance type)
  - Any private methods or fields of base
  - Any protected or private members of any other class

	public	protected	private
class	✓	✓	✓
children	✓	✓	×
rest	✓	×	×

Access Permissions

- We can specify how/what we want to access in the derived class

Inheritance Type	**Base public**	**Base protected**	**Base private**
Public			
Protected			
Private			

```cpp
// Base class
class Animal {
public:
    void eat() {
        cout << "This animal eats food." << endl;
    }

    void sleep() {
        cout << "This animal is sleeping." << endl;
    }
};

// Derived class (inherits from Animal)
class Dog : public Animal {
public:
    void bark() {
        cout << "The dog barks." << endl;
    }
};
```

Access Permissions

- We can specify how/what we want to access in the derived class

Inheritance Type	**Base public**	**Base protected**	**Base private**
Public	Public	Protected	Inaccessible
Protected	Protected	Protected	Inaccessible
Private	Private	Private	Inaccessible

```
// Base class
class Animal {
public:
    void eat() {
        cout << "This animal eats food." << endl;
    }

    void sleep() {
        cout << "This animal is sleeping." << endl;
    }
};

// Derived class (inherits from Animal)
class Dog : public Animal {
public:
    void bark() {
        cout << "The dog barks." << endl;
    }
};
```

Constructors

- A derived class constructor needs to call a specific base class constructor explicitly.
- This cannot be done using an executable instruction in the body of the constructor.
- Base class object is constructed first.

```cpp
#include <iostream>
using namespace std;

// Base class with a parameterized constructor
class Parent {
    int x;
public:
    // Base class's parameterized constructor
    Parent(int i) {
        x = i;
        cout << "Inside base class's parameterized constructor" << endl;
    }
};

// Derived class that calls base class constructor in its initializer list
class Child : public Parent {
public:
    // Derived class's parameterized constructor
    Child(int x) : Parent(x) {
        cout << "Inside derived class's parameterized constructor" << endl;
    }
};

int main() {
    Child obj1(10);
    return 0;
}
```

Destructors

- Destructors get called in the reverse order than the constructors.
- First derived class, then base class destructor
- A special consideration is required when a Base class pointer / reference points to a derived class object, and is deleted.

```cpp
#include <iostream>
using namespace std;

// Base class with constructor & destructor
class Parent {
    int x;
public:
    // Base class constructor
    Parent(int i) {
        x = i;
        cout << "Inside Parent constructor, x = " << x << endl;
    }

    // Base class destructor
    ~Parent() {
        cout << "Inside Parent destructor" << endl;
    }
};

// Derived class with constructor & destructor
class Child : public Parent {
public:
    // Derived class constructor calls base constructor in initializer list
    Child(int x) : Parent(x) {
        cout << "Inside Child constructor" << endl;
    }

    // Derived class destructor
    ~Child() {
        cout << "Inside Child destructor" << endl;
    }
};

int main() {
    cout << "Creating object:\n";
    Child obj1(10);

    cout << "Object going out of scope:\n";
    return 0;
}
```

Pointers and Inheritance

- C++ has quite strong rules towards types.
- Student* pointer cannot point to Instructor class object.
- However, a base class pointer can point to derived class object.
    - Helpful in keeping track of all objects derived from the same class together.
    - We can call appropriate methods of different derived classes with the same pointer.
    - Keeping track of all students together.

```cpp
#include <iostream>
using namespace std;

class Student {
public:
   void display() {
       cout << "I am a Student." << endl;
    }
};

class BTechStudent : public Student {
  private:
       string BTechproject[50];

};

class MTechStudent : public Student {
    private:
       string MTechproject[50];
};

int main() {
    Student* students[2];

    students[0] = new BTechStudent();
    students[1] = new MTechStudent();

    for (int i = 0; i < 2; ++i) {
        students[i]->display();
    }

    for (int i = 0; i < 2; ++i)
        delete students[i];

    return 0;
}
```

Pointers and Inheritance

- Pointers can also be dynamically assigned at run time

```cpp
#include <iostream>
using namespace std;

class Student {
public:
   void display() {
       cout << "I am a Student." << endl;
    }
};

class BTechStudent : public Student {
  private:
       string BTechproject[50];

};

class MTechStudent : public Student {
    private:
       string MTechproject[50];
};

int main() {
    Student* student;
    int choice;

    cout<<"Enter 1 to create BTech student and enter 2 to create MTech Student"<<endl;

    cin >> choice;

    if (choice == 1) {
        student = new BTechStudent();
    } else if (choice == 2) {
        student = new MTechStudent();
    }

    student->display();
    delete student;
    return 0;
}
```

Pointers and Inheritance

- Deleting a derived object automatically calls derived destructor and then the base destructor.

- However, deleting a base pointer pointing to derived object calls only base destructor.

Pointers and Inheritance

- Deleting a base pointer pointing to derived object calls only base destructor.

- If you want to call the destructor of the derived class (and then base class) in such a case, then you need to mark the base destructor virtual.
    - Virtual methods in the next week

Multiple Inheritance

- C++ allows deriving from multiple base classes.
  - Java doesn't.
- The derived class inherits properties of both the base classes.
- If there is ambiguity (same method in both bases), compiler issues an error.

```cpp
#include <iostream>
using namespace std;

// First base class
class Father {
public:
    void fatherFunc() {
        cout << "Function from Father class." << endl;
    }
};

// Second base class
class Mother {
public:
    void motherFunc() {
        cout << "Function from Mother class." << endl;
    }
};

// Derived class inheriting from both Father and Mother
class Child : public Father, public Mother {
public:
    void childFunc() {
        cout << "Function from Child class." << endl;
    }
};

int main() {
    Child obj;

    obj.fatherFunc();  // from Father class
    obj.motherFunc();  // from Mother class
    obj.childFunc();   // from Child class

    return 0;
}
```

Friend Function

- Special function that is not a member of a class but is granted access to the class's private and protected members.
- Friend function can manipulate or access the internal data of the class even though it is defined outside the class.

```cpp
#include <iostream>
using namespace std;

class MyClass {
private:
    int secret;

public:
    MyClass(int val) : secret(val) {}

    // Declare friend function
    friend void revealSecret(MyClass obj);
};

// Friend function definition
void revealSecret(MyClass obj) {
    // Can access private members directly
    cout << "The secret value is: " << obj.secret << endl;
}

int main() {
    MyClass obj(42);

    // Call the friend function
    revealSecret(obj);

    return 0;
}
```

Friend Function

- Special function that is not a member of a class but is granted access to the class's private and protected members.
- Friend function can manipulate or access the internal data of the class even though it is defined outside the class.

```cpp
#include <iostream>
using namespace std;


class ClassB;

class ClassA {
private:
    int numA;
public:
    ClassA() : numA(12) {}

    friend int add(ClassA, ClassB);
};

class ClassB {
private:
    int numB;
public:
    ClassB() : numB(1) {}

    friend int add(ClassA, ClassB);
};


int add(ClassA objectA, ClassB objectB) {

    return (objectA.numA + objectB.numB);
}

int main() {
    ClassA objectA;
    ClassB objectB;
    cout << "Sum: " << add(objectA, objectB) << endl;
    return 0;
}
```

See you in the lab on Friday

Try out examples

Practise problems will be available by tomorrow