Inheritance and composition

Victor Eijkhout, Susan Lindsey

Fall 2025

last formatted: October 7, 2025

Composition

1. Has-a relationship

A class usually contains data members. These can be simple types or other classes. This allows you to reflect relations between things you are modeling.

```
1 class Person {
2   string name;
3   ....
4 };
5 class Course {
6 private:
7   Person the_instructor;
8   int year;
9 };`
```

This is called the has-a relation:

Course has-a *Person*

2. Literal and figurative has-a

A line segment has a starting point and an end point. *LineSegment* code design:

Store both points:

```
1 class Segment {
2 private:
3   Point p_start,p_end;
4 public:
5   Point end_point() {
6     return p_end; };
7 }
8 int main() {
9   Segment seg;
10  Point somepoint =
11    seg.end_point();
```

or store one and derive the other:

```
1 class Segment {
2 private:
3   Point starting_point;
4   float length,angle;
5 public:
6   Point end_point() {
7     /* some computation
8        from the
9        starting point */ };
10 }
```

Implementation vs API: implementation can be very different from user interface.

3. Constructors in has-a case

Class for a person:

```
1 class Person {
2 private:
3   string name;
4 public:
5   Person( string name ) {
6     /* ... */
7   };
8 };
```

Class for a course, which contains a person:

```
1 class Course {
2 private:
3   Person instructor;
4   int enrollment;
5 public:
6   Course( string instr,int n ) {
7     /* ???? */
8   };
9 };
```

Declare a `Course` variable as: `Course("Eijkhout",65);`

4. Constructors in the has-a case

Possible constructor:

```
1 Course( string teachername,int nstudents ) {
2   instructor = Person(teachername);
3   enrollment = nstudents;
4 };
```

Preferred:

```
1 Course( string teachername,int nstudents )
2   : instructor(Person(teachername)),
3     enrollment(nstudents) {
4 };
```

5. Rectangle class

To implement a rectangle with sides parallel to the x/y axes, two designs are possible. For the function:

```
1 float Rectangle::area();
```

it is most convenient to store width and height.
For inclusion testing:

```
1 bool Rectangle::contains(Point);
```

it would be convenient to store bottomleft/topright points.

For now, use the *first* option.

Exercise 1

Make a class *Rectangle* (sides parallel to axes) with a constructor:

```
1 Rectangle(Point botleft,float width,float height);
```

Can you figure out how to use member initializer lists for the constructors?

Implement methods:

```
1 float area(); float rightedge_x(); float topedge_y();
```

and write a main program to test these.

Inheritance

6. Hierarchical object relations

- Hierarchical relations between classes:
  each object in class A is also in class B.

7. Example of class hierarchy

- Class *Employee*:

```
1 class Employee {
2 private:
3   int number,salary;
4 /* ... */
5 };
```

- class *Manager* is subclass of *Employee*
  (every manager is an employee, with number and salary)
- Manager has extra field *n_minions*

How do we implement this?

8. Another example: multiple subclasses

- Example: both triangle and square are polygons.
- You can implement a method `draw` for both triangle/square
- ... or write it once for polygon, and then use that.

9. Terminology

- *`Polygon`* / *`Employee`* is the *base class*.
- *`Triangle`* / *`Manager`* is a *derived class*.
- Derived classes *inherit* data and methods from the base class: they are accessible in objects of the derived class.

10. Examples for base and derived cases

General *FunctionInterpolator* class with method *value_at*. Derived classes:

- *LagranceInterpolator* with *add_point_and_value*;
- *HermiteInterpolator* with *add_point_and_derivative*;
- *SplineInterpolator* with *set_degree*.

11. General case, special case

You can have classes where an object of one class is a special case of the other class. You declare that as

```
1 class General {
2 protected: // note!
3  int g;
4 public:
5  void general_method() {};
6 };
7
8 class Special : public General {
9 public:
10   void special_method() { g = ... };
11 };
```

12. Inheritance: derived classes

Derived class `Special` *inherits* methods and data from base class `General`:

```
1 int main() {
2   Special special_object;
3   special_object.general_method();
4   special_object.special_method();
5 }
```

Members of the base class need to be `protected`, not `private`, to be inheritable.

13. Constructors

When you run the special case constructor, usually the general constructor needs to run too. Here we invoke it explicitly:

```
1 class General {
2 public:
3   General( double x,double y ) {};
4 };
5 class Special : public General {
6 public:
7   Special( double x ) : General(x,x+1) {};
8 };
```

14. Access levels

Methods and data can be

- `private`, because they are only used internally;
- `public`, because they should be usable from outside a class object, for instance in the main program;
- `protected`, because they should be usable in derived classes.

Exercise 2

Take your code where a *Rectangle* was defined from one point, width, and height.

Make a class *Square* that inherits from *Rectangle*. It should have the function *area* defined, inherited from *Rectangle*.

First ask yourself: what should the constructor of a *Square* look like?

Exercise 3

Revisit the *LinearFunction* class. Add methods *slope* and *intercept*.

Now generalize *LinearFunction* to *StraightLine* class. These two are almost the same except for vertical lines. The *slope* and *intercept* do not apply to vertical lines, so design *StraightLine* so that it stores the defining points internally. Let *LinearFunction* inherit.

15. Overriding methods

- A derived class can inherit a method from the base class.
- A derived class can define a method that the base class does not have.
- A derived class can *override* a base class method:

```
1 class Base {
2 public:
3   virtual f() { ... };
4 };
5 class Deriv : public Base {
6 public:
7   virtual f() override { ... };
8 };
```

16. More

- Multiple inheritance: an X is-a A, but also is-a B.
  This mechanism is somewhat dangerous.
- Virtual base class: you don't actually define a function in the base class, you only say 'any derived class has to define this function'.