

Class relations: has-a

Victor Eijkhout, Susan Lindsey

Fall 2023

last formatted: September 12, 2023

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:

```
class Person {  
private:  
    string name;  
public:  
    Person( string name ) {  
        /* ... */  
    };  
};
```

Class for a course, which contains a person:

```
class Course {  
private:  
    Person instructor;  
    int enrollment;  
public:  
    Course( string instr,int n )  
    {  
        /* ??? */  
    };  
};
```

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

4. Constructors in the has-a case

Possible constructor:

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

Preferred:

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

5. Axi-parallel rectangle class

Intended API:

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

It would be convenient to store width and height; for

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

it would be convenient to store bottomleft/topright points.

Exercise 1

1. Make a class `Rectangle` (sides parallel to axes) with a constructor:

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

The logical implementation is to store these quantities. Implement methods:

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

and write a main program to test these.

2. Add a second constructor

```
Rectangle(Point botleft, Point topright);
```

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

Optional exercise 2

Make a copy of your solution of the previous exercise, and redesign your class so that it stores two `Point` objects. Your main program should not change.