Class relations: has-a

Victor Eijkhout, Susan Lindsey

Fall 2019

Has-a relationship

A class usually contains data members. These can be simple types or other classes. This allows you to make structured code.

```
class Course {
private:
  Person the_instructor;
  int year;
}
class Person {
  string name;
  ....
}
```

This is called the has-a relation.

Literal and figurative has-a

A line segment has a starting point and an end point.

A Segment class can store those points:

```
class Segment {
private:
  Point starting_point,
    ending_point;
public:
  Point get_the_end_point() {
    return ending_point; };
}
  ...
  Segment somesegment;
  Point somepoint =
    somesegment.
    get_the_end_point();
```

or store one and derive the other:

```
class Segment {
private:
  Point starting_point;
  float length,angle;
public:
  Point get_the_end_point() {
    /* some computation
       from the
       starting point */ };
}
```

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

Multiple constructors

```
class Segment {
  private:
  // up to you how to implement!
  public:
  Segment( Point start,float length,float angle )
  { .... }
  Segment( Point start,Point end ) { ... }
```

Depending on how you actually implement the class, the one constructor will simply store the defining data, and the other will do some conversion from the given data to the actually stored data.

Member initialization and default constructors

```
class Inner { /* ... */ };
class Outer {
private:
  Inner inside_thing;
```

Two possibilities for constructor:

```
Outer( Inner put_it )
: inside_thing(put_it) {};
```

The `Inner` object is copied during construction of `Outer` object.

```
Outer( Inner put_it ) {
  inside_thing = put_it;
};
```

The `Outer` object is created, including construction of `Inner` object, then the argument is copied into place: ⇒ needs default constructor on `Inner`.

Exercise 1

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

  ```
  Rectangle(Point bl,float w,float h);
  ```

  The logical implementation is to store these quantities. Implement methods

  ```
  float area(); float rightedge(); float topedge();
  ```

- Add a second constructor

  ```
  Rectangle(Point bl,Point tr);
  ```

  Can you figure out how to use member initializer lists for the constructors?
- Write another version of your class so that it stores two `Point` objects.

Polymorphism in constructors

You have to decide what to store and what to derive, but you can construct two ways:

```
class Segment {
private:
 // up to you how to implement!
public:
  Segment( Point start,float length,float angle )
    { .... }
  Segment( Point start,Point end ) { ... }
```

Advantage: with a good API you can change your mind about the implementation without changing the calling code.