Advanced Programming Concepts with C++ CSI2372 – Fall 2019

Jochen Lang &
Mohamed Taleb
EECS

This Lectures

OO

- **Object-oriented design**
  - Use of const and references with objects, Ch. 2.3.1, 2.4.1, 7.3.2
  - Brace initialization revisited, Ch. 3.3.1, 7.5.5
  - Class relationships: association, Ch. 7.1.4, 7.2

Pass by Reference – Objects

- **Reconsider our Point2D class with pass by reference**
  - Avoids copy of object (no copy constructor is called)
  - Enables a method or function to modify the variable for the calling context

```
Point2D a( 0.0, 1.0 ), b( -1.0, 2.0 );
Point2D c = a.subtract( b );
…

Point2D Point2D::subtract( Point2D& _oPoint ) {
    Point2D res;
    res.d_x = d_x - _oPoint.d_x;
    res.d_y = d_y - _oPoint.d_y;
    return res;
}
```

Pass Multiple Results by Reference Example: `Point2D.isSmaller`

- **Return two boolean for x comparison and y-comparison**

```
class PointD {
public:
    void isSmaller( Point2D& _oPoint, bool& xCompare,
                    bool& yCompare );
};
```

```
void Point2D::isSmaller( Point2D& _oPoint, bool& xCompare,
                         bool& yCompare ) {
  if ( d_x < _oPoint.d_x ) {
    xCompare = true;
  } else {
    xCompare = false;
  } …
```

Invalid Return of a Reference

- **We could avoid another copy by**

```
…
Point2D a( 0.0, 1.0 ), b( -1.0, 2.0 );
Point2D c = a.subtract( b );
…

Point2D& Point2D::subtract( Point2D& _oPoint ) {
    Point2D res;
    res.d_x = d_x - _oPoint.d_x;
    res.d_y = d_y - _oPoint.d_y;
    return res;
}
```

- **Why is this bad?**

Valid Return of a Reference

- **We can use the following:**

```
Point2D a( 0.0, 1.0 ), b( -1.0, 2.0 );
Point2D c = a.minusEquals( b );
…

Point2D& Point2D::minusEquals( Point2D& _oPoint ) {
    d_x -= _oPoint.d_x;
    d_y -= _oPoint.d_y;
    return (*this);
}
```

- **Why is this ok?**

Make Use of const modifier

- **Constant variables and references are good!**
  - Clarifies what a function/method does
  - Avoids accidental modifications
  - Potentially increases execution speed since the compiler can optimize more aggressively
- **What to declare constant?**
  - Methods which do not change attributes of an object
  - Arguments which will not be changed in a method
  - References which won't have the aliased variable changed
  - Constants (incl. object where attributes won't change after initialization).

Example: Point2D

- **Make arguments, methods constant**
- **Before:**

```
class Point2D {
    double d_x, d_y;
public:
    Point2D( double _x = 0.0, double _y = 0.0 );
    Point2D add( Point2D& _oPoint );
    Point2D subtract( Point2D& _oPoint );
    Point2D& minusEquals( Point2D& _oPoint );
    double dot( Point2D& _oPoint );
};
```

- **After?**

Example: Point2D

- **Make arguments, methods constant**

```
class Point2D {
    double d_x, d_y;
public:
    Point2D( double _x = 0.0, double _y = 0.0 );
    Point2D add( const Point2D& _oPoint ) const;
    Point2D subtract( const Point2D& _oPoint ) const;
    Point2D& minusEquals( const Point2D& _oPoint );
    double dot( const Point2D& _oPoint ) const;
};
```

Review: List Initializers in C++11

- Can use initializer lists even with non-arrays or non struct
  - Restricts automatic conversions (good), i.e., narrowing not allowed

```
int iA = 1048576, iB(1048576);
int iC{1048576}, iD = {1048576};
short sA = iA, sB(iA);
short sC{iA}, sD = {iA};
```

"Same" Initialization

Illegal: narrowing from int to short

In Class Initializers in C++11

- **In class initializers can be used similar to Java**
  - Some objects or built-in types may have the same initialization for all or most constructors but default initialization is not desired.
    - Avoids code duplication

```
class Circle2D {
  Point2D d_center{ Point2D( -1e39, -1e39 ) }; // C++11 only!
  double d_radius{-1.0}; // C++11 only!
public:
  Circle2D() = default;
  Circle2D( Point2D _center, double _radius ) :
       d_center{Point2D(_center)}, d_radius{_radius}  {}
};
```

Aggregate Classes

- **A `struct` in C++ is the same as a `class` except the default access is public**
- **`struct` are typically used for aggregation**
- **A class is an aggregate if:**
  - All data members are public.
  - No constructors are defined.
  - No in-class initializers
  - No base classes or virtual functions.
- **Such classes or structs are also called POD (Plain Old Data).**
- **We can use brace initialization for these.**

Brace Initialization with Classes

- **Aggregate Classes**

```
struct SizedPoint2D {
  Point2D startP;
  double size, endP;
}; …
SizedPoint2D sP{Point2D(0.5,3.0),2.0};
```

- **Non-aggregate classes – just used for uniform syntax. It will call the corresponding constructor.**

```
Point2D p{0.5,3.0};
```

- **Standard library container types, e.g., std::array, std::string or std::vector make use of a special template and a "sequence constructor" to allow brace initialization.**

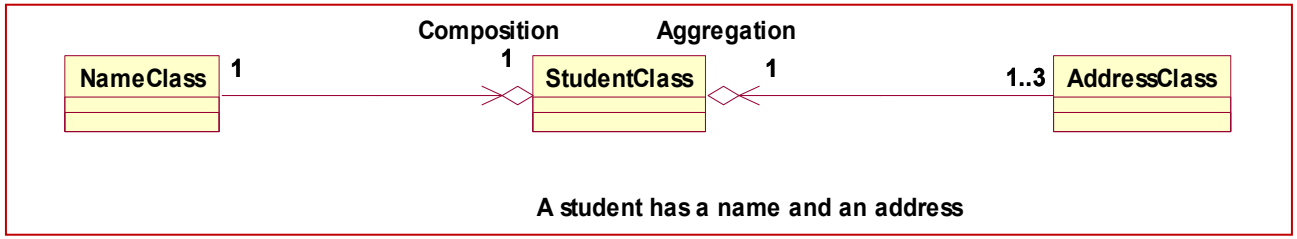
```
std::vector<int> vi{1,2,3};
```

Class Relationships – Overview

- **Association**
  - The interaction and communication among classes
- **Aggregation (or Composition)**
  - The "has a" relationship
  - Containment of objects of other class types

The **composition** is a relationship between two objects. An object can contain another object.

A **composition** is actually a special case of the aggregation relationship.



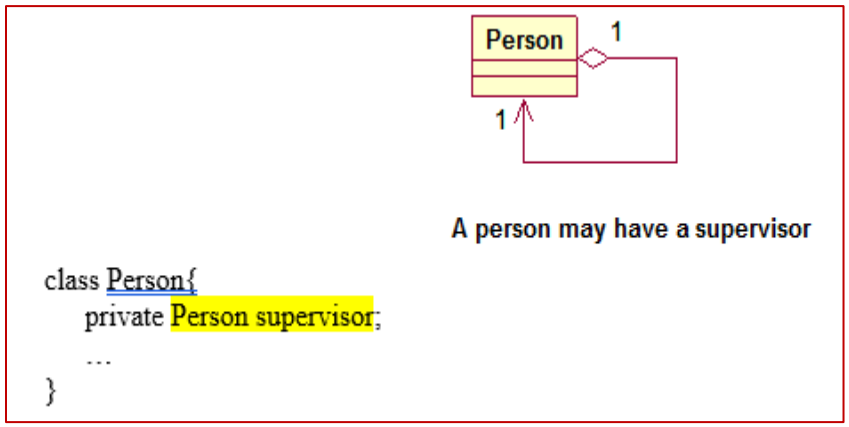
Aggregation models "**has-a**" relationships and represents an ownership relationship between two objects.

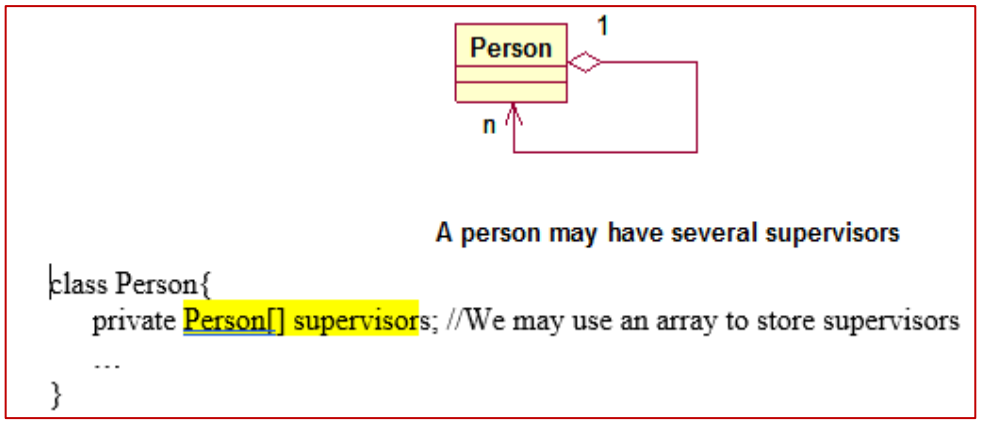
```
class NameClass{
    ....
}
```

```
class StudentClass{
    private NameClass name;
    private AddressClass address;
    ...
}
```

```
class AddressClass{
    ...
}
```

Class Relationships – Overview

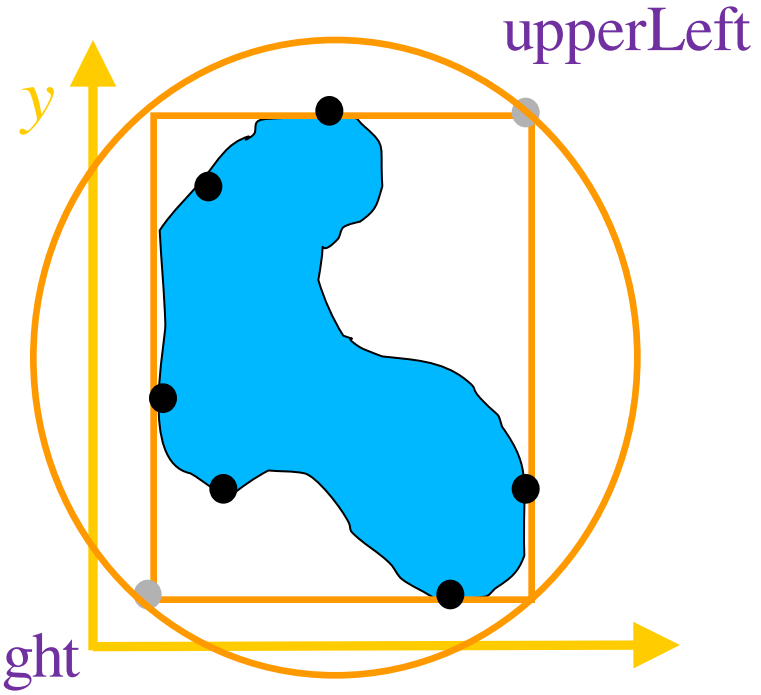




- **Generalization and Inheritance**
  - The “is a” relationship
  - Inheritance from a general class to a more specific one

Example Problem Description

- **Find bounding primitive that encloses the blue shape**
  - Smallest AABox
  - Circle*
  - Input:
    Array of boundary points



*Note: Smallest circle can be found in O(n) where n are the number of boundary points

Class Example: Point2D and Axis-Aligned Bounding Box (AABox)

- Define a `AABox` based on two `Point2D`

```
class Point2D {
    double d_x;
    double d_y;
public:
    Point2D( double _x, double _y );
};

class AABox {
    Point2D d_lowerLeft;
    Point2D d_upperRight;
public:
    AABox( const Point2D& _lowerLeft,
           const Point2D& _upperRight );
};
```

Class Relationships – Association

- **Association**
  - The interaction and communication among classes
  - Example: The class AABox communicates with the class Point2D via public methods

```
bool AABox::enclose(std::array<Point2D,4> extrema ) {
  …
  for (int i=0; i<extrema.size(); ++i) {
    lowerLeft.d_x = std::min(extrema[i].d_x, lowerLeft.d_x);
```

```
class Point2D { …
public:
  Point2D min( const Point2D& _oPoint ) const;
  Point2D max( const Point2D& _oPoint ) const;
  bool isSmaller( const Point2D& _oPoint ) const;
};
```

Next

OO

- **Object-oriented design**
  - Class relationships: aggregation, generalization and inheritance
  - Pointer attributes and this pointer
  - Copy construction and assignment