Creative Coding II

Object oriented design

FILMUNIVERSITÄT
BABELSBERG
KONRAD WOLF

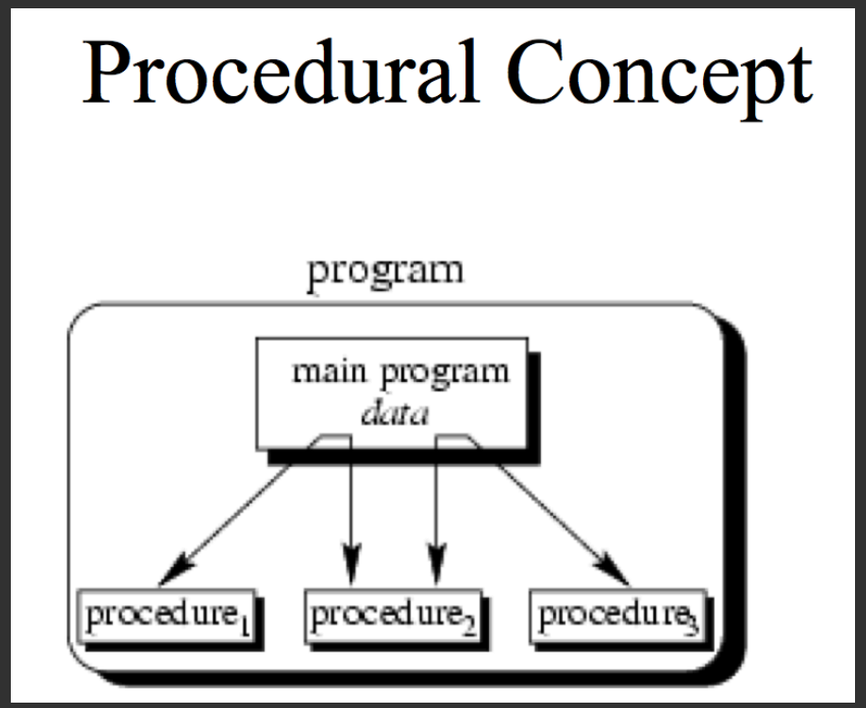
Prof. Dr.-Ing. Angela Brennecke
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

Content

- Object-oriented design
- Aspects of abstraction
- Inheritance

Object-Oriented Class Design

- Object-oriented design is a programming paradigm that is based on the idea of creating objects and defining their relationships
- Objects are user-defined data structures or types



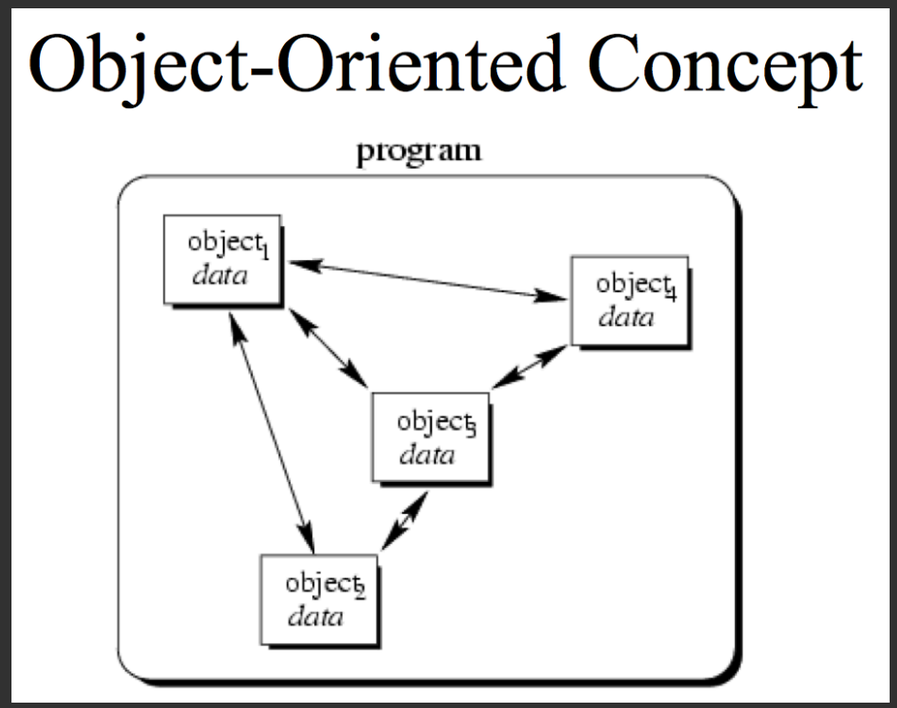


Image credit:
Neelam Sounderajan,
Ohio State University.

Object-Oriented Class Design

- All data types usually are representations of a certain concept
  - The C++ built-in type “float” together with its operations represents the mathematical concept of a real number
  - The fully user-defined (Class) type “ofApp” together with its operations represents the software concept of an openFrameworks application
  - …

Classes & Objects

- To create an object, you have to define its functionality in a corresponding class

- Classes are the blueprint for the actual objects

- Based on the class definition various different object instances can be created
  - A **class** corresponds to the **concept**
  - An **object** corresponds to the actual **instance**

Classes & Objects

- Classes are user-defined data structures and consist of
  - a class name
  - constructor & destructor
  - member functions
  - member variables
  - specification of access levels (private, public,…)
  - specification of relationships to other classes (friends, parents, …)

Classes

- Typical class interface of an openFrameworks application class

- The class interface is usually defined in the header file *.h …

```
#pragma once
#include "ofMain.h"

// ofApp derives from ofBaseApp
class ofApp : public ofBaseApp
{
    // private section – everything here is private and only
    // accessible from within the class ofApp

public:
    // everything after keyword public can be accessed from
    // outside of the class, i.e., the public area represents
    // the interface of the class
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);

private:
    // everything after keyword private can not be accessed from
    // outside of the class but only from this class alone
    // the private area represents the "hidden" implementation
    // details of the class
    std::string appFirstWords;
    bool startDrawing;
};
```

Classes

- … whereas the actual implementation is defined in the definition file *.cpp

```
#include "ofApp.h"

//--------------------------------------------------------------
void ofApp::setup()
{
    ofBackground(0);
    ofSetBackgroundAuto(false);
}

//--------------------------------------------------------------
void ofApp::update()
{

}

//--------------------------------------------------------------
void ofApp::draw()
{

}

//--------------------------------------------------------------
void ofApp::keyPressed(int key)
{

}

//--------------------------------------------------------------
void ofApp::keyReleased(int key){

```

Class Members

- Declaring a class Box with member functions and member variables in a class header *.h file

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

class Box
{
public:
    // Member functions declaration
    double getVolume(void);
    void setLength( double len );
    void setBreadth( double bre );
    void setHeight( double hei );

private:
    double length;          // Length of a box
    double breadth;         // Breadth of a box
    double height;          // Height of a box
};
```

Class Members

- Adding the function definitions in an according *.cpp file

```
23 // Member functions definitions
24 double Box::getVolume(void)
25 {
26     return length * breadth * height;
27 }
28
29 void Box::setLength( double len )
30 {
31     length = len;
32 }
33
34 void Box::setBreadth( double bre )
35 {
36     breadth = bre;
37 }
38
39 void Box::setHeight( double hei )
40 {
41     height = hei;
42 }
```

Object Instances & Member Access

- Accessing & manipulating the box objects

```
44 // Main function for the program
45 int main()
46 {
47     Box Box1;                // Declare Box1 of type Box
48     Box Box2;                // Declare Box2 of type Box
49     double volume = 0.0;     // Store the volume of a box here
50
51     // box 1 specification
52     Box1.setLength(6.0);
53     Box1.setBreadth(7.0);
54     Box1.setHeight(5.0);
55
56     // box 2 specification
57     Box2.setLength(12.0);
58     Box2.setBreadth(13.0);
59     Box2.setHeight(10.0);
60
61     // volume of box 1
62     volume = Box1.getVolume();
63     cout << "Volume of Box1 : " << volume <<endl;
64
65     // volume of box 2
66     volume = Box2.getVolume();
67     cout << "Volume of Box2 : " << volume <<endl;
68     return 0;
69 }
```

Constructor & Destructor

- Every class has two special functions called
  - Constructor — possibly many different types
  - Destructor — one, and one only

- The constructor is required to “construct” the object and to initialize all of the data members of the class

- The destructor is required to properly destroy the object when its lifetime ends

Constructor & Destructor

```
 7 class Box
 8 {
 9 public:
10
11     Box();       // default constructor
12     ~Box();      // default destructor
13
14     // Member functions declaration
15     // ...
16
17 private:
18     double length;
19     double breadth;
20     double height;
21 };
```

```
24 Box::Box()
25 {
26     // inititializes the
27     // data members with
28     // default values, i.e., 0.0
29 }
```

```
44 // Main function for the program
45 int main()
46 {
47     Box Box1;                // Declare Box1 of type Box
48     Box Box2;                // Declare Box2 of type Box
49     double volume = 0.0;     // Store the volume of a box here
50
51     // box 1 specification
```

Default Constructor

```
 7 class Box
 8 {
 9 public:
10
11     Box();       // default constructor
12     ~Box();      // default destructor
13
14     // Member functions declaration
15     // ...
16
17 private:
18     double length;
19     double breadth;
20     double height;
21 };
```

```
32 Box::Box()
33 :
34 length{10},
35 breadth{12},
36 height{50}
37 {
38     // initializes the
39     // data members with
40     // specific values
41 }
```

```
44 // Main function for the program
45 int main()
46 {
47     Box Box1;                // Declare Box1 of type Box
48     Box Box2;                // Declare Box2 of type Box
49     double volume = 0.0;     // Store the volume of a box here
50
51     // box 1 specification
```

Default Constructor

- Every class must have a default constructor that is called when an object of that class is being declared

- If the class does not have an explicit default constructor, the compiler generates it automatically

- If not specified otherwise, the default constructor initializes all data members to their default values

User-Defined Constructors

```
 7  class Box
 8  {
 9  public:
10
11      Box();       // default constructor
12      Box(double theLength);
13      Box(double theBreadth);     // error
14      Box(double theLength, double theBreadth);
15      Box(double length, double breadth, double height);
16      ~Box();     // default destructor
17
18      // Member functions declaration
19      // ...
20
21  private:
22      double length;          // Length of a box
23      double breadth;         // Breadth of a box
24      double height;          // Height of a box
25  };
```

Constructor cannot be redeclared

User-Defined Constructors

```
47 Box::Box(double theLength)
48 :
49 length{theLength},
50 breadth{12},
51 height{50}
52 {
53     // initializes the
54     // data members with
55     // specif
56 }
```

```
58 Box::Box(double theLength, double theBreadth)
59 :
60 length{theLength},
61 breadth{theBreadth},
62 height{50}
63 {
64     // initializes the
65     // data members with
66     // specific values
67 }
```

User-Defined Constructors

- User-defined constructors can be used to directly initialize member variables to other than default values
- User-defined constructors are not limited in number but have to be different from each other

Destructor

- Like with the constructor, every class must have a default destructor that is called when an object of that class goes out of scope — meaning, the object lifetime ends

- If the class does not have an explicit default destructor, the compiler generates one automatically

- Destructors are particularly important when a class member variable allocates dynamic memory

Classes and Objects Revisited

```
 7 class Box
 8 {
 9 public:
10
11     Box();       // default construct
12     ~Box();      // default destructo
13
14     // Member functions declaration
15     // ...
16
17 private:
18     double length;          // Length
19     double breadth;         // Breadt
20     double height;          // Height
21 };
```

```
44 // Main function for the program
45 int main()
46 {
47     Box Box1;                // Declare Box1 of type Box
48     Box Box2;                // Declare Box2 of type Box
49     double volume = 0.0;     // Store the volume of a box here
50
51     // box 1 specification
52     Box1.setLength(6.0);
53     Box1.setBreadth(7.0);
54     Box1.setHeight(5.0);
55
56     // box 2 specification
57     Box2.setLength(12.0);
58     Box2.setBreadth(13.0);
59     Box2.setHeight(10.0);
60
61     // volume of box 1
62     volume = Box1.getVolume();
63     cout << "Volume of Box1 : " << volume <<endl;
64
65     // volume of box 2
66     volume = Box2.getVolume();
67     cout << "Volume of Box2 : " << volume <<endl;
68     return 0;
69 }
```

Abstraction

- One key aspect of object oriented design is abstraction
- Abstraction means to define the interface and to separate it from the implementation details
  - Show the interface — the “*what is there*”
  - Hide the implementation details — the “*how is it done*”

Access Levels

- Classes support different levels of access to the class' member functions and variables that are specified by
  - The public keyword
  - The protected keyword
  - The private keyword

```
 7 class Box
 8 {
 9 public:
10
11     Box();       // default constructor
12     ~Box();      // default destructor
13
14     // Member functions declaration
15     // ...
16
17 private:
18     double length;          // Length of a
19     double breadth;         // Breadth of a
20     double height;          // Height of a
21 };
```

Access Levels: Public

- Every member variable and/or function defined under public can be accessed from outside of the class

- The public keyword specifies the interface level

Access Levels: Protected

- Every member variable and/or function defined under protected can be accessed from inside of the class and from inside of the inherited class(es) only — not from outside of the class

- The protected keyword specifies the inheritance and hierarchical level

Access Levels: Private

- Every member definition in a class is declared as private by default — if not specified otherwise

- Every member variable and/or function defined under private can only be accessed from inside of the class — not from outside of the class

- The private keyword specifies the implementation details level

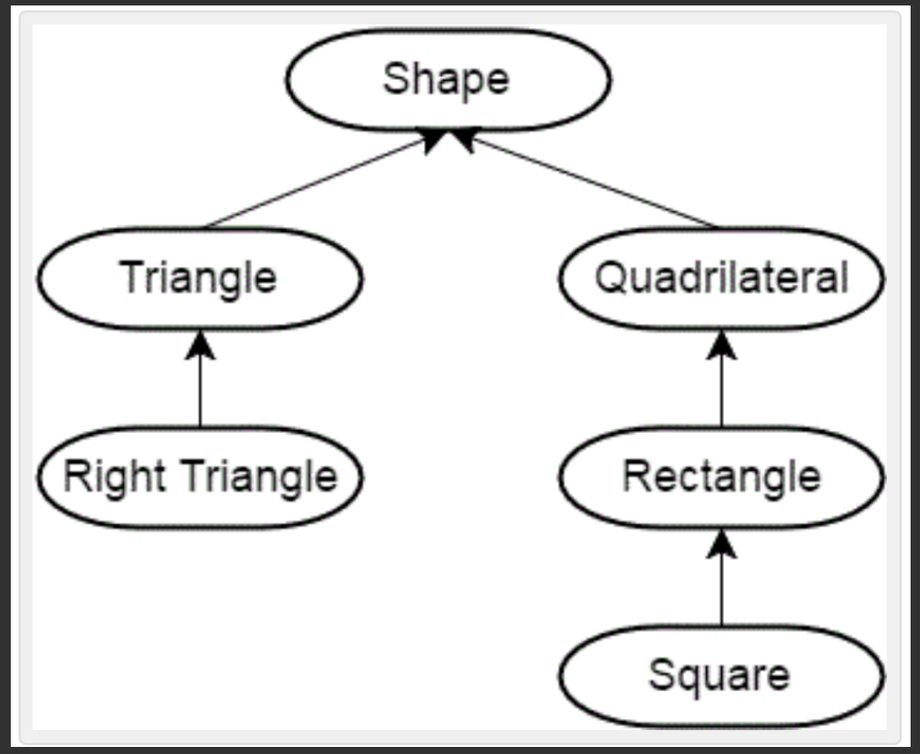
Access Levels: Design Aspects

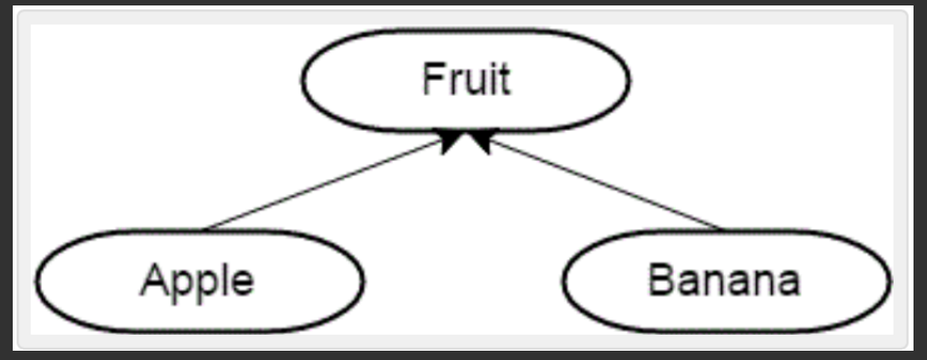
- A rule of thumb for good class design is to ensure
  - Private data members
  - Public member functions to access the underlying data

- This way, a solid public interface can be designed
  - It remains “as is” even when data members change
  - The manipulation of the member variables is defined inside the class only

Inheritance

- Inheritance typically models hierarchical relationships between objects and to introduce concept levels

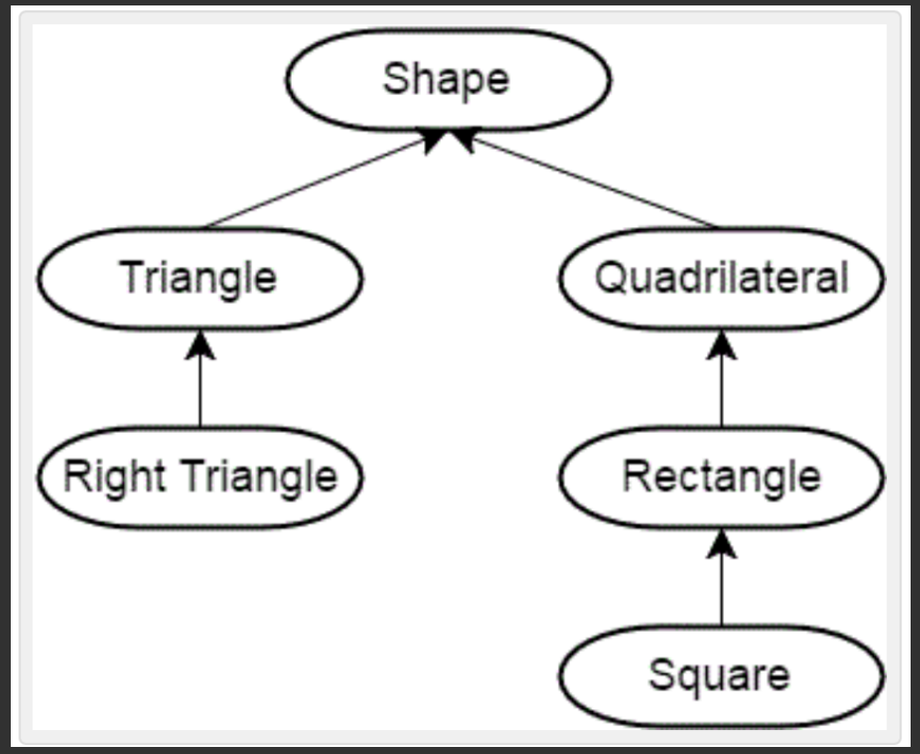


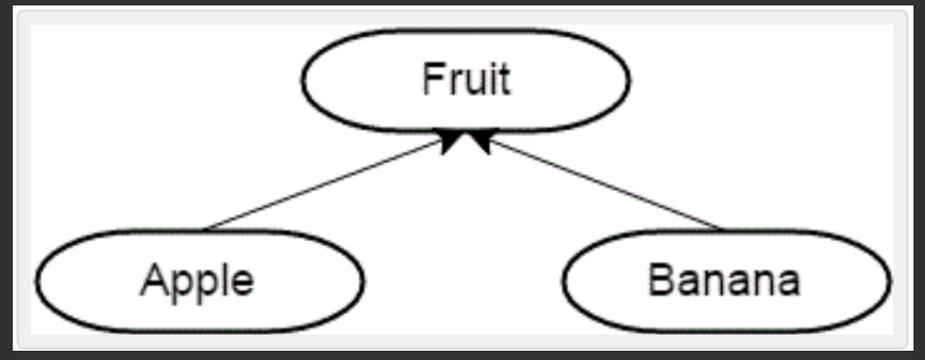


Images credit: http://www.learncpp.com/cpp-tutorial/111-introduction-to-inheritance/

Inheritance

- Hierarchies mostly show a progression over time
  or from general to specific attributes & functionality

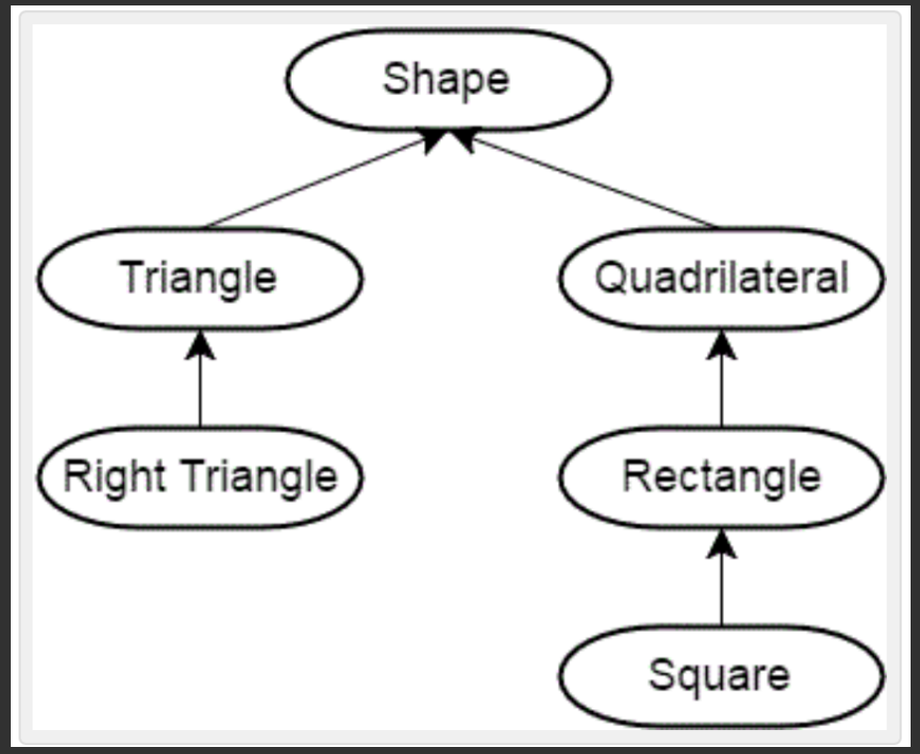


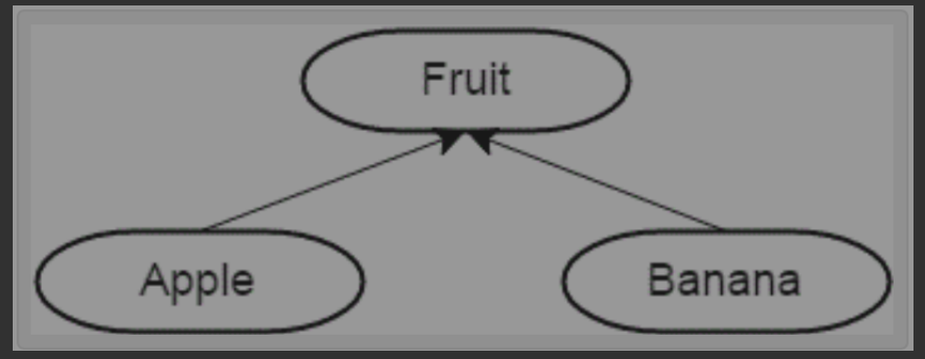


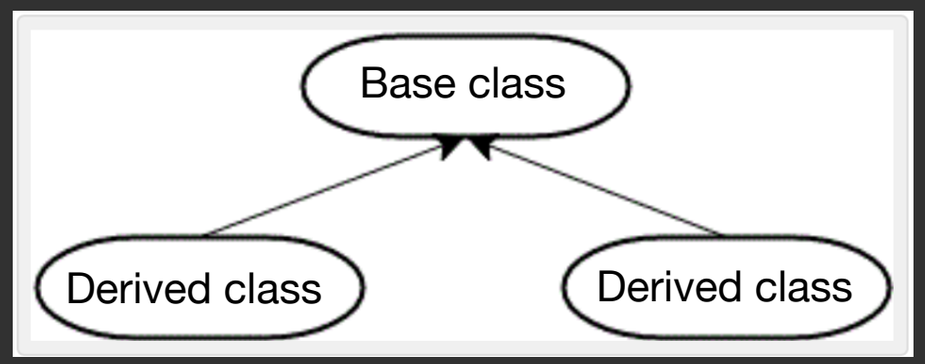
Images credit: http://www.learncpp.com/cpp-tutorial/111-introduction-to-inheritance/

Inheritance

- Usually in C++ we speak of a base class & a derived class



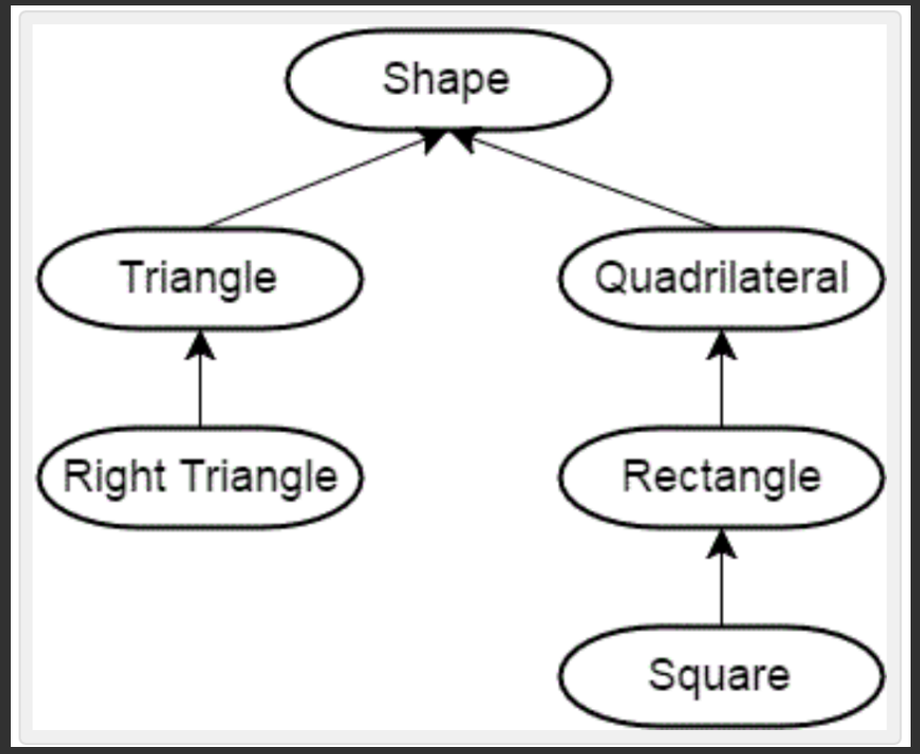


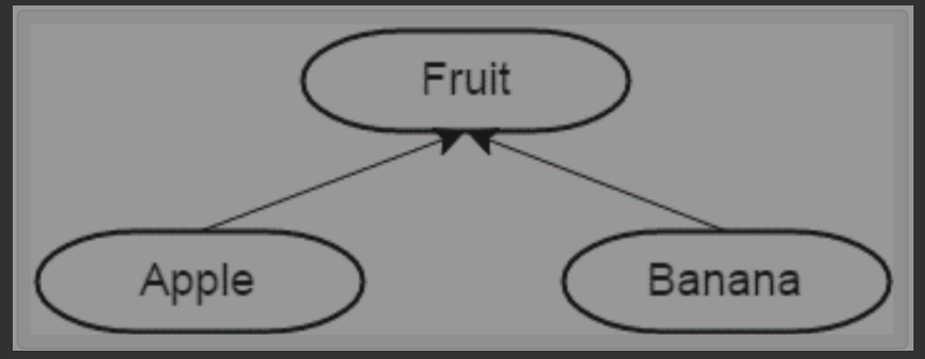


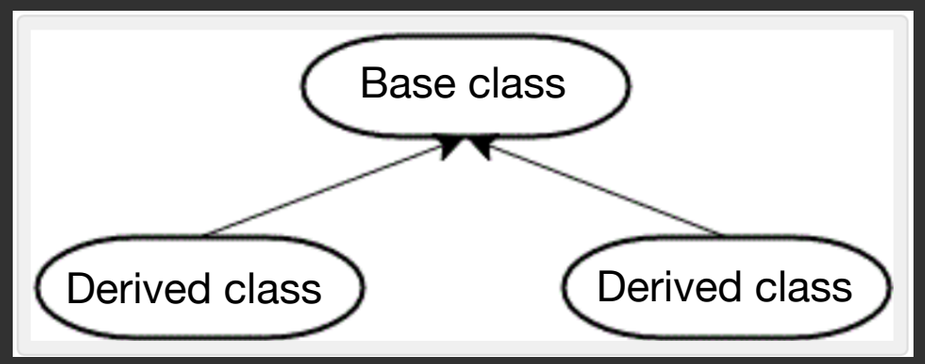
Images credit: http://www.learncpp.com/cpp-tutorial/111-introduction-to-inheritance/

Inheritance

- The derived class inherits all attributes & functionality of its base class — without access to private base members







Images credit: http://www.learncpp.com/cpp-tutorial/111-introduction-to-inheritance/

Code Example

```
11 #include <iostream>
12
13 class fruit
14 {
15 public:
16     // constructors
17     fruit();
18     fruit(std::string theName);
19
20     void printName();
21
22 private:
23     std::string myName;
24 };
```

```
11 #include "fruit.h"
12
13 class apple : public fruit
14 {
15
16 public:
17     // d
18     // i
19     enum
20     {
21
22
```

```
11 #include "fruit.h"
12
13
14 class banana : public fruit
15 {
16 public:
17
18     banana(); // default constructor
19
```

```
9  #include "fruit.h"
10 #include "apple.h"
11 #include "banana.h"
12
13
14 int main()
15 {
16     fruit aFruit;
17     fruit anotherFruit("anotherFruit");
18     apple anApple{apple::appleType::BRAEBURN};
19     banana aBanana;
20
21     aFruit.printName();
22     anotherFruit.printName();
23
24     anApple.printName();    // the derived classes inherit the
25     aBanana.printName();    // base class function printName()
26
27     return 0;
28 }
```

Object Oriented Design Revisited

- A class („box“) provides an object specification
- Many objects can be of type class („box“)

- Object construction can be customized

- Access levels are a design tool that helps define how the object can be used

Object Oriented Design Revisited

- A class interface is usually based on
  - a set of public access & manipulation functions independent of the underlying data types & structures
  - a set of private data members & functions that define and manage the underlying data types & structures
- Inheritance supports object hierarchies

Now, code

Prof. Dr.-Ing. Angela Brennecke
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF