

Namespaces and Class Hierarchies

Namespaces

representing a point
defining a
namespace

Class Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple Inheritance

inheriting from two
classes
ambiguities and
refactoring

1 **Namespaces**
representing a point
defining a namespace

2 **Class Hierarchies**
a hierarchy of shapes
polymorphism: changing shape

3 **Multiple Inheritance**
inheriting from two classes
ambiguities and refactoring

MCS 360 Lecture 9
Introduction to Data Structures
Jan Vershelde, 13 September 2010

Namespaces and Class Hierarchies

Namespaces

representing a point

defining a namespace

Class

Hierarchies

a hierarchy of shapes

polymorphism:
changing shape

Multiple

Inheritance

inheriting from two classes

ambiguities and refactoring

- 1 Namespaces
representing a point
defining a namespace

- 2 Class Hierarchies
a hierarchy of shapes
polymorphism: changing shape

- 3 Multiple Inheritance
inheriting from two classes
ambiguities and refactoring

representing a point

Namespaces

representing a point

defining a namespace

Class

Hierarchies

a hierarchy of shapes

polymorphism:
changing shape

Multiple

Inheritance

inheriting from two classes

ambiguities and refactoring

Suppose we need to store a point:

- 1 data: integer coordinates;
- 2 functions: get values for the coordinates and a `to_string()` method.

We start with a point in the plane and define a class `Point`.

the file `point1.h`

Namespaces

representing a point

defining a
namespace

Class

Hierarchies

a hierarchy of shapes

polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classesambiguities and
refactoring

```
#ifndef POINT1_H
#define POINT1_H

#include <string>

class Point
{
public:

    Point(int a, int b);
    int get_x() const;
    int get_y() const;
    std::string to_string() const;

private:

    int x; /* x-coordinate of the point */
    int y; /* y-coordinate of the point */
};
```

the file `point1.cpp`

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

```
#include "point1.h"
#include <sstream>

Point::Point(int a, int b)
{
    x = a; y = b;
}

int Point::get_x() const        // Point::get_y()
{                               // is similar
    return x;
}

std::string Point::to_string() const
{
    std::ostringstream s;
    s << '(' << x << ',' << y << ')';
    return s.str();
}
```

testing the class

Namespaces

representing a point

defining a
namespace

Class

Hierarchies

a hierarchy of shapes

polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classesambiguities and
refactoring

```
#include "point1.h"
#include <iostream>

using namespace std;

int main()
{
    Point p(2,3);

    cout << "A point with coordinates "
         << p.to_string() << "." << endl;

    return 0;
}
```

Namespaces and Class Hierarchies

Namespaces

representing a point

defining a namespace

Class

Hierarchies

a hierarchy of shapes

polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes

ambiguities and
refactoring

1 Namespaces

representing a point

defining a namespace

2 Class Hierarchies

a hierarchy of shapes

polymorphism: changing shape

3 Multiple Inheritance

inheriting from two classes

ambiguities and refactoring

using a namespace

Namespaces

representing a point
defining a
namespace

Class
Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple
Inheritance

inheriting from two
classes
ambiguities and
refactoring

Suppose we want also points in 3-space,

- okay to have `point2.h` and `point3.h`,
- but like to call a type `Point`.

We use namespaces `D2` and `D3`.

```
#include "point2.h"
#include "point3.h"

int main()
{
    D2::Point p(2,3);
    D3::Point q(2,3,1);
}
```

If only `D3` is used, do `using namespace D3`.

defining a namespace D2

in the file `point2.h`

Namespaces

representing a point

defining a namespace

Class

Hierarchies

a hierarchy of shapes

polymorphism:
changing shape

Multiple

Inheritance

inheriting from two classes

ambiguities and refactoring

```
namespace D2 {  
  
    class Point  
    {  
        public:  
            Point(int a, int b);  
            int get_x() const;  
            int get_y() const;  
            std::string to_string() const;  
  
        private:  
            int x; /* x-coordinate of the point */  
            int y; /* y-coordinate of the point */  
  
    }; // end of class definition  
}; // end of namespace
```

defining a namespace D3

in the file `point3.h`

Namespaces

representing a point

defining a namespace

Class

Hierarchies

a hierarchy of shapes

polymorphism:
changing shape

Multiple

Inheritance

inheriting from two classes

ambiguities and refactoring

```
namespace D3 {  
  
    class Point  
    {  
        public:  
            Point(int a, int b, int c);  
            int get_x() const;  
            int get_y() const;  
            int get_z() const;  
            std::string to_string() const;  
  
        private:  
            int x; /* x-coordinate of the point */  
            int y; /* y-coordinate of the point */  
            int z; /* z-coordinate of the point */  
    };  
};
```

using declaration and directive

Namespaces

representing a point

defining a namespace

Class

Hierarchies

a hierarchy of shapes

polymorphism:
changing shape

Multiple

Inheritance

inheriting from two classes

ambiguities and refactoring

In `point2.cpp` we put the using *declaration*:

```
using namespace D2;
```

and `point3.cpp` starts with using namespace D3.

To use console output `cout`, without using namespace `std`, then we must include the using *directive*

```
using std::cout
```

With using directives we can selectively include names from a namespace.

Namespaces and Class Hierarchies

Namespaces

representing a point
defining a namespace

Class Hierarchies

a **hierarchy of shapes**
polymorphism:
changing shape

Multiple Inheritance

inheriting from two classes
ambiguities and refactoring

1 Namespaces
representing a point
defining a namespace

2 Class Hierarchies
a hierarchy of shapes
polymorphism: changing shape

3 Multiple Inheritance
inheriting from two classes
ambiguities and refactoring

a hierarchy of shapes

Namespaces

representing a point
defining a namespace

Class

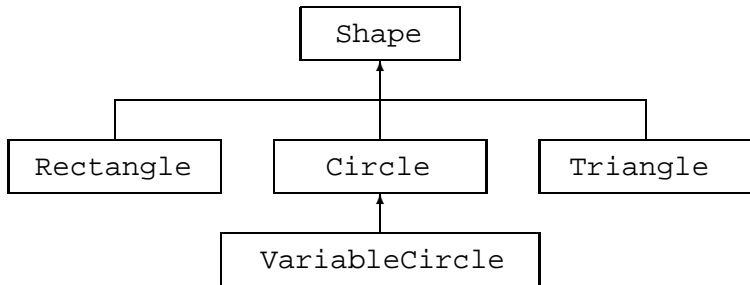
Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two classes
ambiguities and refactoring



- A shape represents a planar geometrical figure.
- The abstract class `Shape` stores the name.
- An object of the class `VariableCircle` has a center and a radius that can change.

the abstract class Shape

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

```
#include <string>

class Shape
{
    private:
        std::string label;

    public:
        Shape(std::string name)
            { label = name; }
        std::string get_name() const
            { return label; }
        void relabel(std::string name)
            { label = name; }
        virtual double area() const = 0;
        virtual void read() = 0;
        virtual std::string to_string() const = 0;
};
```

inheriting from Shape

Namespaces

representing a point
defining a namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two classes
ambiguities and refactoring

```
#include "shape.h"
#include <string>

class Rectangle : public Shape
{
    protected: // visible to derived classes
        double width, height;

    public:
        Rectangle() : Shape("")
            { width = 0.0; height = 0.0; }
        Rectangle(std::string name, double w,
            double h) :
            Shape(name), width(w), height(h) {}

        double area() const;
        void read();
        std::string to_string() const;
};
```

the file `rectangle.cpp`

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

```
#include "rectangle.h"
#include <sstream>
#include <iostream>

double Rectangle::area() const
{
    return width*height;
}
```

With this definition we override the pure virtual `area` function of the class `Shape`.

prompting for a rectangle

Observe the difference between a using declaration and a using directive.

```
void Rectangle::read()  
{  
    using namespace std; // using declaration  
    using std::string;   // using directive  
  
    string name; // if no directive, std::string  
  
    cout << " give name : "; cin >> name;  
    this->relabel(name);  
    cout << " give width : "; cin >> width;  
    cout << "give height : "; cin >> height;  
}
```

Note: the pointer `*this` refer to the object.

string representation

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a **hierarchy** of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

Useful to define the operator<<:

```
std::string Rectangle::to_string() const
{
    std::ostringstream s;

    s << "rectangle " << this->get_name()
      << " has width " << width
      << " and height " << height;

    return s.str();
}
```

working with rectangles

Namespaces

representing a point
defining a namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two classes
ambiguities and refactoring

```
#include "rectangle.h"
using namespace std;

int main()
{
    Rectangle r("A",2,1.23);

    cout << r.to_string() << endl;
    cout << "area of " << r.get_name()
         << " is " << r.area() << endl;
}
```

shows

```
$ /tmp/test_shape
rectangle A has width 2 and height 1.23
area of A is 2.46
$
```

pointing to a shape

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

Dynamic creation of a rectangle requires a pointer and calling `new`:

```
Rectangle *t;  
t = new Rectangle();  
cout << "reading second rectangle ..."  
      << endl;  
t->read();  
cout << "area of " << t->get_name()  
      << " is " << t->area() << endl;
```

Note: `t->read()` equals `(*t).read()`.

Namespaces and Class Hierarchies

Namespaces

representing a point
defining a namespace

Class Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple Inheritance

inheriting from two classes
ambiguities and refactoring

- 1 Namespaces
representing a point
defining a namespace

- 2 Class Hierarchies
a hierarchy of shapes
polymorphism: changing shape

- 3 Multiple Inheritance
inheriting from two classes
ambiguities and refactoring

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

We cannot allocate an object of class `Shape` because `Shape` is an abstract class.

```
Shape *s; // cannot allocate
// s = new Shape("some shape");

s = t;
cout << "second rectangle again..."
      << endl;
cout << s->to_string() << endl;
```

but we can access the rectangle `t` via a pointer to an object of the class `Shape`.

the Circle class

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

Area πr^2 of a circle with radius r requires GNU C mathematical library:

```
#include <cmath>

double Circle::area() const
{
    return M_PI*pow(radius,2.0);
}
```

Where `M_PI` approximates π and `pow` computes r^2 .

The rest of the `Circle` class is very similar to `Rectangle`.

make and makefile

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

For larger programs, the make facility is handy:

```
$ make test_shape
```

executes

```
g++ -c rectangle.cpp circle.cpp  
g++ -o /tmp/test_shape rectangle.o \  
    circle.o test_shape.cpp
```

as defined in the makefile

```
test_shape:  
    g++ -c rectangle.cpp circle.cpp  
    g++ -o /tmp/test_shape rectangle.o \  
        circle.o test_shape.cpp
```

Notice the TABs before the `g++`.

test_shape continued...

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

```
Circle *c;
```

```
c = new Circle("C",1.0);  
cout << c->to_string() << endl;  
cout << "area of " << c->get_name()  
    << setprecision(17)  
    << " is " << c->area() << endl;
```

```
s = c;  
cout << "getting circle again..."  
    << endl;  
cout << s->to_string() << endl;
```

With the pointer *s* we can access once a rectangle and once a circle.

Namespaces and Class Hierarchies

Namespaces

representing a point
defining a namespace

Class Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple Inheritance

inheriting from two classes
ambiguities and refactoring

1 Namespaces
representing a point
defining a namespace

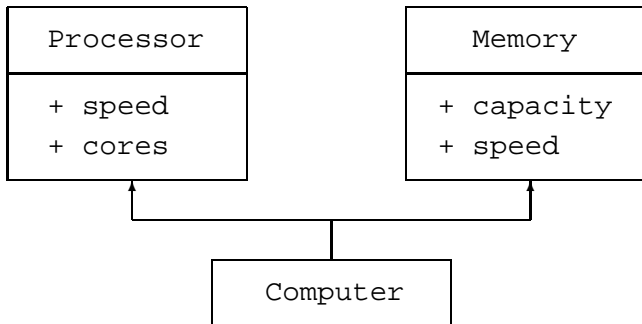
2 Class Hierarchies
a hierarchy of shapes
polymorphism: changing shape

3 Multiple Inheritance
inheriting from two classes
ambiguities and refactoring

multiple inheritance

A computer consists of

- processor, clock speed, #cores;
- memory, capacity, and speed.



the class Processor

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

```
class Processor
{
    public:

        Processor(double s, int c);
        double get_speed() const;
        int get_cores() const;
        std::string to_string() const;

    private:

        double speed; /* clock frequency */
        int cores;    /* number of cores */
};
```

the class Memory

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

```
class Memory
{
    public:

        Memory(double s, int c);
        int get_speed() const;
        int get_capacity() const;
        std::string to_string() const;

    private:

        int speed;    /* clock frequency */
        int capacity; /* capacity in Gb */
};
```

the class Computer

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

```
#include "processor.h"
#include "memory.h"
#include <string>

class Computer : public Processor, public Memory
{
public:

    Computer(double ps, int pc, int ms, int mc) :
        Processor(ps,pc), Memory(ms,mc) {}

    std::string to_string() const;
};
```

the main program

Namespaces

representing a point

defining a
namespace

Class

Hierarchies

a hierarchy of shapes

polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classesambiguities and
refactoring

```
#include "computer.h"
#include <iostream>
using namespace std;

int main()
{
    Computer c(2.26,2,1067,2);

    cout << "your computer is " << endl;
    cout << c.to_string() << endl;
}
```

shows

```
your computer is
  Processor : 2.26 Ghz, 2 cores
  Memory : 2 GB, 1067 MHz
```

Namespaces and Class Hierarchies

Namespaces

representing a point
defining a namespace

Class Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple Inheritance

inheriting from two classes
ambiguities and refactoring

1 Namespaces
representing a point
defining a namespace

2 Class Hierarchies
a hierarchy of shapes
polymorphism: changing shape

3 Multiple Inheritance
inheriting from two classes
ambiguities and refactoring

Ambiguities and Refactoring

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes
ambiguities and
refactoring

Multiple inheritance may lead to inefficiencies and confusion. For example:

- an employee has name, address, salary, hours;
- a student has name, address, major, gpa.

A student worker is an employee and a student.

Given the classes `Employee` and `Student`, the class `StudentWorker` inherits from `Employee` and `Student`.
Problem: name and address stored twice.

Refactoring: store common data of `Employee` and `Student` in one class `Person`.

Summary + Assignments

Namespaces

representing a point
defining a namespace

Class Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple Inheritance

inheriting from two classes
ambiguities and refactoring

Ended Chapter 3: *Inheritance and Class Hierarchies*. Namespaces are important because we may want to compare our own implementations of a vector, stack, list, stack, queue, tree, map, with what is in the standard template library of C++.

Assignments:

- 1 Make two classes `Vector`, one for the plane and another for 3-space, with respective namespaces `PlanarVectors` and `SpatialVectors`. In addition to the constructor, provide a method to compute the length of a vector and a `to_string()` method. Write code to test the two classes.
- 2 Add a class `Triangle` as a third class inheriting from `Shape`. Give code to test the `area` function.

more assignments

Namespaces

representing a point
defining a
namespace

Class

Hierarchies

a hierarchy of shapes
polymorphism:
changing shape

Multiple

Inheritance

inheriting from two
classes

ambiguities and
refactoring

- 3 Define the insertion operator `<<` for objects in the shape hierarchy. Which class will you change? Give code for a test program to show your `<<` works on rectangles and circles.
- 4 Make a class `VariableCircle`, inheriting from `Circle`. Objects of the `VariableCircle` class have an origin (by default at (0.0,0.0)) and their radius can change. Write code to test the operations of this class.
- 5 Extend our computer model via a class `Disk` with similar members as classes `Processor` and `Memory`. Modify the class `Computer` so it inherits from `Disk`.