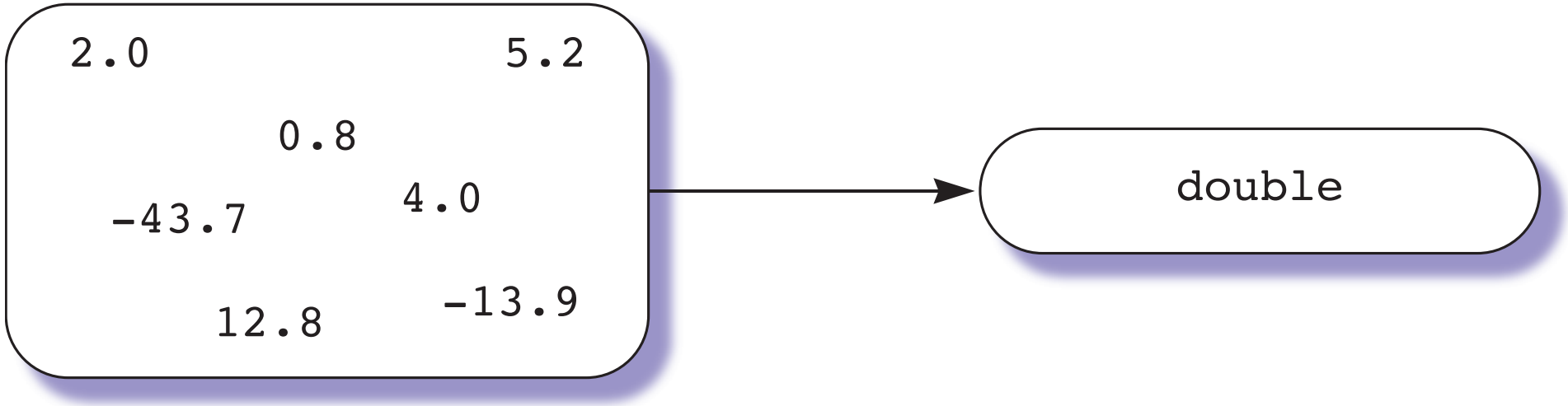
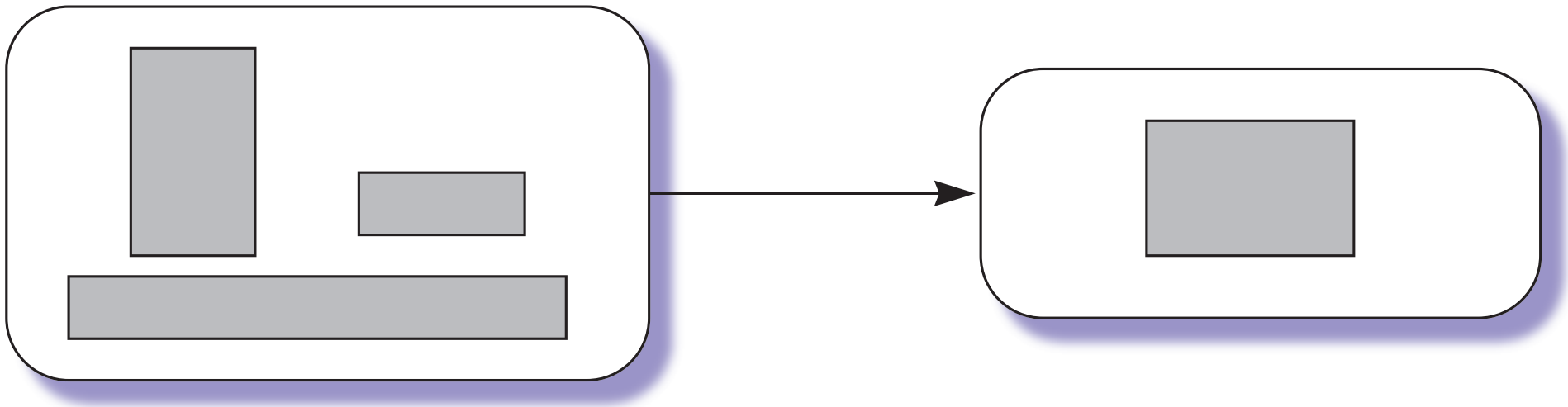
Abstraction

Type abstraction for type `double`.



Structure abstraction



```
struct Rectangle {
    double length;
    double width;
};

myRectangle.length = 2.0;
```

Statement abstraction

```
load length, r1
load width, r2
add r1, r2, r3
mul 2.0, r3, r3
store r3, perim
```

→

```
perim = 2.0 * (length + width);
```

Procedure abstraction

```
double perimeter(Rectangle r) {
    return 2.0 * (r.length + r.width);
}
```

→

```
cout << perimeter(myRectangle);
```

```
cout << perimeter(myRectangle);
cout << perimeter(yourRectangle);
```

```
int gcd(int m, int n) {
    if (0 == n) {
        return m;
    } else {
        return gcd(n, m % n);
    }
}
```

```
temp = gcd(num, denom);
```

Class abstraction

```
struct Rectangle {
    double length;
    double width;
};
```

```
double area(Rectangle r);

double perimeter(Rectangle r);
```

```
class Rectangle {
private
    double length;
    double width;
public
    double area();
    double perimeter();
};
```

Rectangle
– length: double – width: double
+ area( ): double + perimeter( ): double

OO terminology

- *class* corresponds to *type*
- *object* corresponds to *variable*
- *method* corresponds to *procedure* or *function*

UML terminology	C++ terminology
class	class
superclass	base class
subclass	derived class
attribute	data member
operation / method	member function
visibility	access specifier
parameterized class	template
abstract	pure virtual

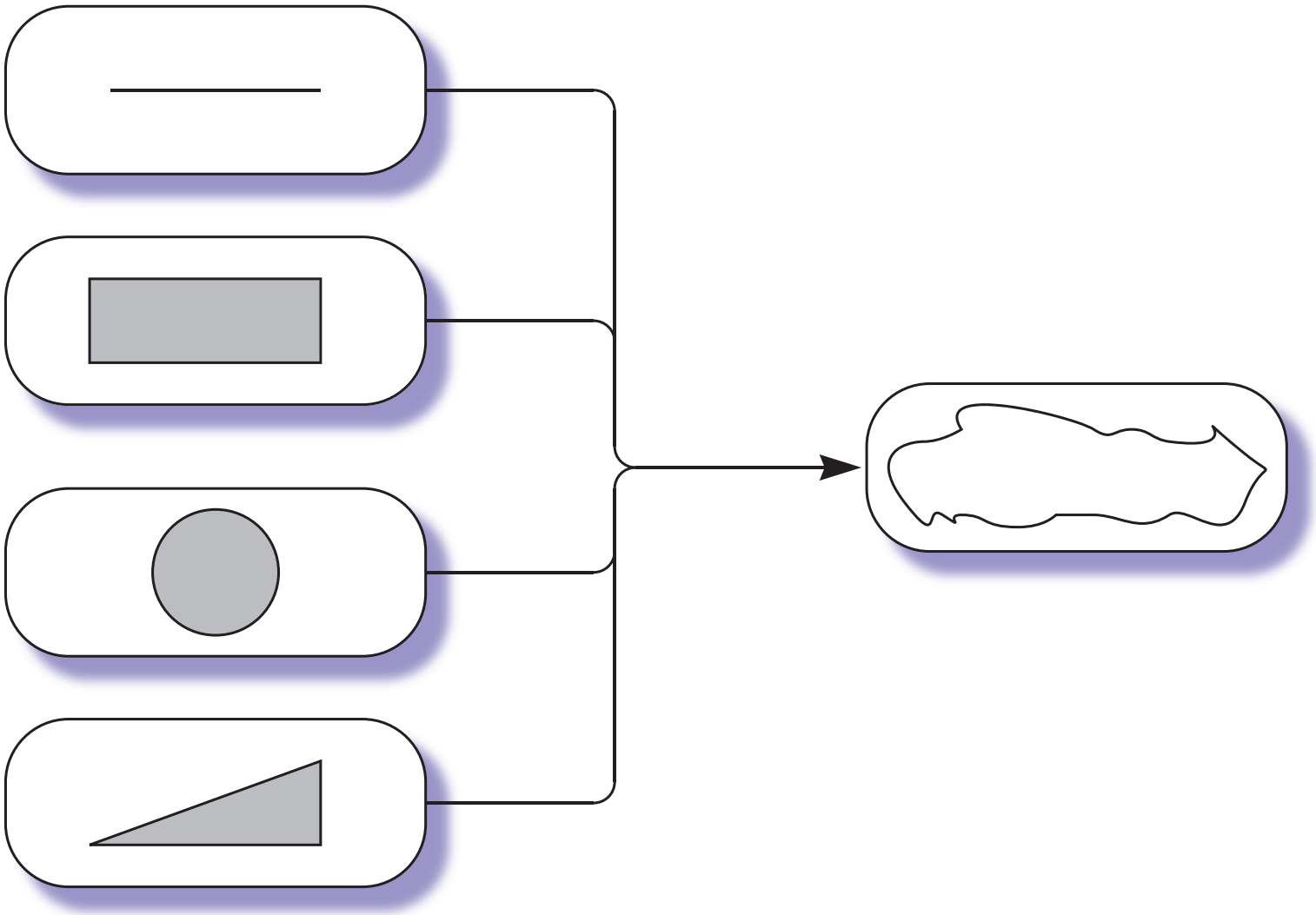
```
class Rectangle {
private:
    double length;
    double width;
public:
    double area();
    double perimeter();
};
```

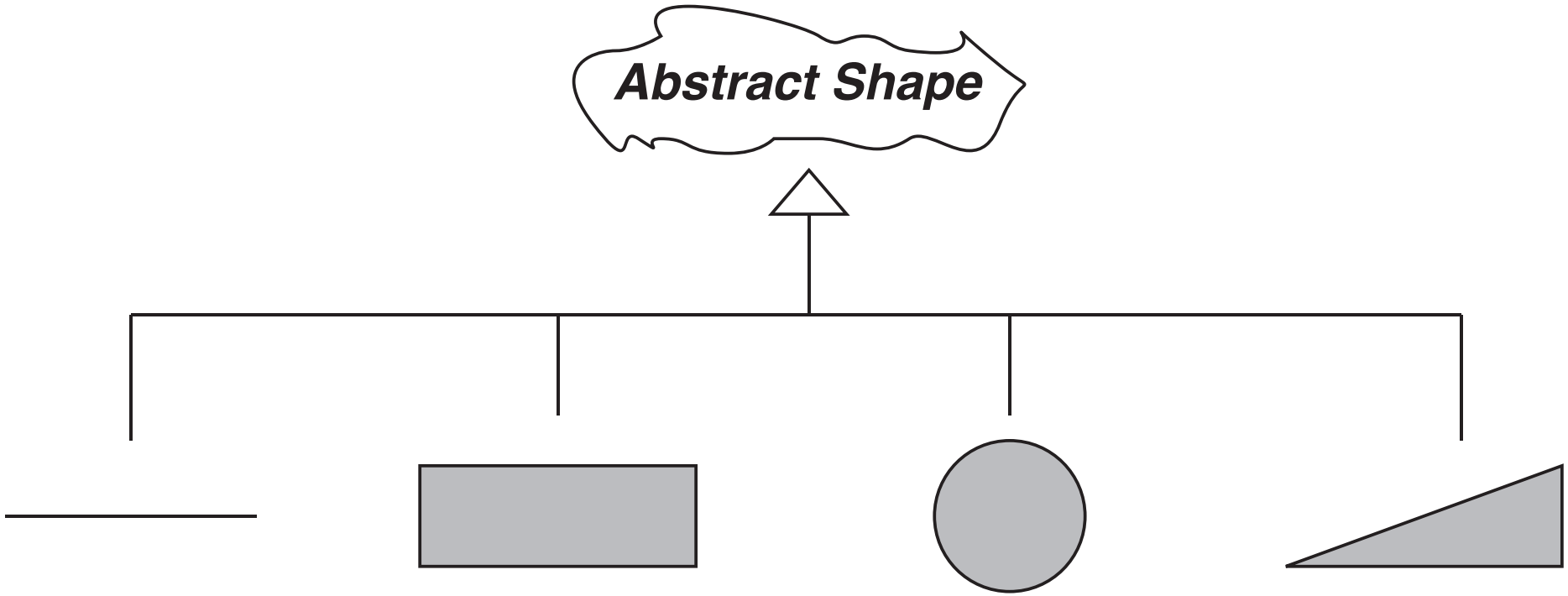
```
class Rectangle {
private:
    double length;
    double width;
public:
    double area();
    double perimeter();
};

double Rectangle::perimeter() {
    return 2.0 * (length + width);
}
```

```
class Rectangle {
private:
    double length;
    double width;
public:
    double area();
    double perimeter();
};

double Rectangle::perimeter() {
    return 2.0 * (length + width);
}

Rectangle myRectangle;
```

```
class Rectangle {
private:
    double length;
    double width;
public:
    double area();
    double perimeter();
};

double Rectangle::perimeter() {
    return 2.0 * (length + width);
}

Rectangle myRectangle;

cout << myRectangle.perimeter();
```

Behavior abstraction




Abstract Shape

The abstract shape class

```
#ifndef AShape_hpp
#define AShape_hpp

#include <iostream> // ostream.
using namespace std;

class AShape {
public:
    virtual ~AShape() = default;
    // Virtual destructor necessary for subclassing.

    virtual double area() = 0;
    // Post: The area of this shape is returned.
```

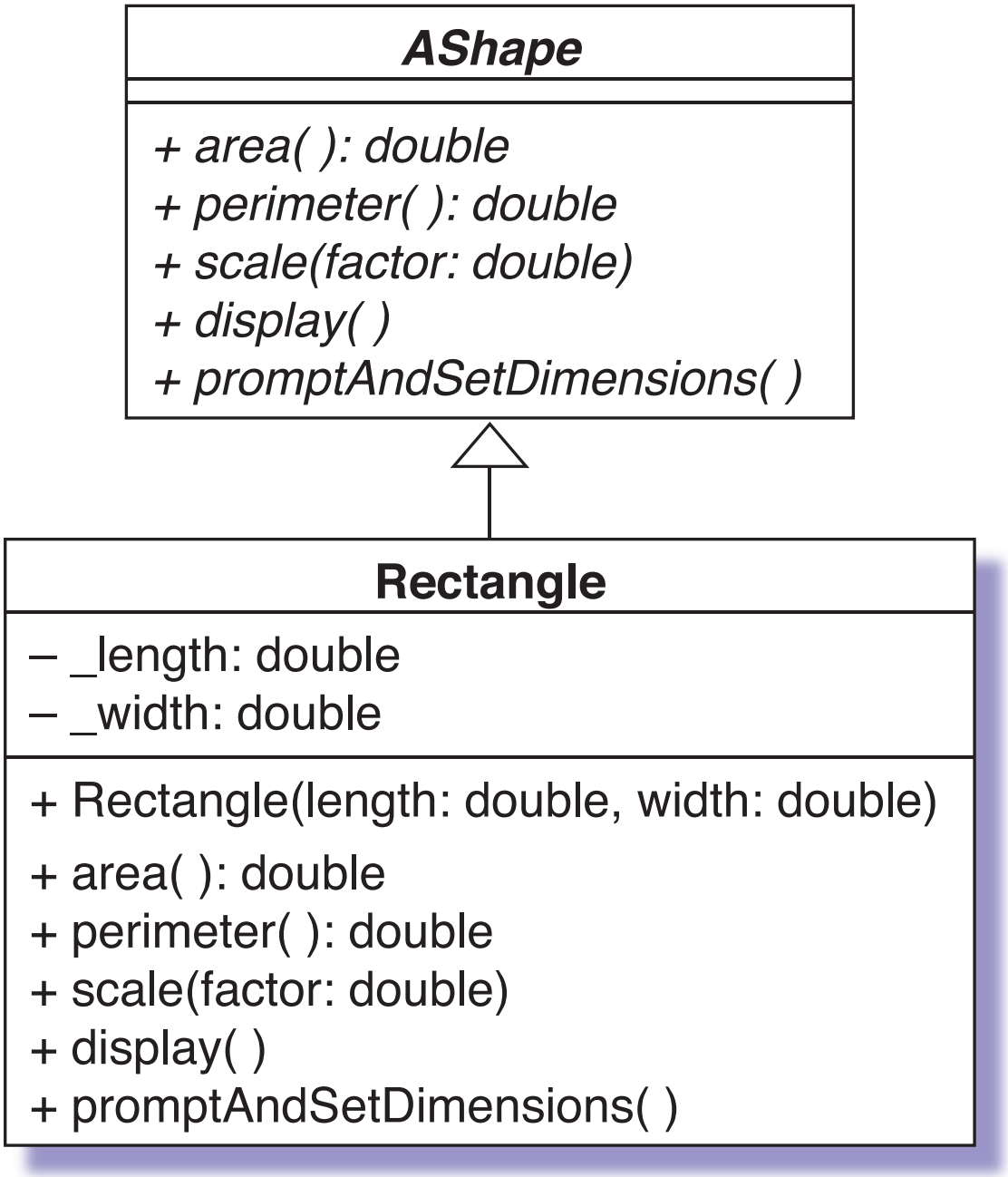
AShape
+ area( ): double *+ perimeter( ): double* *+ scale(factor: double)* *+ display( )* *+ promptAndSetDimensions( )*

```
    virtual double perimeter() = 0;
    // Post: The perimeter of this shape is returned.

    virtual void scale(double factor) = 0;
    // Pre: factor > 0.0
    // Post: This shape's dimensions are multiplied by factor.

    virtual void display(ostream &os) = 0;
    // Post: This shape's name and dimensions are printed to os.

    virtual void promptAndSetDimensions() = 0;
    // Post: This shape's dimensions are prompted and set.
    // No dimension is negative.
};

#endif
```

The rectangle subclass

```
class Rectangle : public AShape {
private:
    double _length;
    double _width;

public:
    explicit Rectangle(double length = 0.0, double width = 0.0);
    // Pre: length >= 0.0 and width >= 0.0.
    // Post: This rectangle is initialized with
    // length length and width width.

    double area() override;
    double perimeter() override;
    void scale(double factor) override;
    void display() override;
    void promptAndSetDimensions() override;
};
```

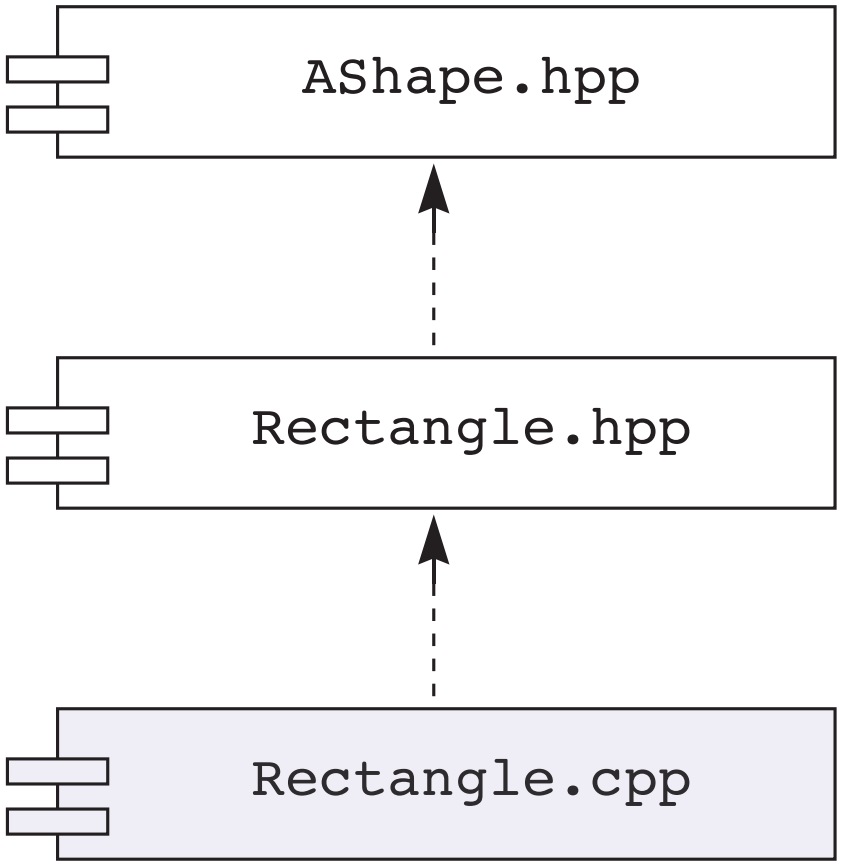
Figure 1.10


AShape
+ area( ): double
+ perimeter( ): double
+ scale(factor: double)
+ display( )
+ promptAndSetDimensions( )
Rectangle
– _length: double
– _width: double
+ Rectangle(length: double, width: double)
+ area( ): double
+ perimeter( ): double
+ scale(factor: double)
+ display( )
+ promptAndSetDimensions( )

```
#include "Rectangle.hpp"
#include "Utilities.hpp"

Rectangle::Rectangle(double length, double width) {
    if (length < 0.0 || width < 0.0) {
        cerr << "Rectangle precondition violated: "
             << "length and width cannot be negative." << endl;
        throw -1;
    }
    _length = length;
    _width = width;
}

double Rectangle::area() {
    return _length * _width;
}

double Rectangle::perimeter() {
    return 2.0 *(_length + _width);
}
```

```
void Rectangle::scale(double factor) {
    cerr << "scale: Exercise for the student." << endl;
    throw -1;
}

void Rectangle::display(ostream &os) {
    os << "Rectangle" << endl;
    os << "Length: " << _length << endl;
    os << "Width: " << _width << endl;
}

void Rectangle::promptAndSetDimensions() {
    _length = promptDoubleGE("Length?", 0.0);
    _width = promptDoubleGE("Width?", 0.0);
}
```

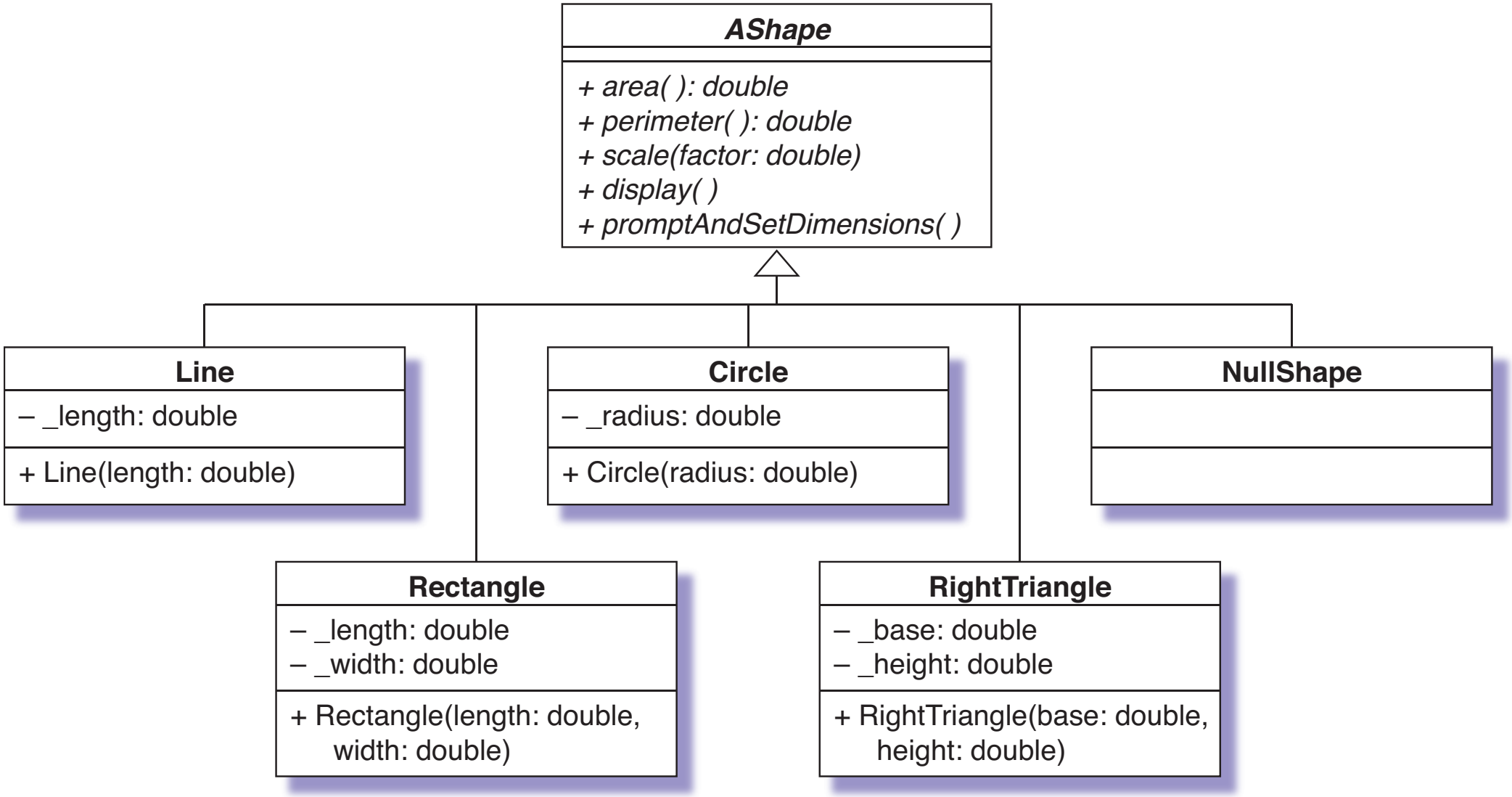
Component diagram



The C++ `try/catch/throw` statement

```
Rectangle::Rectangle(double length, double width) {
    try {
        if (length < 0.0 || width < 0.0) {
            throw -1;
        }
        _length = length;
        _width = width;
    }
    catch(int a) {
        cerr << "Rectangle precondition violated: "
             << "length and width cannot be negative." << endl;
        exit(1);
    }
}
```

Class diagram for abstract and concrete shapes



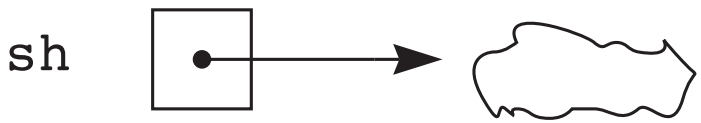
Demo dp4ds Shape project

Specification of shape main program

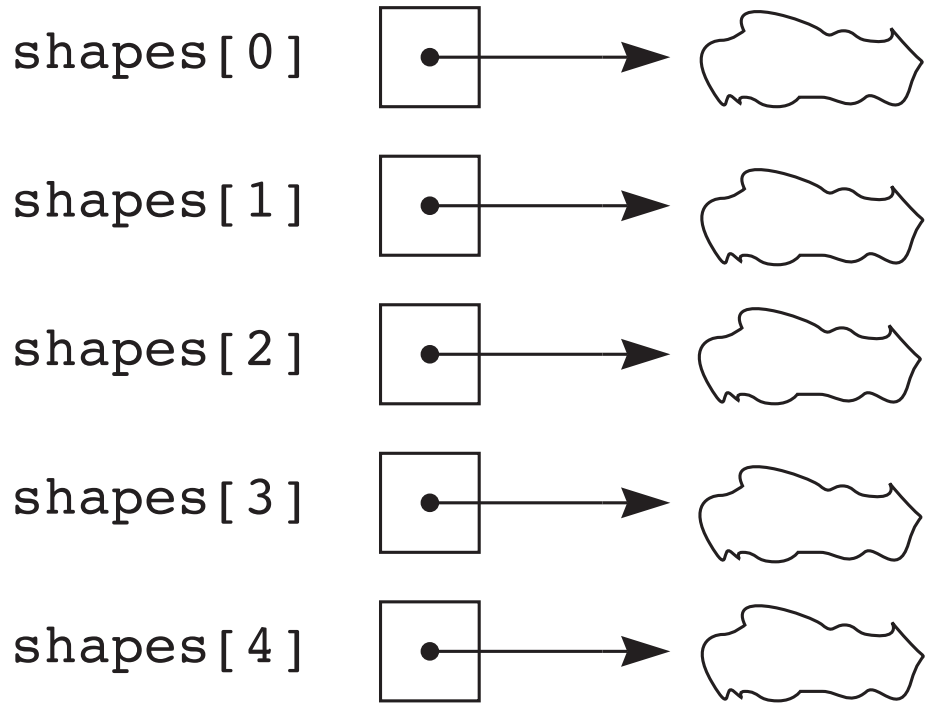
```
void initialize(shared_ptr<AShape> shapes[], int cap);
// Pre: shapes[0..cap - 1] is allocated.
// Post: All shapes[0..cap - 1] are initialized to the null shape.

void promptLoop(shared_ptr<AShape> shapes[], int cap);
// Loop to prompt the user with the top-level main prompt.
// Post: User has selected the quit option.

void makeShape(shared_ptr<AShape> &sh);
// Prompts user for dimensions.
// Post: Original sh is deleted and new sh is created.

char shapeType();
// Prompts user for shape letter, lowercase or uppercase.
// Post: Uppercase character L, R, C, T, or M is returned.
```

```
void clearShape(shared_ptr<AShape> &sh);
// Post: sh is made the null shape.

void printArea(shared_ptr<AShape> sh);
// Post: sh's area is printed to standard output.

void printPerimeter(shared_ptr<AShape> sh);
// Post: The perimeter of this sh is printed to standard output.

void scaleShape(shared_ptr<AShape> sh);
// Prompts user for scale factor.
// Post: sh's dimensions are multiplied by the factor.

void displayShape(shared_ptr<AShape> sh);
// Post: sh's name and dimensions are printed to standard output.
```

Visual depiction of parameters



(a) Visual depiction of parameter `sh`.



(b) Visual depiction of parameter `shapes`.

The ShapeMain program

```
int main() {
    const int NUM_SHAPES = 5;
    shared_ptr<AShape> shapes[NUM_SHAPES];
    initialize(shapes, NUM_SHAPES);
    promptLoop(shapes, NUM_SHAPES);
    return EXIT_SUCCESS;
}
```

```
void initialize(shared_ptr<AShape> shapes[], int cap) {
    for (int i = 0; i < cap; i++) {
        shapes[i] = make_shared<NullShape>();
    }
}
```

When the `make_shared<>()` function executes, it

- allocates storage from the heap for the attributes of the object, and
- returns a pointer to the newly allocated storage.

Figure 1.20

```
void promptLoop(shared_ptr<AShape> shapes[], int cap) {
    char response = '\0';
    do {
        cout << "\nThere are [0.." << cap - 1 << "] shapes." << endl;
        cout << "(m)ake  (c)lear  (a)rea  (p)erimeter  (s)cale  (d)isplay  "
             << "(q)uit: ";
        cin >> response;
        switch (toupper(response)) {
        case 'M':
            makeShape(shapes[promptIntBetween("Which shape?", 0, cap-1)]);
            break;
        case 'C':
            clearShape(shapes[promptIntBetween("Which shape?", 0, cap-1)]);
            break;
        case 'A':
            printArea(shapes[promptIntBetween("Which shape?", 0, cap-1)]);
            break;
```

```
        case 'P':
            printPerimeter(shapes[promptIntBetween("Which shape?", 0, cap-1)]);
            break;
        case 'S':
            scaleShape(shapes[promptIntBetween("Which shape?", 0, cap-1)]);
            break;
        case 'D':
            displayShape(shapes[promptIntBetween("Which shape?", 0, cap-1)]);
            break;
        case 'Q':
            break;
        default:
            cout << "\nIllegal command." << endl;
            break;
        }
    } while (toupper(response) != 'Q');
}
```

Figure 1.21

```
void makeShape(shared_ptr<AShape> &sh) {
    switch (shapeType()) {
    case 'L':
        sh = make_shared<Line>();
        break;
    case 'R':
        sh = make_shared<Rectangle>();
        break;
    case 'C':
        sh = make_shared<Circle>();
        break;
    case 'T':
        sh = make_shared<RightTriangle>();
        break;
    case 'M':
        //Exercise for the student.
        break;
    default:
        break;
    }
    sh->promptAndSetDimensions();
}
```

Figure 1.21

```
char shapeType() {
    char ch;
    cout << "(l)ine  (r)ectangle  (c)ircle  right(t)riangle  (m)ystery: ";
    cin >> ch;
    ch = toupper(ch);
    while (ch != 'L' && ch != 'R' && ch != 'C' && ch != 'T' && ch != 'M') {
        cout << "Must be l, r, c, t, or m.  Which type? ";
        cin >> ch;
        ch = toupper(ch);
    }
    return ch;
}
```

When the `make_shared<>()` operator executes on an object, it

- allocates storage from the heap for the attributes of the object,
- calls the constructor based on the number and types of parameters in the parameter list, and
- returns a pointer to the newly allocated storage.

```
void clearShape(shared_ptr<AShape> &sh) {
    sh = make_shared<NullShape>() ;
}

void printArea(shared_ptr<AShape> sh) {
    cout << "\nArea: " << sh->area() << endl;
}

void printPerimeter(shared_ptr<AShape> sh) {
    cout << "\nPerimeter: " << sh->perimeter() << endl;
}

void scaleShape(shared_ptr<AShape> sh) {
    sh->scale(promptDoubleGE("Scale factor?", 0.0));
}

void displayShape(shared_ptr<AShape> sh) {
    cout << endl;
    sh->display();
}
```

The abstraction process