Object-oriented programming:
Abstract classes

Topics:
- Abstract operations (and pure virtual functions)
- Abstract classes

Problem: Abstract operations

Some classes might define operations for which there is no reasonable default method

Example: Class Shape
- Defines an area() operation
- But there is no general method for computing area of a shape
  - Circle: area is pi times square of the radius
  - Rectangle: area is length times width

Solution: Abstract operations (called pure virtual functions in C++)
Pure virtual function

Defn: Mechanism by which a class can declare an operation w/o providing a method

Syntax:

```cpp
class BaseClass {
    public:
        virtual void pvf() = 0;
};

class DerivedClass : public BaseClass {
    public:
        void pvf() { ... }
};
```

Example: Class Shape

```cpp
class Shape {
    public:
        virtual unsigned area() = 0; // Observe: Pure specifier
};

Class Rectangle : public Shape {
    public:
        Rectangle( unsigned l, unsigned h)
            : length(l), height(h) {}
        unsigned area() { return length * width; }
    protected:
        unsigned length, height;
};
```
Abstract Class

**Defn:** A Class that cannot be instantiated

**Illegal**

```
Shape var;
void f(Shape x)
Shape g();
Shape* x = new Shape;
```

**Legal**

```
Shape& var;
void Foo(Shape& x)
Shape* Bar();
Shape* x = new Rectangle(...);
```

Declaring an abstract class

In C++, a class is abstract if it:
- declares (or inherits) a pure-virtual function; or
- has a protected constructor

Example:
```cpp
class GUIElement {
    public:
        void move(unsigned x, unsigned y);
    protected:
        unsigned xPosition, yPosition;
        GUIElement( unsigned x=0, unsigned y=0 )
            : xPosition(x), yPosition(y) {}
};
```
Uses of abstract classes

Defining an abstract “placeholder” that can hold objects of various types
- E.g., Shape
- Useful for building recursive (self-referential) object structures

Factoring common code into an abstract concept

Definition of role-classes for use in collaboration-based designs

Collaborative Exercise

Design classes for arithmetic expression trees. Each arithmetic operator class should provide operations for retrieving operand expressions. Define at least the following classes:
- Variable
- Literal
- Negate
- Add, Subtract, Multiply, Divide

**Hint:** You will need to invent some abstract classes
Class BinaryExpr

class BinaryExpr : public Expr {
public:
    const Expr* getLeftOperand() const { return leftOperand; }
    const Expr* getRightOperand() const { return rightOperand; }

protected:
    const Expr* leftOperand;
    const Expr* rightOperand;
    BinaryExpr( const Expr* l, const Expr* r )
        : leftOperand( l ), rightOperand( r ) {}
};

Note: Constructor is not public!