Chapter 14: More About Classes
**Class Members**

class Rectangle
{
    private:
        double width;
        double length;
    public:
        void setWidth(double);
        void setLength(double);
        double getWidth();
        double getLength();
};

When the Rectangle class is compiled...

How many copies of the width and length variables will there be?

How many copies of the public functions will there be?
All the Rectangles...

`getWidth()` is the same for all!

Every rectangle has its own length and width...
Class Members

class Rectangle
{
    private:
        double width;
        double length;
    public:
        void setWidth(double);
        void setLength(double);
        double getWidth();
        double getLength();
};

Every object has its own width and length

All objects share the class functions
There are two kinds of member variables:

- **instance variable**: a member variable in a class. Each object has its own copy.
- **static variable**: one variable shared among all objects of a class.
### STATIC MEMBER VARIABLE

#### Contents of `Tree.h`

```cpp
// Tree class
class Tree {

private:

    static int objectCount; // Static member variable - shared by all

public:

    // Constructor
    Tree()
    { objectCount++; }

    // Accessor function for objectCount
    int getObjectCount() const
    { return objectCount; }

};

// Definition of the static member variable, written
// outside the class.
int Tree::objectCount = 0;
```

Static member declared here.

Static member initialized here.
Three Instances of the Tree Class, But Only One objectCount Variable

- Every time a Tree object is created, the static variable is incremented:

```java
Tree oak; // default constructor: objectCount++;
Tree elm; // objectCount++;
Tree pine; // objectCount++;
```
STATIC MEMBER FUNCTION

- **static** member function:
  - can be used to access **static** member variable
  - can be called before any objects are defined
- Declared with **static** before return type:

  ```cpp
  static int getObjectCount() const
  {
    return objectCount;
  }
  ```
- Static member functions can only access **static** member data. - Why?
- Can be called independent of objects:

  ```cpp
  int num = Tree::getObjectCount();
  ```
Modified Version of Tree.h

```cpp
// Tree class
class Tree {
private:
    static int objectCount; // Static member variable.
public:
    // Constructor
    Tree()
    { objectCount++; }

    // Accessor function for objectCount
    static int getObjectCount() const
    { return objectCount; }
};

// Definition of the static member variable, written
// outside the class.
int Tree::objectCount = 0;
```

Now we can call the function like this:
```cpp
cout << "There are " << Tree::getObjectCount() << " objects.\n";
```
Friends of Classes

- Private members of a class are not accessible outside the class.
- However, sometimes it is convenient to share data directly.
- In C++, you can declare a friend: a function or class that is not a member of a class, but has access to private members of the class.
- A friend function can be a stand-alone function or a member function of another class.
- It is declared a friend of a class with friend keyword in the function prototype.
FRIEND FUNCTION DECLARATIONS

- Stand-alone function:
  ```
  friend void setAVal(intVal&, int);
  // declares setAVal function to be
  // a friend of this class
  ```

- Member function of another class:
  ```
  friend void SomeClass::setNum(int num);
  // setNum function from SomeClass
  // class is a friend of this class
  ```

- NB: make sure you have #included everything correctly – see example code
FRIEND CLASS DECLARATIONS

- Class as a friend of a class:
  ```cpp
  class FriendClass
  {
      ...
  };

  class NewClass
  {
      public:
          friend class FriendClass; // declares
          // entire class FriendClass as a friend
          // of this class
          ...
  };
  ```
MEMBERWISE ASSIGNMENT

- `int x = 1;`
- `int y = x; // copies value of x into y`
- You can also use = to assign one object to another, or to initialize an object with another object’s data.

- Copies member to member. *e.g.*, 
  
  ```
  instance2 = instance1; // means:
  copy all member values from instance1 and assign to the corresponding member variables of instance2
  ```

- Use at initialization:
  ```
  Rectangle r1(5, 3);
  Rectangle r2 = r1; // now r2 is also 5,3
  ```
MEMBERWISE ASSIGNMENT

- What could possibly go wrong with memberwise assignment?
- What if one of the members is an array?
- Can you do something like this:
  ```
  int array1[3] = {1,2,3};
  int array2[3] = array1;
  ```
- In the next lecture, we will learn how to copy objects that contain arrays

- Any questions?