Classes and Objects in C++

Shahram Rahatlou

SAPIENZA
UNIVERSITÀ DI ROMA

Corso di Programmazione++

Roma, 11 March 2008

Today's Lecture

- Classes
  - data members and member functions

- Constructors
  - Special member functions

- private and public members

- Helper and utility methods
  - setters
  - getters
  - accessors

Classes in C++

- A class is a set of data and functions that define the characteristics and behavior of an object
  - Characteristics also known as attributes
  - Behavior is what an object can do and is referred to also as its interface

```
class Result {
  public:

   // constructors
   Result() { }
   Result(const double& mean, const double& stdDev) {
     mean_ = mean;
     stdDev_ = stdDev;
   }

   // accessors
   double getMean() { return mean_; };
   double getStdDev() { return stdDev_; };

  private:
   double mean_;
   double stdDev_;
};
```

Interface
or
Member Functions

Data members or attributes

Don’t’s forget ; at the end of definition!

Data Members (Attributes)

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

class Datum {
    double value_;
    double error_;
};
```

- Data defined in the scope of a class are called data members of that class
- Data members are defined in the class and can be used by all member functions
- Contain the actual data that characterizes the content of the class
- Can be public or private
    - public data members are generally bad and symptom of bad design
    - More on this topic later in the course

Interface: Member Functions

- Member functions are methods defined inside the scope of a class
  - Have access to all data members

```
// Student
#include <iostream>
#include <string>
using namespace std;

class Student {
  public:
   // default constructor
   Student() { name_ = ""; }

   // another constructor
   Student(const string& name) { name_ = name; }

   // getter method: access to info from the class
   string name() { return name_; }

   // setter: set attribute of object
   void setName(const string& name) { name_ = name; }

   // utility method
   void print() {
     cout << "My name is: " << name_ << endl;
   }

  private:
   string name_; // data member
};
```

name_ is a datamember

No declaration of name_ in member functions!

name is a local variable only within setName()

Arguments of Member Functions

- All C++ rules discussed so far hold
- You can pass variables by value, pointer, or reference
- You can use the constant qualifier to protect input data and restrict the capabilities of the methods
  - This has implications on declaration of methods using constants
  - We will discuss constant methods and data members next week
- Member functions can return any type
  - Exceptions! Constructors and Destructor
    - Have no return type
    - More on this later

Access specifiers **public** and **private**

- Public functions and data members are available to anyone
- Private members and methods are available ONLY to other member functions

```
 1 #include <iostream>
 2 using std::cout;
 3 using std::endl;
 4
 5 class Datum {
 6   public:
 7     Datum() { }
 8     Datum(double val, double error) {
 9       value_ = val;
10       error_  = error;
11     }
12
13   double value() { return value_; }
14   double error() { return error_; }
15
16   void setValue(double value) { value_ = value; }
17   void setError(double error) { error_ = error; }
18
19   double value_; // public data member!!!
20
21   private:
22     double error_; // private data member
23 };
```

Access elements of an object through member selection operator "."

```
25 int main() {
26
27   Datum d1(1.1223,0.23);
28
29   cout << "d1.value(): " << d1.value()^
30        << " d1.error(): " << d1.error()
31        << endl;
32
33
34   cout << "d1.value_: " << d1.value_
35        << " d1.error_: " << d1.error_
36        << endl;
37
38   return 0;
39 }
```

Accessing private members is a compilation error!

```
$ g++ -o class1 class1.cc
class1.cc: In function `int main()':
class1.cc:22: error: `double Datum::error_' is private
class1.cc:35: error: within this context
```

private members

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

class Datum {
  public:
    Datum(double val, double error) {
      value_ = val;
      error_  = error;
    }

    double value() { return value_; }
    double error() { return error_; }

    void setValue(double value)
      { value_ = value; }
    void setError(double error)
      { error_ = error; }

    void print() {
      cout << "datum: " << value_
           << " +/- " << error_
           << endl;
    }

  private:
    double value_; // private data member!!!
    double error_; // private data member
};
```

```
int main() {

  Datum d1(1.1223,0.23);
  // setter with no return value
  d1.setValue( 8.563 );

  // getter to access private data
  double x = d1.value();

  cout << "d1.value(): " << d1.value()
       << " d1.error(): " << d1.error()
       << endl;

  d1.print();

  return 0;
}
```

```
$ g++ -o class2 class2.cc
$ ./class2
d1.value(): 8.563 d1.error(): 0.23
datum: 8.563 +/- 0.23
```

private methods

- Can be used only inside other methods but not from outside

```
 1 // class3.cc
 2 #include <iostream>
 3 using namespace std;
 4
 5 class Datum {
 6   public:
 7     Datum() { reset(); } // reset data members
 8
 9     double value() { return value_; }
10     double error() { return error_; }
11
12     void setValue(double value) { value_ = value; }
13     void setError(double error) { error_ = error; }
14
15     void print() {
16       cout << "datum: " << value_ << " +/- "
               << error_ << endl;
17     }
18
19   private:
20     void reset() {
21       value_ = 0.0;
22       error_ = 0.0;
23     }
24
25     double value_;
26     double error_;
27 };
```

```
int main() {

  Datum d1;
  d1.setValue( 8.563 );
  d1.print();

  return 0;
}
```

```
$ g++ -o class3 class3.cc
$ ./class3
datum: 8.563 +/- 0
```

```
30 int main() {
31
32   Datum d1;
33   d1.setValue( 8.563 );
34   d1.print();
35   d1.reset();
36
37   return 0;
38 }
```

```
$ g++ -o class4 class4.cc
class4.cc: In function `int main()':
class4.cc:20: error: `void Datum::reset()' is private
class4.cc:35: error: within this context
```

Hiding Implementation from Users/Clients

- How to decide what to make public or private?
- Principle of Least Privilege
  - elements of a class, data or functions, must be private unless proven to be needed as public!
- Users should rely solely on the interface of a class
- They should never use the internal details of the class
- That's why having public data members is a VERY bad idea!
  - name and characteristics of data members can change
  - Functionalities and methods remain the same
  - You must be able to change internal structure of the class without affecting the clients!

Bad Example of Public Data Members

```
class Datum {
  public:
    Datum(double val, double error) {
      value_ = val;
      error_  = error;
    }

    double value() { return value_; }
    double error() { return error_; }

    void setValue(double value) { value_ = value; }
    void setError(double error) { error_ = error; }

    void print() {
      cout << "datum: " << value_ << " +/- " << error_ << endl;
    }

  //private:       // all data are public!
    double value_;
    double error_;
};
```

```
int main() {

  Datum d1(1.1223,0.23);
  double x = d1.value();
  double y = d1.error_;
  cout << "x: " << x << "\t y: " << y << endl;

  return 0;
}
```

application uses directly the data member!

```
$ g++ -o class6 class6.cc
$ ./class6
x: 1.1223        y: 0.23
```

Bad Example of Public Data Members

Change the names of data members

No change of functionality so no one should be affected!

Same Application as before

```
class Datum {
  public:
    Datum(double val, double error) {
      val_ = val;
      err_  = error;
    }

    double value() { return val_; }
    double error() { return err_; }

    void setValue(double value) { val_ = value; }
    void setError(double error) { err_ = error; }

    void print() {
      cout << "datum: " << val_ << " +/- " << err_ << endl;
    }

  //private:       // alla data are public!
    double val_;  // value_ → val_
    double err_;  // error_ → err_
};
```

```
28 int main() {
29
30   Datum d1(1.1223,0.23);
31   double x = d1.value();
32   double y = d1.error_;
33
34   cout << "x: " << x << "\t y: " << y << endl;
35
36   return 0;
37 }
```

Our application is now broken!

But Datum has not changed its behavior!

Bad programming!

Only use the interface of an object not its internal data!

Private data members prevent this

```
$ g++ -o class7 class7.cc
class7.cc: In function `int main()':
class7.cc:32: error: 'class Datum' has no member named ‘error_'
```

Constructors

```
class Datum {
  public:
    Datum() { }
    Datum(double val, double error) {
      value_ = val;
      error_  = error;
    }

  private:
    double value_; // public data member!!!
    double error_; // private data member
};
```

- Special member functions
  - Required by C++ to create a new object
  - MUST have the same name of the class
  - Used to initialize data members of an instance of the class
  - Can accept any number of arguments
    - Same rules as any other C++ function applies

- Constructors have no return type!

- There can be several constructors for a class
  - Different ways to declare and an object of a given type

Different Types of Constructors

- Default constructor
  - Has no argument
  - On most machines the default values for data members are assigned
- Copy Constructor
  - Make a new object from an existing one
- Regular constructor
  - Provide sufficient arguments to initialize data members

```
class Datum {
  public:
    Datum() { }


    Datum(double x, double y) {
      value_ = x;
      error_ = y;
    }


    Datum(const Datum& datum) {
      value_ = datum.value_;
      error_ = datum.error_;
    }


  private:
    double value_;
    double error_;
};
```

Using Constructors

```
// class5.cc
#include <iostream>
using namespace std;

class Datum {
  public:
    Datum() { }

    Datum(double x, double y) {
      value_ = x;
      error_ = y;
    }

    Datum(const Datum& datum) {
      value_ = datum.value_;
      error_ = datum.error_;
    }

    void print() {
      cout << "datum: " << value_
           << " +/- " << error_
           << endl;
    }

  private:
    double value_;
    double error_;
};
```

```
int main() {

  Datum d1;
  d1.print();

  Datum d2(0.23,0.212);
  d2.print();

  Datum d3( d2 );
  d3.print();


  return 0;
}
```

```
$ g++ -o class5 class5.cc
$ ./class5
datum: NaN +/- 8.48798e-314
datum: 0.23 +/- 0.212
datum: 0.23 +/- 0.212
```

Default Constructors on Different Architectures

```
$ uname -a
CYGWIN_NT-5.1 lajolla 1.5.18(0.132/4/2) 2005-07-02 20:30 i686 unknown
unknown Cygwin
$ gcc -v
Reading specs from /usr/lib/gcc/i686-pc-cygwin/3.4.4/specs
…
gcc version 3.4.4 (cygming special) (gdc 0.12, using dmd 0.125)

$ g++ -o class5 class5.cc
$ ./class5
datum: NaN +/- 8.48798e-314
datum: 0.23 +/- 0.212
datum: 0.23 +/- 0.212
```

Windows XP with CygWin

```
$ uname -a
Linux pccms02.roma1.infn.it 2.6.14-1.1656 FC4smp #1 SMP Thu Jan 5 22:24:06 EST
 2006 i686 i686 i386 GNU/Linux
$ gcc -v
Using built-in specs.
Target: i386-redhat-linux
…
gcc version 4.0.2 20051125 (Red Hat 4.0.2-8)
$ g++ -o class5 class5.cc
$ ./class5
datum: 6.3275e-308 +/- 4.85825e-270
datum: 0.23 +/- 0.212
datum: 0.23 +/- 0.212
```

Fedora Core4

Default Assignment

```
// ctor.cc
#include <iostream>
using std::cout;
using std::endl;

class Datum {
  public:
    Datum(double x) { x_ = x; }
    double value()  { return x_; }
    void setValue(double x) { x_ = x; }
    void print() {
      cout << "x: " << x_ << endl;
    }

  private:
    double x_;
};
```

```
int main() {

  Datum d1(1.2);
  d1.print();

 // no default ctor. compiler error if uncommented
  //Datum d2;
  //d2.print();

  Datum d3 = d1; // default assigment by compiler
  d3.print();
  cout << "&d1: " << &d1
       << "\t &d3: " << &d3 << endl;
  return 0;
}
```

d3.x_ = d1.x_

done by compiler

```
$ g++ -o ctor ctor.cc
$ ./ctor
x: 1.2
x: 1.2
&d1: 0x23ef10    &d3: 0x23ef08
```

Question

- Can a constructor be private?
  - Is it allowed by the compiler?
  - How to instantiate an object with no public constructor?

- Find a working example of a very simple class for next week!

Accessors and Helper/Utility Methods

- Methods that allow read access to data members
- Can also provide functionalities commonly needed by users to elaborate information from the class
  - for example formatted printing of data
- Usually they do not modify the objects, i.e. do not change the value of its attributes

```
class Student {
  public:

   // getter method: access to data members
   string name() { return name_; }

   // utility method
   void print() {
     cout << "My name is: " << name_ << endl;
   }

  private:
   string name_; // data member
};
```

```
class Datum {
  public:

    double value() { return value_; }
    double error() { return error_; }

    void print() {
      cout << "datum: " << value_
           << " +/- " << error_
           << endl;
    }

  private:
    double value_; // public data member!!!
    double error_; // private data member
};
```

Getter Methods

- getters are helpers methods with explicit names returning individual data members
  - Do not modify the data members simply return them
  - Good practice: call these methods as getFoo() or foo() for member foo_
- Return value of a getter method should be that of the data member

```
class Datum {
  public:
    Datum(double val, double error) {
      val_ = val;
      err_  = error;
    }

    double value() { return val_; }
    double error() { return err_; }

    void setValue(double value) { val_ = value; }
    void setError(double error) { err_ = error; }

    void print() {
      cout << "datum: " << val_ << " +/- " << err_
           << endl;
    }

  private:
    double val_;
    double err_;
};
```

```
// Student
#include <iostream>
#include <string>
using namespace std;

class Student {
  public:
   // default constructor
   Student() { name_ = ""; }

   // another constructor
   Student(const string& name) { name_ = name; }

   // getter method: access to info from the class
   string name() { return name_; }

   // setter: set attribute of object
   void setName(const string& name) { name_ = name; }

   // utility method
   void print() {
     cout << "My name is: " << name_ << endl;
   }

  private:
   string name_; // data member
};
```

Setter Methods

- Setters are member functions that modify attributes of an object after it is created
  - Typically defined as void
  - Could return other values for error handling purposes
  - Very useful to assign correct attributes to an object in algorithms
  - As usual abusing setter methods can cause unexpected problems

```
// class8.cc
#include <iostream>
using namespace std;

class Datum {
  public:
    Datum(double val, double error) {
      value_ = val;
      error_  = error;
    }

    double value() { return value_; }
    double error() { return error_; }

    void setValue(double value) { value_ = value; }
    void setError(double error) { error_ = error; }

    void print() {
      cout << "datum: " << value_ << " +/- “
           << error_ << endl;
    }

  private:
    double value_;
    double error_;
};
```

```
int main() {

  Datum d1(23.4,7.5);
  d1.print();

  d1.setValue( 8.563 );
  d1.setError( 0.45 );
  d1.print();

  return 0;
}
```

```
$ g++ -o class8 class8.cc
$ ./class8
datum: 23.4 +/- 7.5
datum: 8.563 +/- 0.45
```