CIT237
Chapter 13: Introduction to Classes – Part 3

October 28, 2019

Reminders

- Quiz 5 will be held at the start of class on
  Wednesday, November 6.
- The material covered on Quiz 5 will be:
  - Lectures of October 21 through October 30.
  - Chapters 13 and 14.
  - Further updates will be provided as the date approaches.
- Programming Project 2:
  - The EXTENDED due date is TODAY:
    Monday, October 28
  - If you are having difficulty, talk to me during the Lab portion of today's class.

Member Function Overloading

- Non-constructor member functions can also be overloaded:

```
void setCost(double);
void setCost(char *);
```

- Each overloaded member function must have a unique parameter list (unique signature)

Using Private Member Functions

- A `private` member function can only be called by another member function

- It is used for internal processing by the class, not for use by code outside of the class

Inline Member Functions

- Member functions can be defined
  - inline: in class declaration
  - after the class declaration

- Inline appropriate for short function bodies:

```
int getWidth() const
    { return width; }
```

Class with Inline Member Functions

```
class RectangleWithInlineFunctions
{
   private:
      double width, length;
   public:
      void setWidth(double);
      void setLength(double);
      double getWidth() const
         { return width; }
      double getLength() const
         { return length; }
      double getArea() const
         { return width * length; }
};
```

Tradeoffs – Inline vs. Regular Member Functions

- Regular functions – when called, compiler stores return address of call, allocates memory for local variables, etc.

- Code for an inline function is copied into program in place of call – larger executable program, but no function call overhead, hence faster execution

Arrays of Objects

- Objects can be the elements of an array:

```
InventoryItem inventory[40];
```

- Default constructor for object is used when array is defined

Initilizing Arrays of Objects

- Must use initializer list to invoke constructor that takes arguments.
- For example, the InventoryItem constructor invoked below takes one argument:

```
InventoryItem inventory[3] =
  { "Hammer", "Wrench", "Pliers" };
```

Constructors with Multiple Parameters

- If the constructor requires more than one argument, the initializer must take the form of a function call:

```
InventoryItem inventory[3] =
 {InventoryItem("Hammer", 6.95, 12),
  InventoryItem("Wrench", 8.75, 20),
  InventoryItem("Pliers", 3.75, 10)};
```

Different Array Elements can Invoke Different Constructors

- It is not necessary to call the same constructor for each object in an array:

```
InventoryItem inventory[3] =
  {“Hammer”,
   InventoryItem(“Wrench”, 8.75, 20),
   “Pliers”};
```

Accessing Objects in an Array

- Objects in an array are referenced using subscripts

- Member functions are referenced using dot notation:

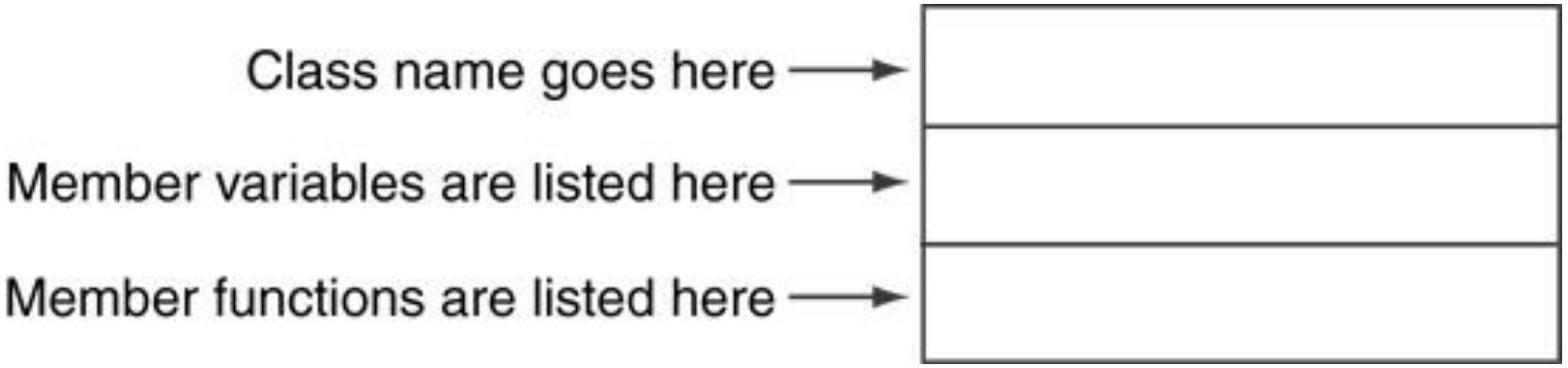
```
inventory[2].setUnits(30);
cout << inventory[2].getUnits();
```

The Unified Modeling Language

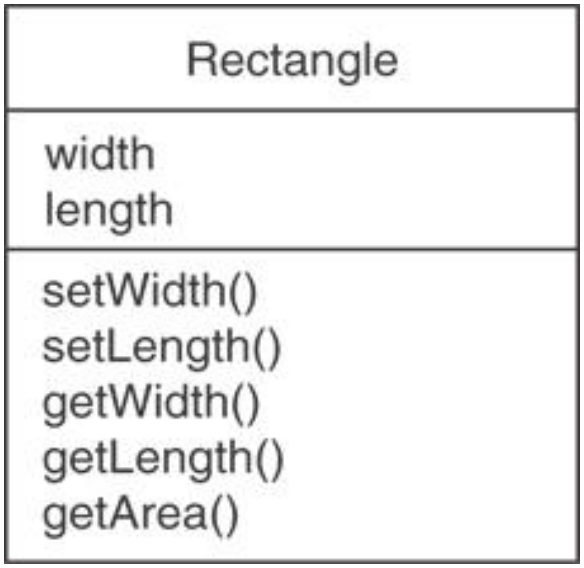
- *UML* stands for *Unified Modeling Language*.
- The UML provides a set of standard diagrams for graphically depicting object-oriented systems

UML Class Diagram

- A UML diagram for a class has three main sections.



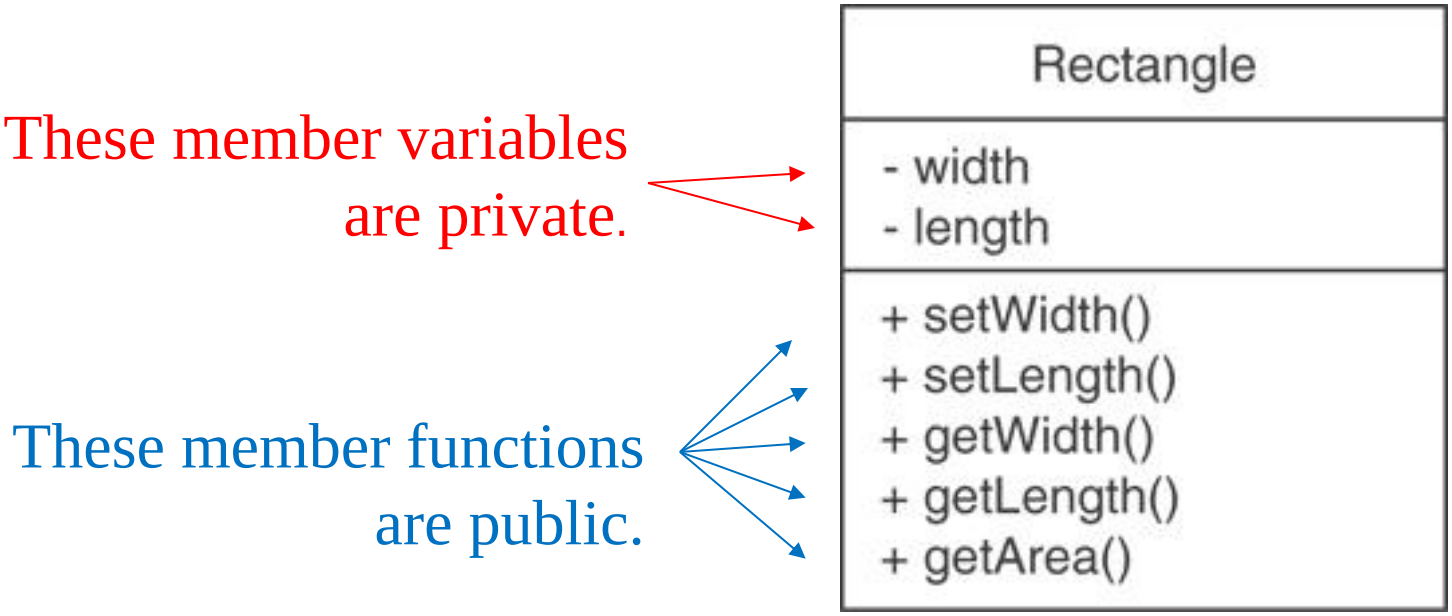
Example: A Rectangle Class




```
class Rectangle
{
   private:
      double width;
      double length;
   public:
      bool setWidth(double);
      bool setLength(double);
      double getWidth() const;
      double getLength() const;
      double getArea() const;
};
```

UML Access Specification Notation

- In UML you indicate a private member with a minus (-) and a public member with a plus(+).



UML Data Type Notation

- To indicate the data type of a member variable, place a colon followed by the name of the data type after the name of the variable.

```
- width : double
- length : double
```

UML Parameter Type Notation

- To indicate the data type of a function's parameter variable, place a colon followed by the name of the data type after the name of the variable.

```
+ setWidth(w : double)
```

UML Function Return Type Notation

- To indicate the data type of a function's return value, place a colon followed by the name of the data type after the function's parameter list.

```
+ setWidth(w : double) : void
```