BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

Programming and Computer Applications-2

Introduction to Classes

Instructor : PhD, Associate Professor Leyla Muradkhanli

Procedural and Object-Oriented Programming

- **Procedural programming** focuses on the process/actions that occur in a program

- **Object-Oriented programming** is based on the data and the functions that operate on it.

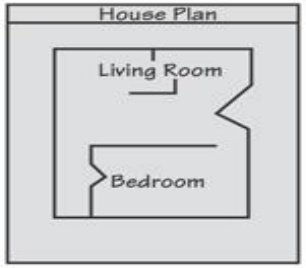
Object-Oriented Programming Terminology

- **class:** like a `struct` (allows bundling of related variables), but variables and functions in the class can have different properties than in a `struct`
- **object**: an instance of a `class`, in the same way that a variable can be an instance of a `struct`

Classes and Objects

- A Class is like a blueprint and objects are like houses built from the blueprint

Blueprint that describes a house.



Instances of the house described by the blueprint.

Object-Oriented Programming Terminology

- **attributes:** members of a class

- **methods** or **behaviors:** member functions of a class

Introduction to Classes

- Objects are created from a **`class`**
- Format:

```
class ClassName
{
        declaration;
        declaration;
};
```

Class Example

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

Access Specifiers

- Used to control access to members of the class
- **`public:`** can be accessed by functions outside of the class
- **`private:`** can only be called by or accessed by functions that are members of the class

Class Example

Private Members

Public Members

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

More on Access Specifiers

- Can be listed in any order in a class
- Can appear multiple times in a class
- If not specified, the default is `private`

Using `const` With Member Functions

- `const` appearing after the parentheses in a member function declaration specifies that the function will not change any data in the calling object.

```
double getWidth() const;
double getLength() const;
double getArea() const;
```

Defining a Member Function

- When defining a member function:
  - Put prototype in class declaration
  - Define function using class name and scope resolution operator `(::)`

```
int Rectangle::setWidth(double w)
{
    width = w;
}
```

Accessors and Mutators

- **Mutator:** a member function that stores a value in a private member variable, or changes its value in some way

- **Accessor:** function that retrieves a value from a private member variable. Accessors do not change an object's data, so they should be marked `const`.

Defining an Instance of a Class

- An object is an instance of a class
- Defined like structure variables:

```
Rectangle r;
```

- Access members using dot operator:

```
r.setWidth(5.2);
cout << r.getWidth();
```

- Compiler error if attempt to access **`private`** member using dot operator

```
// This program demonstrates a simple class.
#include <iostream>
using namespace std;

// Rectangle class declaration.
class Rectangle
{
   private:
      double width;
      double length;
   public:
      void setWidth(double);
      void setLength(double);
      double getWidth() const;
      double getLength() const;
      double getArea() const;
};
//*************************************************
// setWidth assigns a value to the width member.   *
//*************************************************
void Rectangle::setWidth(double w)
{
   width = w;
}
```

```
//*********************************************
// setLength assigns a value to the length member. *
//*********************************************

void Rectangle::setLength(double len)
{
   length = len;
}
//*********************************************
// getWidth returns the value in the width member. *
//*********************************************

double Rectangle::getWidth() const
{
   return width;
}
//**********************************************
// getLength returns the value in the length member. *
//**********************************************
double Rectangle::getLength() const
{
   return length;
}
```

```
//*************************************************
// getArea returns the product of width times length. *
//*************************************************

double Rectangle::getArea() const
{
   return width * length;
}
//*************************************************
// Function main                                   *
//*************************************************
int main()
{
   Rectangle r;     // Define an instance of the Rectangle class
   double rectWidth;  // Local variable for width
   double rectLength; // Local variable for length

   // Get the rectangle's width and length from the user.
   cout << "This program will calculate the area of a\n";
   cout << "rectangle. What is the width? ";
   cin >> rectWidth;
   cout << "What is the length? ";
   cin >> rectLength;
```

```
// Store the width and length of the rectangle
   // in the box object.
   r.setWidth(rectWidth);
   r.setLength(rectLength);

   // Display the rectangle's data.
   cout << "Here is the rectangle's data:\n";
   cout << "Width: " << r.getWidth() << endl;
   cout << "Length: " << r.getLength() << endl;
   cout << "Area: " << r.getArea() << endl;
   return 0;
}
```

Program Output

```
This program will calculate the area of a
rectangle. What is the width? 10 [Enter]
What is the length? 5 [Enter]
Here is the rectangle's data:
Width: 10
Length: 5
Area: 50
```

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; }
```

Constructors

A constructor is a member function of a class which initializes objects of a class.
In C++,Constructor is automatically called when object(instance of class) create.
It is special member function of the class.

Constructors

- Member function that is automatically called when an object is created
- Purpose is to construct an object
- Constructor function name is class name
- Has no return type

Default Constructors

- A default constructor is a constructor that takes no arguments.

- If you write a class with no constructor at all, C++ will write a default constructor for you, one that does nothing.

- A simple instantiation of a class (with no arguments) calls the default constructor:

```
Rectangle r;
```

Passing Arguments to Constructors

- To create a constructor that takes arguments:
  - indicate parameters in prototype:

```
Rectangle(double, double);
```

  - Use parameters in the definition:

```
Rectangle::Rectangle(double w, double len)
{
    width = w;
    length = len;
}
```

Passing Arguments to Constructors

- You can pass arguments to the constructor when you create an object:

```
Rectangle r(10, 5);
```

Example

```
// example: class constructor
#include <iostream>
using namespace std;
class Rectangle {
    int width, height;
  public:
    Rectangle (int,int);
    int area () {return (width*height);}
};
Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}
```

Example

```
int main () {
  Rectangle r1(3,4);
  Rectangle r2(5,6);
  cout << "r1 area: " << r1.area() << endl;
  cout << "r2 area: " << r2.area() << endl;
  return 0;
}
```

```
rect area: 12
rectb area: 30
```

More About Default Constructors

- If all of a constructor's parameters have default arguments, then it is a default constructor. For example:

```
Rectangle(double = 0, double = 0);
```

- Creating an object and passing no arguments will cause this constructor to execute:

```
Rectangle r;
```

Destructors

- Member function automatically called when an object is destroyed
- Destructor name is ~classname, *e.g.*, **`~Rectangle`**
- Has no return type; takes no arguments
- Only one destructor per class, *i.e.*, it cannot be overloaded
- If constructor allocates dynamic memory, destructor should release it

Overloading Constructors

- A class can have more than one constructor

- Overloaded constructors in a class must have different parameter lists:

```
Rectangle();
Rectangle(double);
Rectangle(double, double);
```

Overloading Constructors

```
// overloading class constructors
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
  public:
    Rectangle ();
    Rectangle (int,int);
    int area (void) {return (width*height);}
};
```

Overloading Constructors

```
Rectangle::Rectangle () {
  width = 5;
  height = 5;
}
Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}
```

Overloading Constructors

```
int main () {
  Rectangle r1(3,4);
  Rectangle r2;
  cout << "r1 area: " << r1.area() << endl;
  cout << "r2 area: " << r2.area() << endl;
  return 0;
}
```

rect area: 12

rectb area: 25

Only One Default Constructor and One Destructor

- Do not provide more than one default constructor for a class: one that takes no arguments and one that has default arguments for all parameters

```
Square();
Square(int = 0);  // will not compile
```

- Since a destructor takes no arguments, there can only be one destructor for a class

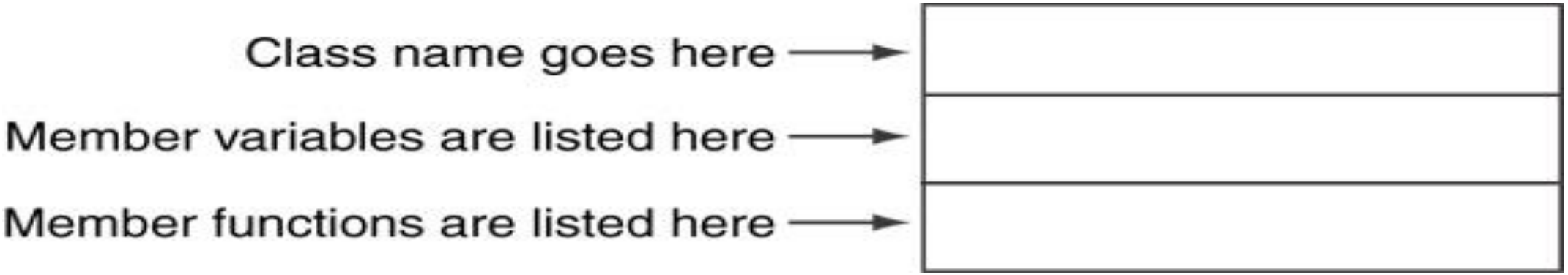
The Unified Modeling Language

- *UML* stands for *Unified Modeling Language*.

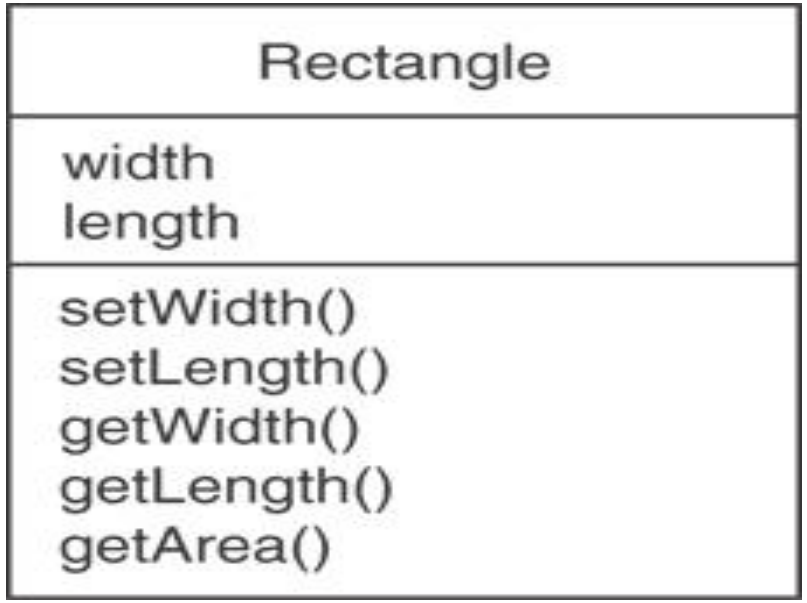
- The UML provides a set of standard diagrams for graphically depicting object-oriented systems

UML Class Diagram

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



Example: A Rectangle Class



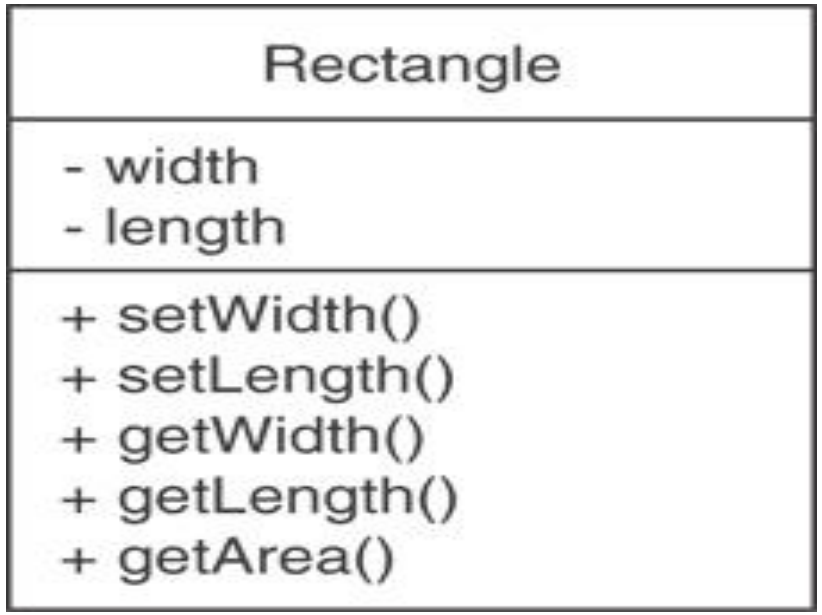

```
class Rectangle
{
    private:
        double width;
        double length;
    public:
        bool setWidth(double);
        bool setLength(double);
        double getWidth() const;
        double getLength() const;
        double getArea() const;
};
```

UML Access Specification Notation

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

These member variables are private.

These member functions are public.



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
```

The Rectangle Class

Rectangle
- width : double - length : double
+ setWidth(w : double) : bool + setLength(len : double) : bool + getWidth() : double + getLength() : double + getArea() : double