Object Oriented Programming (CS1143)

Week 2

Department of Computer Science

Capital University of Science and Technology (CUST)

Outline

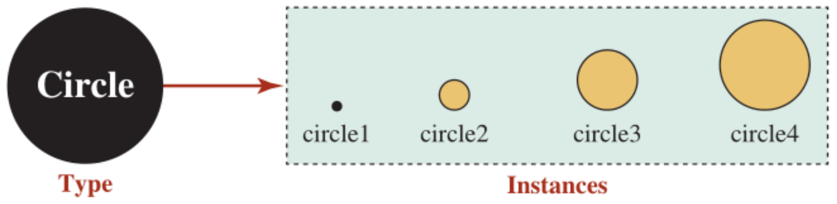
- Introduction to Classes and Objects
- Data Members
- Member Functions
- Constant Member Functions

What is OOP?

- Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects.

Types and Instances

- A type is a concept from which an instance is created.
- In other words, a type is an abstraction; an instance of that type is a concrete entity.
  - The word circle is a type. We can draw circles with different radii as instances of that type.
  - Student is a type. All of you are its instances
- The relationship between a type and its instances is a one-to-many relation. We can have many instances from one single type.



Attributes and Behaviors

- Any instance we encounter in this world has a set of **attributes** and a set of **behaviors**.

Attributes

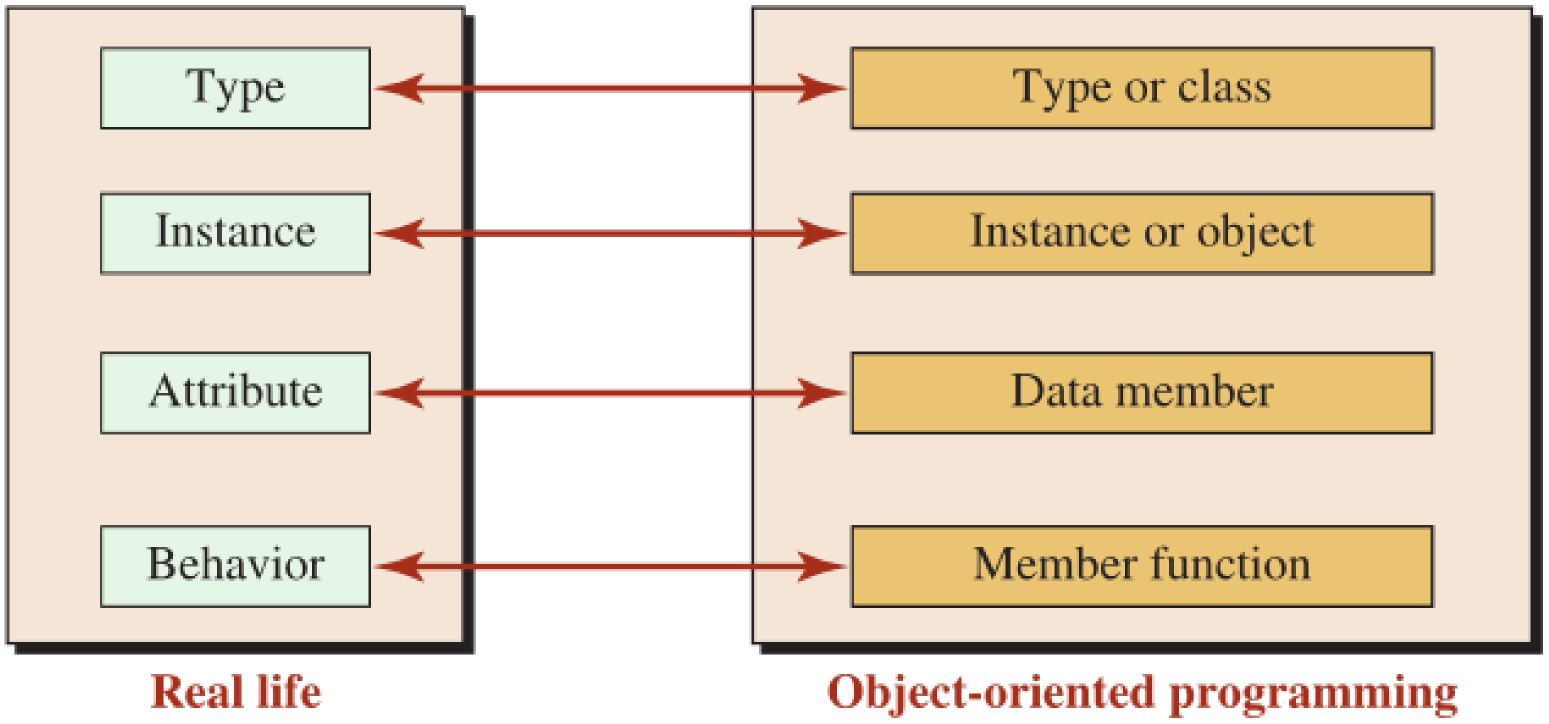
- An attribute is a characteristic of an instance that we are interested in.
- For example, if the instance is a student in a university, we may be interested only in the student’s name, year, courses taken, and grades.
- If the instance is a circle, we may be interested only in its radius, perimeter, and area.

Behaviors

- A behavior is an operation that we assume an instance can perform on itself.
- For example, if an instance is a student, we assume that he can give his name, courses taken. etc.,
- If an instance is a circle, we assume that it can give its radius, its perimeter, and its area.

Classes and Objects

- In C++, a user defined type can be created using a construct named class.
- An instance of a class is referred to as an object.
- Attributes and behaviors of an object are represented as data members and member functions


Type
Type or class
Instance
Instance or object
Attribute
Data member
Behavior
Member function
Real life
Object-oriented programming

Data Members

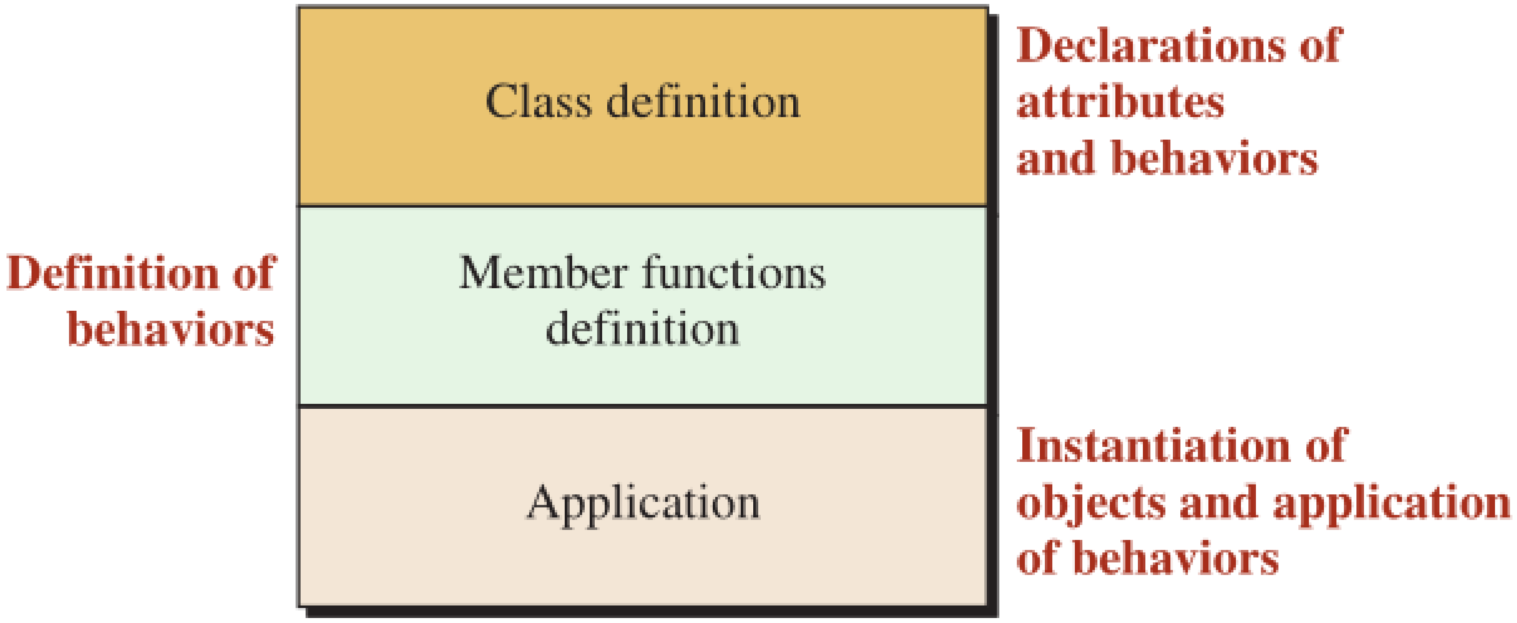
- A data member of an object is a variable whose value represents an attribute.
    - For example, the radius of a circle object can be represented by the value of a variable of type double.
- Some attributes of an instance are not independent; they may depend on values of other attributes.
    - For example, the perimeter and the area of a circle may be two attributes, but we do not represent them as data members because both depend on the value of the radius.

Member Functions

- A member function in object-oriented programming is a function that simulates one of the behaviors of an object.
  - For example, we can write a function that allows a circle to give its radius, its area, and its perimeter.
  - We can also write a function that a circle can use to set its radius.

Object Oriented Programming

- To write object-oriented programs, we need to create a class, as a type, and then instantiate objects as instances of that type.
- We need three sections:
  - the class definition,
  - the member function definition,
  - the application (which uses the objects created from the class).
- In the **class definition section**, we declare the data members and member functions.
- In the **member function definition section**, we define all member functions.
- In the **application section**, we instantiate objects and apply the member function to those objects

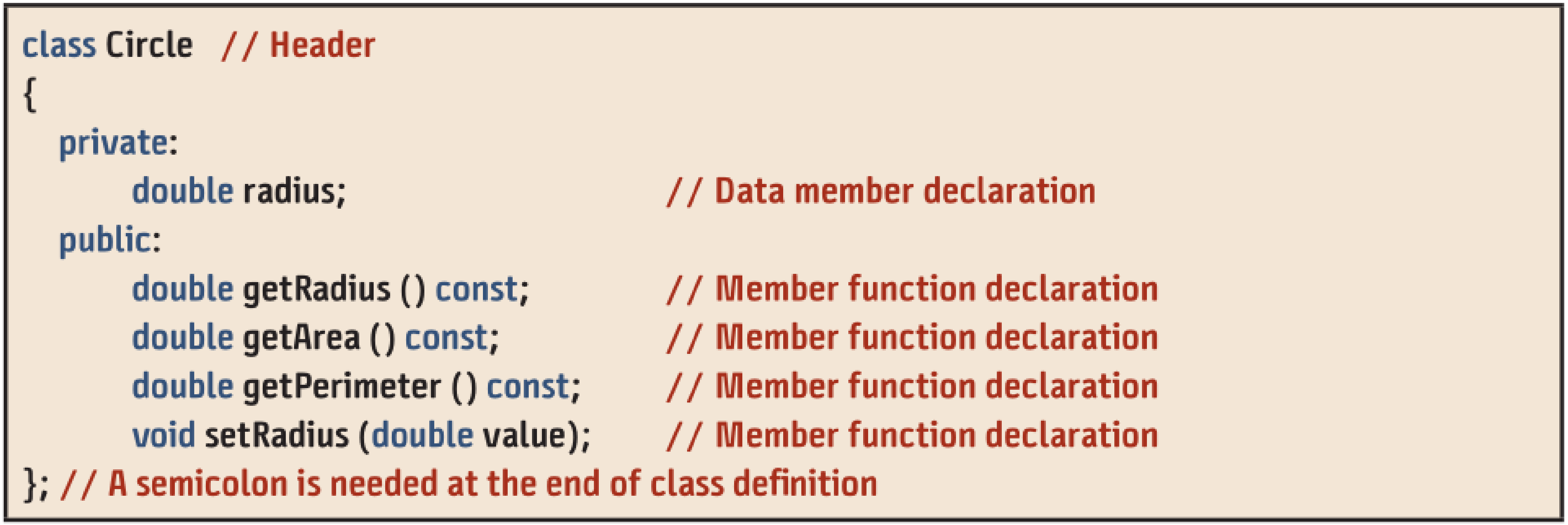

Class definition
Member functions definition
Application
Declarations of attributes and behaviors
Definition of behaviors
Instantiation of objects and application of behaviors

Class Definition

Class Definition

- A class definition is made of three parts:
  - a header
  - a body
  - a semicolon.
- A class header is made of the reserved word class followed by the name given by the designer.
  - We follow a convention that recommends that class names start with an uppercase letter to distinguish them from library classes, which start with lowercase letters.
- The class body is a block (starting with an opening brace and ending with a closing brace) that holds the declaration of data members and member functions in the block.
- The third element of a class declaration is a semicolon that terminates the definition.

A Circle class



```
class Circle   // Header
{
   private:
         double radius;                       // Data member declaration
   public:
         double getRadius () const;           // Member function declaration
         double getArea () const;             // Member function declaration
         double getPerimeter () const;        // Member function declaration
         void setRadius (double value);       // Member function declaration
}; // A semicolon is needed at the end of class definition
```

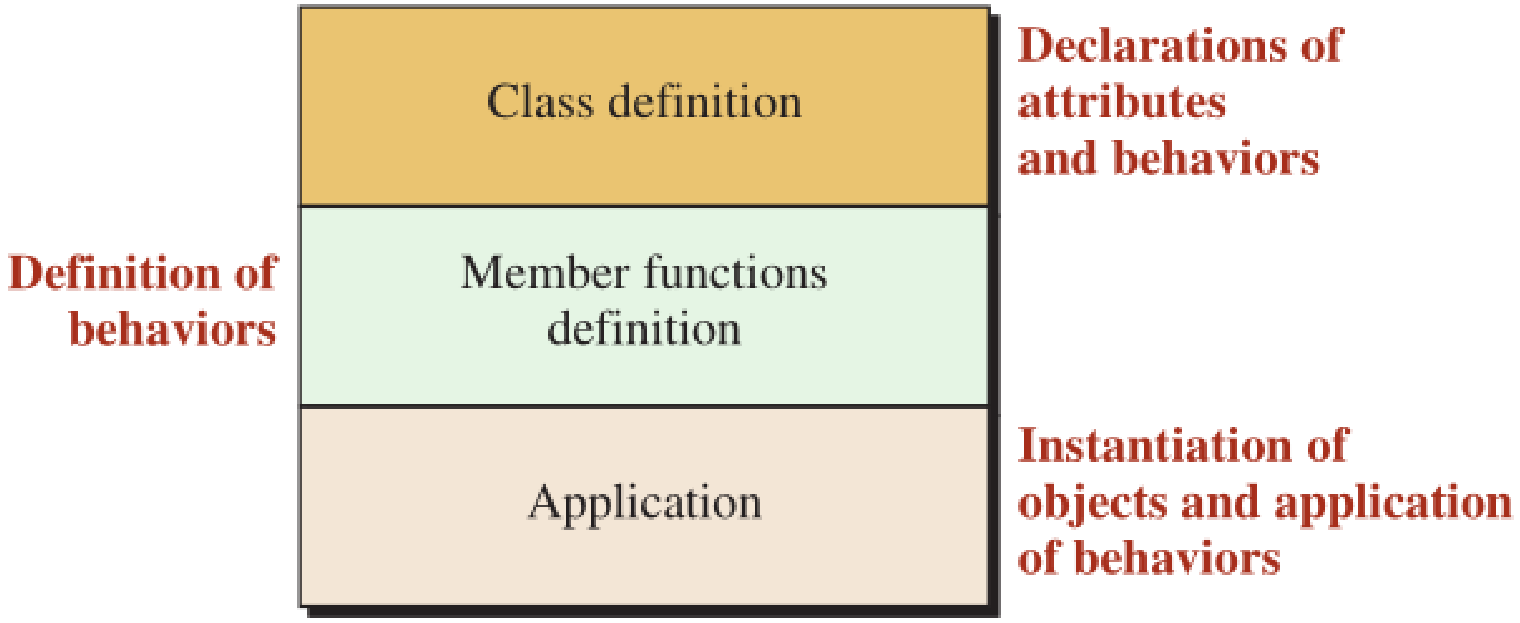
Declaring Data Members

- The data members of a class simulate the attributes of the objects that are instantiated from the class.
- Some of the attributes are dependent on the others and can be calculated given the other attributes. In such a case, **always choose the most basic attribute**
- For example, in our circle class, we have three attributes: radius, area, and perimeter. We have chosen the radius to be a data member.

Data members in a class must not depend on each other.

Declaring Member Functions

- The second part of the class definition declares the member functions of the class; that is, it declares all functions that are used to simulate the behavior of the class.
- Some functions have the const qualifier at the end and some do not.
- **Constant member functions** are
  those functions which are denied permission to change the values of the data members of their class.


Class definition
Declarations of attributes and behaviors
Member functions definition
Definition of behaviors
Application
Instantiation of objects and application of behaviors

Member Functions Definition

Member Functions Definition

- The declaration of a member function gives its prototype; each member function also needs a definition.
- The definition of each member function is similar to the definition that we have used previously but there are two differences.
    - The first is the qualifier (const) that is applied to some member function.
    - The second is the name of the function that must be qualified with the name of the class.
- In C++, we need to mention the class name first followed by a class scope (::) symbol while writing the function definition.

```
double Circle :: getRadius () const
{
   return radius;
}
double Circle :: getArea () const
{
   const double PI  = 3.14;
   return (PI * radius * radius);
}
double Circle :: getPerimeter () const
{
   const double PI  = 3.14;
   return (2 * PI * radius);
}
void Circle :: setRadius (double value)
{
   radius = value;
}
```

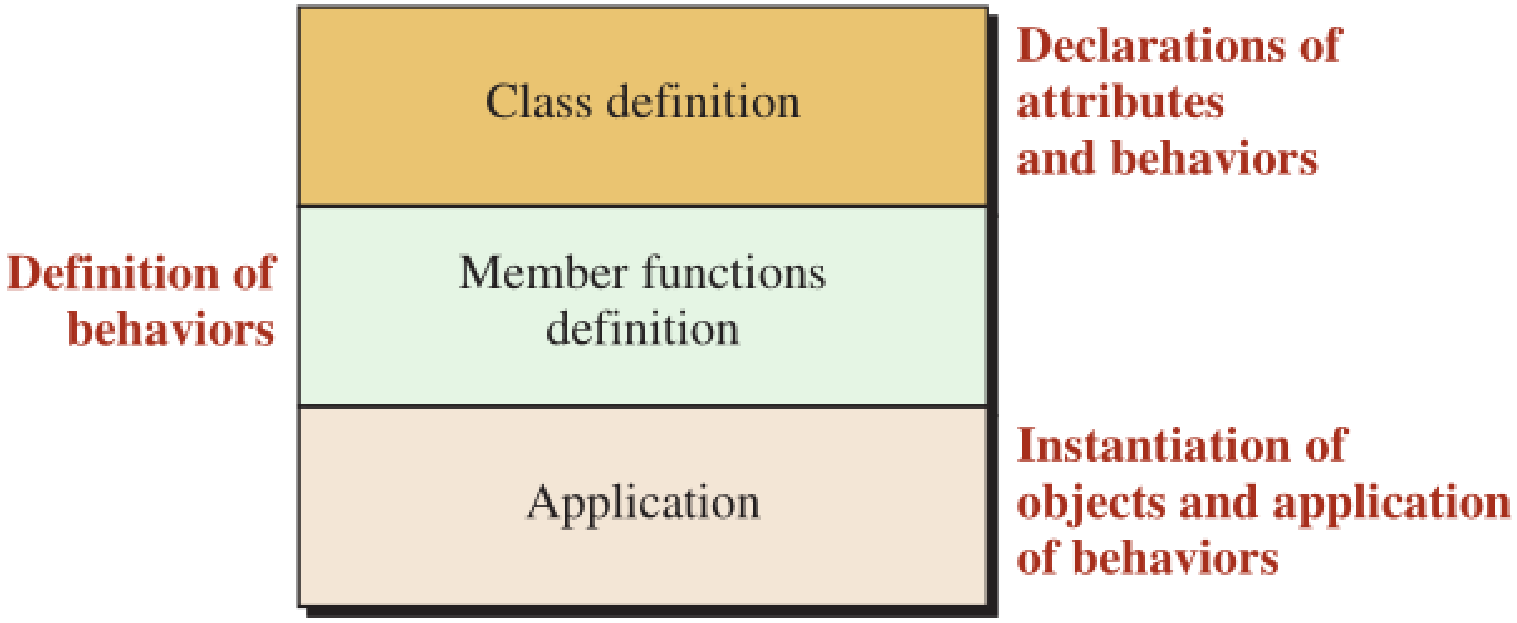
```
class Circle
{
   • • •
    public:
        double getRadius () const;
       • • •
}
```

No scope resolution in the function declaration

```
double Circle :: getRadius () const;
{
   • • •
}

• • •
```

Scope resolution in in the function definition


Class definition
Member functions definition
Application
Declarations of attributes and behaviors
Definition of behaviors
Instantiation of objects and application of behaviors

Application

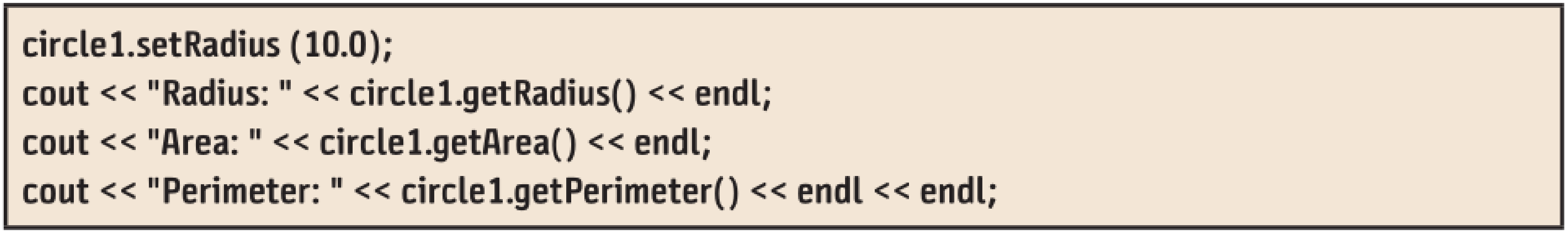
Application

- We need an application section to instantiate objects of the class and apply the member functions on those objects.
- We can instantiate an object of the class in the following way

```
Circle circle1;
```

Applying operations on objects

- We can apply operations on objects in the following way.

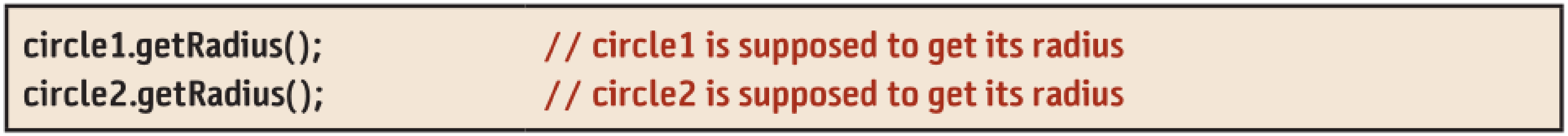


```
circle1.setRadius (10.0);
cout << "Radius: " << circle1.getRadius() << endl;
cout << "Area: " << circle1.getArea() << endl;
cout << "Perimeter: " << circle1.getPerimeter() << endl << endl;
```

- The first line sets the radius of circle1. The second line gets the value of the circle’s radius.
- The next two lines calculate and print the area and perimeter of the object named circle1

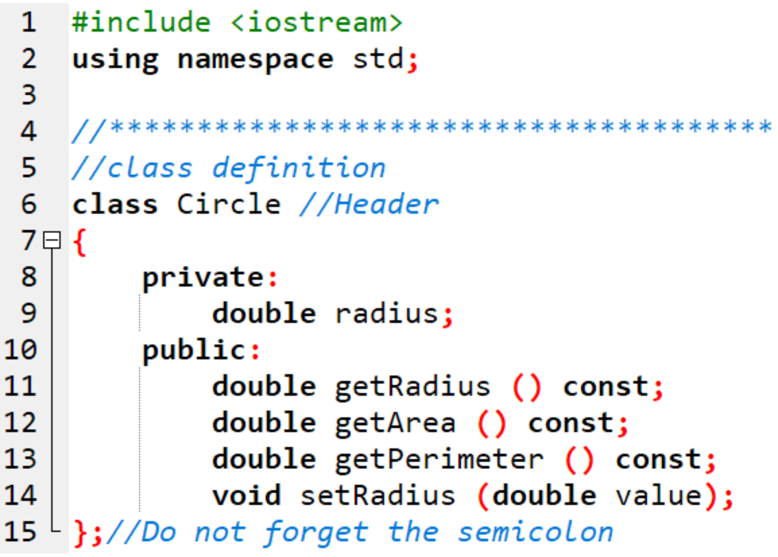
Member Selection

- We are using a dot between the object name and the member function that is supposed to operate on the object.
- This is called the member select operator
- In other words, we can apply the same function on different objects using this operator



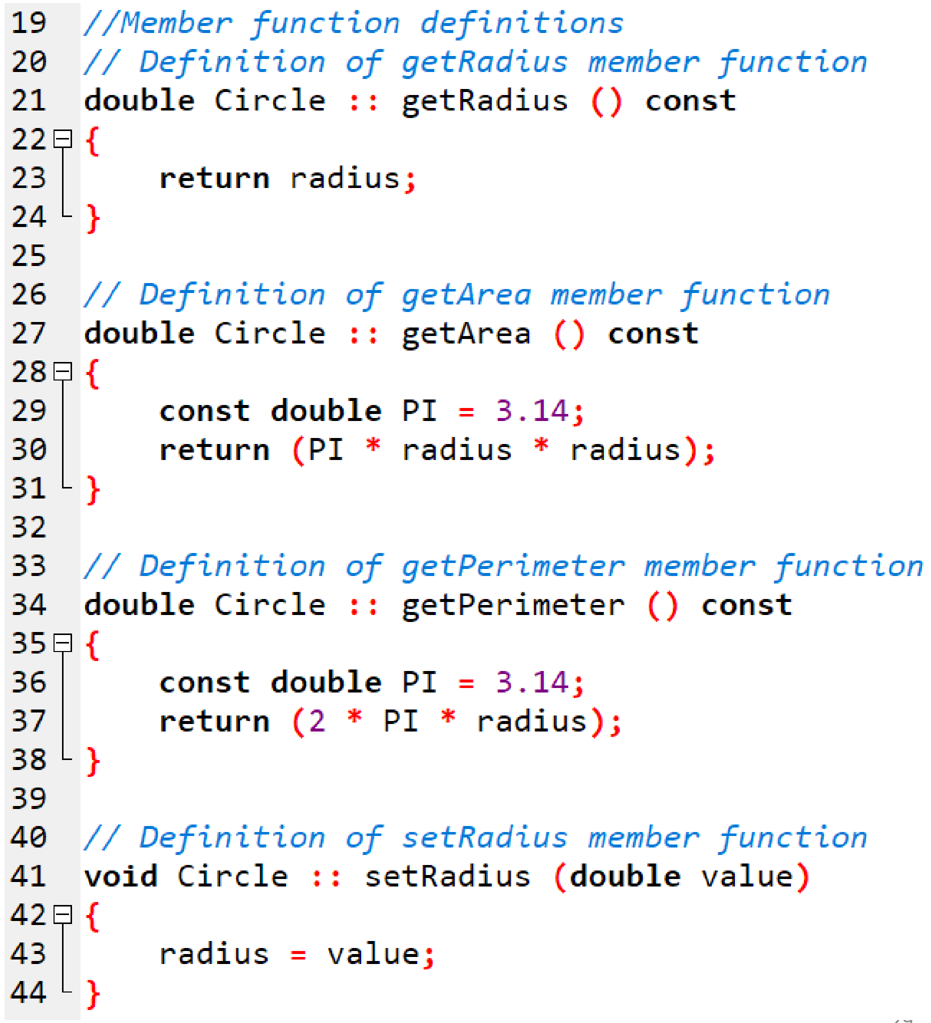
```
circle1.getRadius();            // circle1 is supposed to get its radius
circle2.getRadius();            // circle2 is supposed to get its radius
```

Complete Program



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

//************************************
//class definition
class Circle //Header
{
    private:
        double radius;
    public:
        double getRadius () const;
        double getArea () const;
        double getPerimeter () const;
        void setRadius (double value);
};//Do not forget the semicolon
```

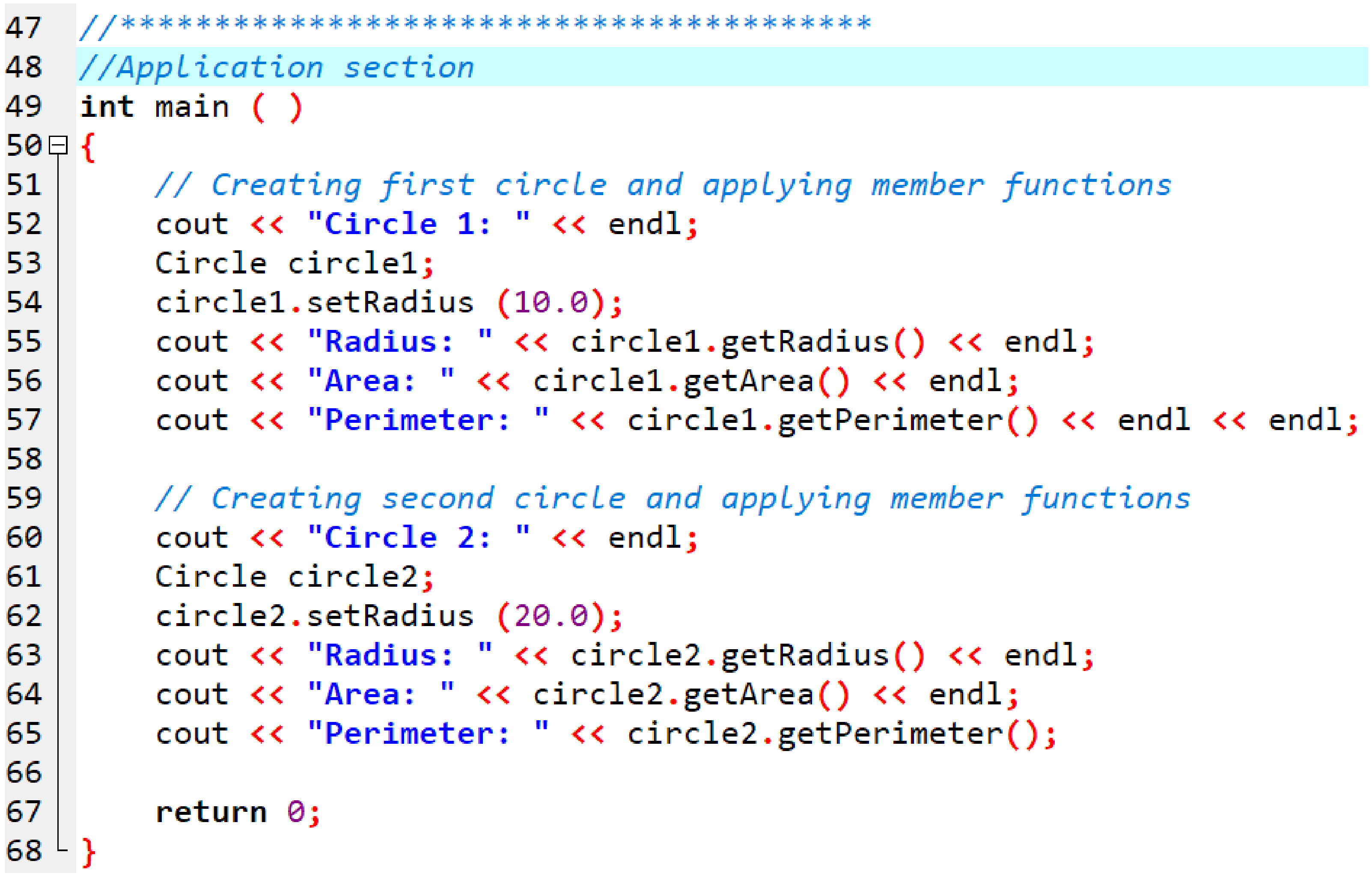


```
//Member function definitions
// Definition of getRadius member function
double Circle :: getRadius () const
{
    return radius;
}

// Definition of getArea member function
double Circle :: getArea () const
{
    const double PI = 3.14;
    return (PI * radius * radius);
}

// Definition of getPerimeter member function
double Circle :: getPerimeter () const
{
    const double PI = 3.14;
    return (2 * PI * radius);
}

// Definition of setRadius member function
void Circle :: setRadius (double value)
{
    radius = value;
}
```



```cpp
//****************************************
//Application section
int main ( )
{
    // Creating first circle and applying member functions
    cout << "Circle 1: " << endl;
    Circle circle1;
    circle1.setRadius (10.0);
    cout << "Radius: " << circle1.getRadius() << endl;
    cout << "Area: " << circle1.getArea() << endl;
    cout << "Perimeter: " << circle1.getPerimeter() << endl << endl;

    // Creating second circle and applying member functions
    cout << "Circle 2: " << endl;
    Circle circle2;
    circle2.setRadius (20.0);
    cout << "Radius: " << circle2.getRadius() << endl;
    cout << "Area: " << circle2.getArea() << endl;
    cout << "Perimeter: " << circle2.getPerimeter();

    return 0;
}
```

Separating Class Definition from Implementation

Benefits

- It hides implementation from definition. You can feel free to change the implementation. The client program that uses the class does not need to change as long as the definition is not changed.
- As a software vendor, you can just provide the customer with the header file and class object code without revealing the source code for implementing the class. This protects the software vendor’s intellectual property.

How to do it?

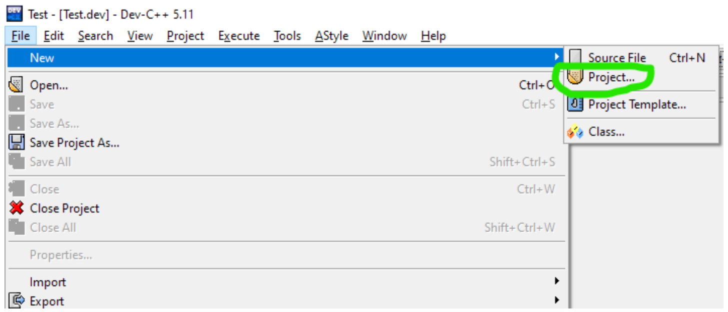
- Class Definition must be in a file “ClassName.h”
- Member Function Definition must be in a file “ClassName.cpp”
  - Include the .h file in this .cpp file using

    #include "Circle.h"
- Application can be in a third file.
  - Include the .h file in this .cpp file using

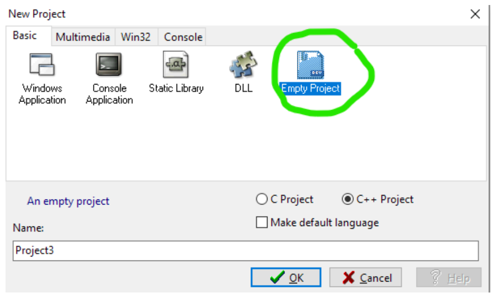
    #include "Circle.h"

Making project in Dev-C++

1. Make a new folder on Desktop. Rename it to “MyProject”.
2. Open Dec-C++.
3. Make new project.



4. Select Empty project



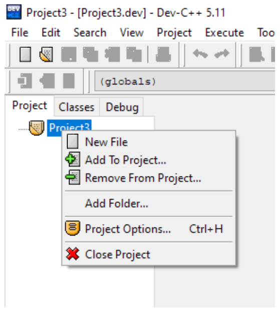
An empty project

C Project [✓ C++ Project]

☐ Make default language

Name: Project3

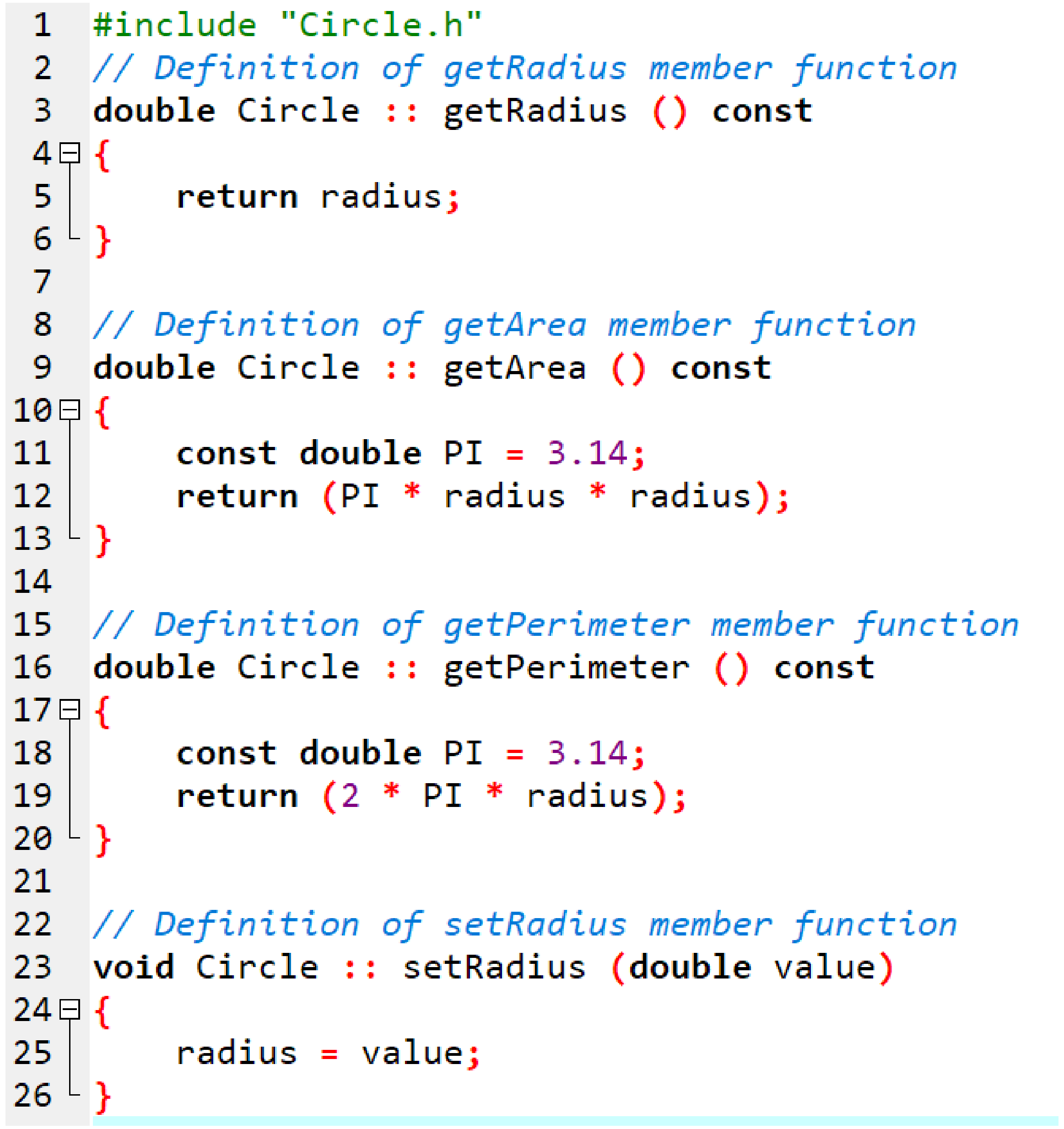
5. Save it to the folder created in step 1.
6. Add new files to the project.



Circle.h

```
class Circle
{
	private:
		double radius;
	public:
		double getRadius () const;
		double getArea () const;
		double getPerimeter () const;
		void setRadius (double value);
};
```

Circle.cpp



```cpp
#include "Circle.h"
// Definition of getRadius member function
double Circle :: getRadius () const
{
    return radius;
}

// Definition of getArea member function
double Circle :: getArea () const
{
    const double PI = 3.14;
    return (PI * radius * radius);
}

// Definition of getPerimeter member function
double Circle :: getPerimeter () const
{
    const double PI = 3.14;
    return (2 * PI * radius);
}

// Definition of setRadius member function
void Circle :: setRadius (double value)
{
    radius = value;
}
```

main.cpp

```cpp
#include "Circle.h"

#include <iostream>
using namespace std;

int main ( )
{
    // Creating first circle and applying member functions
    cout << "Circle 1: " << endl;
    Circle circle1;
    circle1.setRadius (10.0);
    cout << "Radius: " << circle1.getRadius() << endl;
    cout << "Area: " << circle1.getArea() << endl;
    cout << "Perimeter: " << circle1.getPerimeter() << endl << endl;

    // Creating second circle and applying member functions
    cout << "Circle 2: " << endl;
    Circle circle2;
    circle2.setRadius (20.0);
    cout << "Radius: " << circle2.getRadius() << endl;
    cout << "Area: " << circle2.getArea() << endl;
    cout << "Perimeter: " << circle2.getPerimeter();

    return 0;
}
```

That is all for Week 2