Chapter 9: Objects and Classes

Sections 9.1}9.6, 9.9

Textbooks: Y. Daniel Liang, Introduction to Programming with C++, 3rd Edition

These slides were adapted by Prof. Gheith Abandah from the Computer Engineering Department of the University of Jordan for the Course: Computer Skills for Engineers (0907101)

Outline

- Introduction
- Defining Classes for Objects
- Example: Defining Classes and Creating Objects
- Constructors
- Constructing and Using Objects
- Separating Class Definition from Implementation
- Data Field Encapsulation

Introduction

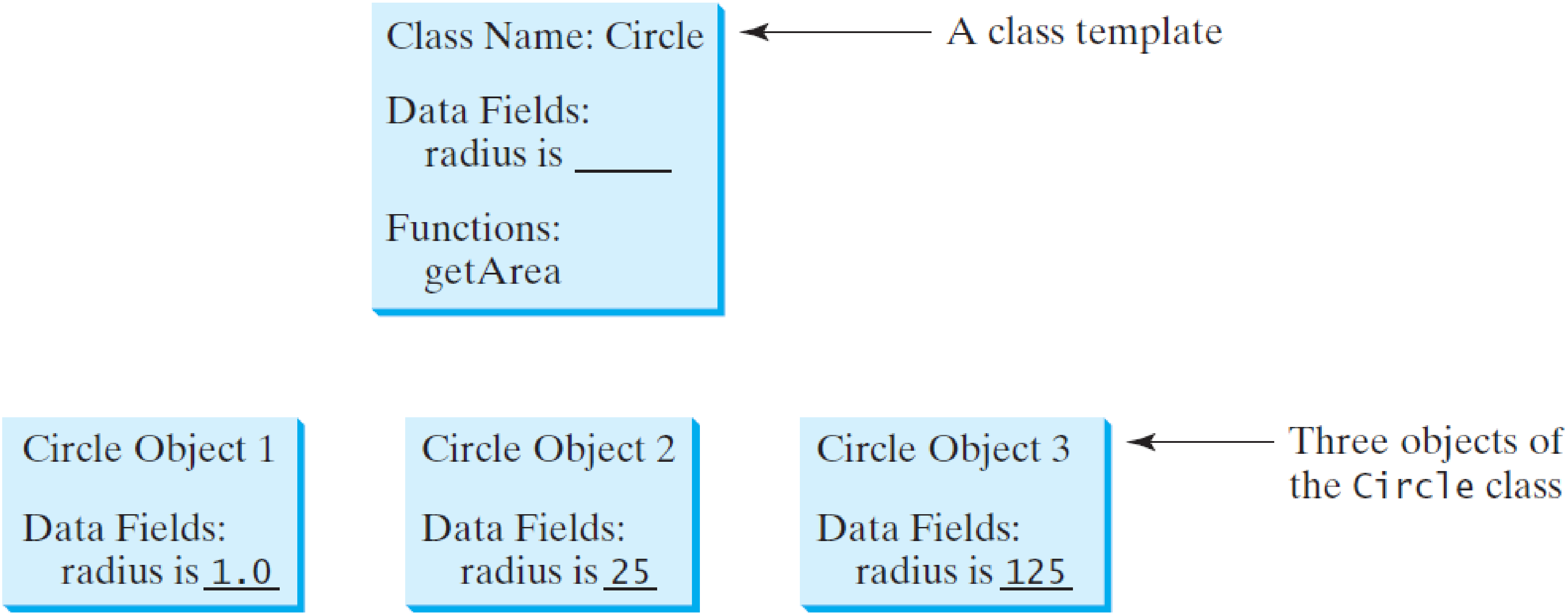
- *Object-oriented programming* (OOP) involves programming using objects.
- An *object* represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.
- An object has a unique identity, state, and behaviors.
- The *state* of an object consists of a set of *data fields* (also known as *properties*) with their current values.

Outline

- Introduction
- Defining Classes for Objects
- Example: Defining Classes and Creating Objects
- Constructors
- Constructing and Using Objects
- Separating Class Definition from Implementation
- Data Field Encapsulation

Classes and Objects

A class defines the properties and behaviors for objects..



Class Name: Circle

Data Fields:
radius is _____

Functions:
getArea

A class template

Circle Object 1

Data Fields:
radius is 1.0

Circle Object 2

Data Fields:
radius is 25

Circle Object 3

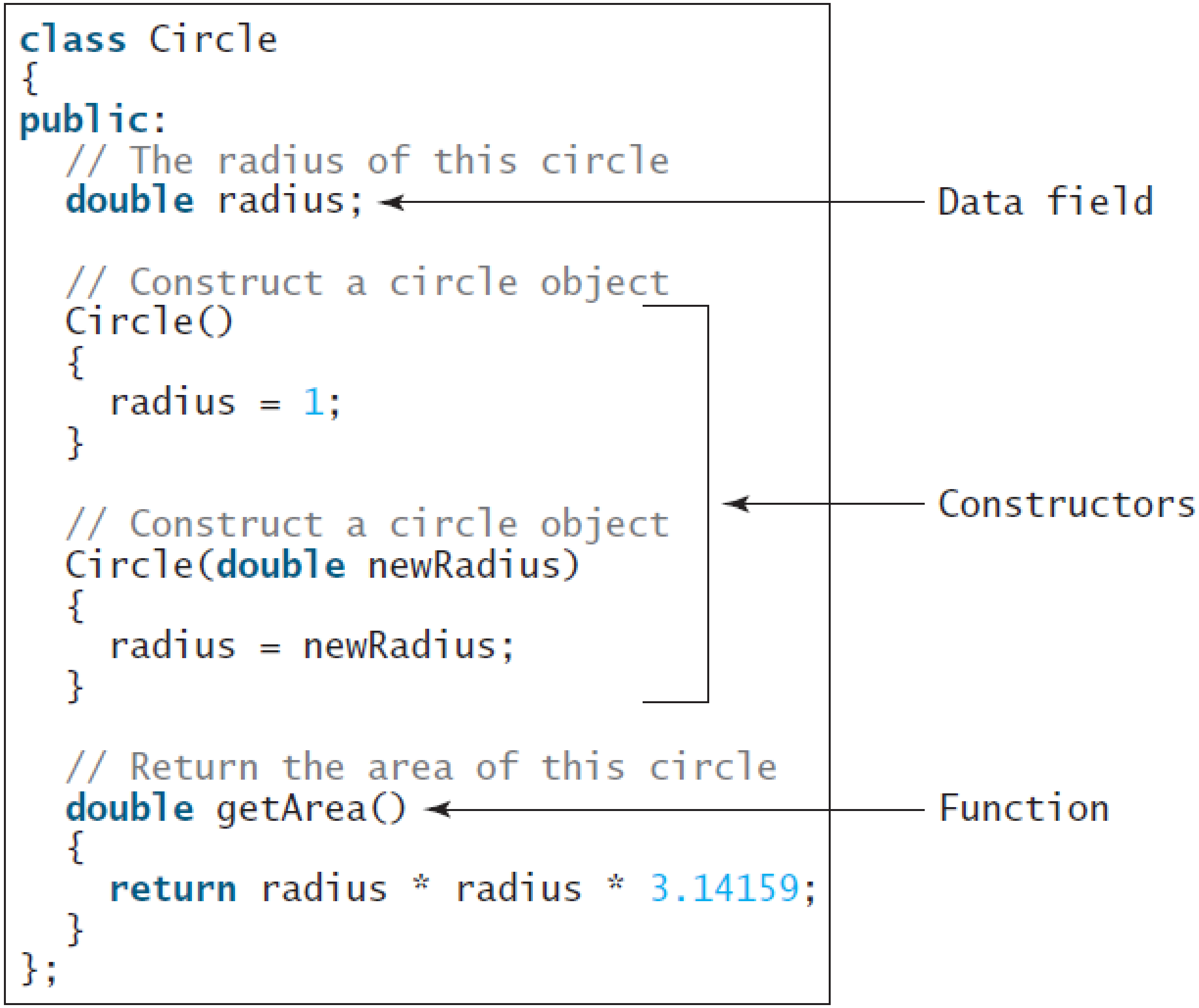
Data Fields:
radius is 125

Three objects of
the `Circle` class

Classes

- *Classes* are constructs that define objects of the same type.
- A class uses *variables* to define data fields and *functions* to define behaviors.
- Additionally, a class provides a special type of functions, known as *constructors*, which are invoked to construct objects from the class.

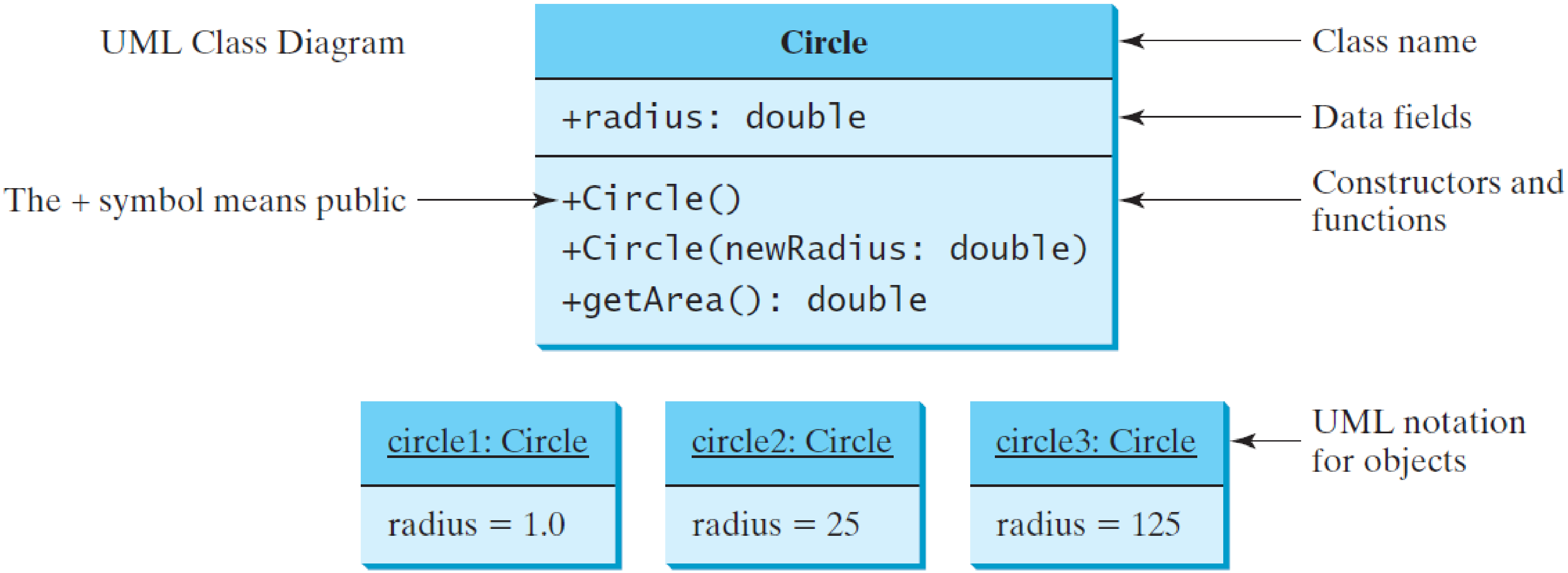
Example of the class for `Circle` objects



```
class Circle
{
public:
  // The radius of this circle
  double radius;

  // Construct a circle object
  Circle()
  {
    radius = 1;
  }

  // Construct a circle object
  Circle(double newRadius)
  {
    radius = newRadius;
  }

  // Return the area of this circle
  double getArea()
  {
    return radius * radius * 3.14159;
  }
};
```

Data field
Constructors
Function

UML Class Diagram



UML Class Diagram

Circle

+radius: double

+Circle()

+Circle(newRadius: double)

+getArea(): double

Class name

Data fields

Constructors and functions

The + symbol means public

circle1: Circle

radius = 1.0

circle2: Circle

radius = 25

circle3: Circle

radius = 125

UML notation for objects

class Replaces struct

- The C language has the **struct** type for representing records.
- For example, you may define a **struct** type for representing students as shown in (a).
- C++ **class** allows functions in addition to data fields. **class** replaces **struct**, as in (b)

```
struct Student
{
  int id;
  char firstName[30];
  char mi;
  char lastName[30];
};
```

(a)

```
class Student
{
public:
  int id;
  char firstName[30];
  char mi;
  char lastName[30];
};
```

(b)

Outline

- Introduction
- Defining Classes for Objects
- Example: Defining Classes and Creating Objects
- Constructors
- Constructing and Using Objects
- Separating Class Definition from Implementation
- Data Field Encapsulation

A Simple Circle Class

- Objective: Demonstrate creating objects, accessing data, and using functions.

TestCircle

Run

TestCircle.cpp 1/2

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

class Circle
{
public:
    // The radius of this
circle
    double radius;

    // Construct a default
object
    Circle()
    {
        radius = 1;
    }

    // Construct a circle
object
    Circle(double newRadius)
    {
        radius = newRadius;
    }
```

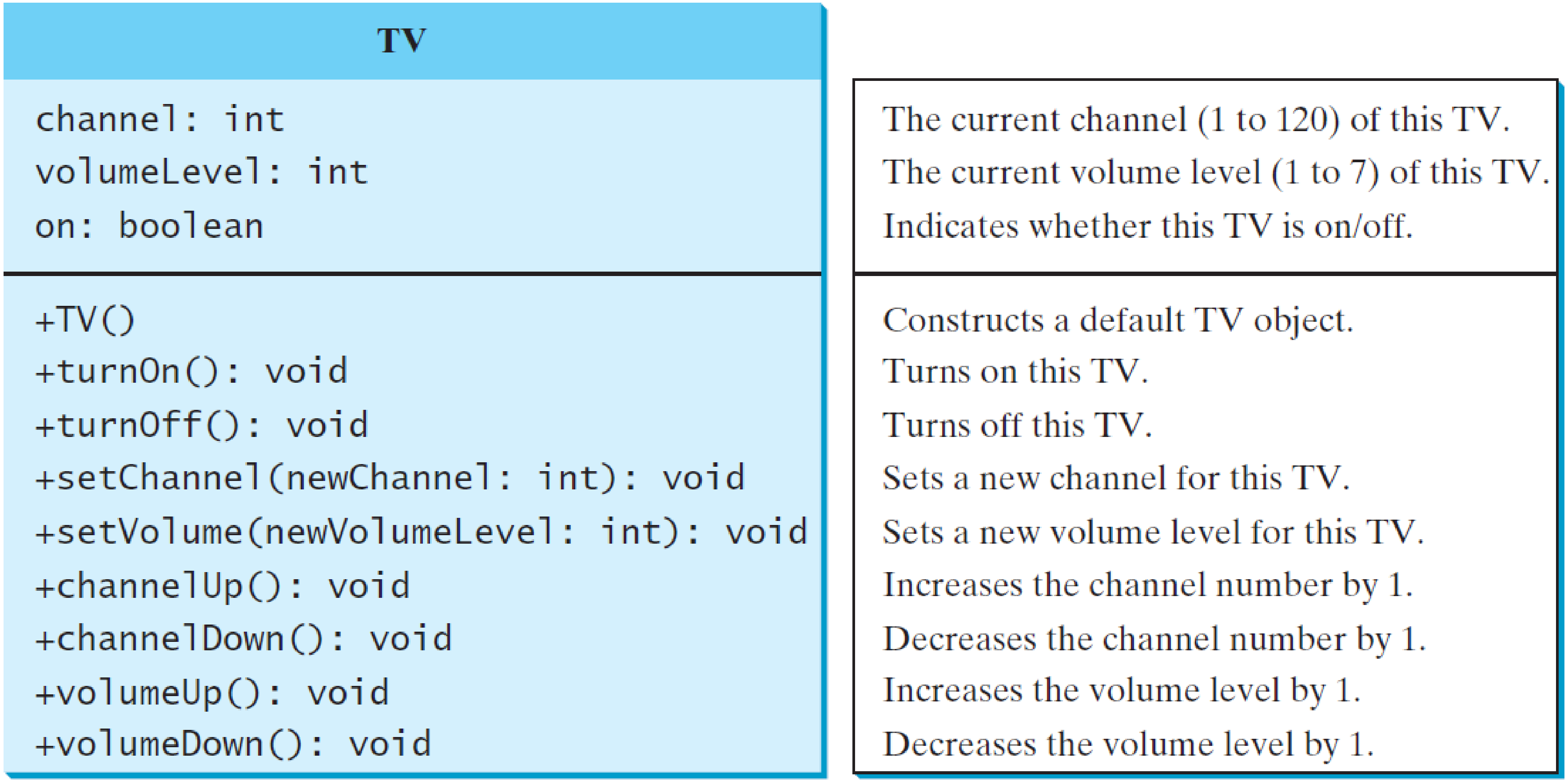
```
    // Return the area of this circle
    double getArea()
    {
        return radius * radius *
3.14159;
    }

    // Return the perimeter of this
circle
    double getPermeter()
    {
        return 2 * radius * 3.14159;
    }

    // Set new radius for this circle
    void setRadius(double newRadius)
    {
        radius = newRadius;
    }
}; // Must place a semicolon here
```

TestCircle.cpp 2/2

```cpp
int main()
{
    Circle circle1(1.0);
    Circle circle2(25);
    Circle circle3(125);

    cout << "The area of the circle of radius "
        << circle1.radius << " is " << circle1.getArea() << endl;
    cout << "The area of the circle of radius "
        << circle2.radius << " is " << circle2.getArea() << endl;
    cout << "The area of the circle of radius "
        << circle3.radius << " is " << circle3.getArea() << endl;

    // Modify circle radius
    circle2.radius = 100;
    cout << "The area of the circle of radius "
        << circle2.radius << " is " << circle2.getArea() << endl;

    return 0;
}
```

```
The area of the circle of radius 1 is 3.14159
The area of the circle of radius 25 is
1963.49
The area of the circle of radius 125 is
49087.3
```

Example: The TV class models TV sets



TV	
channel: int	The current channel (1 to 120) of this TV.
volumeLevel: int	The current volume level (1 to 7) of this TV.
on: boolean	Indicates whether this TV is on/off.
+TV()	Constructs a default TV object.
+turnOn(): void	Turns on this TV.
+turnOff(): void	Turns off this TV.
+setChannel(newChannel: int): void	Sets a new channel for this TV.
+setVolume(newVolumeLevel: int): void	Sets a new volume level for this TV.
+channelUp(): void	Increases the channel number by 1.
+channelDown(): void	Decreases the channel number by 1.
+volumeUp(): void	Increases the volume level by 1.
+volumeDown(): void	Decreases the volume level by 1.

TV Run

TV.cpp 1/4

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

class TV
{
public:
    int channel;
    int volumeLevel; // Default volume level is 1
    bool on; // By default TV is off

    TV()
    {
        channel = 1; // Default channel is 1
        volumeLevel = 1; // Default volume level is 1
        on = false; // By default TV is off
    }

    void turnOn()
    {
        on = true;
    }
```

TV.cpp 2/4

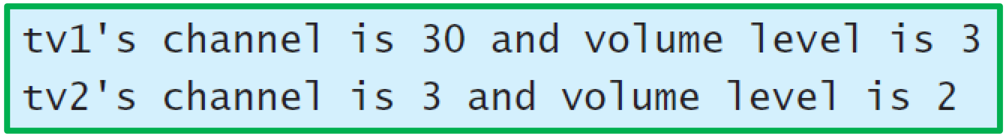
```
void turnOff()
{
    on = false;
}

void setChannel(int newChannel)
{
    if (on && newChannel >= 1 && newChannel <= 120)
        channel = newChannel;
}

void setVolume(int newVolumeLevel)
{
    if (on && newVolumeLevel >= 1 && newVolumeLevel <= 7)
        volumeLevel = newVolumeLevel;
}

void channelUp()
{
    if (on && channel < 120)
        channel++;
}
```

TV.cpp 3/4

```
    void channelDown()
    {
        if (on && channel > 1)
            channel--;
    }

    void volumeUp()
    {
        if (on && volumeLevel < 7)
            volumeLevel++;
    }

    void volumeDown()
    {
        if (on && volumeLevel > 1)
            volumeLevel--;
    }
};
```

TV.cpp 4/4

```cpp
int main()
{
    TV tv1;
    tv1.turnOn();
    tv1.setChannel(30);
    tv1.setVolume(3);

    TV tv2;
    tv2.turnOn();
    tv2.channelUp();
    tv2.channelUp();
    tv2.volumeUp();

    cout << "tv1's channel is " << tv1.channel
        << " and volume level is " << tv1.volumeLevel << endl;
    cout << "tv2's channel is " << tv2.channel
        << " and volume level is " << tv2.volumeLevel << endl;

    return 0;
}
```



```
tv1's channel is 30 and volume level is 3
tv2's channel is 3 and volume level is 2
```

Outline

- Introduction
- Defining Classes for Objects
- Example: Defining Classes and Creating Objects
- Constructors
- Constructing and Using Objects
- Separating Class Definition from Implementation
- Data Field Encapsulation

Constructors

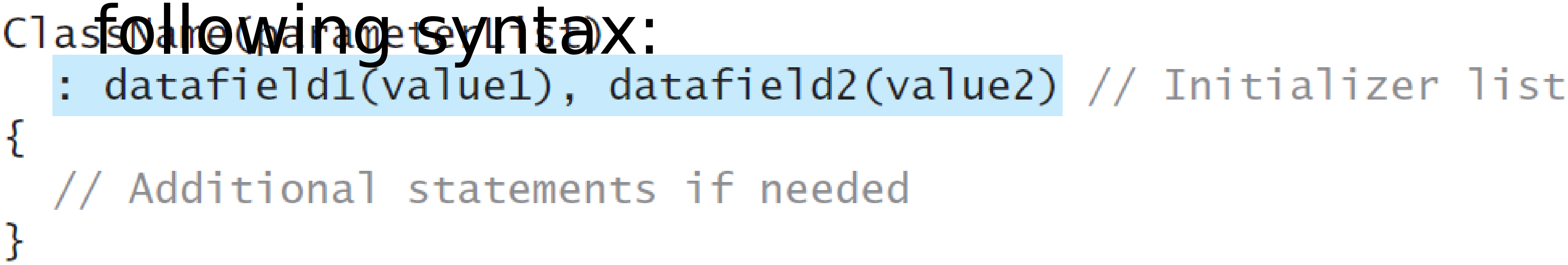
- The constructor has exactly the same name as the defining class.
- Constructors can be overloaded (i.e., multiple constructors with the same name but different signatures).
- A class normally provides a constructor without arguments (e.g., **`Circle()`**). Such constructor is called a *no-arg* or *no-argument constructor*.
- A class may be declared without constructors. In this case, a no-arg constructor with an empty body is implicitly declared in the class. This constructor is called a *default constructor*.

Constructors Features

- Constructors must have the same name as the class itself.
- Constructors do not have a return type—not even **`void`**.
- Constructors play the role of initializing objects.

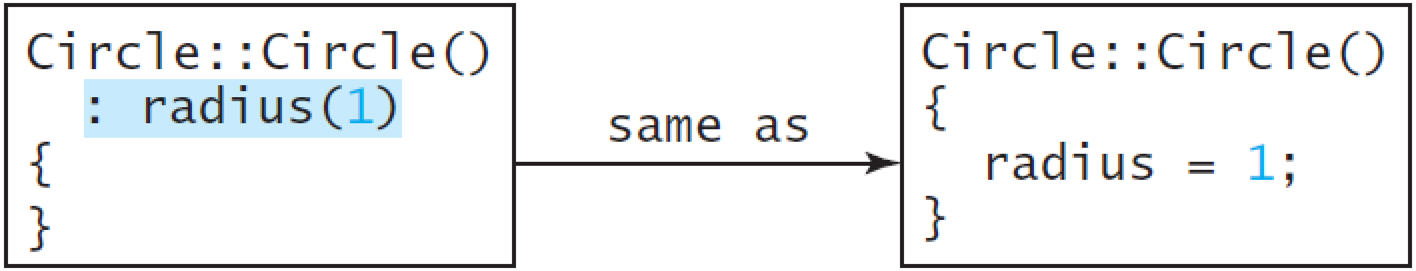
Initializer Lists

- Data fields may be initialized in the constructor using an initializer list in the following syntax:



```
ClassName(parameterList)
  : datafield1(value1), datafield2(value2) // Initializer list
{
  // Additional statements if needed
}
```

- Example:



```
Circle::Circle()
  : radius(1)
{
}
```

```
Circle::Circle()
{
  radius = 1;
}
```

Outline

- Introduction
- Defining Classes for Objects
- Example: Defining Classes and Creating Objects
- Constructors
- Constructing and Using Objects
- Separating Class Definition from Implementation
- Data Field Encapsulation

Object Names

- You can assign a name when creating an object.
- A constructor is invoked when an object is created.
- The syntax to create an object using the no-arg constructor is:

```
ClassName objectName;
```

- For example,

```
    Circle circle1;
```

- The size of and object depends on its data fields only.

```
    cout << sizeof(circle1) << endl;;
    8
```

Constructing with Arguments

- The syntax to declare an object using a constructor with arguments is:

  ```
  ClassName objectName(arguments);
  ```

- For example, the following declaration creates an object named **`circle2`** by invoking the `Circle` class's constructor with a specified radius **`5.5`**.

  ```
  Circle circle2(5.5);
  ```

Access Operator

- After an object is created, its data can be accessed and its functions invoked using the dot operator (**.**), also known as the *object member access operator*:

- **objectName.dataField** references a data field in the object.

- **objectName.function(arguments)** invokes a function on the object.

Naming Objects and Classes

- When you declare a custom class, capitalize the first letter of each word in a class name; for example, the class names **`Circle`**, **`Rectangle`**, and **`Desk`**.
- The class names in the C++ library are named in lowercase.
- The objects are named like variables.

Class is a Type

- You can use primitive data types, like `int`, to declare variables.
- You can also use class names to declare object names.
- In this sense, a class is also a data type.

Memberwise Copy

- You can also use the assignment operator **=** to copy the contents from one object to the other.
- By default, each data field of one object is copied to its counterpart in the other object. For example,

```
circle2 = circle1;
```

- Copies the **radius** in **circle1** to **circle2**.
- After the copy, **circle1** and **circle2** are still two different objects, but with the same radius.

Constant Object Name

- Object names are like array names. Once an object name is declared, it represents an object.
- It cannot be reassigned to represent another object.
- In this sense, an object name is a constant, though the contents of the object may change.

Anonymous Object

- Most of the time, you create a named object and later access the members of the object through its name.
- Occasionally, you may create an object and use it only once. In this case, you don't have to name the object. Such objects are called *anonymous objects*.
- The syntax to create an anonymous object is

  `ClassName()` or **`ClassName(arguements)`**

- You can create an anonymous object just for finding the area by:

```
  cout << "Area:" << Circle(5).getArea() <<
endl;
```

Outline

- Introduction
- Defining Classes for Objects
- Example: Defining Classes and Creating Objects
- Constructors
- Constructing and Using Objects
- Separating Class Definition from Implementation
- Data Field Encapsulation

Separating Definition from Implementation

- C++ allows you to separate class *definition* from *implementation*.
- The class definition describes the contract of the class and the class implementation implements the contract.
- The class declaration simply lists all the data fields, constructor prototypes, and the function prototypes.
- The class implementation implements the constructors and functions.
- The class declaration and implementation are in two separate files. Both files should have the same name, but with different extension names. The class declaration file has an extension name *.h* and the class implementation file has an extension name *.cpp*.

Circle.h Circle.cpp TestCircleWithHeader.cpp Run

Circle.h

```
#ifndef CIRCLE_H
#define CIRCLE_H
class Circle
{
public:
    // The radius of this circle
    double radius;

    // Construct a default circle object
    Circle();

    // Construct a circle object
    Circle(double);

    // Return the area of this circle
    double getArea();
};
#endif
```

Used to prevent a header file from being included multiple times.

Circle.cpp

```
#include "Circle.h"

// Construct a default circle object
Circle::Circle()
{
    radius = 1;
}

// Construct a circle object
Circle::Circle(double newRadius)
{
    radius = newRadius;
}

// Return the area of this circle
double Circle::getArea()
{
    return radius * radius * 3.14159;
}
```

The `::` symbol is the *binary scope resolution operator*

TestCircleWithHeader.cpp

```
#include <iostream>
#include "Circle.h"
using namespace std;

int main()
{
    Circle circle1;
    Circle circle2(5.0);

    cout << "The area of the circle of radius "
        << circle1.radius << " is " << circle1.getArea() <<
endl;
    cout << "The area of the circle of radius "
        << circle2.radius << " is " << circle2.getArea() <<
endl;

    // Modify circle radius
    circle2.radius = 100;
    cout << "The area of the circle of radius "
        << circle2.radius << " is " << circle2.getArea() <<
endl;
```

```
The area of the circle of radius 1 is 3.14159
The area of the circle of radius 5 is 78.5397
The area of the circle of radius 100 is
31415.9
```

Outline

- Introduction
- Defining Classes for Objects
- Example: Defining Classes and Creating Objects
- Constructors
- Constructing and Using Objects
- Separating Class Definition from Implementation
- Data Field Encapsulation

Data Field Encapsulation

The data fields **radius** in the **Circle** class can be modified directly (e.g., **circle1.radius = 5**). This is not a good practice for two reasons:

1. Data may be tampered.
2. Second, it makes the class difficult to maintain and vulnerable to bugs. Suppose you want to modify the **Circle** class to ensure that the radius is non-negative after other programs have already used the class. You have to change not only the **Circle** class, but also the programs (*clients*) that use the **Circle** class. This is because the clients may have modified the radius directly (e.g., **myCircle.radius = 5**).

Accessor and Mutator

- A **get** function is referred to as a *getter* (or *accessor*).
- A **get** function has the following signature:

```
returnType getPropertyName()
```

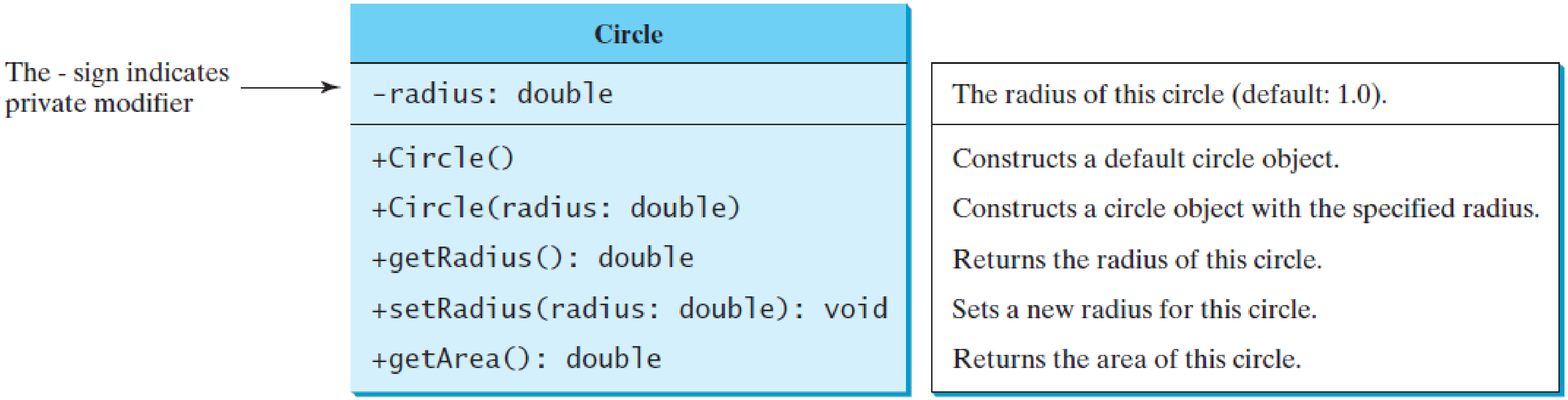
- If the **returnType** is **bool**, the **get** function should be defined as follows by convention:

```
bool isPropertyName()
```

- A **set** function is referred to as a *setter* (or *mutator*).
- A **set** function has the following signature:

```
public void setPropertyName(dataType
propertyValue)
```

Example: The Circle Class with Encapsulation



The - sign indicates private modifier

Circle	
-radius: double	The radius of this circle (default: 1.0).
+Circle()	Constructs a default circle object.
+Circle(radius: double)	Constructs a circle object with the specified radius.
+getRadius(): double	Returns the radius of this circle.
+setRadius(radius: double): void	Sets a new radius for this circle.
+getArea(): double	Returns the area of this circle.

CircleWithPrivateDataFields.h

CircleWithPrivateDataFields.cpp

TestCircleWithPrivateDataFields

Run

CircleWithPrivateDataFields.h

```
#ifndef CIRCLE_H
#define CIRCLE_H

class Circle
{
public:
    Circle();
    Circle(double);
    double getArea();
    double getRadius();
    void setRadius(double);

private:
    double radius;
};

#endif
```

CircleWithPrivateDataFields.cpp

```cpp
#include
"CircleWithPrivateDataFields.h"

// Construct a default circle object
Circle::Circle()
{
    radius = 1;
}

// Construct a circle object
Circle::Circle(double newRadius)
{
    radius = newRadius;
}

// Return the area of this circle
double Circle::getArea()
{
    return radius * radius * 3.14159;
}
```

```cpp
// Return the radius of this
circle
double Circle::getRadius()
{
    return radius;
}

// Set a new radius
void
Circle::setRadius(double
newRadius)
{
    radius = (newRadius >=
0)
              ? newRadius :
0;
}
```

TestCircleWithPrivateDataFields .cpp

```
#include <iostream>
#include "CircleWithPrivateDataFields.h"
using namespace std;

int main()
{
    Circle circle1;
    Circle circle2(5.0);

    cout << "The area of the circle of radius "
        << circle1.getRadius() << " is " << circle1.getArea() <<
endl;
    cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() <<
endl;

    // Modify circle radius
    circle2.setRadius(100);
    cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() <<
endl;
```

```
The area of the circle of radius 1 is 3.14159
The area of the circle of radius 5 is 78.5397
The area of the circle of radius 100 is
31415.9
```

Outline

- Introduction
- Defining Classes for Objects
- Example: Defining Classes and Creating Objects
- Constructors
- Constructing and Using Objects
- Separating Class Definition from Implementation
- Data Field Encapsulation