Lecture 9

C++ Programming

Arne Kutzner

Hanyang University / Seoul Korea

Object Oriented Programming in C++

Object Oriented Programming

- Object Orientation is a programming paradigm comprising several concepts for increasing code reliability and code reusability
- Basic idea / programming approach:
  - Programmer *thinks* about and defines the attributes and behavior of objects, where the objects are often modeled after real-world entities.

Programming Concepts comprised by Object Orientation

Reliability

Abstraction

Encapsulation

Information hiding

Reuse

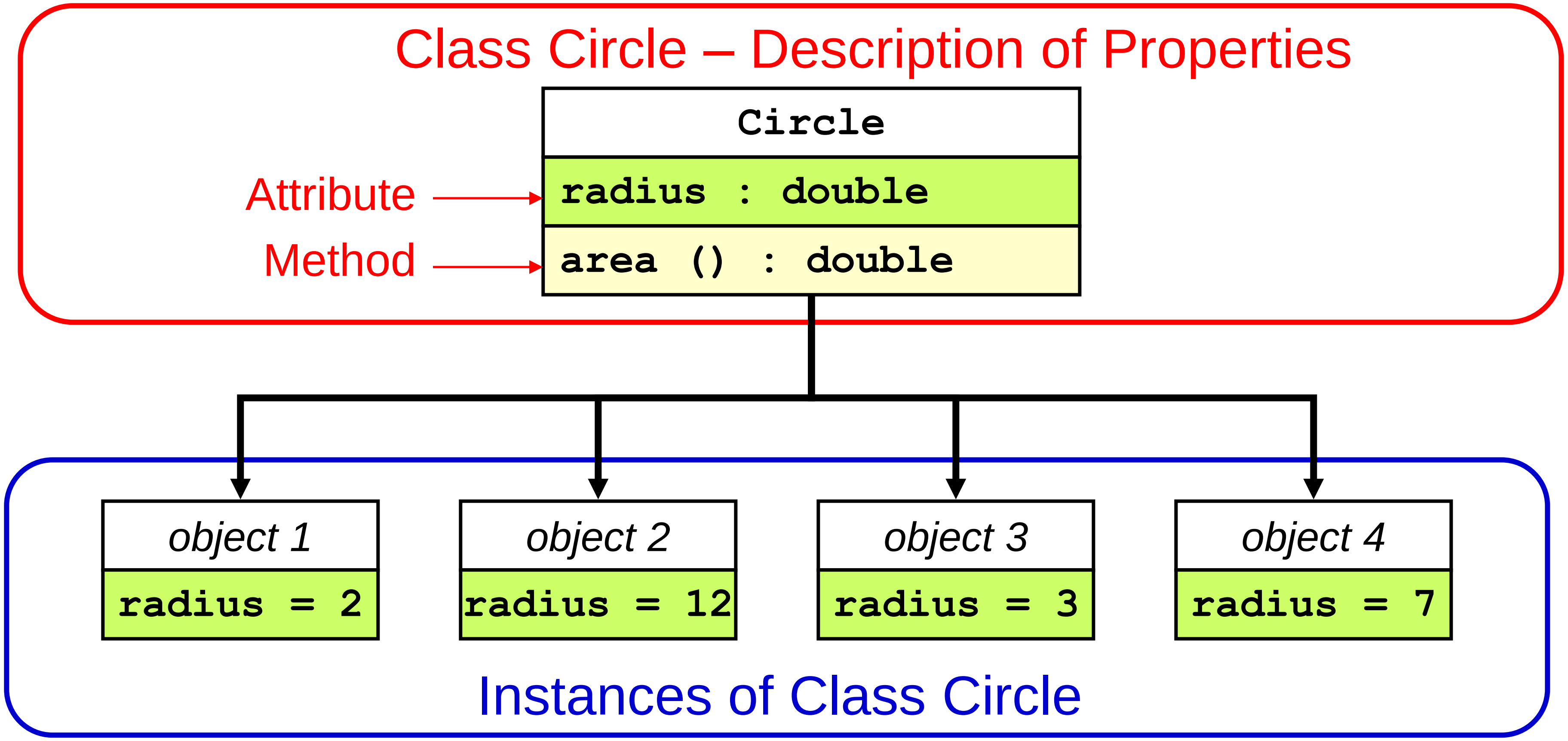
Inheritance

Polymorphism

C++ Classes - Foundations

- A C++ class is an object type.
  - Technically it is an extended form of a C-structure
- Objects are instances of a class and allocate memory during program execution
- A class definition comprises:
  - Data members, called **attributes**.
  - Class related functions, called **methods**.

Classes and Objects – Example



Class Definition in C++

```
class Circle {
  public :
      double radius;
      double area() {
            return radius * radius * 3.141;
      };
};
```

Creating an object

- Defining a class does not result in creation of an object.
- Declaring/defining a variable of a class type creates an object of that class. This process is called **instantiation.**
  Example:

```
void main() {
   Circle c; //object definition
}
```

Accessing Attributes and Methods

- Attributes and methods can be accessed by using the member access operator "." (just like components of structures in C).
  Example:

```
Circle c; // definition
c.radius = 3.0;
double a = c.area();
```

Object Creation plus Initialization using the technique for initializing structs in C

- Example:

```
void main() {
    Circle c = {5.0};
    std::cout << c.area();
}
```

- Works for public attributes only.
- Later, ee will learn a much better way for initializing objects …

Method names and the scope operator “::”

- The scope operator “::” (remember namespaces) can be used in the context of classes as well:
  Usage scenario:
  - The class definition comprises a prototype of a method merely and the actual method definition is outside the class.
  - Example on next slide …

Class Definition in C++ using Prototype

```
class Circle {
  public :
      double radius;
      double area();
};

double Circle::area() {
  return radius * radius * 3.14159;
}
```

$\left.\begin{array}{l}\texttt{double radius;}\\ \texttt{double area();}\end{array}\right\}$ Public Class Interface

What happens here?

```
class foo {
   int i;        // # elements in the array
   int a[10];    // array 10 at most
   // sum the elements
   int sum(void) {
     int i, x=0;
     for (i=0;i<i;i++)
      x+=a[i];
     return(x);
   }
   ...
```

which i is it ?

Resolving the problem …

- You can solve the previous problem using the "::" operator classname::membername.

```
 for (i=0;i<foo::i;i++)
    x+=a[i];
```

Information Hiding

- The “visibility” of attributes and methods in code segments can be controlled by the modifiers **public**, **private** and **protected**.
- The above three modifiers allow the “hiding” of attributes and methods in specific sections of the code. Therefore the notion “Information hiding”

Private vs. Public

- Public data members and methods can be accessed outside the class without limitations.
- Private members and methods are for class internal use only. They are invisible outside of the class they are defined in.
  - Private members can not be initialized using the C-struct initialization (see slide before)
- We later learn the protected modifier …

Circle with a “private” radius and getter and setter methods

```
class Circle {
  private :
      double radius;
  public :
      double area();
      void set_radius(double r) {
          radius = r;
      }
      double get_radius() {
          return radius;
      } // accessor method
}; // class
```

“setter – method”

“getter – method”

Special Member Functions

- **Constructor**s: called when a new object is created (instantiated).
  - can be many constructors, each can take different arguments.

- **Destructor**: called when an object is eliminated
  - only one, has no arguments.

Circle with Constructor

```
class Circle {
   public :
       double radius;
       double area();
       Circle(double r); // Constructor
};

Circle::Circle(double r){
   radius = r;
}

void main() {
   Circle c = Circle(5.0);
   std::cout << c.area();
}
```

Constructor Prototype

Constructor Definition

Constructor Use

Circle with Destructor

```
class Circle {
  public :
      double radius;
      double area();
      Circle(double r); // Constructor
      ~Circle(); // Destructor
};
```

Destructor Prototype

```
Circle::~Circle(){
  std::cout << "Goodbye Object\n";
}
```

Destructor Definition

Default Constructor

- Constructor that is used to create an object when you don’t provide initialization values.

```
class Circle {
  public :
      double radius;
      double area();
      Circle(double r); // Constructor
      Circle(); // Default Constructor
      ~Circle(); // Destructor
};

Circle::Circle(){ // Default Constructor Definition
  radius = 0;
}
```