C++ Classes, Constructor & Object Oriented Programming

Object Oriented Programming

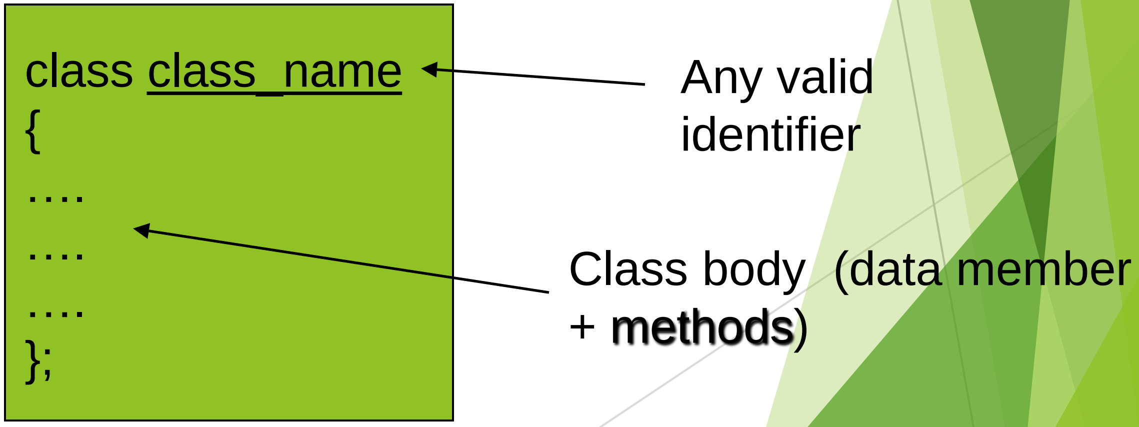
- Programmer *thinks* about and defines the attributes and behavior of objects.
- Often the objects are modeled after real-world entities.
- Very different approach than *function-based* programming (like C).

Object Oriented Programming

- Object-oriented programming (OOP)
  - Encapsulates data (attributes) and functions (behavior) into packages called classes.
- So, Classes are user-defined (programmer-defined) types.
  - Data (data members)
  - Functions (member functions or methods)
- In other words, they are structures + functions

Classes in C++

- A class definition begins with the keyword *class*.
- The body of the class is contained within a set of braces, `{ } ;` (notice the semi-colon).



```
class class_name
{
….
….
….
};
```

Any valid identifier

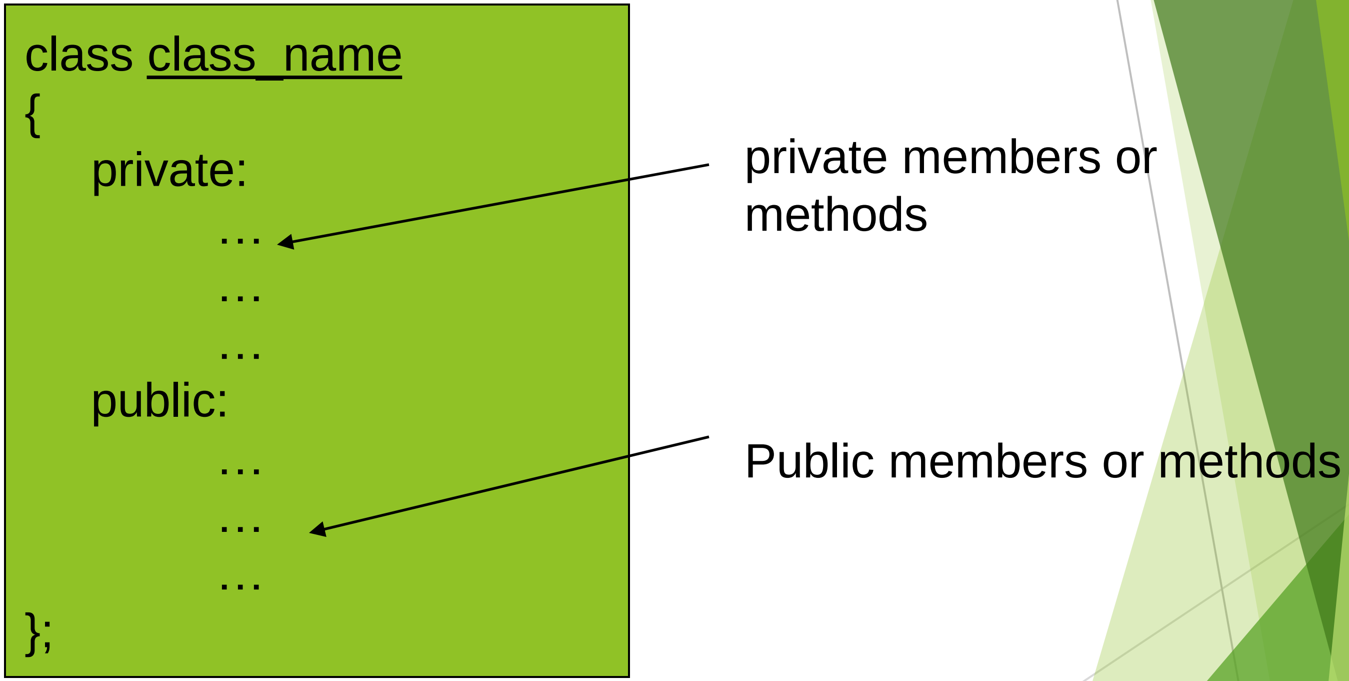
Class body (data member + **methods**)

Classes in C++

- Within the body, the keywords *private:* and *public:* specify the access level of the members of the class.
  - the default is private.

- Usually, the data members of a class are declared in the *private:* section of the class and the member functions are in *public:* section.

Classes in C++



```
class class_name
{
    private:
            …
            …
            …
    public:
            …
            …
            …
};
```

private members or methods

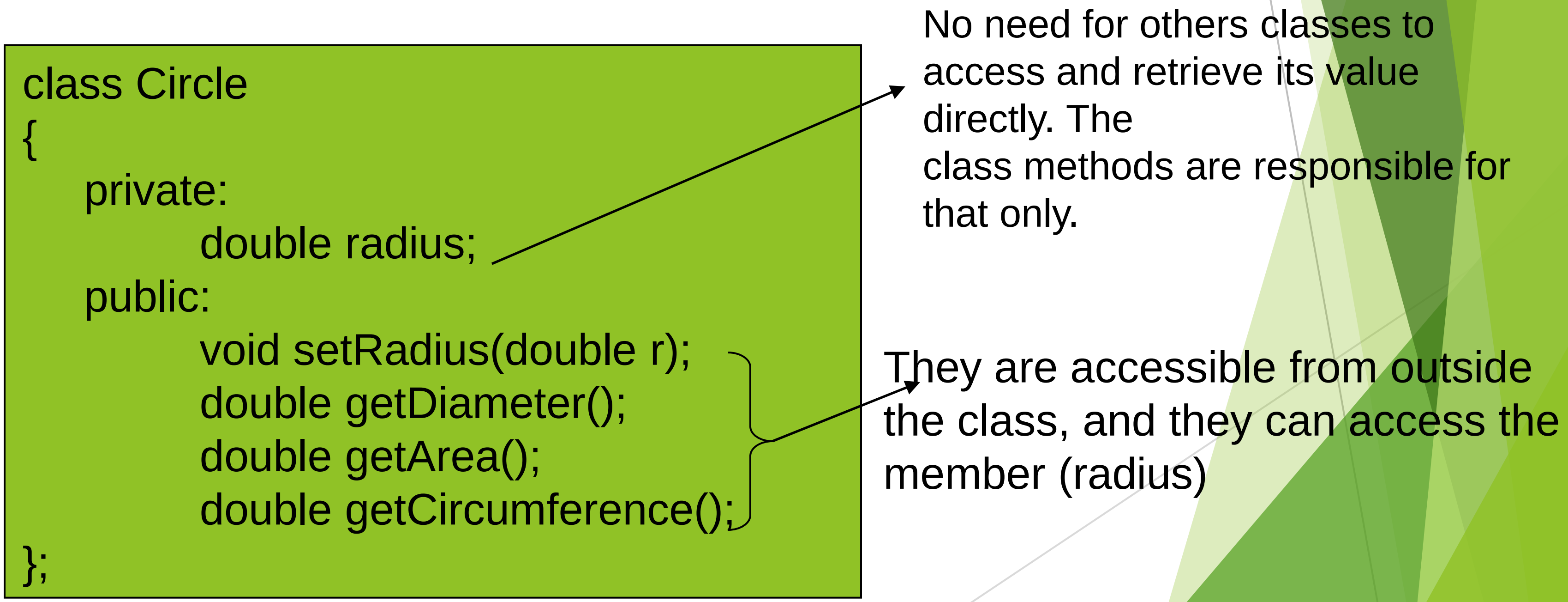
Public members or methods

Classes in C++

- Member access specifiers
  - public:
    - can be accessed outside the class directly.
      - The public stuff is *the interface*.
  - private:
    - Accessible only to member functions of class
    - Private members and methods are for internal use only.

Class Example

- This class example shows how we can encapsulate (gather) a circle information into one package (unit or class)



```
class Circle
{
    private:
            double radius;
    public:
            void setRadius(double r);
            double getDiameter();
            double getArea();
            double getCircumference();
};
```

No need for others classes to access and retrieve its value directly. The
class methods are responsible for that only.

They are accessible from outside the class, and they can access the member (radius)

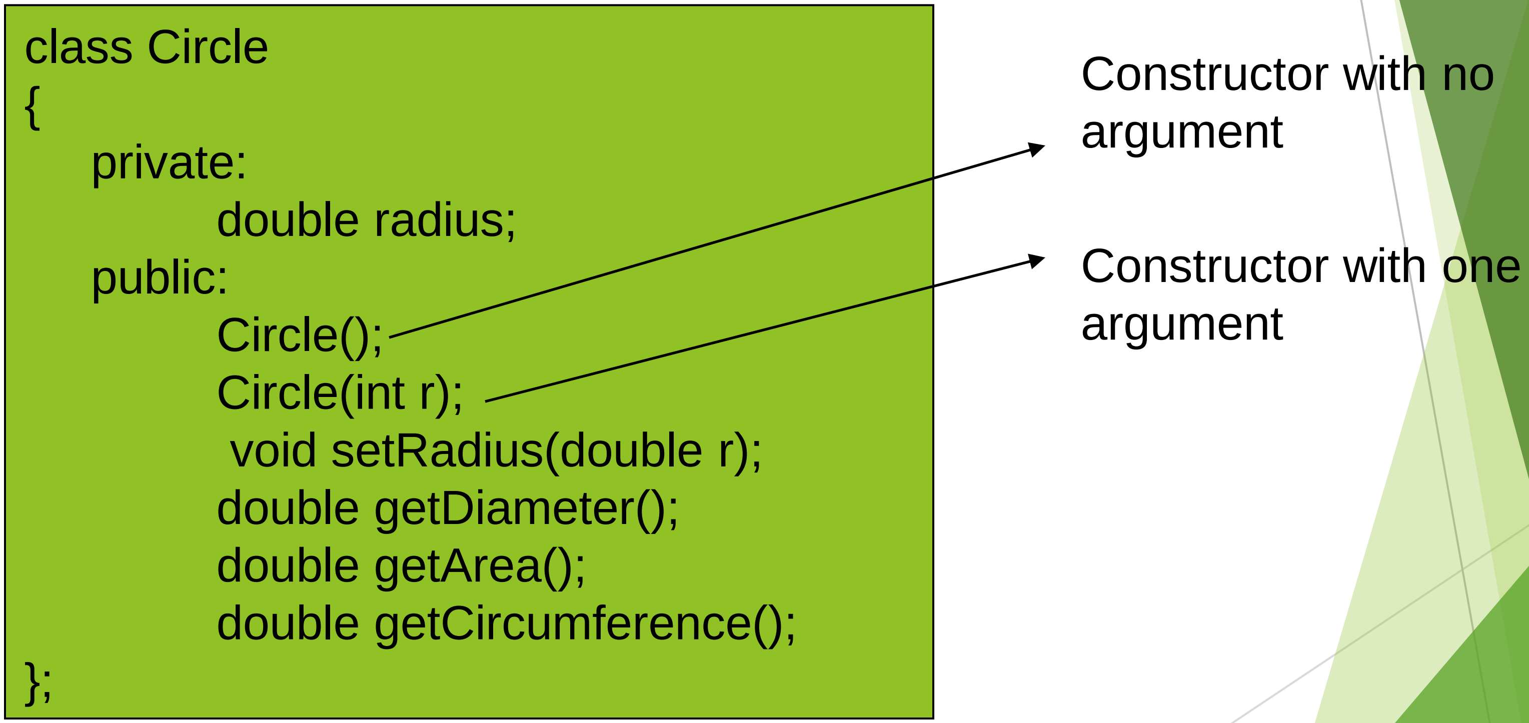
Creating an object of a Class

- Declaring a variable of a class type creates an object. You can have many variables of the same type (class).
  - *Instantiation*
- Once an object of a certain class is instantiated, a new memory location is created for it to store its data members and code
- You can instantiate many objects from a class type.
  - Ex) Circle c; Circle *c;

Special Member Functions

- Constructor:
  - Public function member
  - called when a new object is created (instantiated).
  - Initialize data members.
  - Same name as class
  - No return type
  - Several constructors
    - Function overloading

Special Member Functions



```
class Circle
{
    private:
            double radius;
    public:
            Circle();
            Circle(int r);
             void setRadius(double r);
            double getDiameter();
            double getArea();
            double getCircumference();
};
```

Constructor with no argument

Constructor with one argument

Implementing class methods

- Class implementation: writing the code of class methods.
- There are two ways:

1. Member functions defined outside class
    - Using Binary scope resolution operator (::)
    - “Ties” member name to class name
    - Uniquely identify functions of particular class
    - Different classes can have member functions with same name

- Format for defining member functions

```
ReturnType
    ClassName::MemberFunctionName( ){
    …
}
```

Implementing class methods

2. Member functions defined inside class

- Do not need scope resolution operator, class name;

```
class Circle
{
    private:
            double radius;
    public:
            Circle() { radius = 0.0;}
            Circle(int r);
            void setRadius(double r){radius = r;}
            double getDiameter(){ return radius *2;}
            double getArea();
            double getCircumference();
};
```

Defined
inside
class

```
class Circle
{
    private:
            double radius;
    public:
            Circle() { radius = 0.0;}
            Circle(int r);
            void setRadius(double r){radius = r;}
            double getDiameter(){ return radius *2;}
            double getArea();
            double getCircumference();
};
Circle::Circle(int r)
{
    radius = r;
}
double Circle::getArea()
{
    return radius * radius * (22.0/7);
}
double Circle:: getCircumference()
{
    return 2 * radius  * (22.0/7);
}
```

Defined outside class

Accessing Class Members

- Operators to access class members
  - Identical to those for **`structs`**
  - Dot member selection operator (`.`)
    - Object
    - Reference to object
  - Arrow member selection operator (`->`)
    - Pointers

```
class Circle
{
    private:
            double radius;
    public:
            Circle() { radius = 0.0;}
            Circle(int r);
            void setRadius(double r){radius = r;}
            double getDiameter(){ return radius *2;}
            double getArea();
            double getCircumference();
};
Circle::Circle(int r)
{
    radius = r;
}
double Circle::getArea()
{
    return radius * radius * (22.0/7);
}
double Circle:: getCircumference()
{
    return 2 * radius  * (22.0/7);
}
```

The first

The second constructor is called

Since radius is a private class data member

```
void main()
{
    Circle c1,c2(7);

    cout<<“The area of c1
        <<c1.getArea()<<“\n”;

    //c1.raduis = 5;//syntax error
    c1.setRadius(5);

    cout<<“The circumference of c1:”
       << c1.getCircumference()<<“\n”;

    cout<<“The Diameter of c2:”
       <<c2.getDiameter()<<“\n”;
}
```

```
class Circle
{
    private:
            double radius;
    public:
            Circle() { radius = 0.0;}
            Circle(int r);
            void setRadius(double r){radius = r;}
            double getDiameter(){ return radius *2;}
            double getArea();
            double getCircumference();
};
Circle::Circle(int r)
{
    radius = r;
}
double Circle::getArea()
{
    return radius * radius * (22.0/7);
}
double Circle:: getCircumference()
{
    return 2 * radius  * (22.0/7);
}
```

```
void main()
{
      Circle c(7);
      Circle *cp1 = &c;
      Circle *cp2 = new Circle(7);

      cout<<“The are of cp2:”
                    <<cp2->getArea();

}
```

Destructors

- Destructors
  - Special member function
  - Same name as class
    - Preceded with tilde (~)
  - No arguments
  - No return value
  - Cannot be overloaded
  - Before system reclaims object's memory
    - Reuse memory for new objects
    - Mainly used to de-allocate dynamic memory locations

Another class Example

- This class shows how to handle time parts.

```
class Time
{
    private:
        int *hour,*minute,*second;
    public:
        Time();
        Time(int h,int m,int s);
        void printTime();
        void setTime(int h,int m,int s);
        int getHour(){return *hour;}
        int getMinute(){return *minute;}
        int getSecond(){return *second;}
        void setHour(int h){*hour = h;}
        void setMinute(int m){*minute = m;}
        void setSecond(int s){*second = s;}
        ~Time();
};
```

Destructor

Dynamic locations should be allocated to pointers first

```
Time::Time()
}
        hour = new int;
        minute = new int;
        second = new int;
        *hour = *minute = *second = 0;
{

Time::Time(int h,int m,int s)
}
        hour = new int;
        minute = new int;
        second = new int;
        *hour = h;
        *minute = m;
        *second = s;
{

void Time::setTime(int h,int m,int s)
}
        *hour = h;
        *minute = m;
        *second = s;
{
```

```
void Time::printTime()
}
     cout<<"The time is : ("<<*hour<<":"<<*minute<<":"<<*second<<")"
               >>endl;
{

Time::~Time()
}
       delete hour; delete minute;delete second;
{

void main()
}
       Time *t;
       t= new Time(3,55,54);
       t->printTime();

       t->setHour(7);
       t->setMinute(17);
       t->setSecond(43);

       t->printTime();

       delete t;
{
```

Destructor: used here to de-allocate memory locations

```
Output:
The time is : (3:55:54)
The time is : (7:17:43)
Press any key to continue
```

When executed, the destructor is called

Reasons for OOP

1. Simplify programming
2. Interfaces
   - **Information hiding:**
     - Implementation details hidden within classes themselves
3. Software reuse
   - Class objects included as members of other classes