Data Structures and Object Oriented Programming

Lecture 3

Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

Object-Oriented Programming in C++

OOP Concepts

- What is an object?
  - An object is a software bundle of related ***state*** and ***behavior***
- Software objects are often used to model the real-world objects that you find in everyday life.
- Objects are key to understand object-oriented technology

OOP Concepts

- Many examples of real-world objects:
  - your car, your desk, your television set, your bicycle
- Real-world objects share two characteristics: they all have **state** and **behavior**. For example:
  - Cars have state (name, color, brand, current speed, current gear) and behavior (left-turn, right-turn, reverse, changing gear, applying brakes).

OOP Concepts

- Identifying the state and behavior for real-world objects is a good way to begin thinking in terms of OOP.

- Exercise:

  - Observe the real-world objects that are in your immediate area, for each object that you see, ask yourself two questions:

    - What possible states can this object be in?
    - What possible behaviors can this object perform?

- Write down your observations

OOP Concepts

- Real-world objects vary in complexity:
  - Your desktop lamp has only two possible states (on, off) and two possible behaviors (turn on, turn off).
  - Your desktop radio might have additional states (on, off, current volume, current station) and behaviors (turn on, turn off, increase volume, decrease volume, seek, scan, tune).
- Some objects will also contain other objects.

OOP Concepts

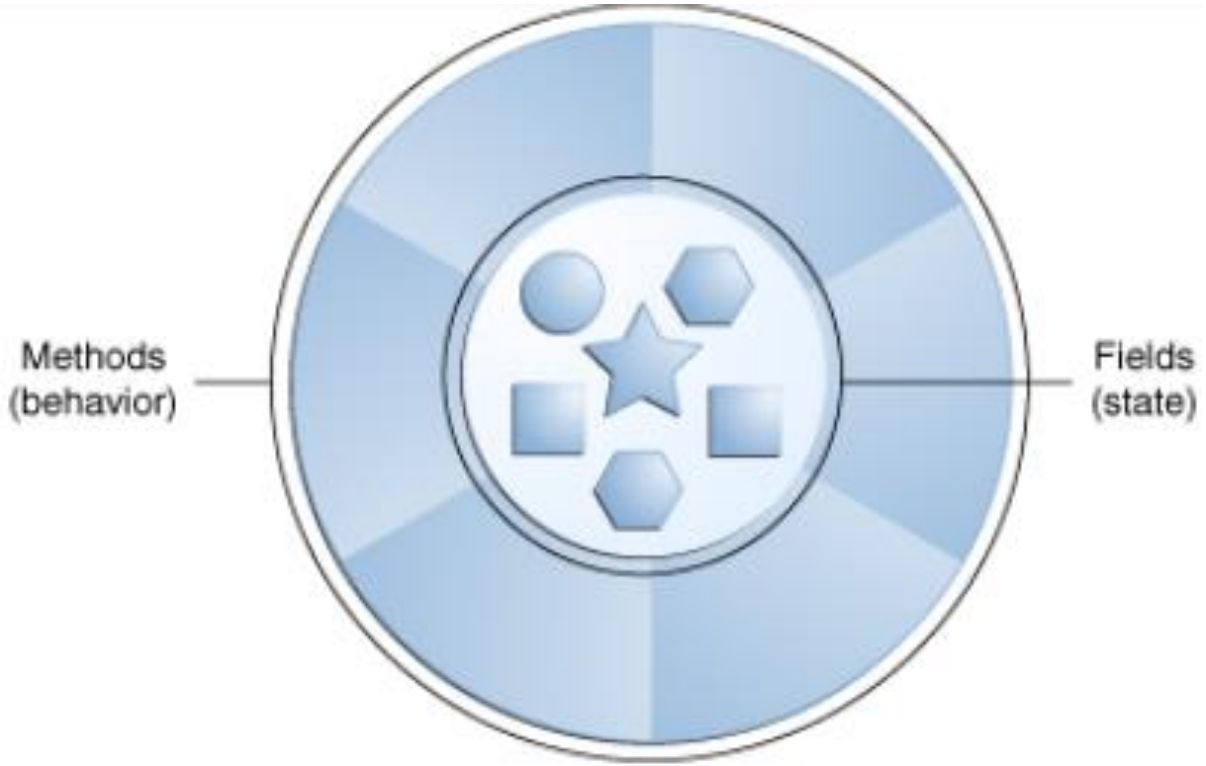
- An object stores its state in **fields** (variables) and exposes its behavior through methods (functions)

- Methods operate on an object's **internal state** and serve as the primary mechanism for object-oriented communication.

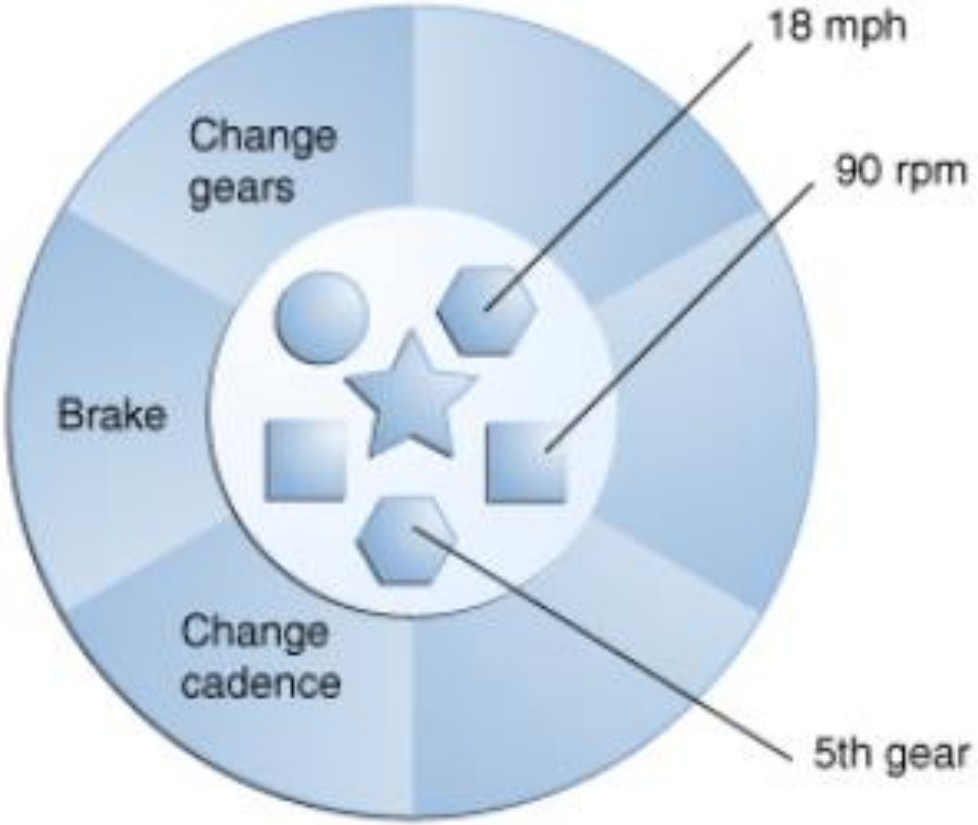
- Hiding internal state and requiring all interaction to be performed through an object's methods is known as **data encapsulation**.

- Data encapsulation is a fundamental principle of OOP.

OOP Concepts



Example: Car



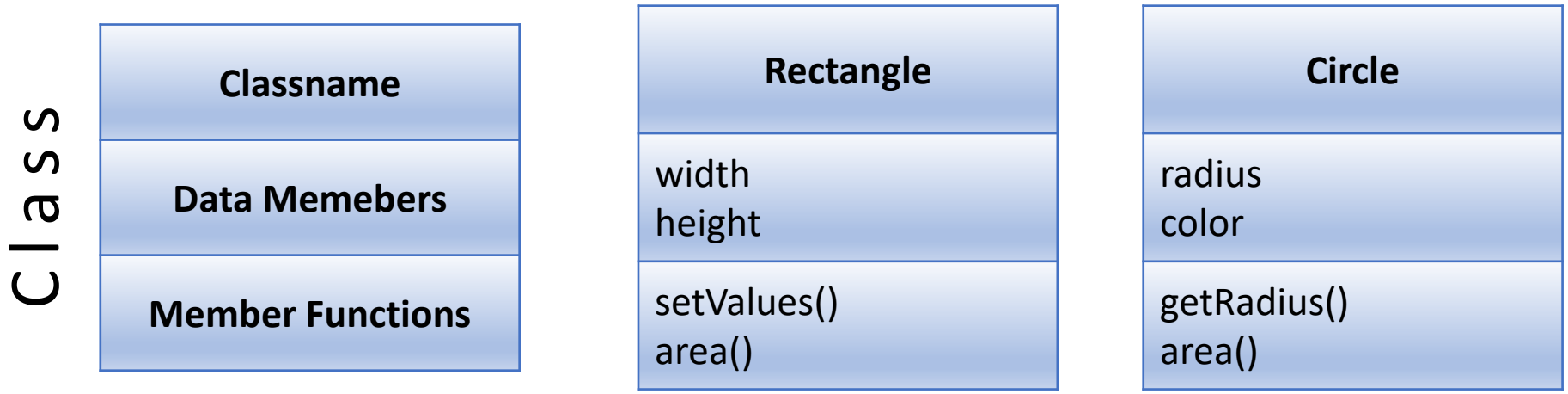
OOP Concepts

- Bundling code into individual software objects provides a number of benefits:

  1. Modularity
  2. Information hiding
  3. Code re-use

Class

- What is *Class*?
  - Class is a blueprint from which individual objects are created
  - An expanded concept of data structures: like data structures, they can contain data members, but they can **also contain functions** as members.



  - Class members can restrict their access through ***access specifiers***

Classes Intro: Access Specifier

- An ***access specifier*** determines what kind of access do you want to give to class members

- Access can be of three types:
  - **Private:** members of a class are accessible only from within other members of the same class
  - **Protected:** members of a class are not accessible outside of its members, but is accessible from the members of any class derived from same class
  - **Public:** members are accessible from anywhere where the object is visible

Class definition

- A class definition starts with the keyword ***class*** followed by the class name

```
class Rectangle {
    int width, height;
public:
    void set_values(int a, int b)
    {
        width = a;
        height = b;
    }

    int area(void)
    {
        return width * height;
    }
};
```

```
class Rectangle {
private:
    int width, height;
public:
    void set_values(int a, int b)
    {
        width = a;
        height = b;
    }

    int area(void)
    {
        return width * height;
    }
};
```

Class definition

- Complete example:

```
class Rectangle {
    int width, height;
public:
    void set_values(int a, int b)
    {
        width = a;
        height = b;
    }

    int area(void)
    {
        return width * height;
    }
};
```

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

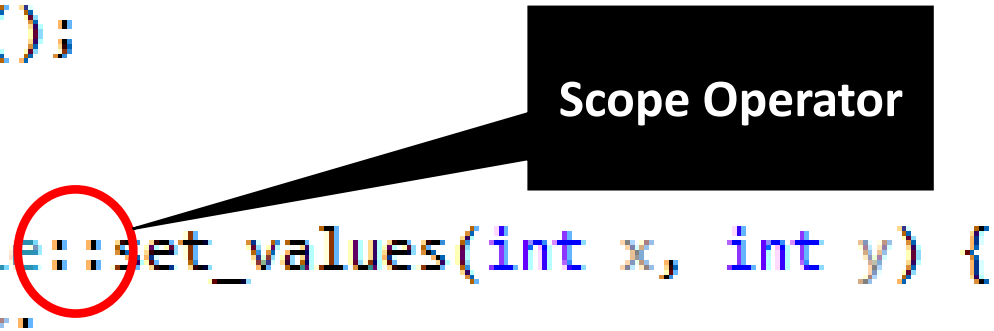
Class: Scope Operator

- Another way:

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

void Rectangle::set_values(int x, int y) {
    width = x;
    height = y;
}

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



Class: Constructor

What would happen if we called member function ***area()*** before having called ***set_values(int, int)***?

- Class can include a special function called its ***constructor***
- Automatically called when new object is created, allowing class to initialize member variables or (allocate storage). **Cannot be call explicitly**
- Declared just like regular member function, but with a name that **matches the class name** and without any return type; **not even void**

Class: Constructor

- Example:

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

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

void Rectangle::set_values(int x, int y) {
    width = x;
    height = y;
}

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

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