Northeastern University London

Classes and Objects I

LCSCI5202: Object Oriented Design

Week 2

Waleed Iqbal – 2025/26

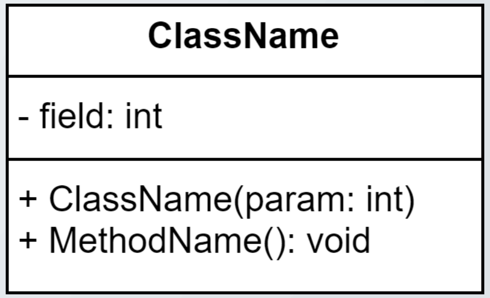
Class definition

- A class is a blueprint for creating objects in C#.
- It can contain fields, methods, properties, and events.

```
class ClassName {
    // Fields
    private int field;

    // Constructor
    public ClassName(int param) {
        field = param;
    }

    // Method
    public void MethodName() {
        Console.WriteLine("This is a method.");
    }
}
```



Instantiation

- In C#, instantiation is the process of creating an instance of a class. An instance of a class is also known as an object.
- To instantiate a class, you use the new operator and call the class' constructor. The constructor is a special method that is used to initialize the state of the object when it is created. For example:

```
ClassName obj = new ClassName(5);
obj.MethodName();
```

Constructor

- A constructor is a special type of method that is used to initialize the state of an object when it is created. It is called automatically when an object is instantiated using the `new` operator.
- It has the same name as the class, and it does not have a return type (not even void). A constructor can be defined with or without parameters.

```
class ClassName {
    public ClassName(int param) {
        field = param;
    }
}
```

Accessing methods and fields

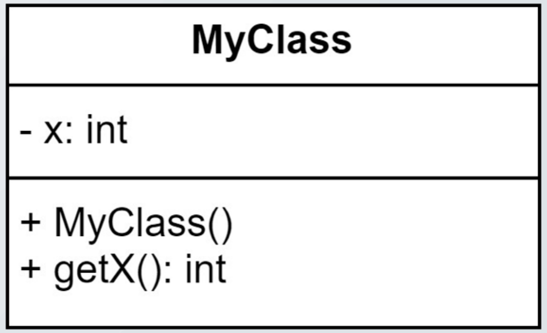
- Once an object is instantiated, you can use it to call its methods and access its fields. For example:

```
MyClass myObject = new MyClass();
int myVariable = myObject.getX();
Console.WriteLine(myVariable);

class MyClass {
   int x;

   public MyClass() {
      x = 0;
   }

   public int getX() {
      return x;
   }
}
```



Writing Methods

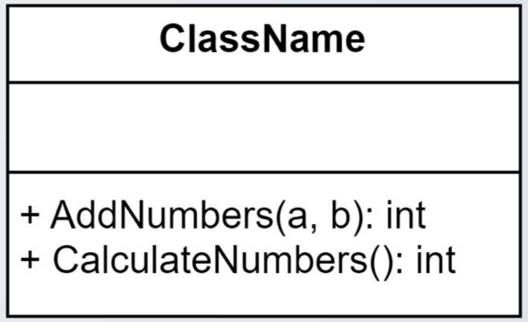
- A method is a block of code that performs a specific task.
- Methods allow code reuse and modular design.

```
class ClassName {
    int AddNumbers(int a, int b) {
        return a + b;
    }
}
```

Using Methods

```
class ClassName {
    int AddNumbers(int a, int b) {
        return a + b;
    }

    void CalculateNumbers() {
        int myResult = AddNumbers(3, 5);
    }
}
```



Writing Methods

- A method is a block of code that performs a specific task.
- Methods allow code reuse and modular design.

```
[access modifier] [return type] MethodName([parameters]) {
    // Method body
    // Code to execute
    return value; // Optional, if return type is not void
}
```

Access Modifiers

- Control the visibility and accessibility of classes, methods, fields, and other members.
- Define where and how a class or its members can be accessed.
    - private: Accessible only within the same class.
    - public: Accessible from anywhere in the program.
    - protected: Accessible within the same class and by derived classes.

Inheritance

- Allows a class (derived class) to acquire the properties and behavior (methods) of another class (base class).
- Promotes code reuse and establishes a parent-child relationship between classes.

Inheritance

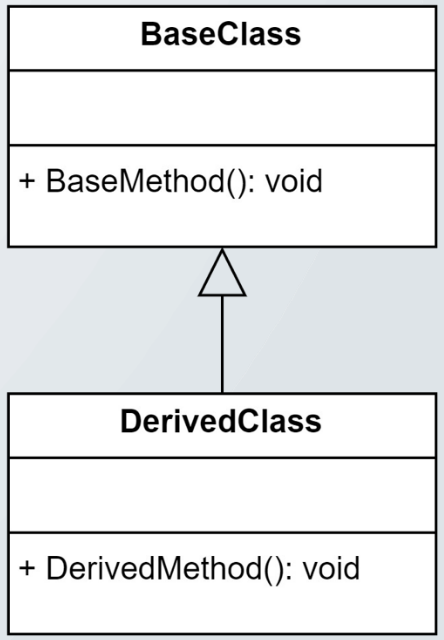
- Base Class: The class whose properties and methods are inherited.
- Derived Class: The class that inherits from the base class.
- base Keyword: Used to access members of the base class from the derived class.
- virtual and override Keywords: Used for modifying inherited methods for customization.

Inheritance

```
DerivedClass obj = new DerivedClass();
obj.BaseMethod();
obj.DerivedMethod();


class BaseClass {
    public void BaseMethod() {
        Console.WriteLine("Base method.");
    }
}

class DerivedClass : BaseClass {
    public void DerivedMethod() {
        Console.WriteLine("Derived method.");
    }
}
```



Method Overriding

- Allows a derived class to provide a specific implementation of a method that is already defined in its base class.
- The base class method must be marked as virtual, and the derived class must use the override keyword.
- Is a form of **polymorphism**, where a method behaves differently based on the object type.

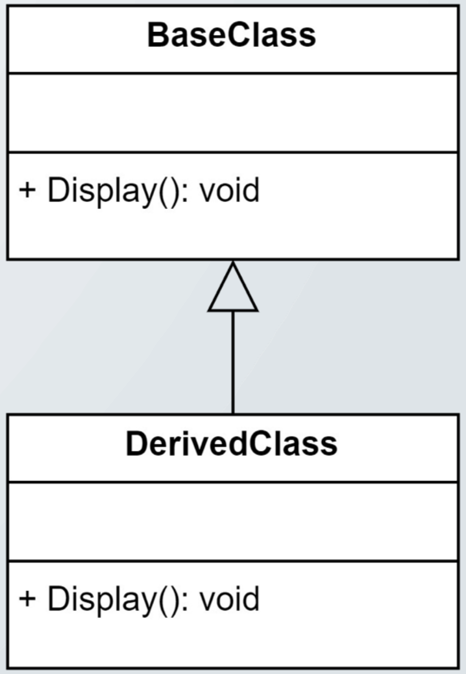
Inheritance

```
BaseClass obj1 = new BaseClass();
obj1.Display();  // Output: Display from BaseClass

DerivedClass obj2 = new DerivedClass();
obj2.Display();  // Output: Display from DerivedClass

class BaseClass {
    public virtual void Display() {
        Console.WriteLine("Display from BaseClass");
    }
}

class DerivedClass : BaseClass {
    public override void Display() {
        Console.WriteLine("Display from DerivedClass");
    }
}
```



Abstract Classes

- A class that cannot be instantiated directly and is meant to serve as a base class for other classes.
- Can contain both abstract methods (without implementation) and regular methods (with implementation).

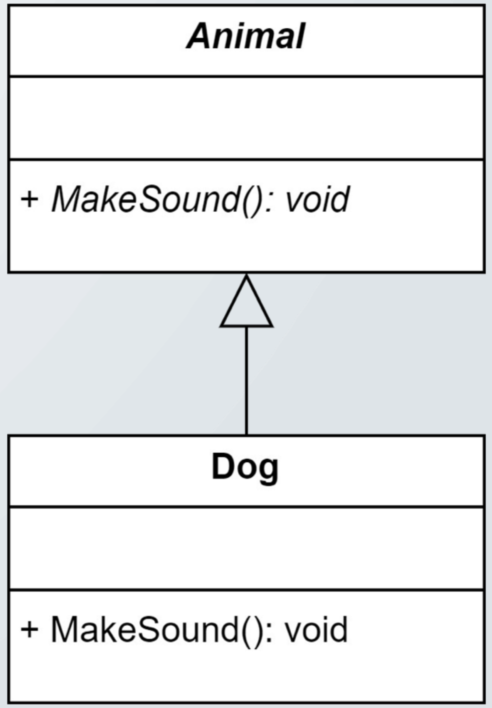
Abstract Classes

```
Dog myDog = new Dog();
myDog.MakeSound();
myDog.Sleep();

public abstract class Animal {
    public abstract void MakeSound();

    public void Sleep() {
        Console.WriteLine("Sleeping...");
    }
}

public class Dog : Animal {
    public override void MakeSound() {
        Console.WriteLine("Bark");
    }
}
```



Summary

- **Classes**: Blueprints for creating objects (contain fields, methods, properties, events).
- **Objects**: Instances of classes, created via constructors.
- **Constructors**: Special methods to initialize object state, called automatically on instantiation.
- **Methods**: Blocks of reusable code; defined with access modifiers (public, private, protected).
- **Access Modifiers**: Control visibility and accessibility.
- **Inheritance**: Enables code reuse and parent-child relationships.
- **Method Overriding**: Derived classes redefine base class methods using virtual and override.
- **Abstract Classes**: Cannot be instantiated; define abstract + implemented methods for subclasses.