# OOP Characteristics, Overload, Override, static

# Lesson Objectives

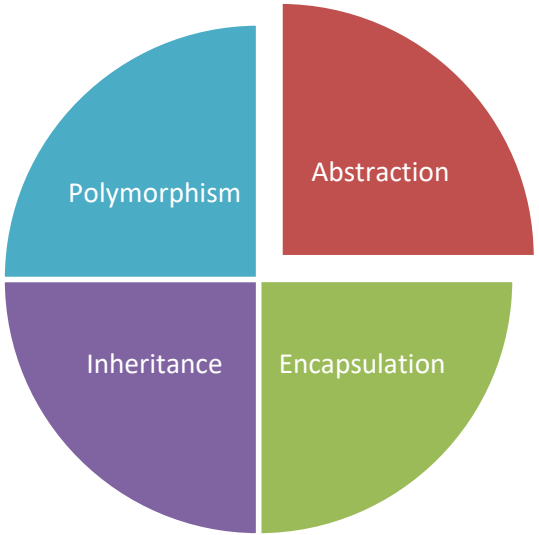
- OOP Characteristics
  - ✓ Inheritance
  - ✓ Polymorphism
- Overload
- Override

Section 1

# OOP CHARACTERISTICS (CONT.)

# OOP Characteristics

- **Abstraction**:
  - ✓ extracting only the required information from objects
- **Encapsulation**
  - ✓ Details of what a class contains need not be visible to other classes and objects that use it
- **Inheritance**
  - ✓ creating a new class based on the attributes and methods of an existing class
- **Polymorphism**
  - ✓ the ability to behave differently in different situations



# Inheritance

- Inheritance enables you to create new classes that reuse, extend, and modify the behaviour defined in other classes.
  - ✓ The class whose members are inherited is called the base class,
  - ✓ The class that inherits those members is called the derived class.
  - ✓ A derived class can have only one direct base class (*single inheritance)*
  - ✓ Inheritance is transitive *(multi-level inheritance)*

# Inheritance syntax

- To inherit from a class, use the : symbol
- **Syntax**

  [<Modifier>] class <class name> : <parent class name>

- **Example:**

  public class Avenger : Baby

# Example

```
1 reference
public class Avenger : Baby
{
    0 references
    public decimal Strength { get; set; }
    0 references
    public string SpecialAbilities { get; set; }

    1 reference
    public void Fly()
    {
        Console.WriteLine("Avenger can fly");
    }
}
```

```
var hulk = new Avenger();
hulk.IsHungry = true;
hulk.Cry();
hulk.Fly();
```

# Inheritance Rules

- **Rule 1:** The execution of any child class starts by invoking its parent class's default constructor by default.
- **Rule 2:** The child class can access its parent class's members but a parent class can never access its child class member.
- **Rule 3:** In the same way an object of a class can also be assigned to its parent class variable and make it as a reference, but in this scenario we are also using the parent reference.

# Inheritance Rules

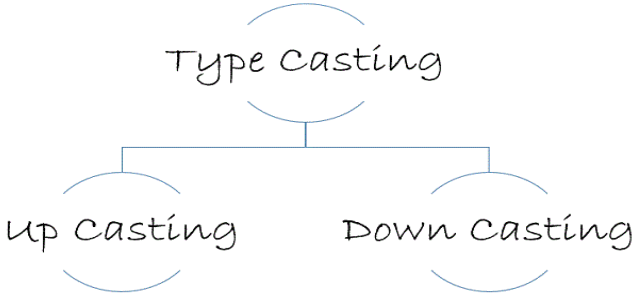
- **Rule 4:** Each and every class we define in .NET languages has an implicit parent class (the class object) defined in the system namespace.
- **Rule 5:** Derived class cannot wider accessible than super class

| Super class | Derived class |
|---|---|
| internal | internal |
| public | public |
| internal | ~~public~~ |
| public | internal |

# Inheritance Rules

- If you don't want other classes to inherit from a class, use the sealed keyword
- Use base keyword to access members of the base class from within a derived class

# Casting Objects

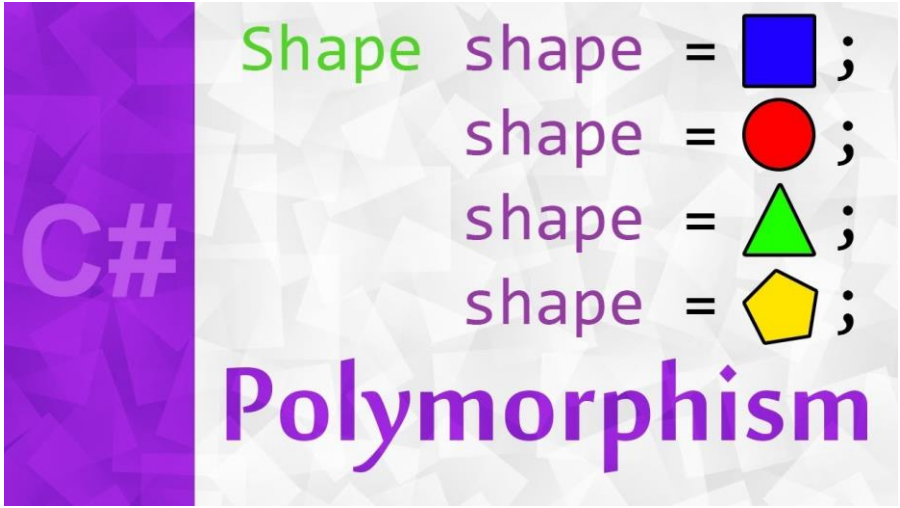



- An assignment of derived class object to a base class reference in C# inheritance is known as up-casting.
- The up-casting is implicit and any explicit typecast is not required.

- An assignment of base class object to derived class object is known as Down-casting.
- Down-Casting is required in C# programming

# Polymorphism

- Polymorphism is the ability to behave differently in different situations.
- It is basically seen in programs where you have multiple methods declared with the same name but with different parameters and different behaviour.



Section 2

# METHOD OVERLOADING, OVERRIDING

# Method Overloading

- Method Overloading is the common way of implementing polymorphism.
- It is the ability to redefine a function in more than one form.
- A user can implement function overloading by defining two or more functions in a class sharing the same name.
- C# can distinguish the methods with different method signatures.
- Constructor is considered as a special method => can have constructor overloading

# Method Overloading

- Overloading types:
  - ✓ Difference number of parameters
  - ✓ Difference data type of parameters
  - ✓ Difference order of parameters with difference data type
  - ✓ CANNOT: difference return values

# Example

```
public void Eat()
{
    Console.WriteLine($"Baby name {this.Name} is eating nothing.");
}

public void Eat(string food)
{
    if (this.IsHungry)
    {
        this.Cry();
        if (!string.IsNullOrEmpty(food) && food.Equals(this.Hobby))
        {
            Console.WriteLine($"Baby name {this.Name} is eating
            {food}");
            this.IsHungry = false;
        }
    }
}

public void Eat(string food, bool enough)
{
    if (this.IsHungry)
    {
        this.Cry();
        if (!string.IsNullOrEmpty(food) && food.Equals(this.Hobby))
        {
            Console.WriteLine($"Baby name {this.Name} is eating
            {food}");
            if (enough)
            {
                this.IsHungry = false;
            }
        }
    }
}
```

# DO and AVOID

## DO

✓ DO try to use descriptive parameter names to indicate the default used by shorter overloads.

## AVOID

✕ AVOID arbitrarily varying parameter names in overloads.

*If a parameter in one overload represents the same input as a parameter in another overload, the parameters should have the same name.*

✕ AVOID being inconsistent in the ordering of parameters in overloaded members.

*Parameters with the same name should appear in the same position in all overloads.*

# DO and DON’T

**DO**

✓ DO make only the longest overload virtual (if extensibility is required).

*Shorter overloads should simply call through to a longer overload.*

✓ DO allow null to be passed for optional arguments.

✓ DO use member overloading rather than defining members with default arguments.

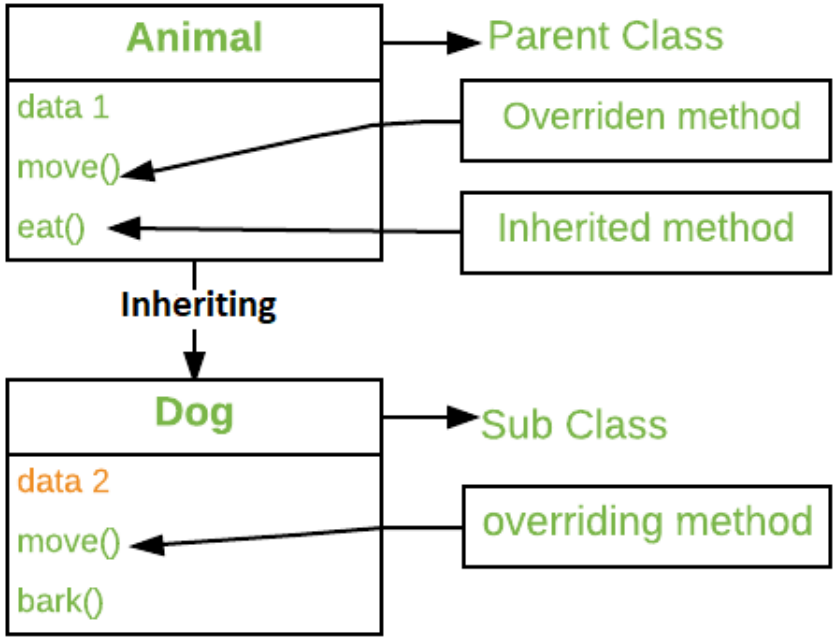
**DON’T**

✕ DO NOT use ref or out modifiers to overload members.

✕ DO NOT have overloads with parameters at the same position and similar types yet with different semantics.

# Method Overriding

- Method Overriding is a technique that allows the invoking of functions from another class (base class) in the derived class
- Creating a method in the derived class with the same signature as a method in the base class is called as method overriding.



# Method Overriding

- In C# we usually use 3 keywords for Method Overriding:
- **virtual**
  - ✓ use within base class method. It is used to modify a method in *base class* for *overridden* that particular method in the derived class.
- **override**
  - ✓ use with derived class method. It is used to modify a *virtual* or *abstract* method into *derived class* which presents in base class.
- **base**
  - ✓ used to access members of the base class from derived class.

# Method Overriding

```
4 references
public class Baby
{
    1 reference
    public string Name { get; set; }
    3 references
    public virtual void Eat()
    {
        Console.WriteLine($"Baby name {this.Name} is eating nothing.");
    }

    /* ... code ... */
}

1 reference
public class Avenger : Baby
{
    3 references
    public override void Eat()
    {
        base.Eat();
        Console.WriteLine("Avenger eats very much!");
    }

    /* ... code ... */
}
```

# Rules

- Method overriding is possible only in derived classes.
  - ✓ *Because a method is overridden in the derived class from base class.*
- A method must be a virtual or non-static method for override.
- Both the override method and the virtual method must have the same access level modifier.

# Rules

- Overriding method can be override then override…

```
public class Baby
{
    public string Name { get; set; }
    public virtual void Eat()
    {
        Console.WriteLine($"Baby name {this.Name} is eating nothing.");
    }
}

public class Avenger : Baby
{
    public override void Eat()
    {
        base.Eat();
        Console.WriteLine("Avenger eats very much!");
    }
}
public class ATeam : Avenger
{
    public override void Eat()
    {
        base.Eat();
        Console.WriteLine("Avenger eats very very much!");
    }
}
```

# Summary: overload vs override

1. Creating more than one method or function having same name but different signatures or the parameters in the same class is called method overloading.
2. It is called the compile time polymorphism
3. It has the same method name but with different signatures or the parameters

1. Creating a method in the derived class with the same signature as a method in the base class is called as method overriding
2. It is called runtime polymorphism
3. It must have same method name as well as the signatures or the parameters.

# Summary: overload vs override

4. Method overloading doesn't need inheritance
5. Method overloading is possible in single class only
6. Access modifier can be any
7. Method overloading is also called early binding.

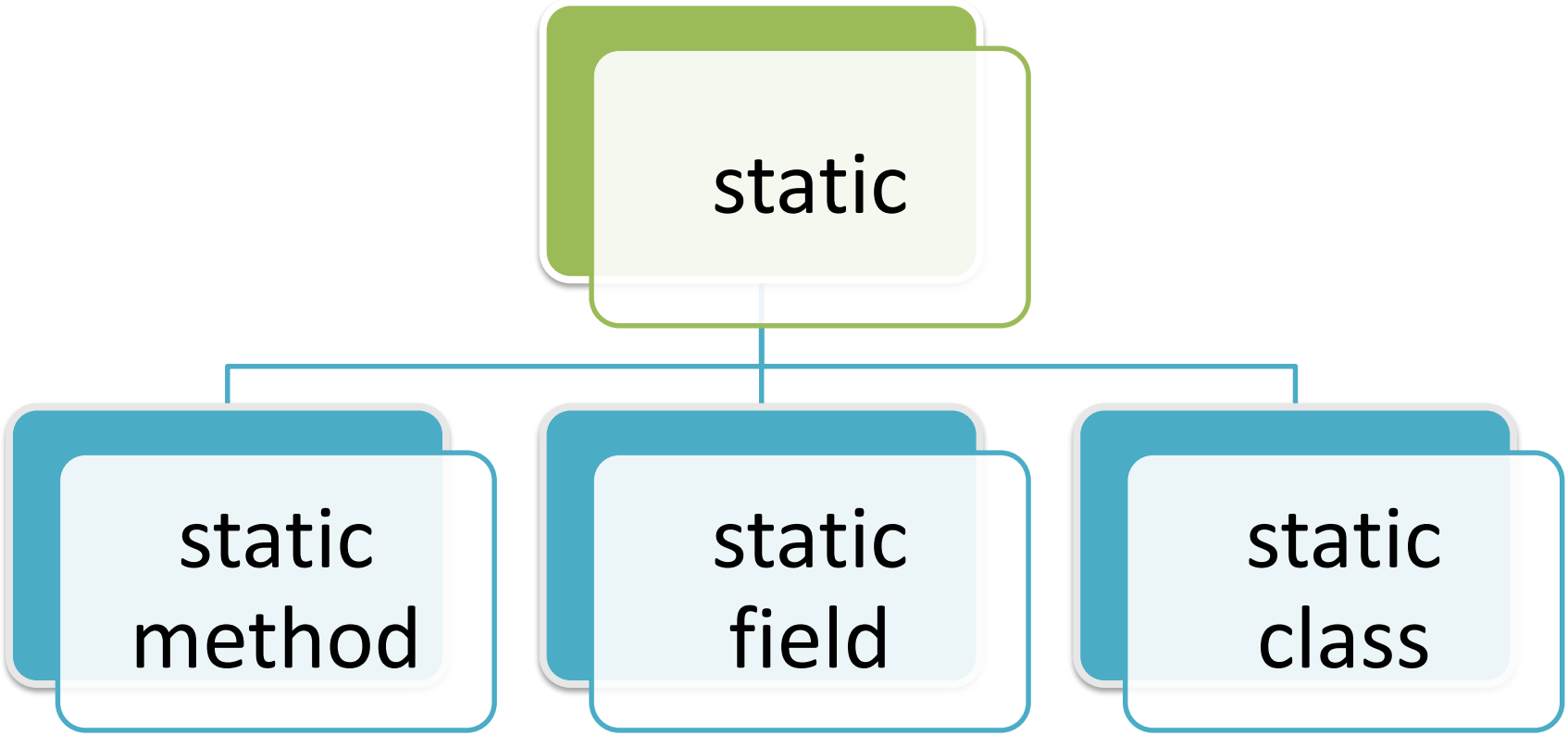
4. Method overriding needs inheritance
5. Method overriding needs hierarchy level of the classes i.e. one parent class and other child class.
6. Access modifier must be public.
7. Method overriding is also called late binding.

Section 3

# STATIC

# Static concept

- Use the static modifier to declare a static member, which belongs to the type itself rather than to a specific object.
- static in use:
  - ✓ To implement common functions. Example: Methods in Math or Convert class
  - ✓ To provide some statistics data like: count number of user, ….


static
static method
static field
static class

# static method

- A static method in C# is a method that keeps only one copy of the method at the Type level, not the object level.
- That means, all instances of the class share the same copy of the method and its data.
- The last updated value of the method is shared among all objects of that Type.
- Static methods are called by using the class name, not the instance of the class.

# static field

- Use static field create a single field that is shared among all objects created from a single class.
- We can access the field by using the name of the class
- Static field can be used inside static method

# static class

- Static classes cannot contain Instance Constructors.
- Static classes contain only static members.
- Static classes cannot be instantiated.
- Static classes are sealed. That means, you cannot inherit other classes from instance classes.

# Lesson Summary

# Thank you