Polymorphism

Polymorphism, Override and Overload Methods

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

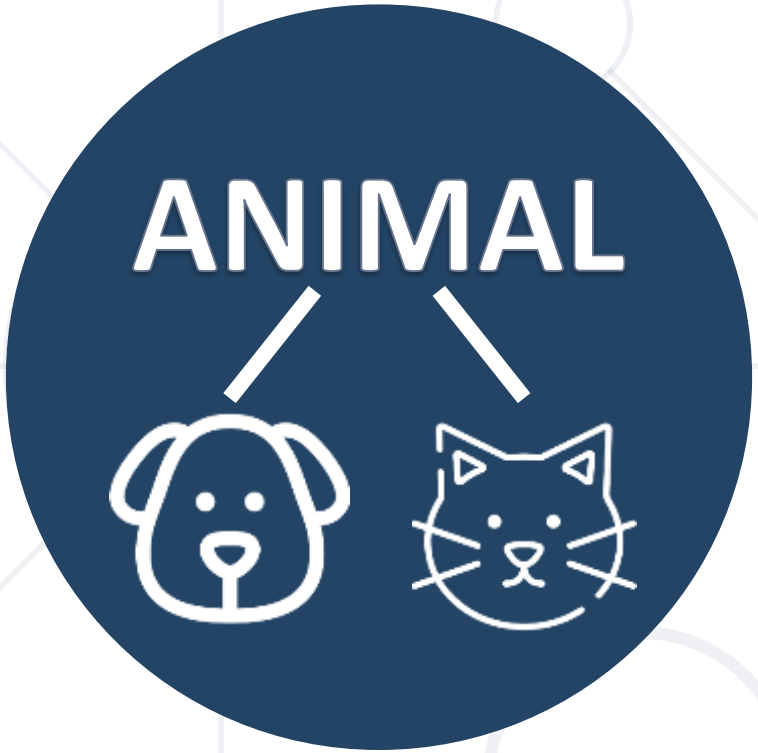
Table of Contents

Questions

Software University

sli.do

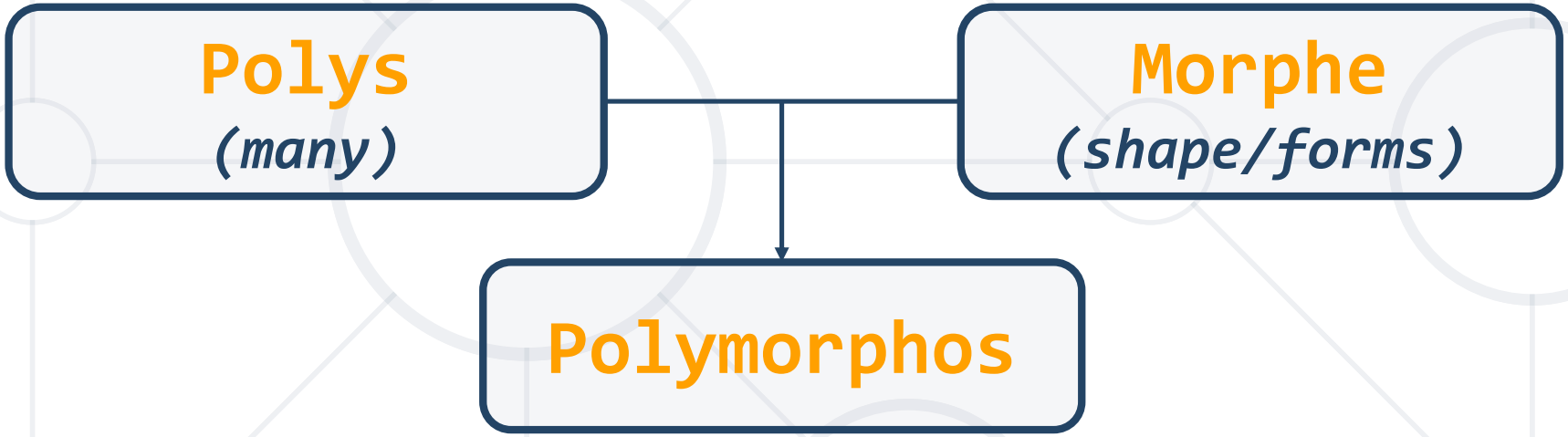
#csharp-advanced



Polymorphism

What is Polimorphism?

- From the Greek



- This is something similar to a word having several different meanings depending on the context
- Polymorphism is often referred to as the third pillar of object-oriented programming, after encapsulation and inheritance

Polymorphism in OOP

- Ability of an **object** to take on **many forms**

```
public interface IAnimal {}
public abstract class Mammal {}
public class Person : Mammal, IAnimal {}
```

```
Person IS-A Person
```

```
Person IS-AN Animal
```

```
Person IS-AN Object
```

```
Person IS-A Mammal
```

Variable Type and Data Type

- **Variables Type** is the compile-time type of the variable
- **Data Type** is the actual runtime type of the variable
- If you need an **object method** you need to **cast it or override it**

```
public class Person : Mammal, IAnimal {}
object objPerson = new Person();
IAnimal person   = new Person();
Mammal mammal    = new Person();
Person person    = new Person();
```

Variable Type

Data Type

Keyword – is

- Runtime check if an **object** is an **instance** of a specific **class**

```
public class Person : Mammal, IAnimal {}
IAnimal person = new Person();
Mammal personOne = new Person();
Person personTwo = new Person();
if (person is Person)
{
  ((Person)person).getSalary();
}
```

Check object type of person

Cast to object type and use its methods

is Type Pattern

- Type pattern - tests whether an expression can be converted to a specified type and casts it to a variable of that type

```
public class Person : Mammal, IAnimal {}
Mammal personOne = new Person();
Person personTwo = new Person();
if (personOne is Person person)
{
  person.GetSalary();
}
```

Checks if object is of type person and casts it

Uses its methods

is Constant Pattern

- When performing pattern matching with the constant pattern, **is** tests whether an expression equals a specified constant
- Checking for **null** can be performed using the constant pattern

```
int i = 0;
int min = 0, max = 10;
while(true)
{
  Console.WriteLine($"i is {i}");
  i++;
  if(i is max or min) break;
}
```

is var Pattern

- A pattern match with the **var pattern** always succeeds

```
Enumerable.Range(0, 100).Where(
    x => x % 10 is var r && r >= 1 && r <= 3)
```

- The value of expr is always assigned to a local variable named varname
- **varname** is a variable of the same type as **expr**
- Note that if **expr** is null, the **is** expression still is true and assigns null to **varname**

Keyword – is

Software University

> Anytime you find yourself writing code of the form "if the object is of type T1, then do something, but if it's of type T2, then do something else", slap yourself.

From *Effective C++*, by Scott Meyers

Keyword – as

- You can use the as operator to perform certain types of conversions between compatible reference types

```
public class Person : Mammal, IAnimal {}

IAnimal person = new Person();
Mammal personOne = new Person();
Person personTwo;
personTwo = personOne as Person;
if (personTwo != null)
{
  // Do something specific for Person
}
```

Convert Mammal to Person

Check if conversion is successful

Types of Polymorphism

- Runtime

```
public class Shape {}
public class Circle : Shape {}
public static void Main()
{
  Shape shape = new Circle();
  shape.Draw();
}
```

- Compile-time

```
public static void Main()
{
  int Sum(int a, int b, int c)
  double Sum(double a, double b)
}
```

Compile-time Polymorphism

- Also known as **Static Polymorphism**

```
public static void Main()
{
  static int MyMethod(int a, int b) {}
  static double MyMethod(double a, double b) { … }
}
```

Method overloading

- Argument lists could differ in:
  - Number of parameters
  - Data type of parameters
  - Order of parameters

Problem: MathOperation

```
MathOperation
+Add(int, int): int
+Add(double, double, double): double
+Add(decimal, decimal, decimal): decimal
```

```
MathOperations mo = new MathOperations();
Console.WriteLine(mo.Add(2, 3));
Console.WriteLine(mo.Add(2.2, 3.3, 5.5));
Console.WriteLine(mo.Add(2.2m, 3.3m, 4.4m));
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1503#0

Solution: MathOperation

```
public int Add(int a, int b)
{
  return a + b;
}
public double Add(double a, double b, double c)
{
  return a + b + c;
}
public decimal Add(decimal a, decimal b, decimal c)
{
  return a + b + c;
}
```

Rules for Overloading a Method

- Name should be the same
- **Signature** must be different
  - **Number** of arguments
  - **Type** of arguments
  - **Order** of arguments
- Return type is not a part of its signature
- Overloading can take place in the **same class** or in its **sub-classes**
- Constructors can be **overloaded**

Runtime Polymorphism (1)

- Has two distinct aspects:
- At run time, objects of a **derived class** may be treated as objects of **a base class in** places, such as method parameters and collections or arrays
  - When this occurs, the **object's declared type** is no longer identical to **its run-time type**

Runtime Polymorphism(2)

- Base classes may define and implement **virtual methods**
  - Derived classes can **override**
  - They provide **their own definition and implementation**
- At run-time, the CLR looks up the run-time type of the object and invokes that override of the virtual method

Runtime Polymorphism (1)

- Also known as **Dynamic Polymorphism**

```
public class Rectangle {
  public virtual double Area() {
    return this.a * this.b;
  }
}
public class Square : Rectangle {
  public override double Area() {
    return this.a * this.a;
  }
}
```

Method overriding

Runtime Polymorphism (2)

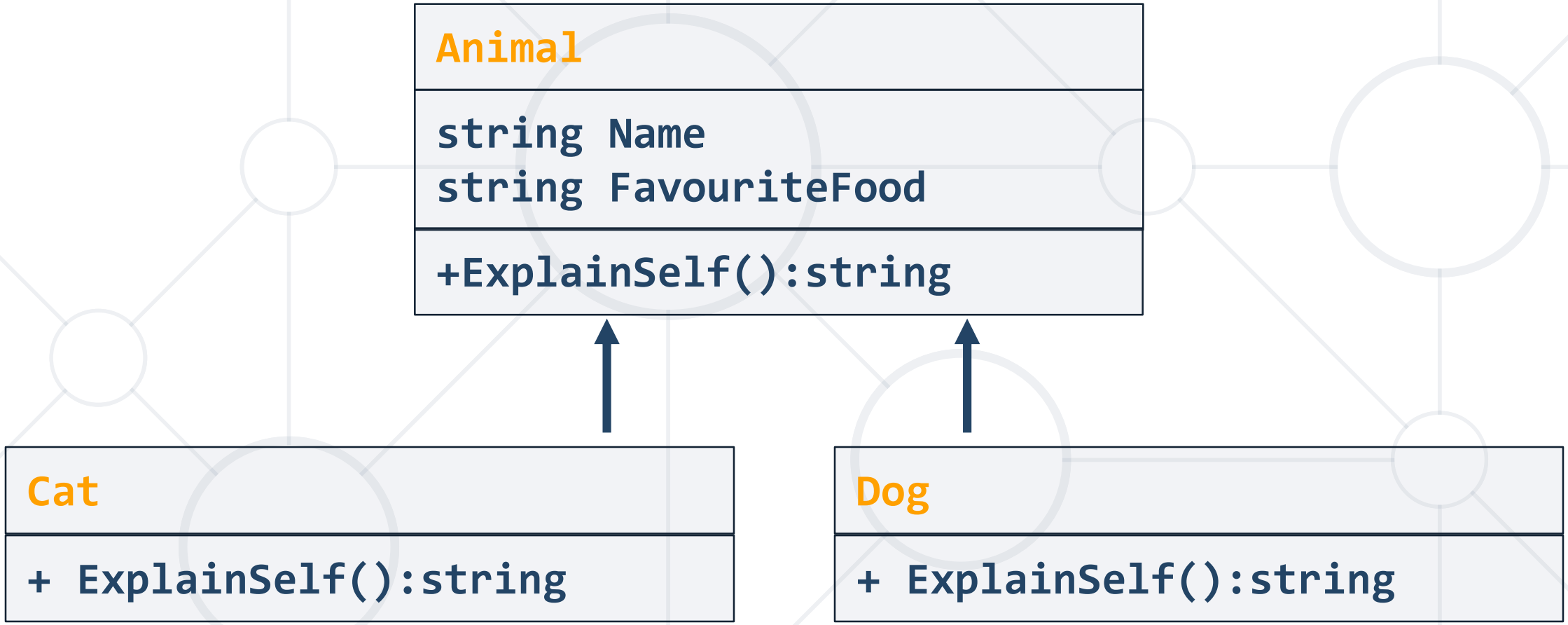
- Usage of **override** method

```
public static void Main()
{
  Rectangle rect = new Rectangle(3.0, 4.0);
  Rectangle square = new Square(4.0);

  Console.WriteLine(rect.Area()); // 12.0
  Console.WriteLine(square.Area()); // 16.0
}
```

Method overriding

Problem: Animals



Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1503#1

Solution: Animals (1)

```
public abstract class Animal {
  // Create Constructor
  public string Name { get; private set; }
  public string FavouriteFood { get; private set; }
  public virtual string ExplainSelf() {
    return string.Format(
      "I am {0} and my favourite food is {1}",
      this.Name,
      this.FavouriteFood);
  }
}
```

Solution: Animals (2)

```
public class Dog : Animal
{
  public Dog(string name, string favouriteFood)
    : base(name, favouriteFood) { }
  public override string ExplainSelf()
  {
    return base.ExplainSelf() +
    Environment.NewLine +
    "BARK";
  }
}
```

Solution: Animals (3)

```
public class Cat : Animal
{
  public Cat(string name, string favouriteFood)
    : base(name, favouriteFood) { }
  public override string ExplainSelf()
  {
    return base.ExplainSelf() +
    Environment.NewLine +
    "MEOW";
  }
}
```

Rules for Overriding Method

- **Overriding** must take place in any sub-classes
- The overriding method and the base must have the **same return type** and the **same signature**
- Base method must have the `virtual` keyword
- Overriding method must have the `abstract` or `override` keyword
- **Private and static** methods **cannot** be overridden
- **Virtual** members can use **`base` keyword** to call the **base class**

Virtual Members

- Virtual members **remain virtual indefinitely**
- A derived class can stop virtual inheritance by declaring an override as `sealed`
  - Sealed methods can be replaced by derived classes by using the `new` keyword
- The `override` modifier extends the base class virtual method
  - The `new` modifier hides an accessible base class method

Summary

- Polymorphism - **Definition** and **Types**
- **is** Keyword
- **as** Keyword
- Overload Methods
- Override Methods

Questions?

SoftUni

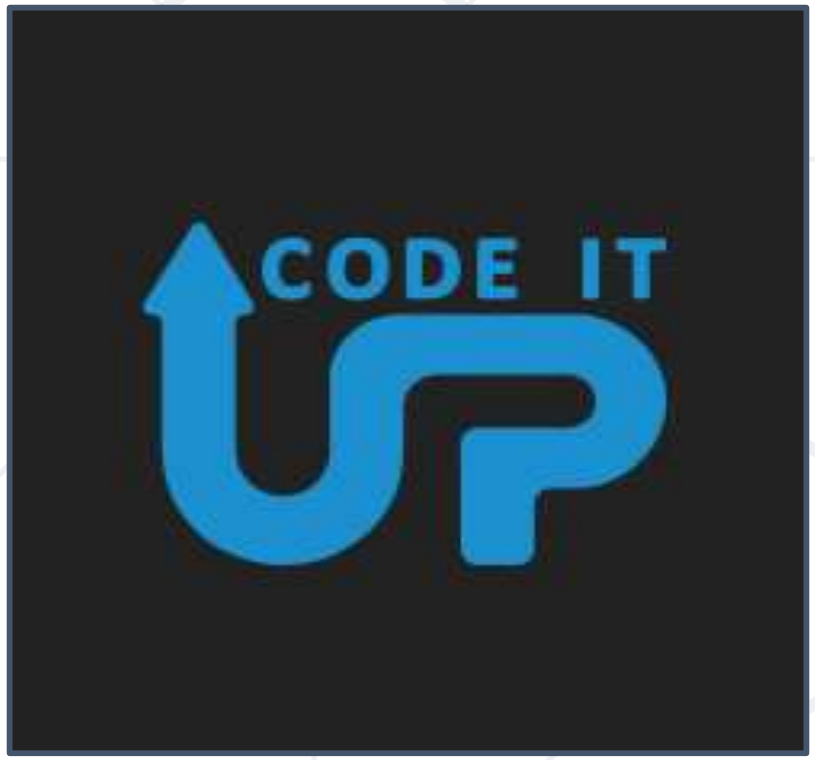
Software University | SoftUni Svetlina | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids

SoftUni Diamond Partners

Software University

Educational Partners

Software University



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg