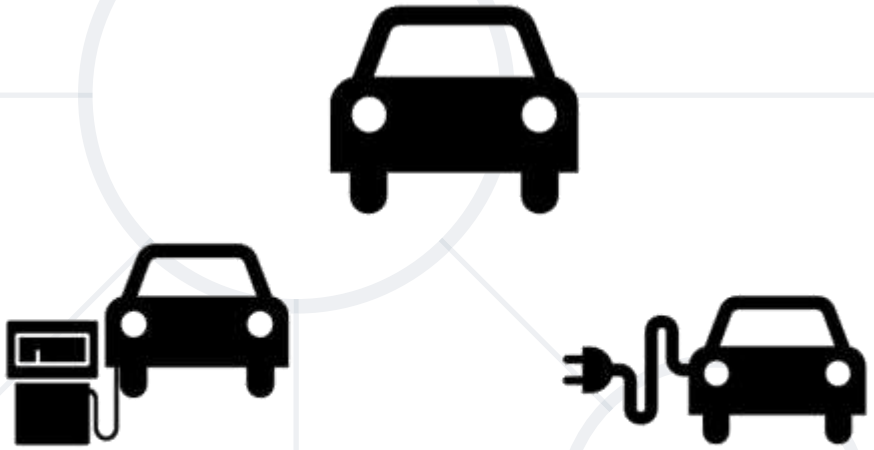
# Полиморфизъм



ONE NAME FOR MANY FORMS


**SoftUni Team**

**Technical Trainers**

Software University

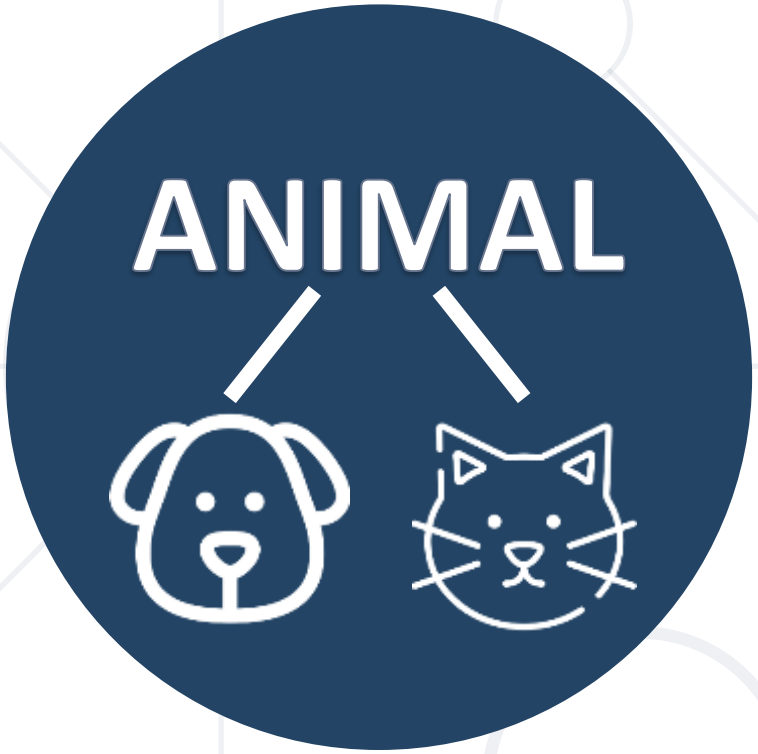
SoftUni

Software University

https://softuni.bg

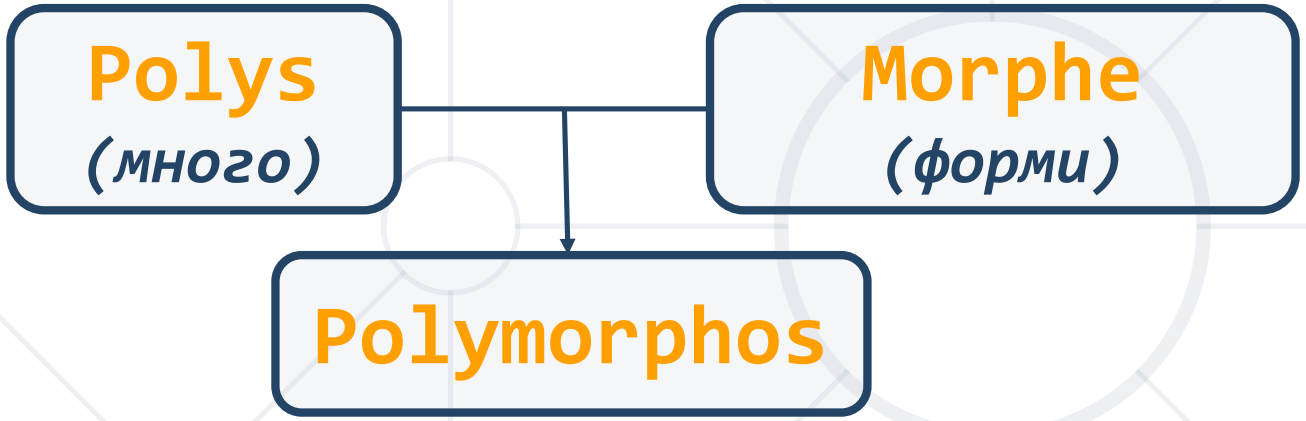
# Съдържание

1. Полиморфизъм
2. Операторите Is и As
3. Видове полиморфизъм
4. Полиморфизъм по време на компилация – презареждане на методи (overload)
5. Полиморфизъм по време на изпълнение– презаписване на методи (override)




# Полиморфизъм

# Какво е полиморфизъм?





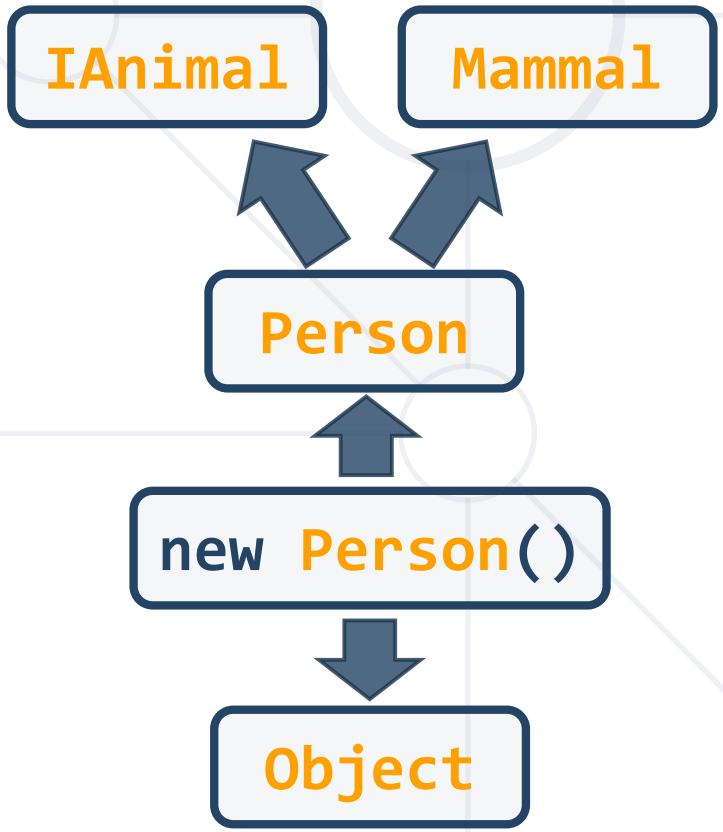
- Полиморфизмът е гръцка дума, която означава “**едно име, много форми**"

# Полиморфизъм в ООП

- Способността на един **обект** да приема **много различни форми**
- Позволява ни да третираме обекти на **производния клас** като обекти на **базовия клас**

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

| | |
|---|---|
| `Person IS-AN Object` | `Person IS-A Mammal` |
| `Person IS-A Person` | `Person IS-AN Animal` |



# Референтен тим и обектен тип

- **Променливите** се запазват в **референтен** тип
- Можете да използвате **само референтни методи**
- Ако се нуждаете от **обектен метод**, можете да го **конвертирате или да го презапишете**

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

**Референтен тип**

**Обектен тип**


# Промяна на типа и проверка на съвместимостта

# Ключовата дума is

- Проверете дали даден **обект** е **инстанция** на конкретен **клас**

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

Проверете обектния тип на person

Конвертираме към обектен тип и използваме неговите методи

# IS Type Pattern

- **is type pattern** – тества дали изразът може да бъде **конвертиран** към специфичен тип **и го конвертира** към променлива от този тип

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

Проверява дали обектът е от тип person и го конвертира

Използва неговите методи

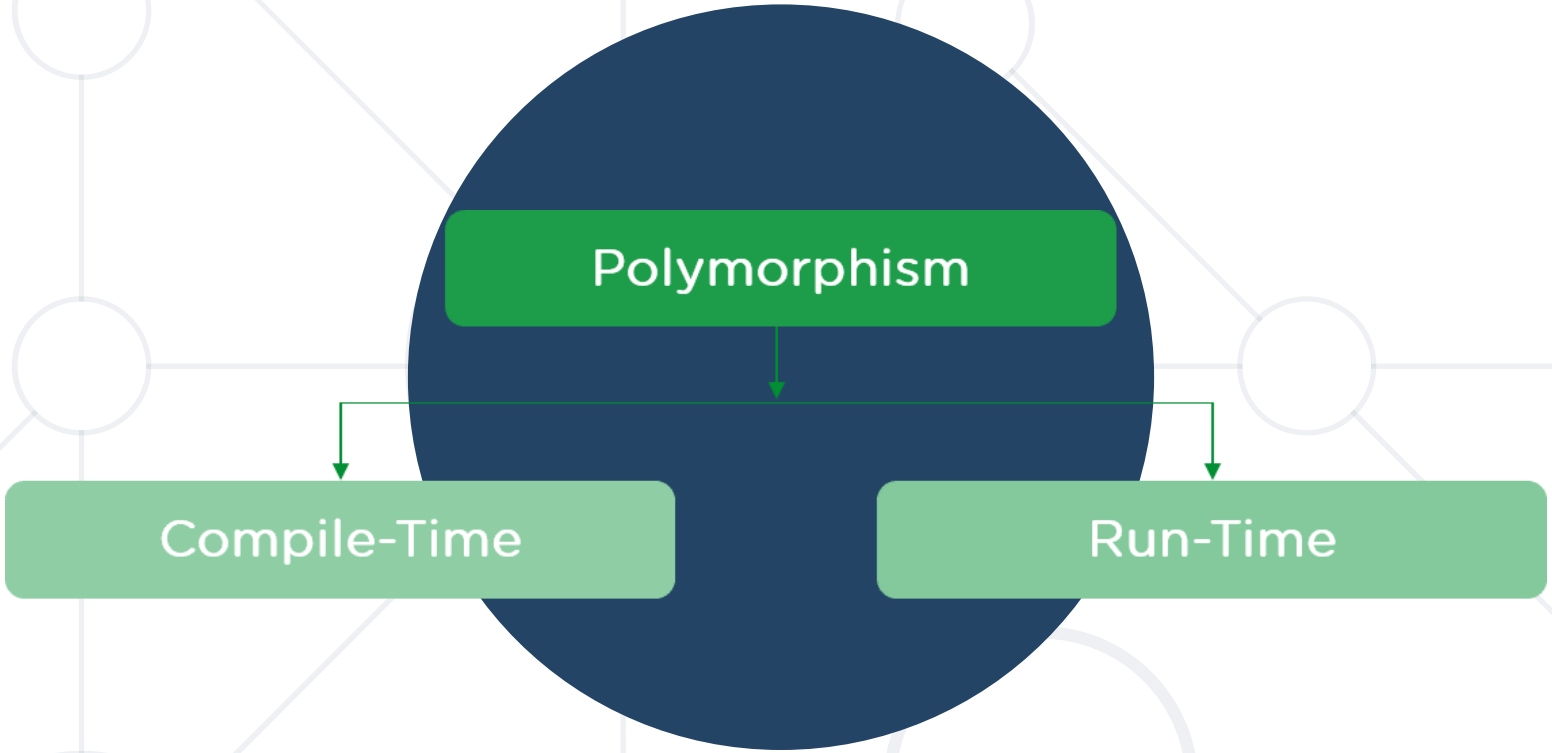
# Ключовата дума As

- Операторът **as** се използва за **конвертиране** между съвместими референтни типове

```
public class Person : Mammal, Animal {}
Animal person = new Person();
Mammal personOne = new Person();
Person personTwo;
personTwo = personOne as Person;
if (personTwo != null)
{
  // Do something specific for Person
}
```

Конвертира Mammal към Person

Проверява дали е конвертиран успешно




# Видове полиморфизъм

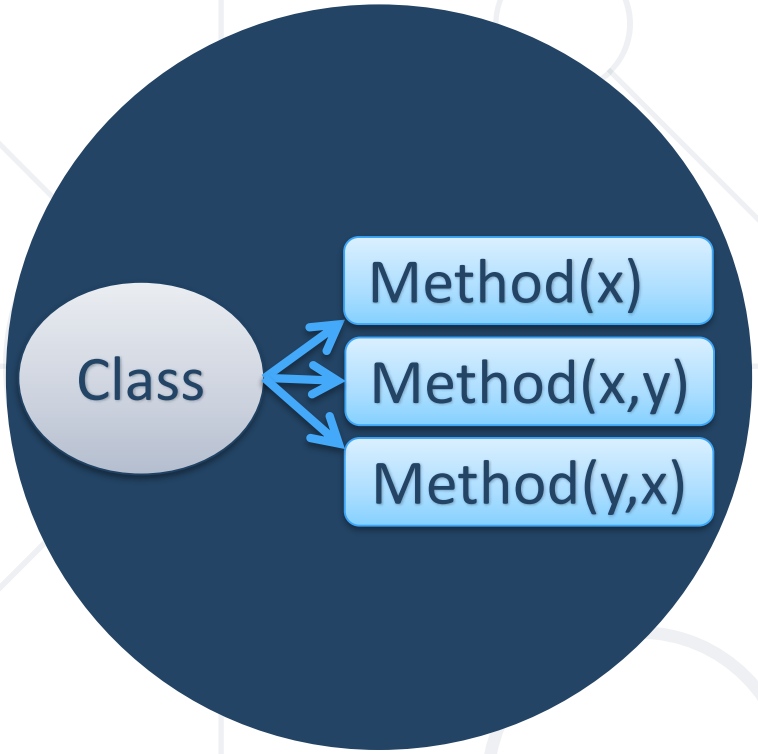
# Видове полиморфизъм

- По време на изпълнение

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

- По време на компилация

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




# Варианти на методи (overloading)

# Полиморфизъм по време на компилация

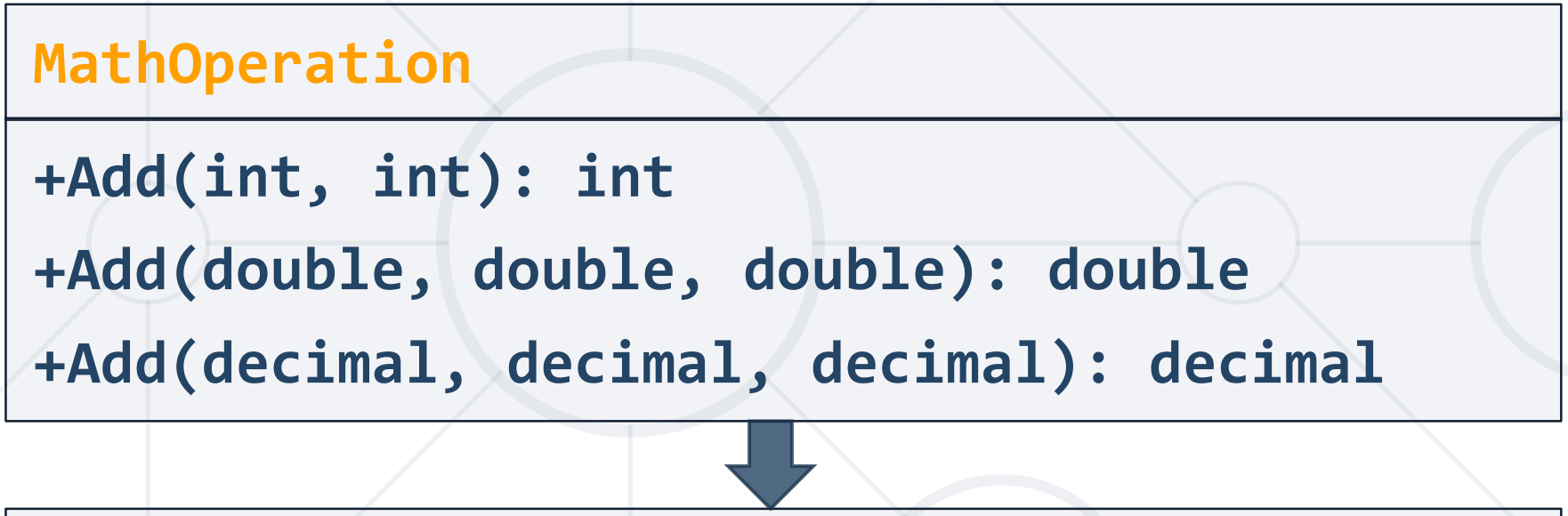
- Още познат като **статичен полиморфизъм** – реализира се чрез **overloading**

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

**Едно и също име на метода, различни имплементации**

- Списъкът с аргументи може да се различава по:
  - **Броя** на аргументите
  - **Типа** на аргументите
  - **Реда** на аргументите

# Задача: Математически операции




```
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));
```

# Решение: Математически операции

```
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;
}
```

Проверете решението си тук : https://judge.softuni.bg/Contests/Practice/Index/3167#0

# Правила при overloading методи (1)

- Сигнатурите **трябва да се различават** по един от следните показатели:
  - **Броя** на аргументите
  - **Типа** на аргументите
  - **Реда** на аргументите
- Типът на върнатата стойност **не е** част от сигнатурата
- Процесът на overloading може да се осъществи в **един и същ клас** или в неговите **подкласове**
- Конструкторите също могат да имат **различни варианти**

# Правила при overloading методи (1)

- **Различен брой** на аргументите

```
class Calculator
{
  public int Add(int a, int b) { return a + b; }
  public int Add(int a, int b, int c) { return a + b + c; }
}
```

```
static void Main(){
  Calculator calc = new Calculator();
  int sum1 = calc.Add(1, 2);
  int sum2 = calc.Add(1, 2, 3);
}
```

# Правила при overloading методи (2)

- **Различен тип** на аргументите

```
class Calculator
{
  public int Add(int a, int b) { return a + b; }
  public double Add(double a, double b) { return a + b; }
}
```

```
static void Main(){
  Calculator calc = new Calculator();
  int sum1 = calc.Add(1, 2);
  double sum2 = calc.Add(1.5, 2.1, 3.2);
}
```

# Правила при overloading методи (3)

- **Различен ред** на аргументите

```
class Guest {
string Identity(string name, int id)
  { return $"{name} + {id}"; }
string Identity(int id, string name)
  { return $"{name} + {id}"; }
}
```

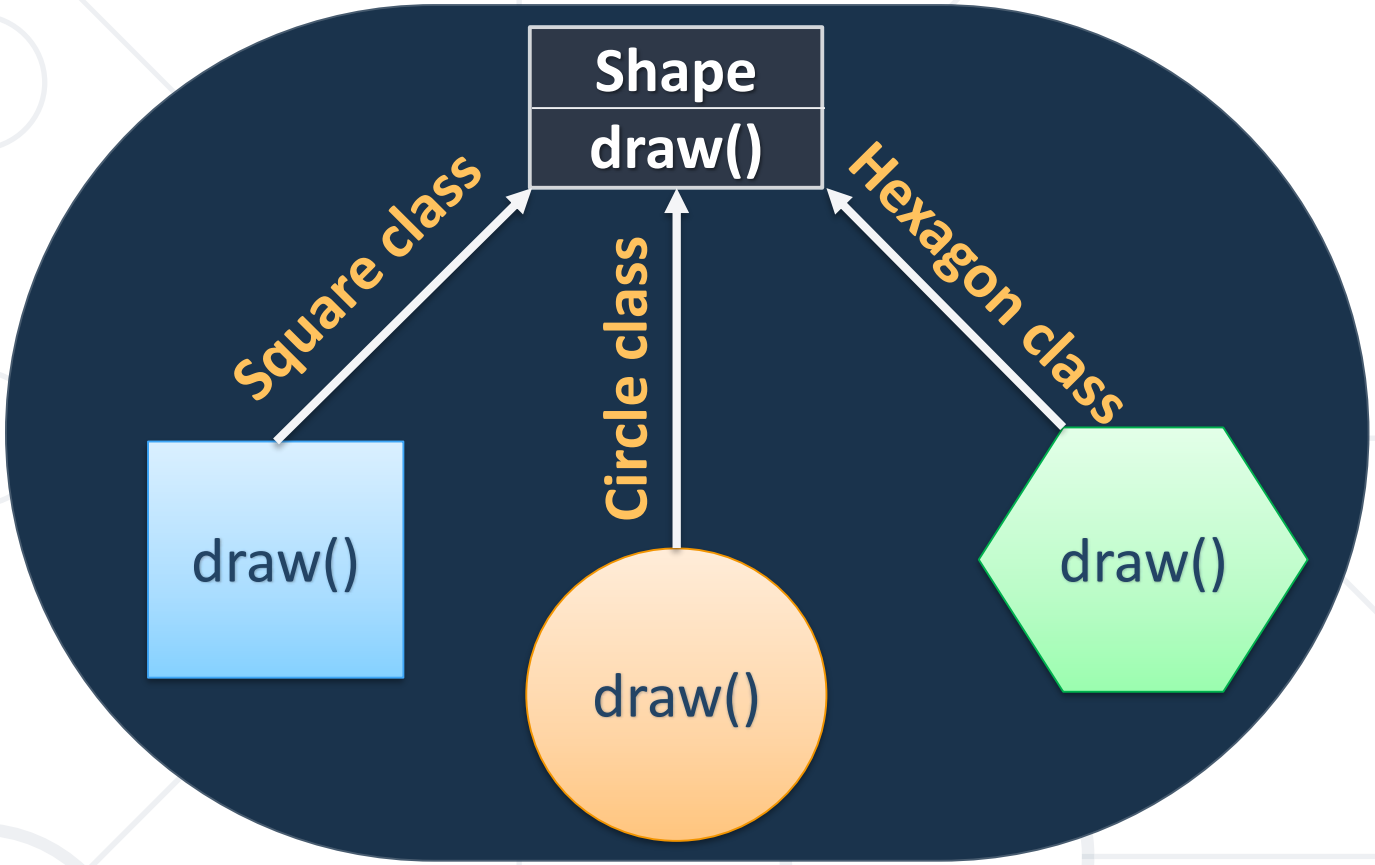
```
static void Main()
{
  Guest guest = new Guest();
  string guest1 = guest.Identity("Stephen", 15);
  string guest2 = guest.Identity(15, "Stephen");
}
```

# Една сигнатура с различен тип на return стойност

- **Не можете** да декларирате **методи** с **една и съща сигнатура**, а само с **различен тип на върнатата стойност (return)**

```
static void Print(string text)
{
  Console.WriteLine("Printing");
}

static string Print(string text)
{
  return "Printing";
}
```




# Презаписване (overriding)

# Полиморфизъм по време на изпълнение (1)

- Известен още като **динамичен полиморфизъм** – реализира се чрез презаписване на метод на базовия клас с ключовата дума `virtual` или `override`

```
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;
  }
}
```

Собствена дефиниция и имплементация

# Полиморфизъм по време на изпълнение (2)

- Използване на **override** метод (презаписване)

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

  Console.WriteLine(rect.Area());
  Console.WriteLine(square.Area());
}
```

Презаписване на метода

# Полиморфизъм по време на изпълнение (3)

- По време на изпълнение, обекти от **производния клас** може да бъдат третирани като обекти от **базовия клас**
- Когато това се случва, **декларираният тип на обекта** вече не е идентичен с неговия **run-time тип**

```
public class Animal
{
…
}
```

```
public class Cat : Animal
{
…
}
```

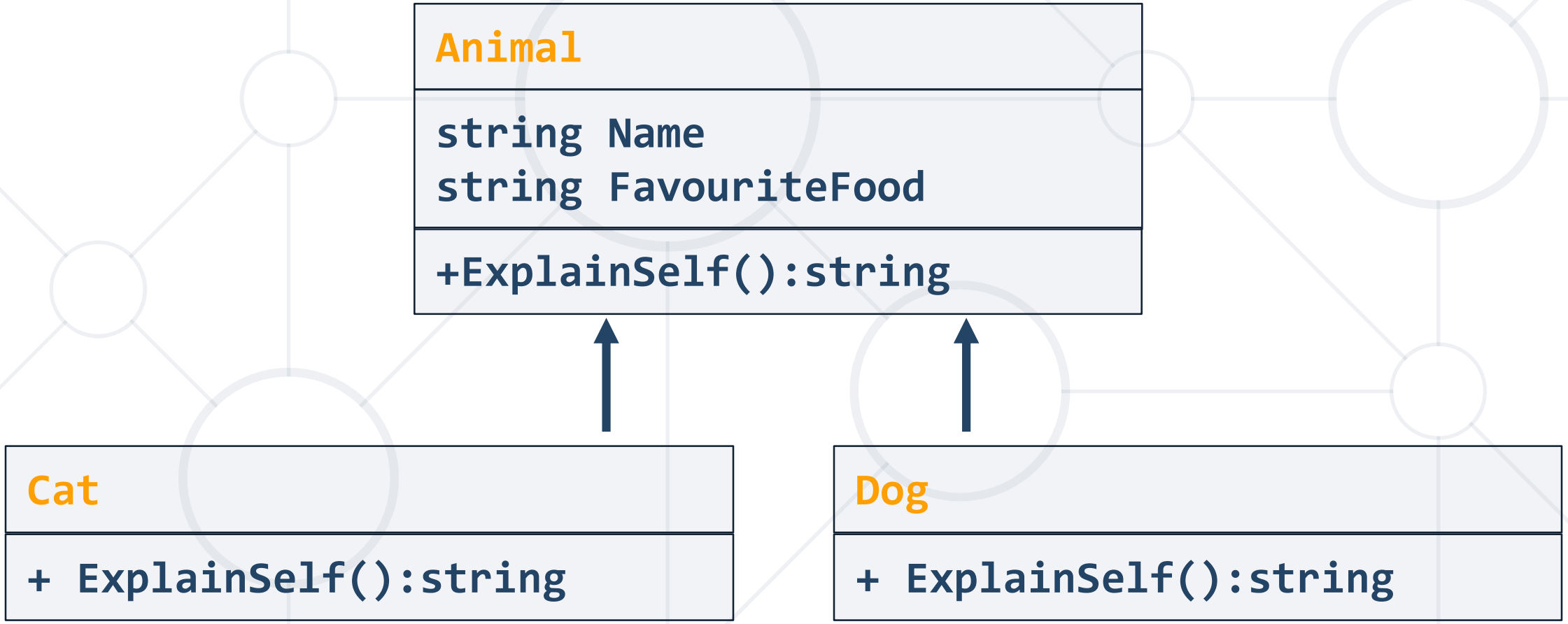
```
Animal cat = new Cat();
```

Декларирай тип

run-time тип

# Задача: Животни

- Имплементирайте следната йерархия от класове:



# Решение: Животни (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);
  }
}
```

# Решение: Животни (2)

```
public class Dog : Animal
{
  public Dog(string name, string favouriteFood)
    : base(name, favouriteFood) { }

  public override string ExplainSelf()
  {
    return base.ExplainSelf() +
    Environment.NewLine +
    "BARK";
  }
}
```

# Решение: Животни (3)

```
public class Cat : Animal
{
  public Cat(string name, string favouriteFood)
    : base(name, favouriteFood) { }

  public override string ExplainSelf()
  {
    return base.ExplainSelf() +
    Environment.NewLine +
    "MEOW";
  }
}
```

Проверете решението си тук: https://judge.softuni.bg/Contests/Practice/Index/3167#1

# Правила за презаписване на метод (overriding) (1)

```
public class Rectangle
{
  public virtual double Area()
  {
    return a * b;
  }
}
```

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

Виртуален метод в базовия клас

Еднаква върната стойност и сигнатура

override или абстрактен метод в подкласа

- **Частни** и **статични** методи **не могат** да бъдат презаписани

# Правила за презаписване на метод (overriding) (2)

- **Виртуалните** членове използват **ключовата дума base**, за да извикат **базовия клас**

```
class Bird
{
  public virtual void Fly()
  {
    Console.Write("Flying");
  }
}
```

```
class Swallow : Bird
{
  public override void Fly()
  {
    base.Fly();
    Console.WriteLine("Hunt");
  }
}
```

Разширява виртуалния метод на базовия клас

Може да добави ново поведение

# Правила за презаписване на метод (overriding) (3)

- Производен клас **може да спре виртуалното наследяване** като декларира override като `sealed` („запечатан“)

```
class Penguin : Bird
{
  public sealed override void Fly() {}
}
```

```
class NewTypePenguin : Penguin
{
  public new void Fly()
  {
    base.Fly();
  }
}
```

- Sealed методите могат да бъдат заместени от производните класове с ключовата дума `new`
- Модификаторът `new` скрива достъпен метод на базовия клас

# Обобщение

- Полиморфизъм – способността на един обект да приема много форми
- Видове полиморфизъм:
  - По време на компилация
    - Осъществява се чрез overloading (презареждане) – едно и също име на метода, но различна имплементация
  - По време на изпълнение
    - Осъществява се чрез overriding (презаписване) – чрез ключовите думи virtual + override

# Въпроси?

SoftUni

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

# Лиценз

- Този курс (презентации, примери, демонстрационен код, упражнения, домашни, видео и други активи) представлява **защитено авторско съдържание**
- Нерегламентирано копиране, разпространение или използване е незаконно
- © СофтУни – https://softuni.org
- © Софтуерен университет – https://softuni.bg