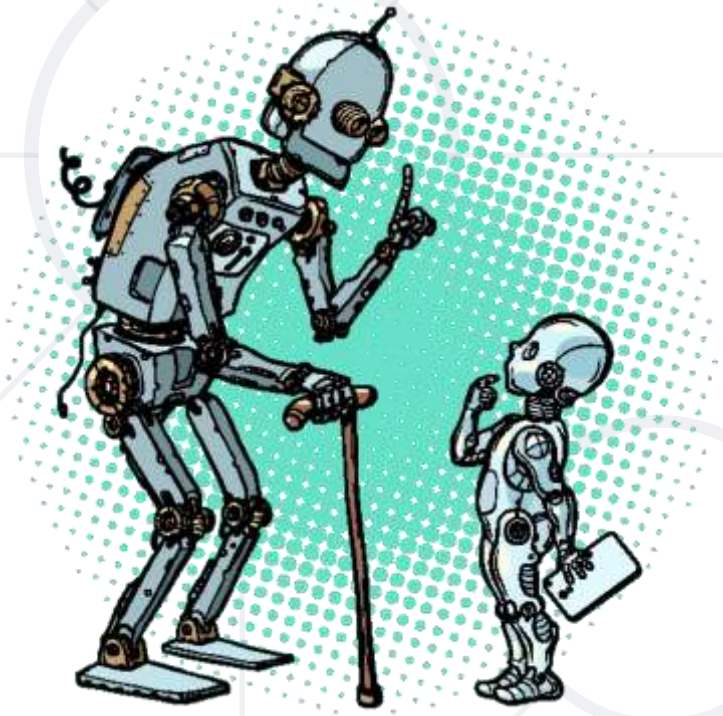
Наследяване

Йерархия на класовете



SoftUni Team

Technical Trainers

Software University

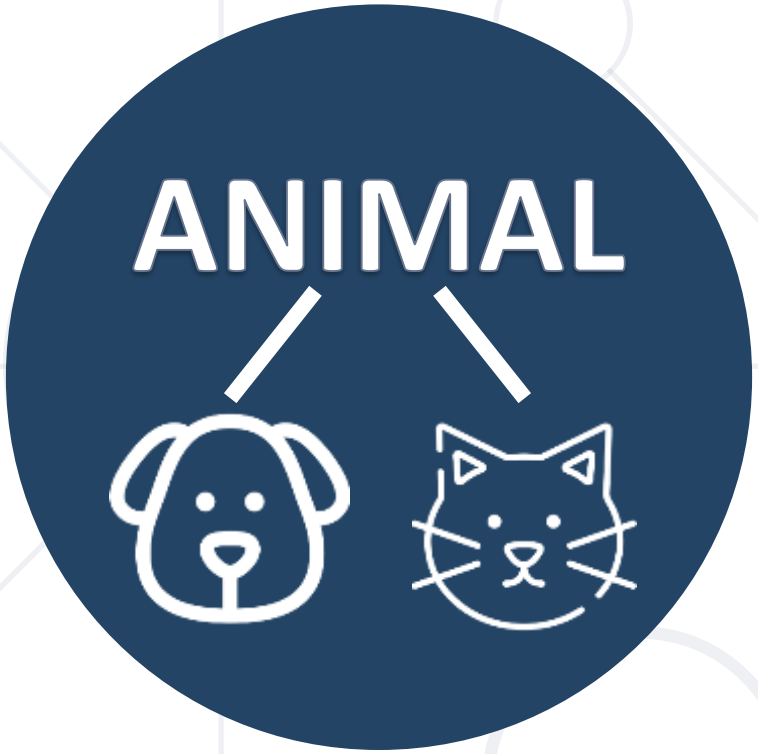
SoftUni

Software University

https://about.softuni.bg/

Съдържание

1. Наследяване
2. Йерархия на класовете
3. Достъп до базови членове на класа
4. Преизползване на класове
5. Видове преизползване на класове
6. Хвърляне на изключения

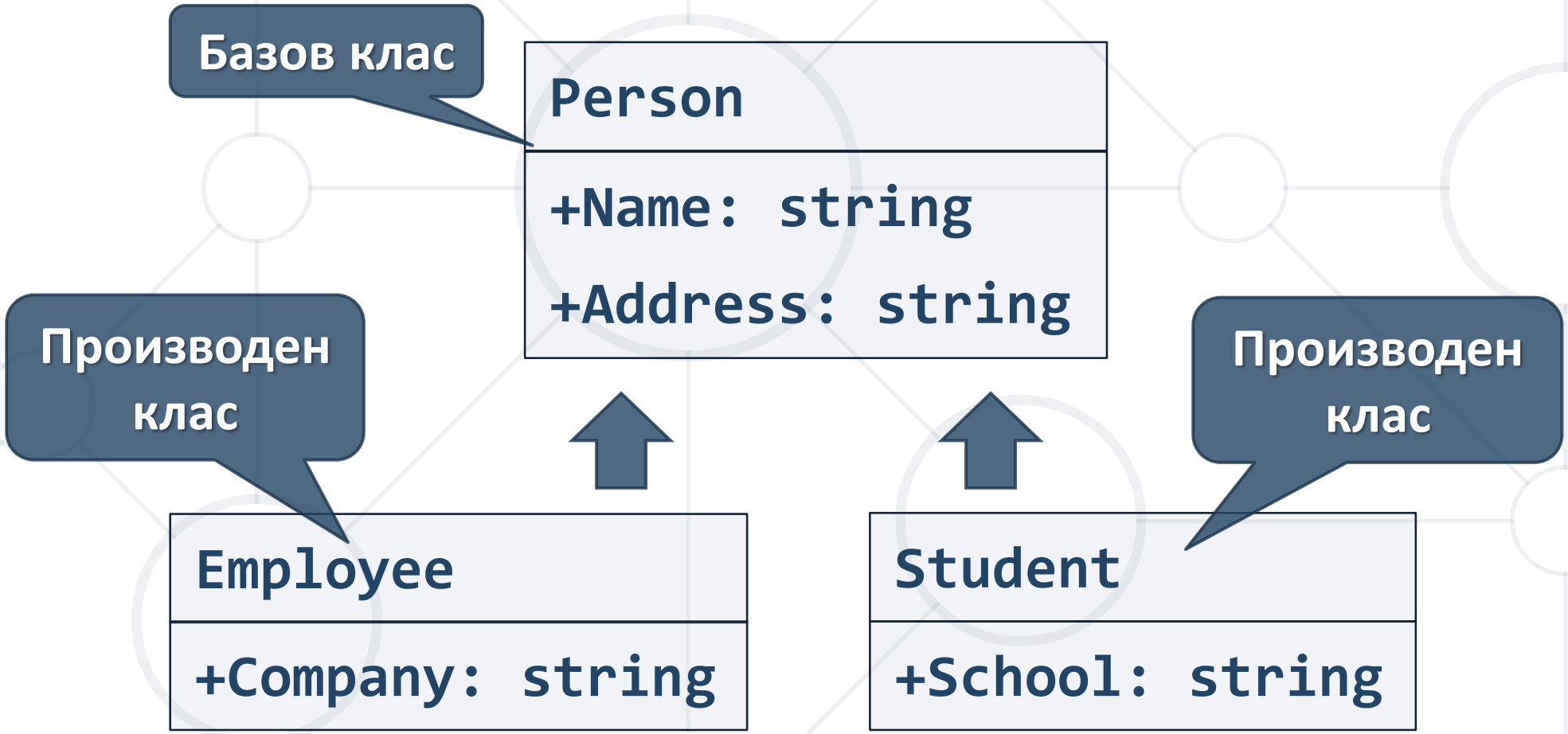


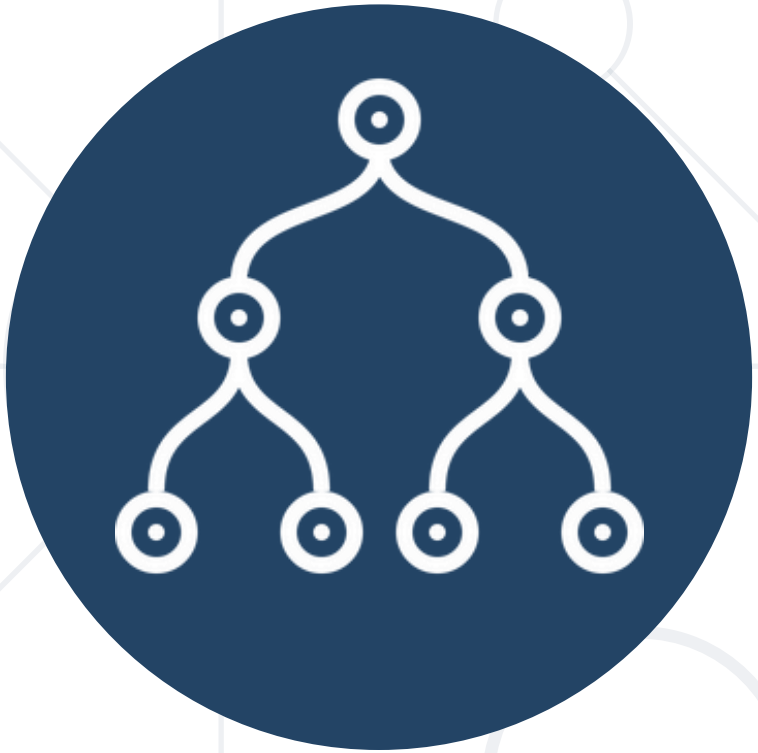
Разширяване на класове

Наследяване

- **Суперклас (superclass)** – родителски (Parent) class, базов (Base) Class
  - Класът предава своите **членове** на **класа дете (Child)**
- **Подклас (subclass)** – **Child class (дете/дъщерен клас)**, **Derived class (производен)**
  - Класът взима членовете си от базовия клас

Наследяване – пример

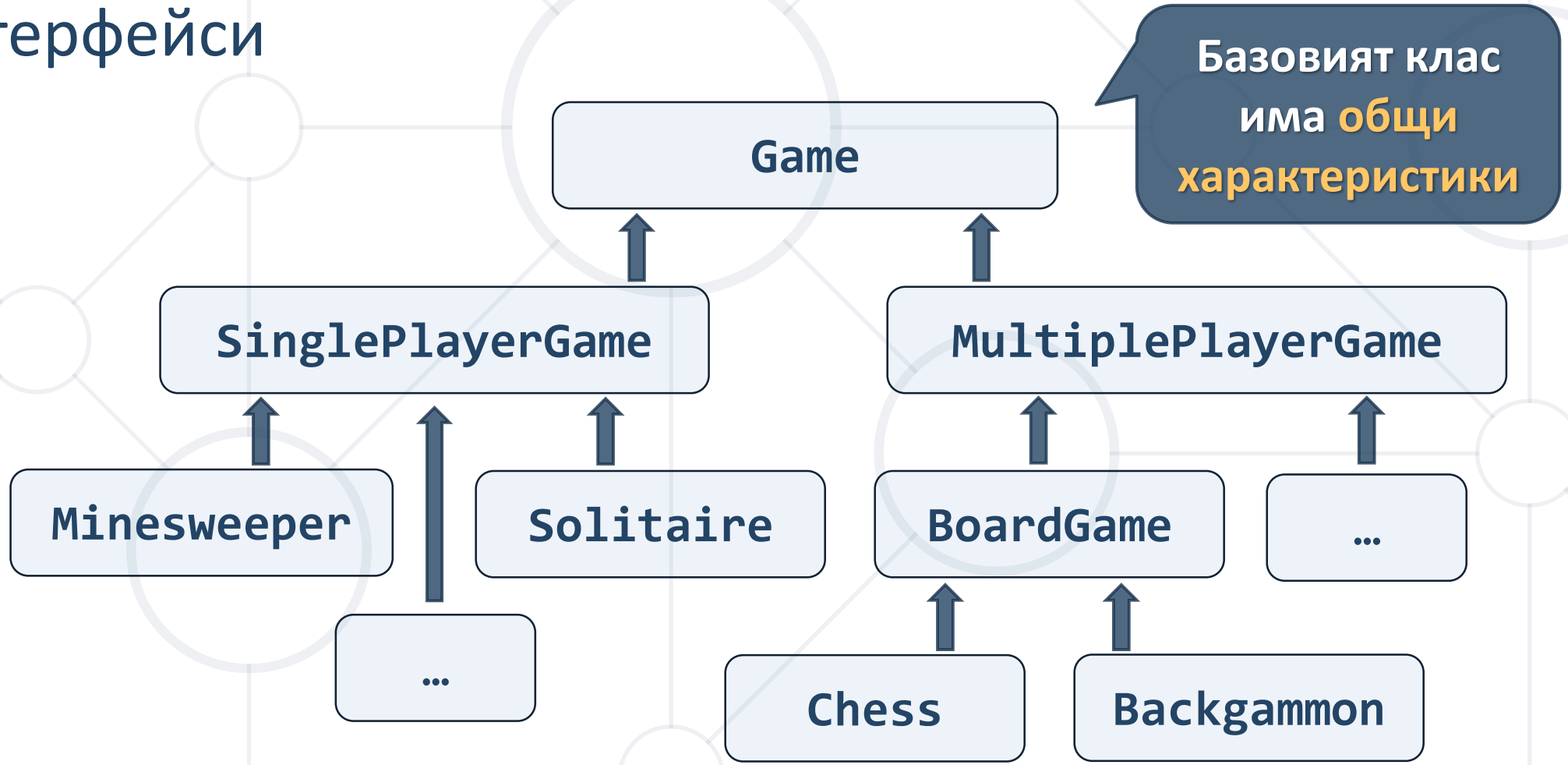




Наследяването създава йерархия

Класови йерархии

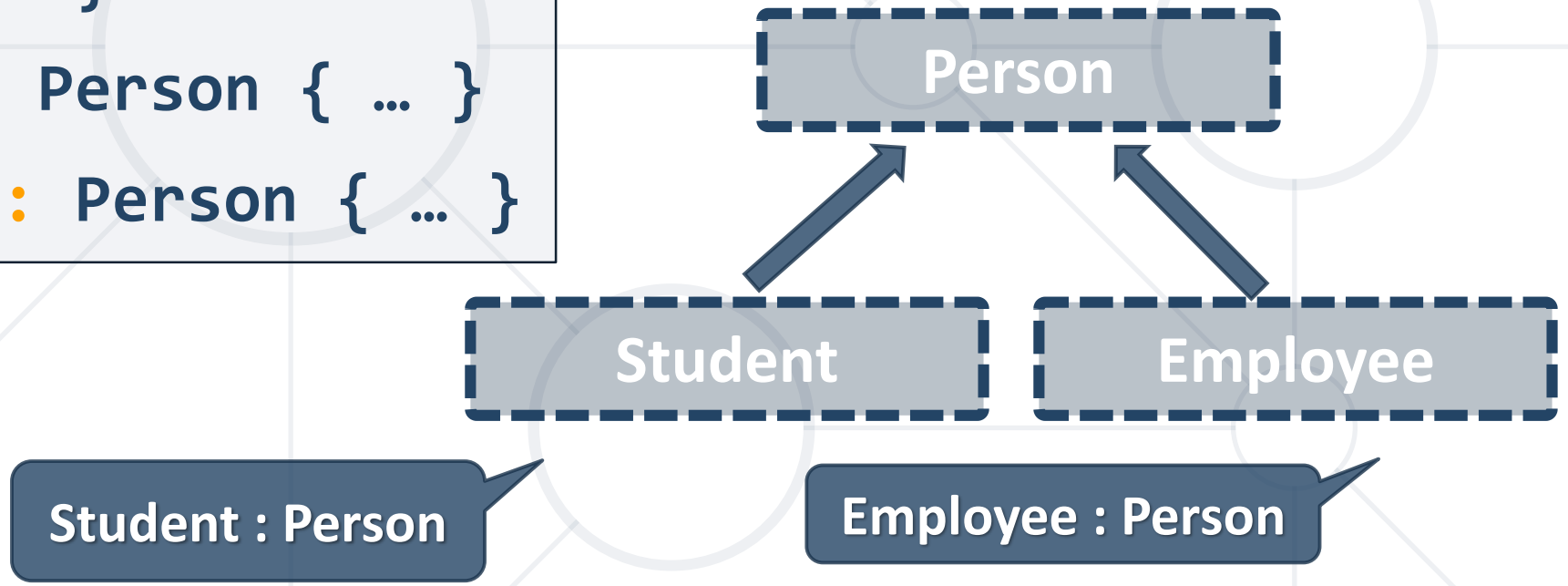
- **Наследяването** води до **йерархии** от класове и/или интерфейси



Наследяване в C#

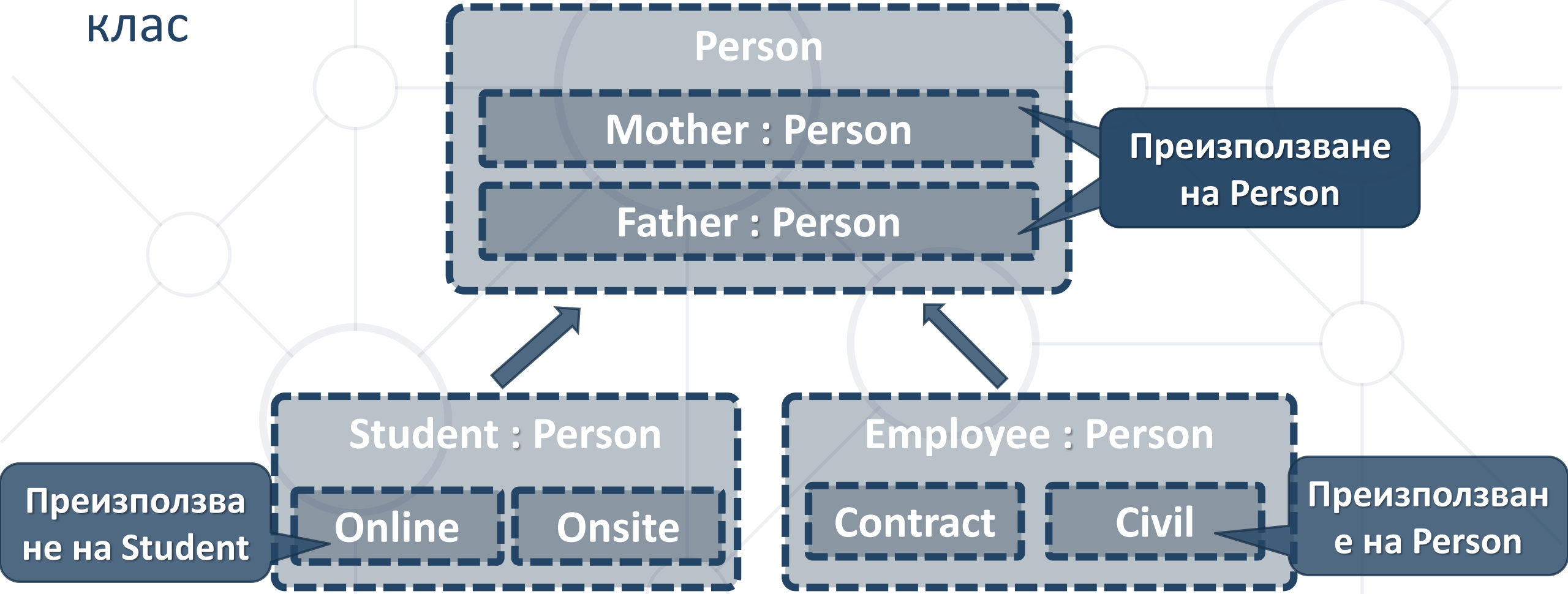
- In C# inheritance is defined by the : operator

```
class Person { … }
class Student : Person { … }
class Employee : Person { … }
```



Наследяване – производен клас

- Производните класове **взимат всички членове** от базовия клас

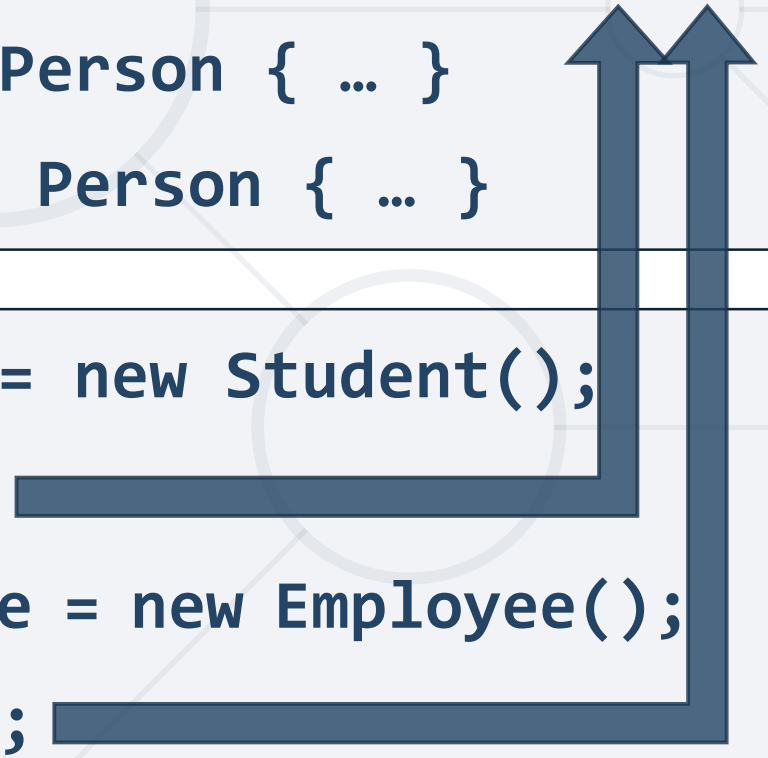


Използване на наследени членове

- Можете да достъпите наследените членове както обикновено

```
class Person { public void Sleep() { … } }
class Student : Person { … }
class Employee : Person { … }
```

```
Student student = new Student();
student.Sleep();
Employee employee = new Employee();
employee.Sleep();
```



Преизползване на конструктори

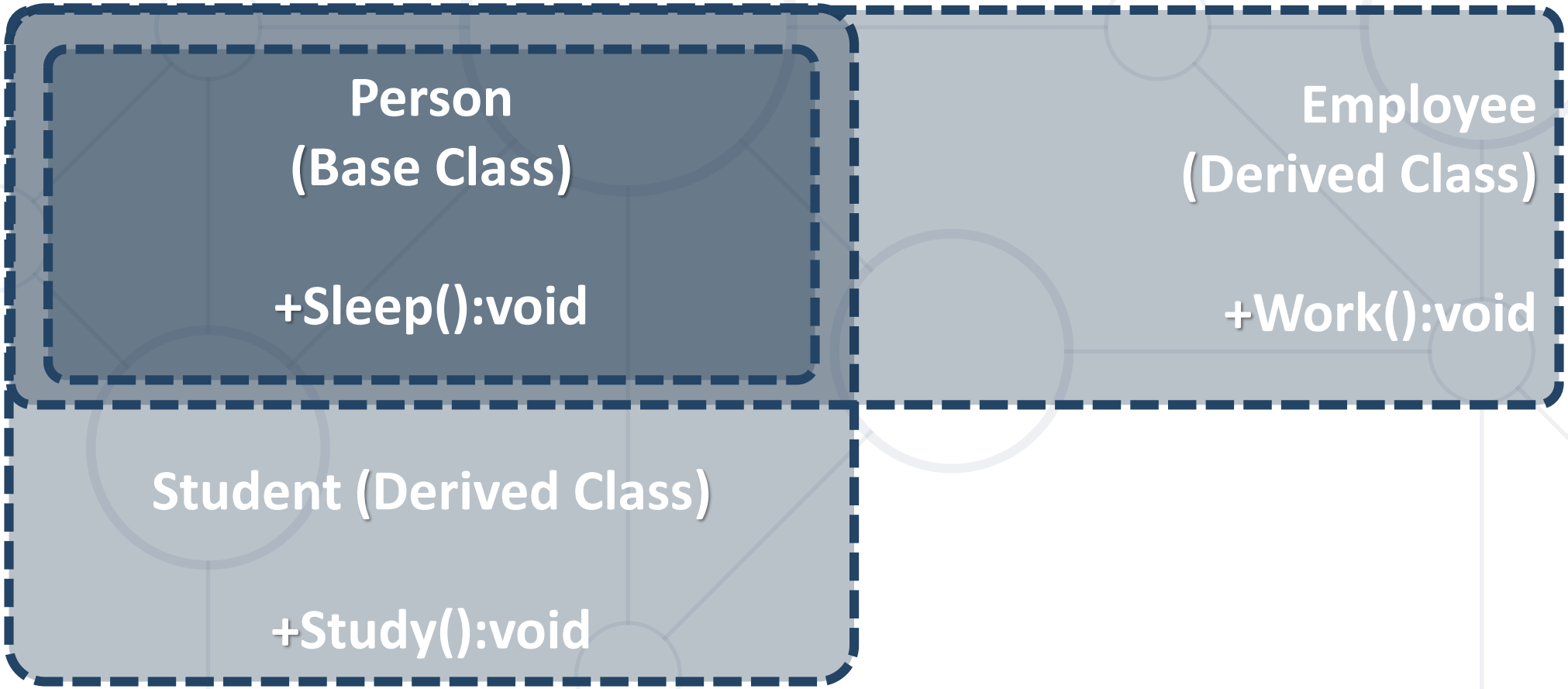
- Конструкторите **не се наследяват**
- Могат да се **преизползват** от дъщерните класове

```
class Student : Person
{
private School school;
  public Student(string name, School school)
    : base(name) {this.school = school;}
}
```

Извиква конструктора на базовия клас (parent)

Наследяване – Разширяване (Extends)

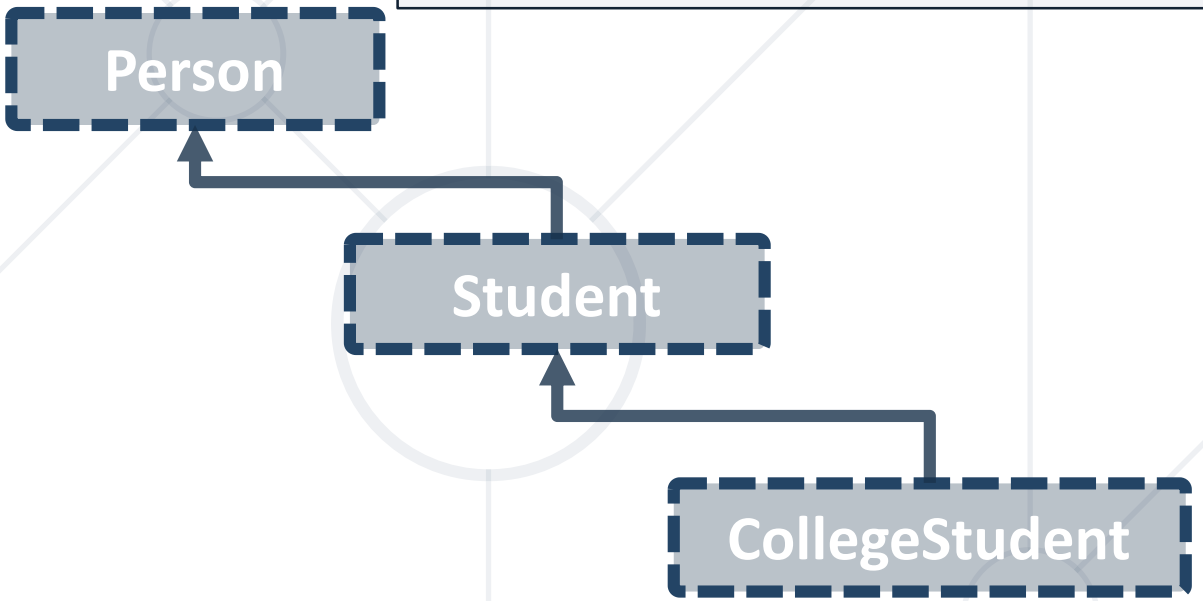
- Инстанцията на производния клас **съдържа** инстанция на базовия клас



Преходна връзка (Transitive Relation)

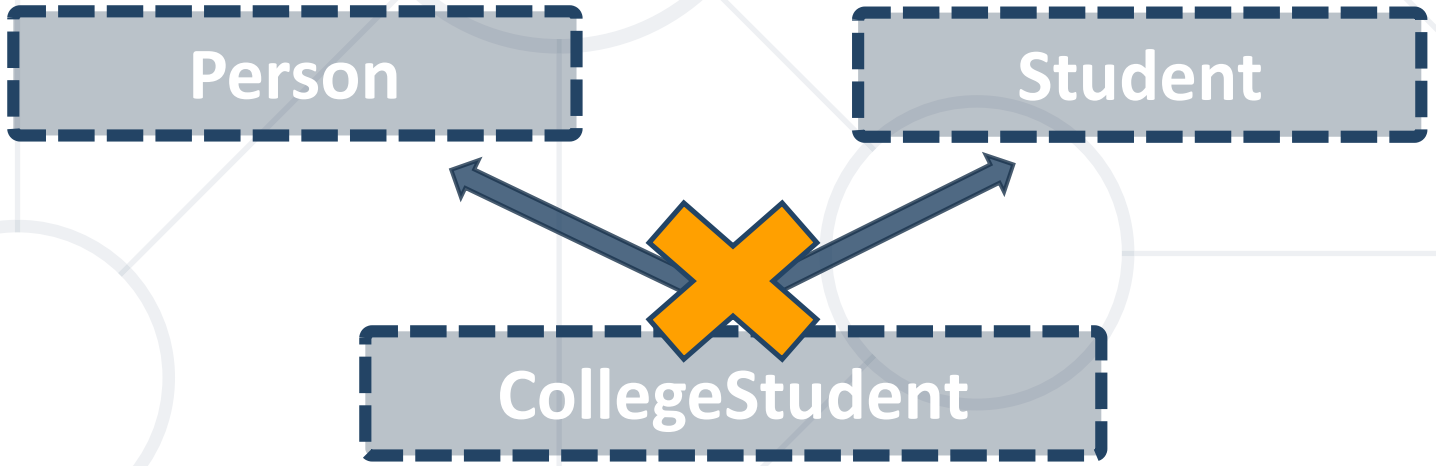
- Наследяването има **преходна връзка**

```
class Person { … }
class Student : Person { … }
class CollegeStudent : Student { … }
```



Множествено наследяване

- В C# няма **множествено** наследяване
- Само **множество интерфейси** могат да бъдат имплементирани



Ключовата дума Base

Достъп до членовете на базовия клас

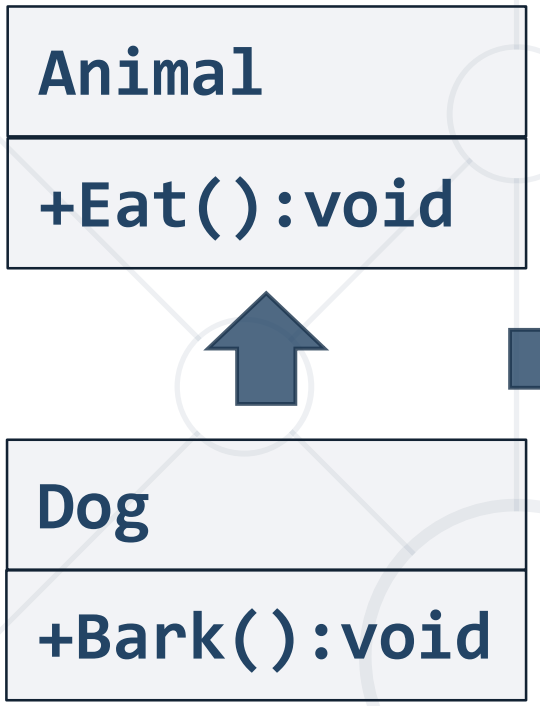
- Използвайте ключовата дума **base**

```
class Person { … }

class Employee : Person
{
  public void Fire(string reasons)
  {
    Console.Writeline($"{base.name} got fired because of {reasons}");
  }
}
```

Задача: Куче наследява животно

- Създайте два класа: **Animal** и **Dog**:



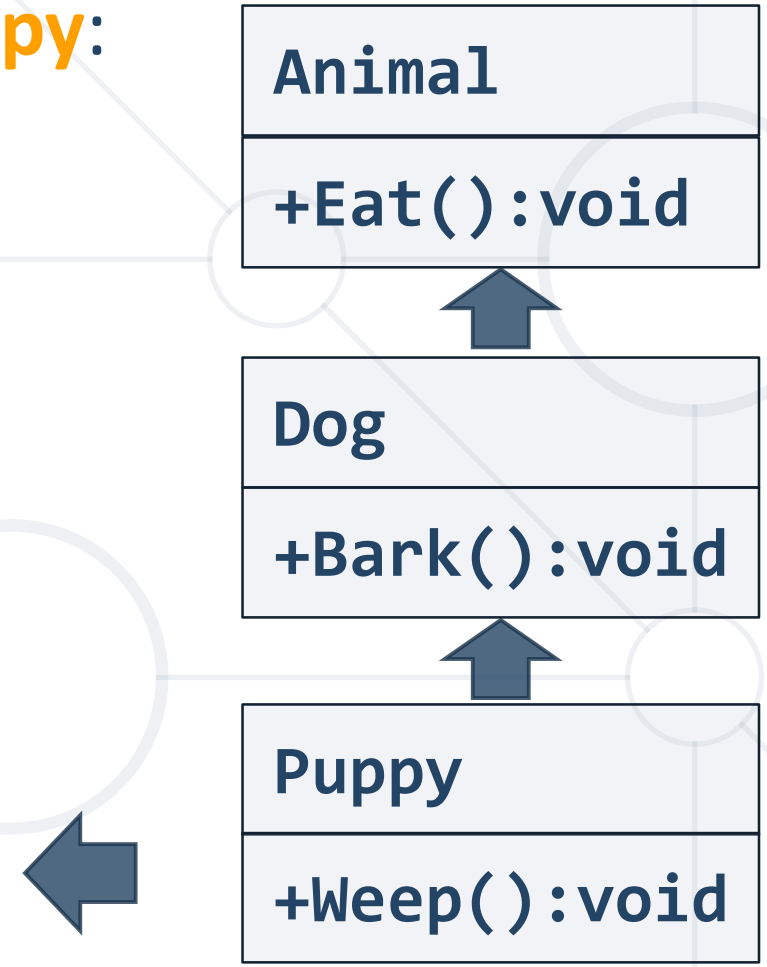

```
Dog dog = new Dog();
dog.Eat();
dog.Bark();
```

- **Animal** с метод **Eat()**, който отпечатва: **"eating..."**
- **Dog** с метод **Bark()**, който отпечатва: **"barking..."**
- **Dog** трябва да наследява **Animal**

Проверете решението си тук: https://judge.softuni.bg/Contests/Practice/Index/3164#0

Задача: Верижно наследяване

- Създайте класовете **Animal**, **Dog** и **Puppy**:
- **Dog** трябва да наследи **Animal**
- **Puppy** трябва да наследи **Dog**

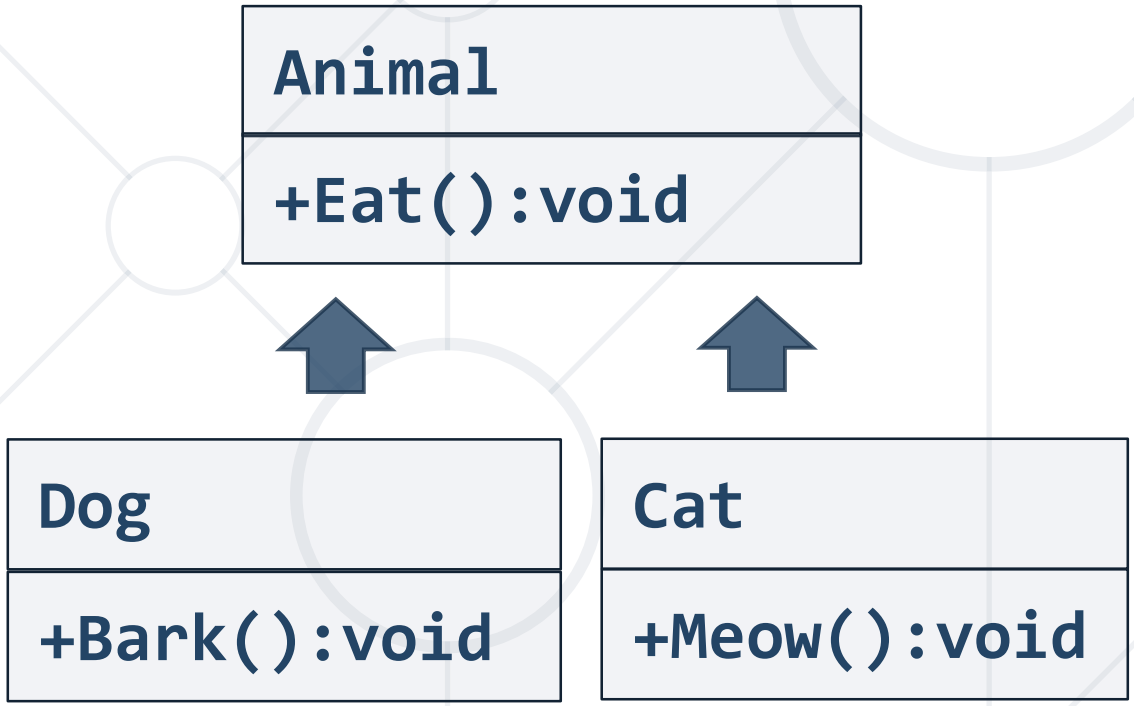



```
Puppy puppy = new Puppy();
puppy.Eat();
puppy.Bark();
puppy.Weep();
```

Проверете решението си тук: https://judge.softuni.bg/Contests/Practice/Index/3164#1

Задача: Наследствена йерархия

- Създайте класовете **Animal**, **Dog** и **Cat**:
- **Dog** и **Cat** трябва да наследят **Animal**




```
Dog dog = new Dog();
dog.Eat();
dog.Bark();

Cat cat = new Cat();
cat.Eat();
cat.Meow();
```

Проверете решението си тук: https://judge.softuni.bg/Contests/Practice/Index/3164#2

Преизползване на код на ниво клас

Наследяване и модификатори за достъп

- Производните класове **имат достъп до всички публични** и **защитени** членове
- **Вътрешните** членове **могат да се достъпят в същия проект**
- **Частните** полета **не се наследяват** от подкласовете

```
class Person
{
    private string id;
    string name;
    protected string address;
    public void Sleep();
}
```

„Засенчване“ (Shadowing) на променливи

- Производните класове **могат да крият** променливи от суперкласа

```
class Person { protected int weight; }
```

```
class Patient : Person
{
  protected float weight;
  public void Method()
  {
    double weight = 0.5d;
  }
}
```

Hides int weight

Hides float weight

„Засенчване“ на променливи - достъп

- Използвайте **base** и **this**, за да уточните достъпа

```
class Patient : Person
{
  protected float weight;
  public void Method()
  {
    double weight = 0.5d;
    this.weight = 0.6f;
    base.weight = 1;
  }
}
```

Локална променлива

Член на базовия клас

Член на инстанцията

Виртуални методи

- `virtual` – дефинира метод, който **може да бъде презаписан**

```
public class Animal
{
  public virtual void Eat() { … }
}
```

```
public class Dog : Animal
{
  public override void Eat() {}
}
```

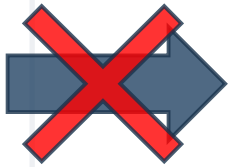
Модификатор Sealed (1)

- Модификаторът **sealed** за бранява на другите класове да **наследяват** текущия клас

```
class Dinosaur
{
   public void Eat() {…}
}
```

```
sealed class TRex : Dinosaur
{
   public void Eat() {…}
}
```

```
class EvolvedTRex : TRex
{
}
```



Модификатор Sealed (2)

- Можете да използвате модификатора **sealed** на **метод** или **свойство** в **базовия** клас:
  - Така можете да **позволите на класовете** да **наследяват** от базовия клас
  - **Забранявате презаписването** на конкретни **виртуални методи** и **свойства**

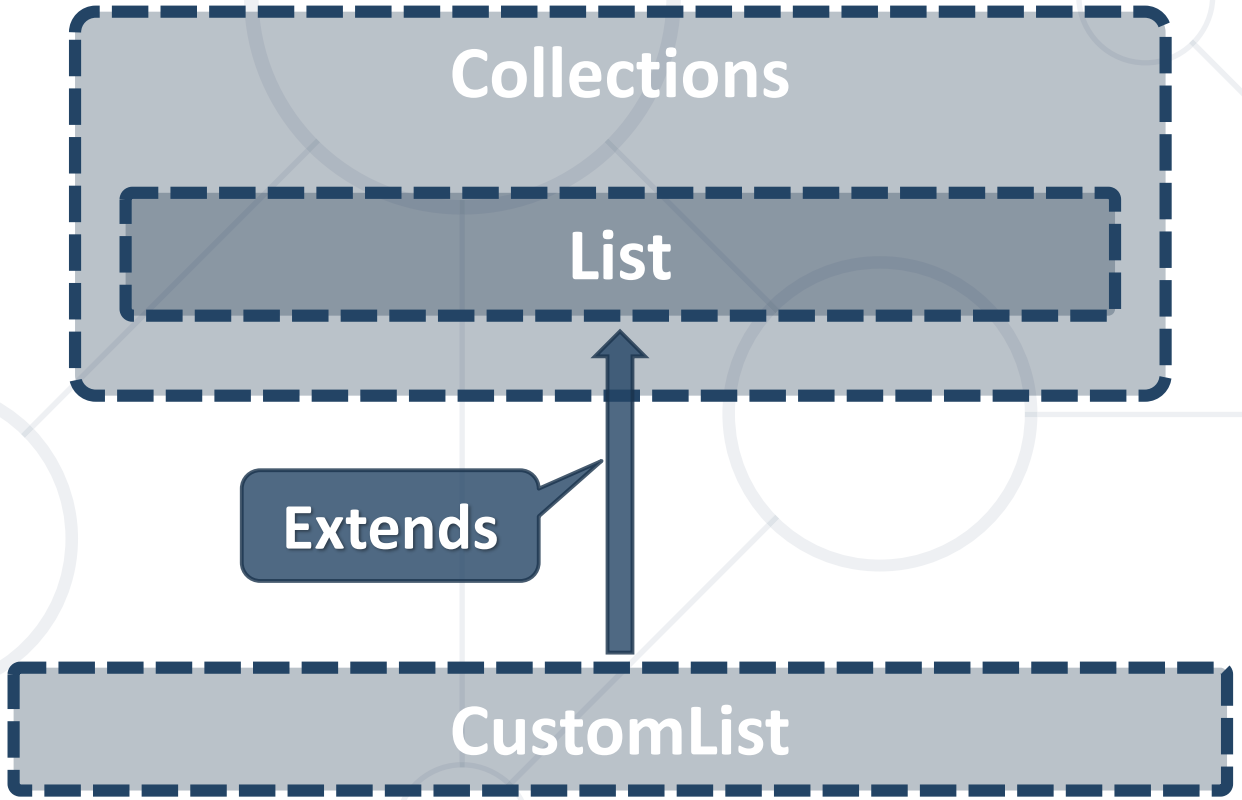
```
class Bird
{
  public virtual void Fly() {}
}
```

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

```
class Penguin : Waimanu
{
  public void Walk() {}
}
```

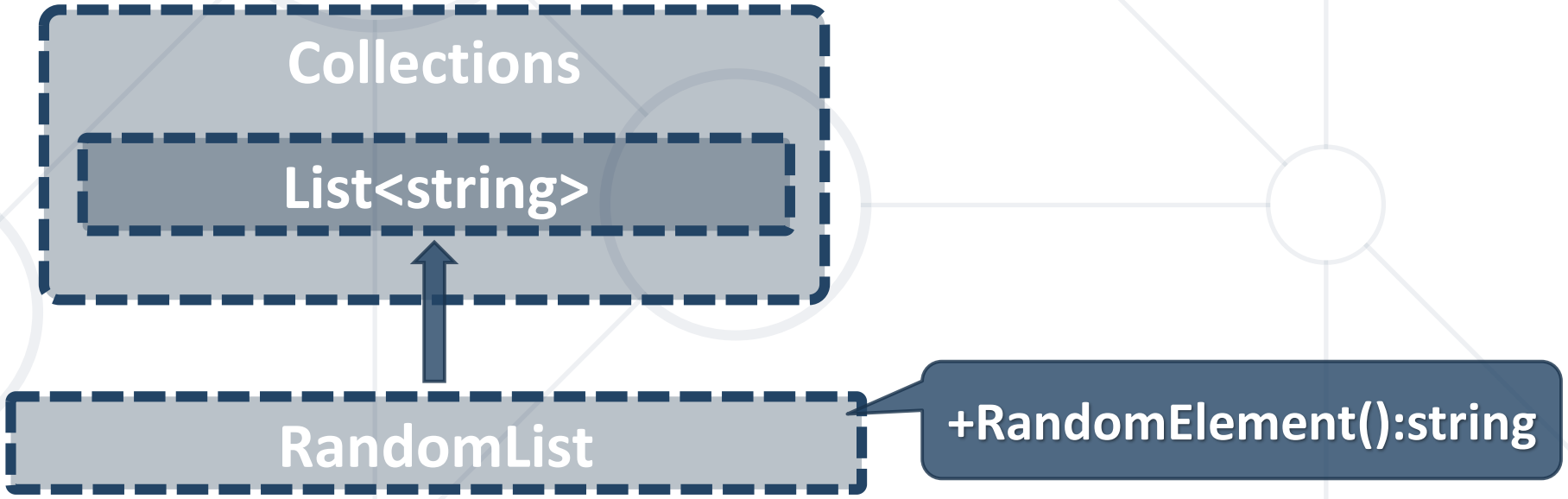
Ползи от наследяването – Разширяване

- Можем да **разширим клас**, който **иначе не можем да променима**



Задача: Случаен списък

- Създайте списък, който има
  - Всички функционалности на `List<string>`
  - Метод, който връща и премахва случаен елемент



Решение: Случаен списък

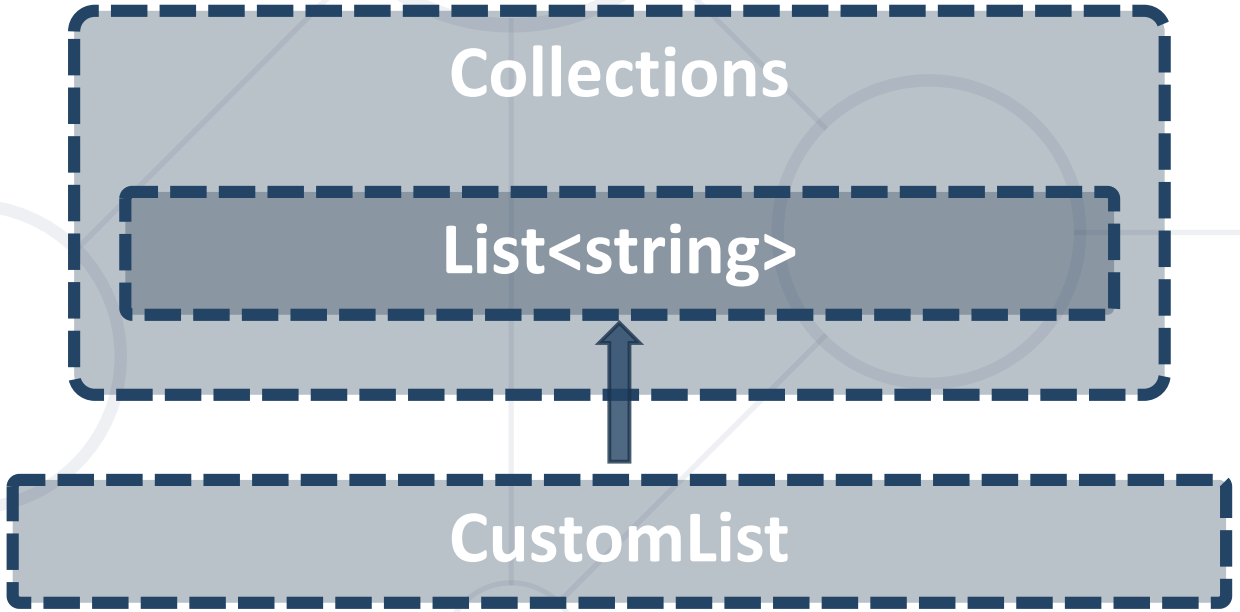
```
public class RandomList : List<string>
{
  private Random rnd; // TODO: Add constructor
  public string RemoveRandomElement()
  {
    int index = rnd.Next(0, this.Count);
    string str = this[index];
    this.RemoveAt(index);
    return str;
  }
}
```

Проверете решението си тук : https://judge.softuni.bg/Contests/Practice/Index/3164#3

Разширяване, композиция, делегиране

Разширяване

- **Повтарянето на код** води до грешки
- **Можем да преизползваме класове** чрез **разширяване**
- Понякога това е единственият начин

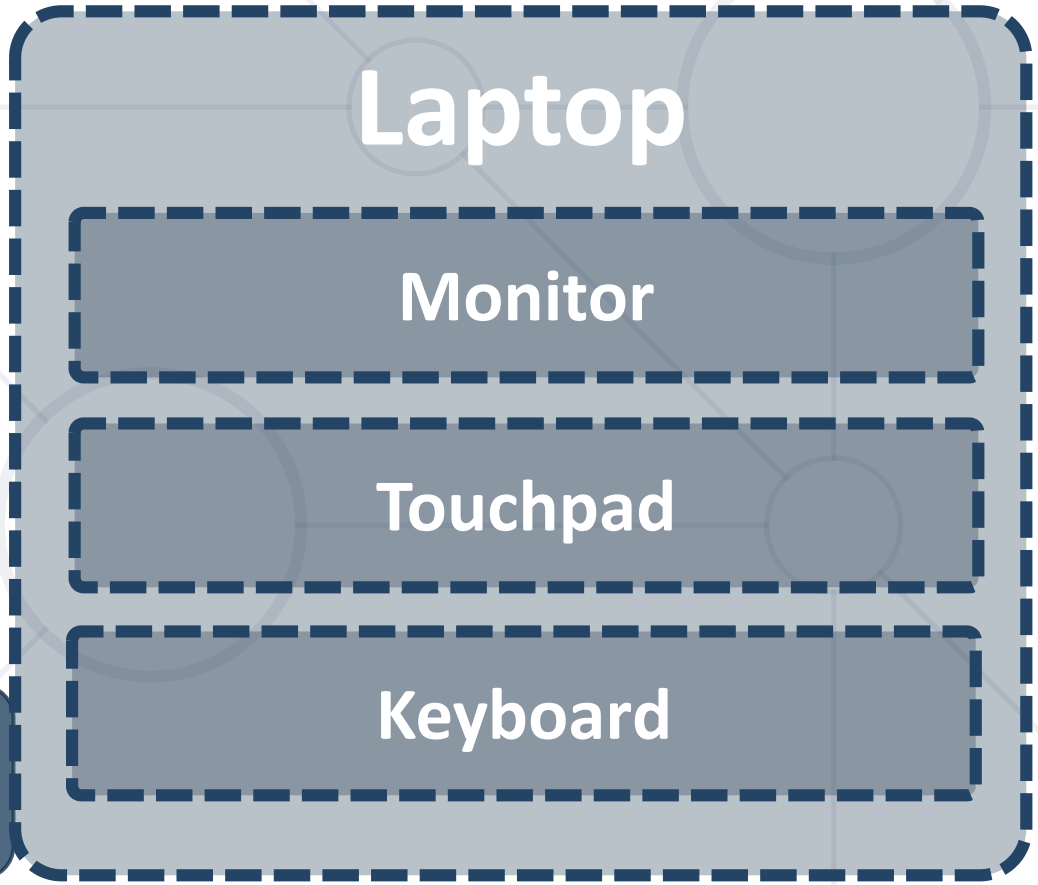


Композиция

- Използваме класове, за да **дефинираме** полета и свойства на класа

```
class Laptop
{
  Monitor monitor;
  Touchpad touchpad;
  Keyboard keyboard;
  …
}
```

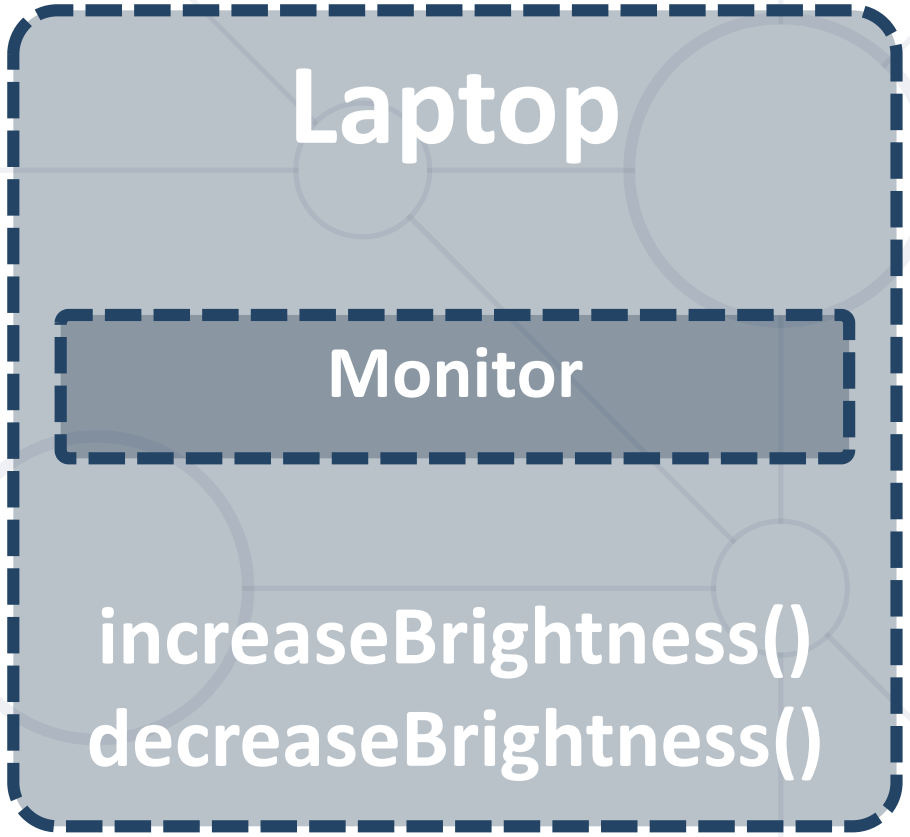
Преизползване на класове



Делегиране

```
class Laptop
{
  Monitor monitor;
  void IncrBrightness() =>
    monitor.Brighten();

  void DecrBrightness() =>
    monitor.Dim();
}
```



Задача: Поредица от стрингове

- Създайте клас **StackOfStrings**, който **наследява** **Stack<string>** и добавя следните методи:

StackOfStrings
+IsEmpty(): boolean +AddRange(elements): void

Решение: Поредица от стрингове

```
public class StackOfStrings : Stack<string>
{
  public bool IsEmpty()
  {
    return this.Count == 0;
  }
  public void AddRange(IEnumerable<string> elements)
  {
    foreach (var element in elements)
      this.Push(element);
  }
}
```

Проверете решението си тук : https://judge.softuni.bg/Contests/Practice/Index/3164#4

Ключовата дума Throw

Ключовата дума Throw

- **Хвърляне на изключение** със съобщение за грешка:

```
throw new ArgumentException("Invalid amount!");
```

- Изключенията приемат **съобщение** + **друго изключение** (причина):

```
try {
  …
}
catch (SqlException sqlEx) {
  throw new InvalidOperationException("Cannot save invoice.",
sqlEx); }
```

- Това се нарича "**верига**" от изключения

Хвърляне на изключения

- Изключения се хвърлят с ключовата дума `throw`
- Когато е хвърлено изключение:
  - Изпълнението на програмата приключва
  - Изключението се пренася по стека
    - Докато не достигне `catch` блок, който да предприеме действие

Повторно хвърляне на изключения

- Изключенията могат да бъдат хвърляни повторно:

```
try {
  Int32.Parse(str);
}
catch (FormatException fe) {
  Console.WriteLine("Parse failed!");
  throw fe; // Re-throw the caught exception
}
```

```
catch (FormatException) {
  throw; // Re-throws the last caught exception
}
```

Хвърляне на изключения – пример

```
public static double Sqrt(double value) {
  if (value < 0)
    throw new System.ArgumentOutOfRangeException("value",
      "Sqrt for negative numbers is undefined!");
  return Math.Sqrt(value);
}
static void Main() {
  try {
    Sqrt(-1);
  }
  catch (ArgumentOutOfRangeException ex) {
    Console.Error.WriteLine("Error: " + ex.Message);
    throw;
  }
}
```

Създаване на собствени изключения

- Собствените изключения наследяват exception класа(e. g. `System.Exception`)

```
public class PrinterException : Exception
{
  public PrinterException(string msg)
    : base(msg) { … }
}
```

- Хвърлят се както всички останали изключения

```
throw new PrinterException("Printer is out of paper!");
```

Задача: Следа от изключения

- **Прочете** всички редове от **файл** и **сумирайте числата**
- Използвайте **class `MyFileReader`**
- Ако **пътят към файла** е null или празен, хвърлете изключение (**`throw new ArgumentException`**) със съобщение`"Invalid Path or File Name."`
- Ако някоя стойност във файла **не може да се конвертира,** хвърлете изключение (**`throw new ArgumentException`**) със съобщение `"Error: On the line {line number} of the file the value was not in the correct format."`
- Ако всичко е **успешно**, **отпечатайте**: `"The sum of all correct numbers is: {numbers sum}"`

Решение: Следа от изключения(1)

```
public class MyFileReader {
  private string path;
  public MyFileReader(string path)
  {
    this.Path = path;
  }
  public string Path
  {
    get { return path; }
    set {
      if (string.IsNullOrEmpty(value)) {
        throw new ArgumentException("Invalid Path or File Name."); }
      path = value;
    }
  }
```

Решение: Следа от изключения (2)

```
  public void ReadAndSum() {
    string[] inputFromFile = File.ReadAllLines(this.Path);

    List<int> numbers = new List<int>();
    int countRow = 0;

    foreach (var value in inputFromFile) {
      countRow++;
      try { numbers.Add(int.Parse(value)); }
      catch (Exception) {
        throw new ArgumentException($"Error: On the line {countRow}
          of the file the value was not in the correct format."); }
    }
  Console.WriteLine($"The sum of all correct numbers is: {numbers.Sum()}");
  }
}
```

Решение: Следа от изключения (3)

```
static void Main() {
  try {
    MyFileReader reader1 = new MyFileReader(@"C:\temp\numbers.txt");
    reader1.ReadAndSum();
  }
  catch (Exception ex) {
    Console.Error.WriteLine("Error: " + ex.Message);
  }
  try {
    MyFileReader reader2 = new MyFileReader(@"");
    reader2.ReadAndSum();
  }
  catch (Exception ex) {
    Console.Error.WriteLine("Error: " + ex.Message);
  }
}
```

Проверете решението си тук: https://judge.softuni.bg/Contests/Practice/Index/3164#5

Обобщение

- Наследяването ни позволява да **преизползваме код**
- **Наследяването** води до **йерархии**
- **Подкласа наследява** членовете от **суперкласа** и може да **презаписва** мет оди
- Следете за класове с **еднаква роля**
- Обмислете **композиция** и **делегиране**

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