Inheritance

Class Hierarchies

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://about.softuni.bg/

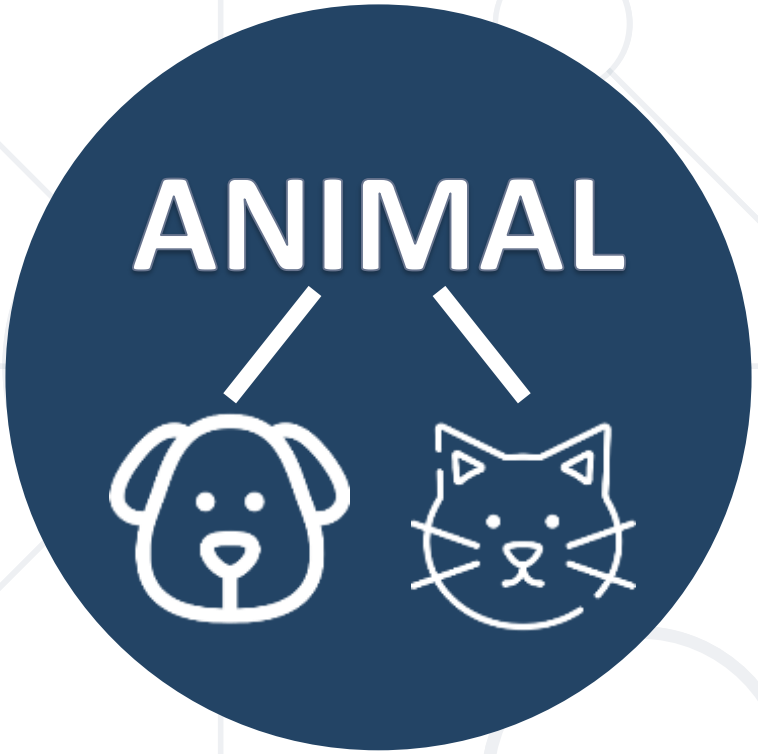
Table of Contents

Have a Question?

Software University

sli.do

#csharp-advanced

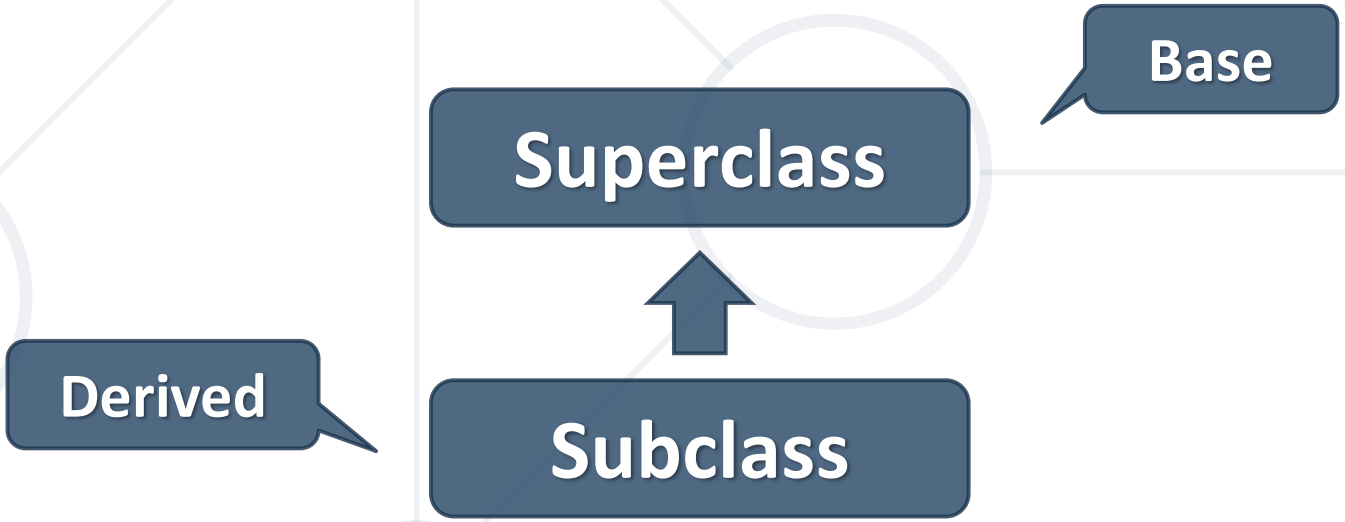


Inheritance

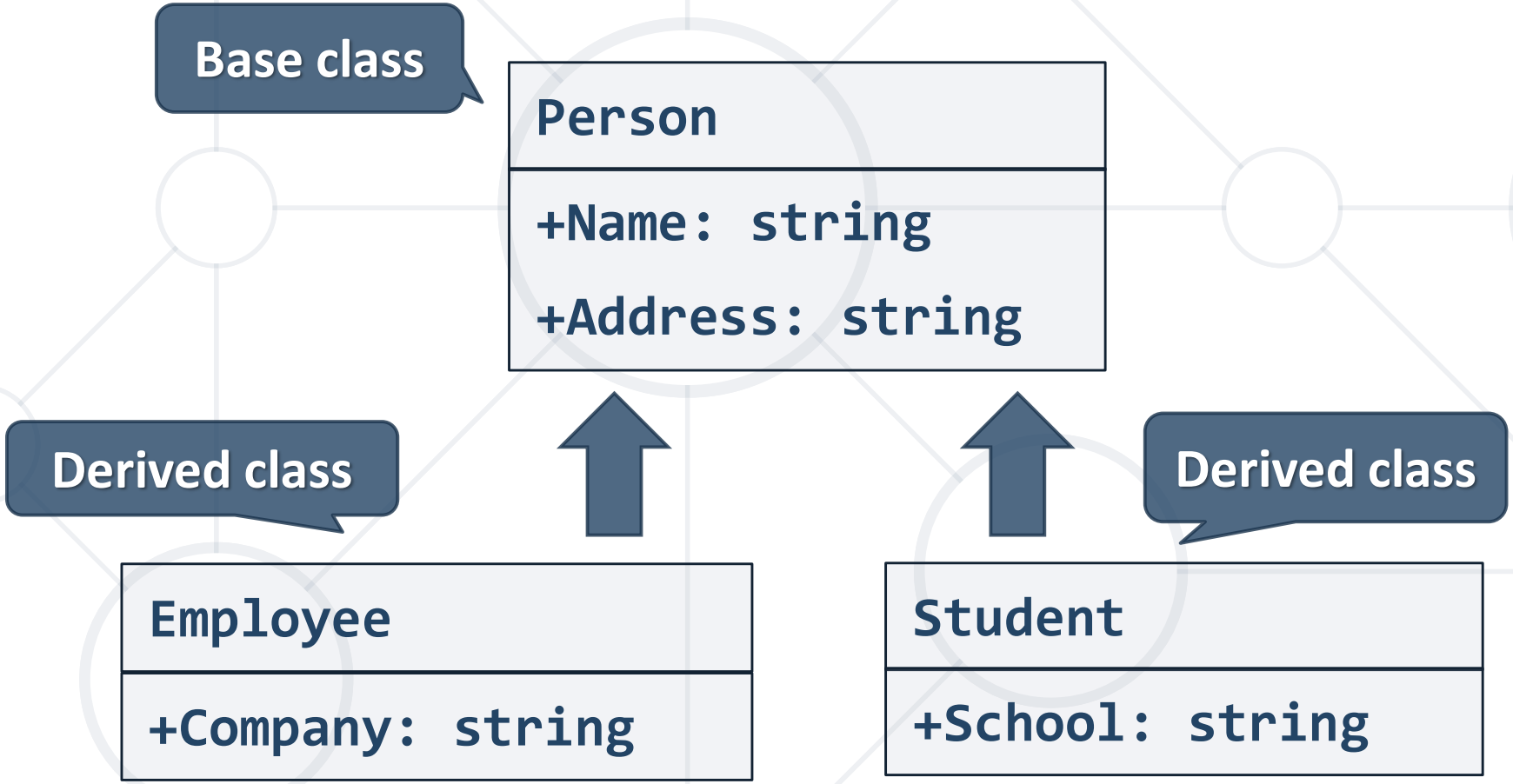
Extending Classes

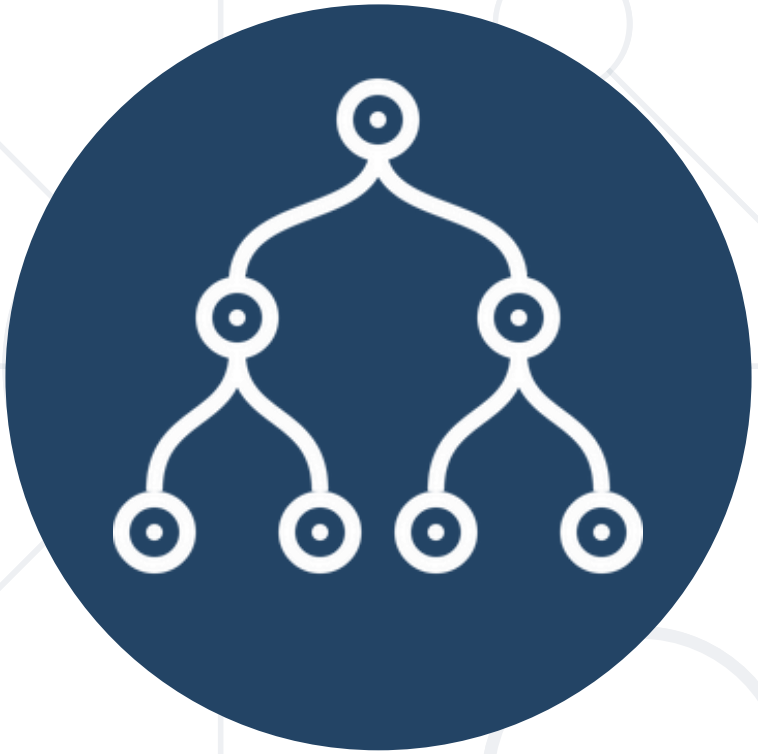
Inheritance

- **Superclass** - Parent class, Base Class
  - The class giving its **members** to its **child class**
- **Subclass** - **Child** class, **Derived class**
  - The class taking members from its base class



Inheritance – Example

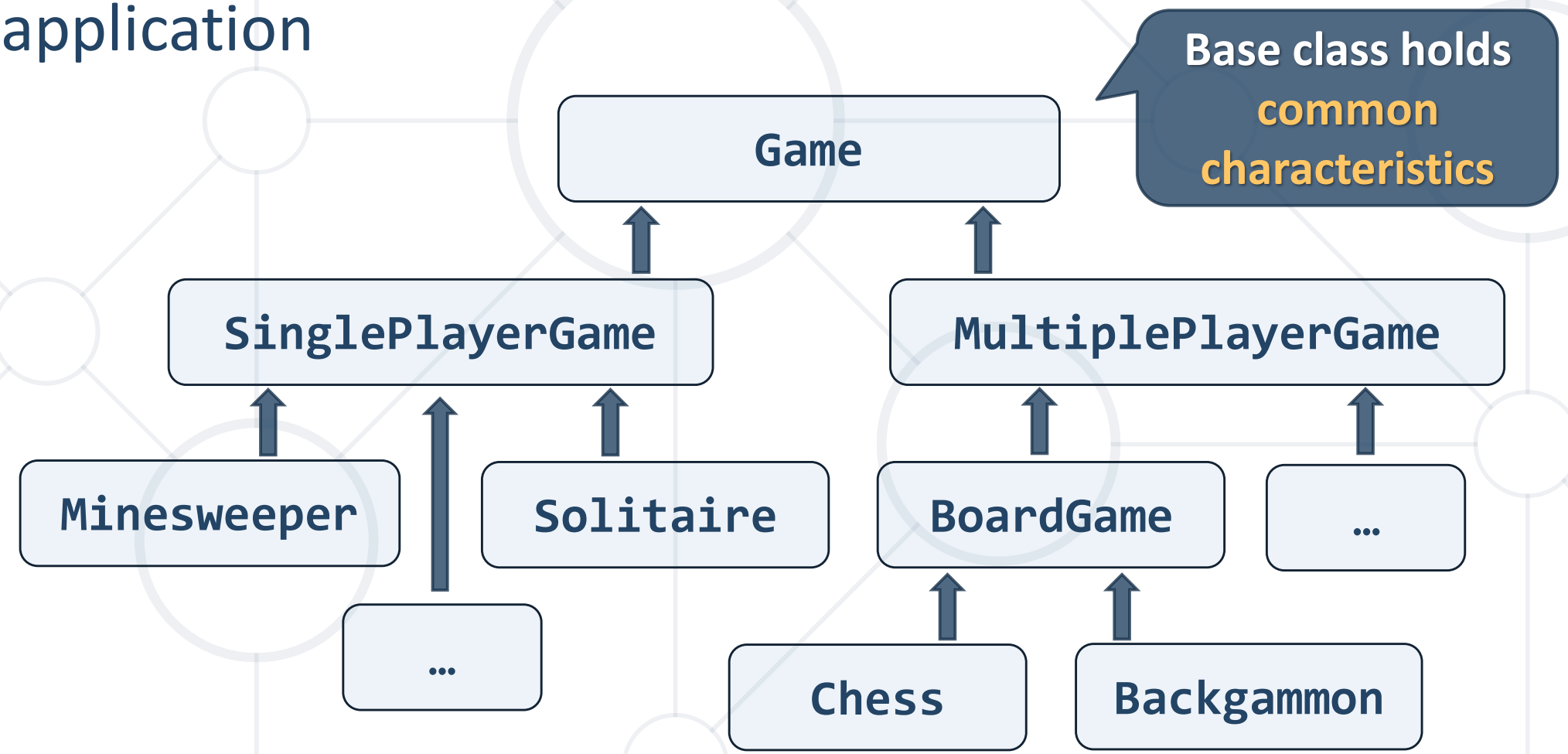




Class Hierarchies

Class Hierarchies

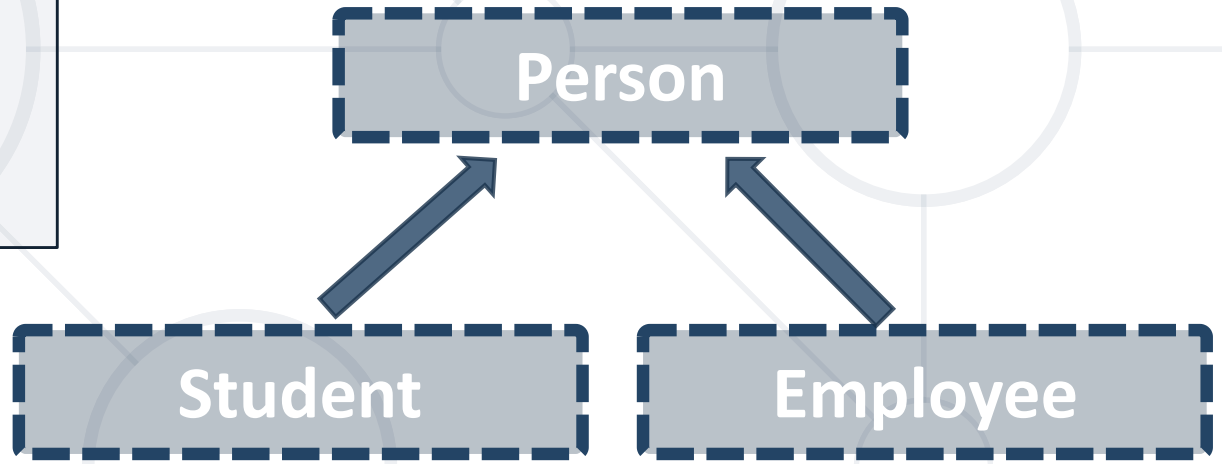
- Inheritance leads to hierarchies of classes and/or interfaces in an application



Inheritance in C#

- In C# inheritance is defined by the : operator

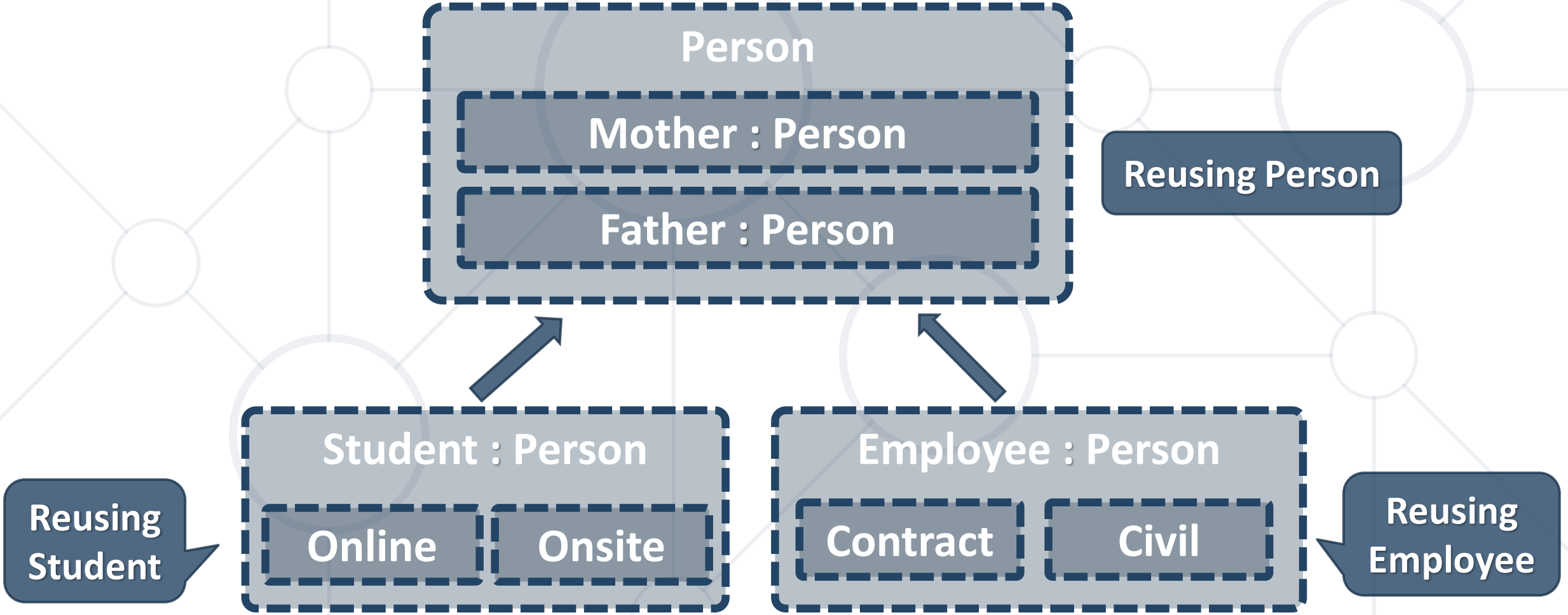
```
class Person { … }
class Student : Person { … }
class Employee : Person { … }
```



Student : Person

Inheritance - Derived Class

- Derived classes **take all members** from base classes



Using Inherited Members

- You can access inherited members as usual

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

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

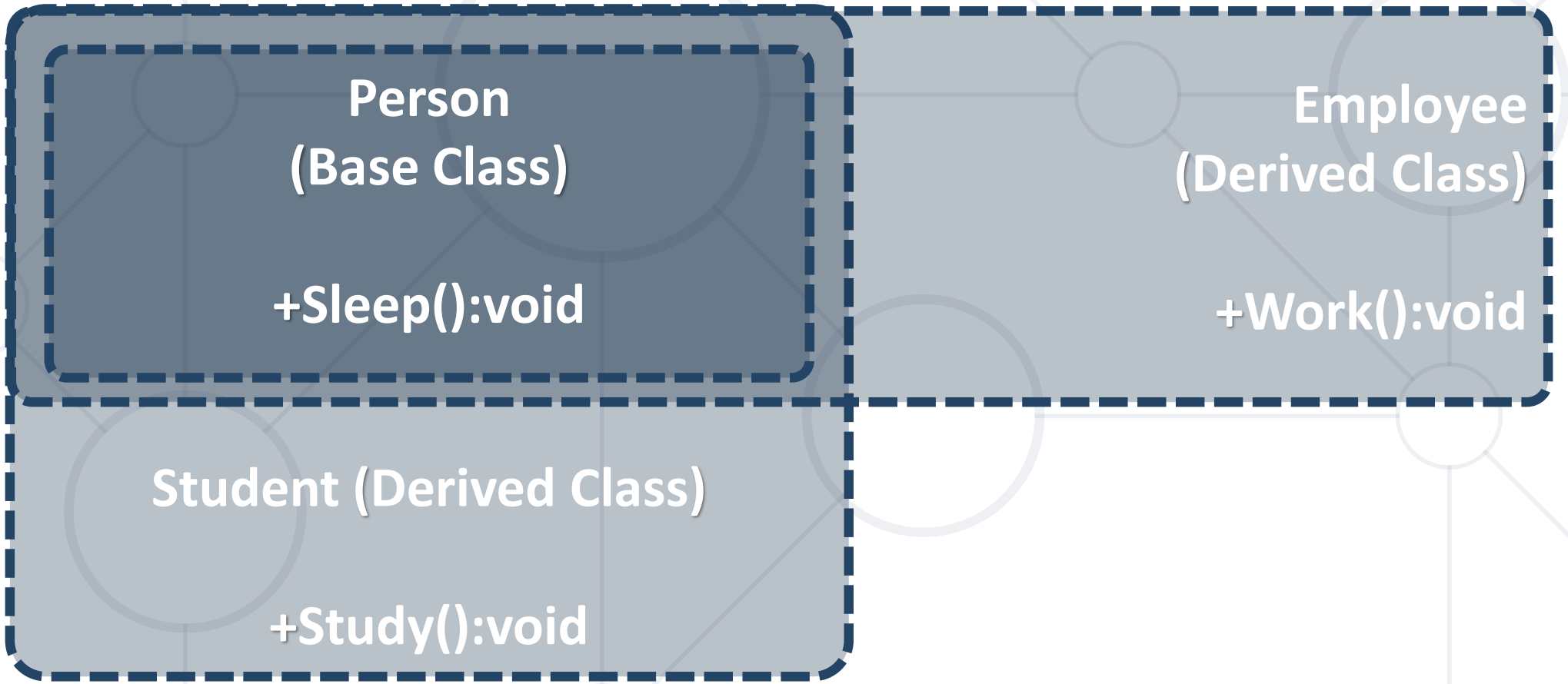
Reusing Constructors

- Constructors are **not inherited**
- They can be **reused** by the child classes

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

Thinking about Inheritance - Extends

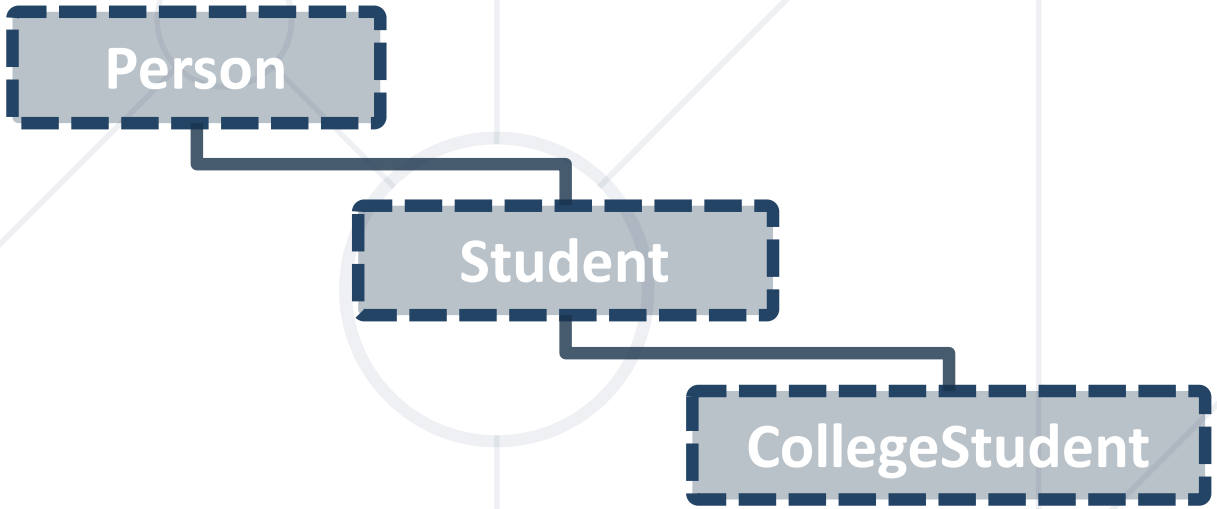
- Derived class instance **contains** instance of its base class



Transitive Relation

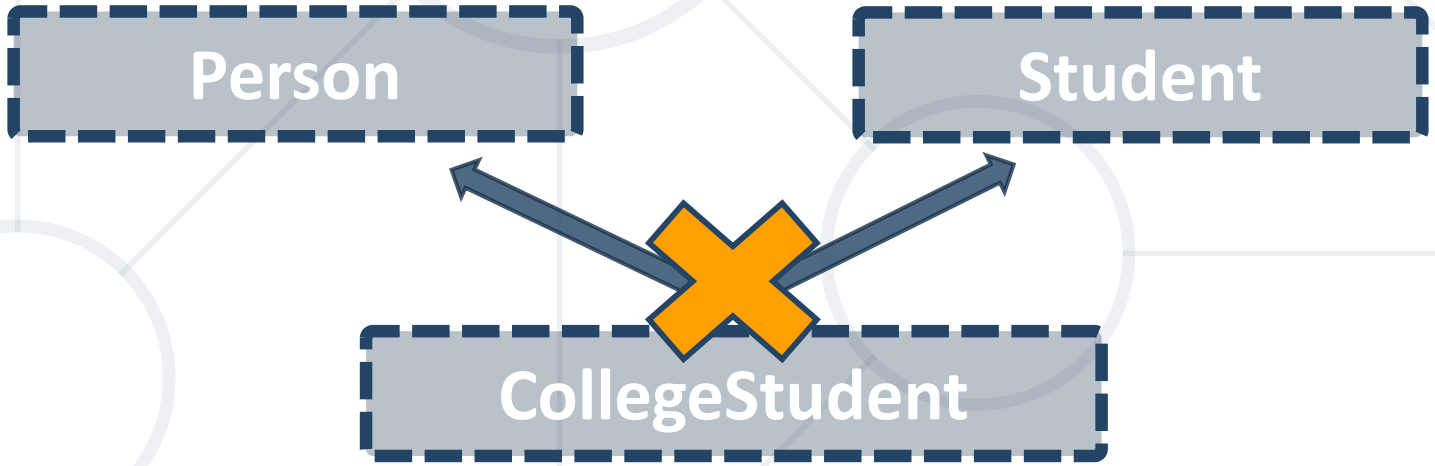
- Inheritance has a **transitive relation**

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



Multiple Inheritance

- In C# there is **no multiple** inheritance
- Only **multiple interfaces** can be implemented



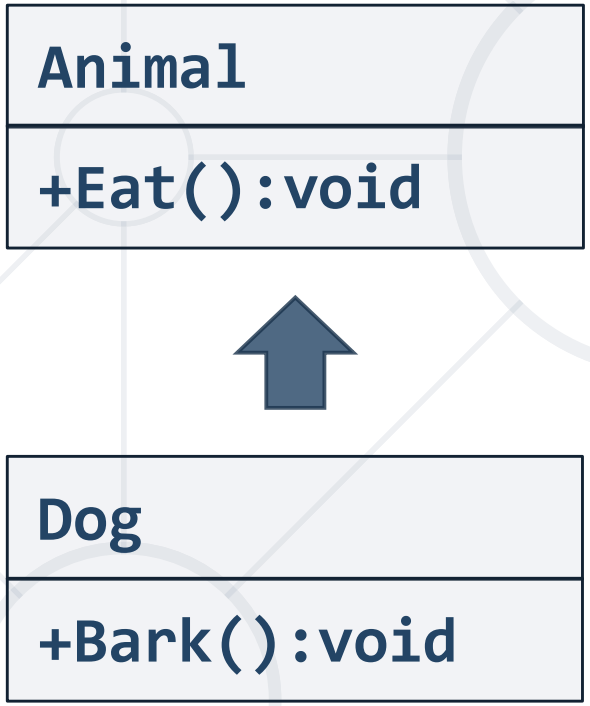
Accessing Base Class Members

Access to Base Class Members

- Use the **base** keyword

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

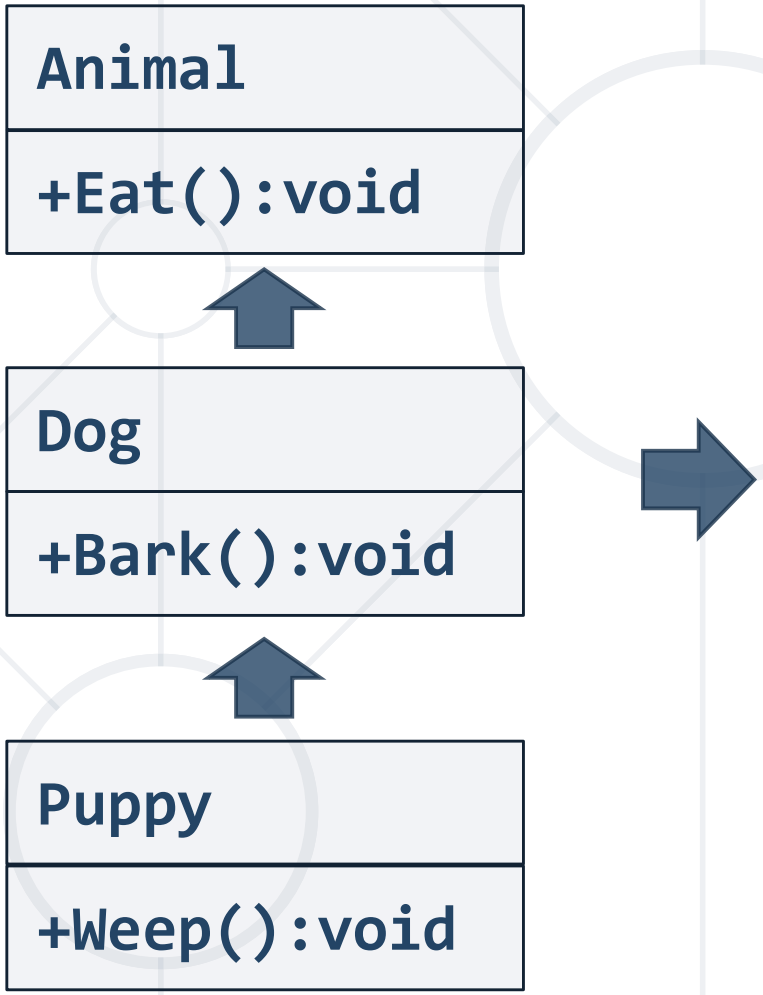
Problem: Single Inheritance




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

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1499#0

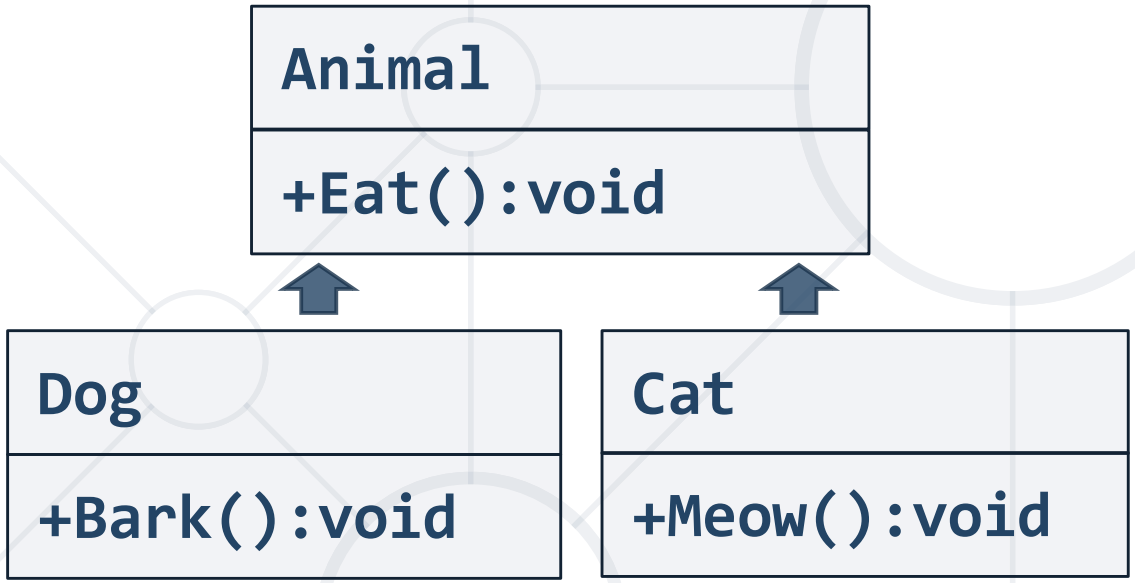
Problem: Transitive Inheritance




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

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1499#1

Problem: Hierarchical Inheritance




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

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

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1499#2

Reusing Classes

Reusing Code at Class Level

Inheritance and Access Modifiers

Software University

- Derived classes **can access all `public`** and **`protected`** members
- **`Internal`** members **are accessed in the same assembly**
- **`Private`** fields are **inherited,** but not visible in subclasses

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

Shadowing Variables

- Derived classes **can hide** superclass variables

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

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

Hides int weight

Hides float weight

Shadowing Variables - Access

- Use `base` and `this` to specify member access

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

Local variable

Instance member

Base class member

Virtual Methods

- `Virtual` - defines a method that **can be overriden**

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

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

Sealed Modifier

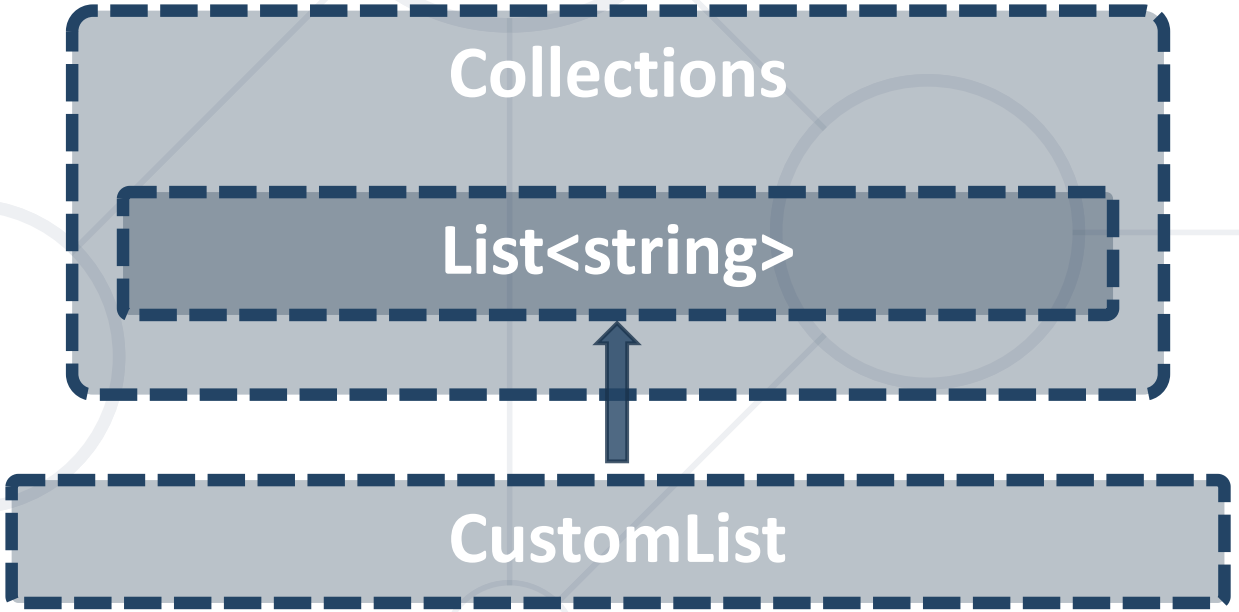
- The `sealed` modifier prevents other classes from **inheriting** from it
- You can use the `sealed` modifier on a **method** or a **property** in a `base` class:
- It enables you to **allow classes** to **derive** from your class
- **Prevents** the **overriding** of specific `virtual` **methods** and properties

Types of Class Reuse

Extension (Inheritance) and Composition

Extension (Inheritance) (IS-A relation)

- **Duplicate code** is error prone
- **Reuse classes** through **extension**
- Sometimes the only way

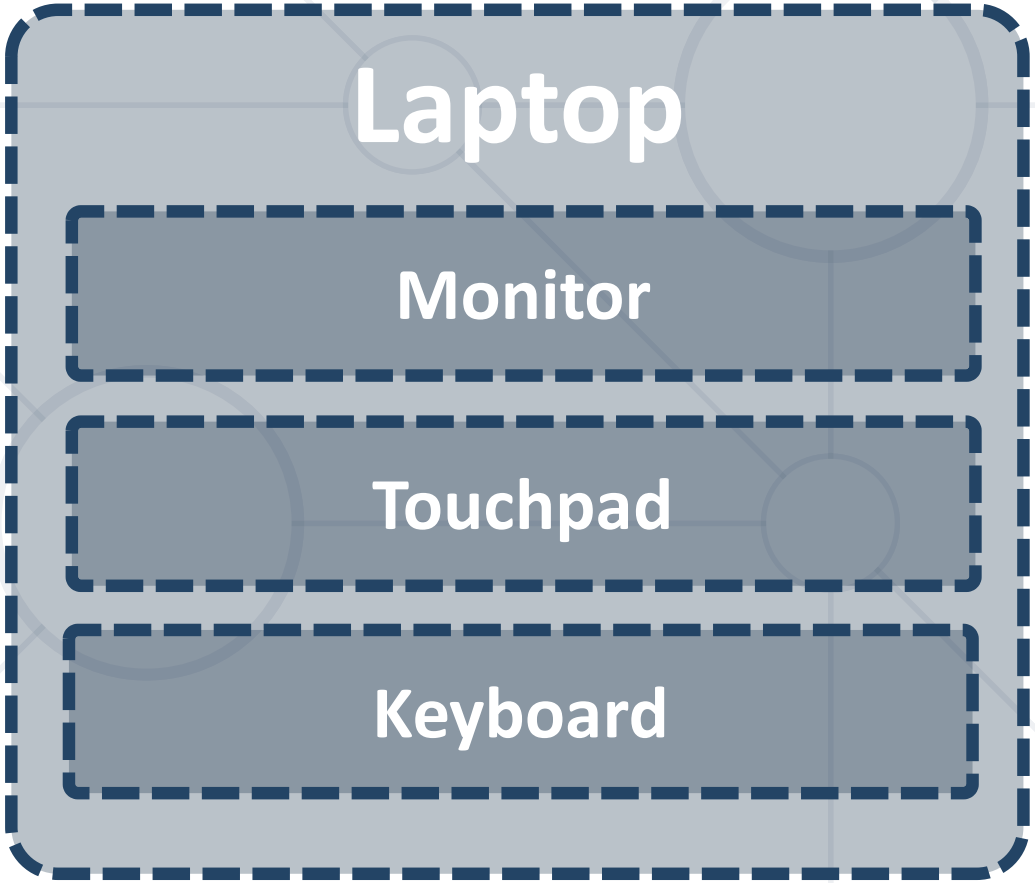


Composition (HAS-A relation)

- Using classes to **define** classes

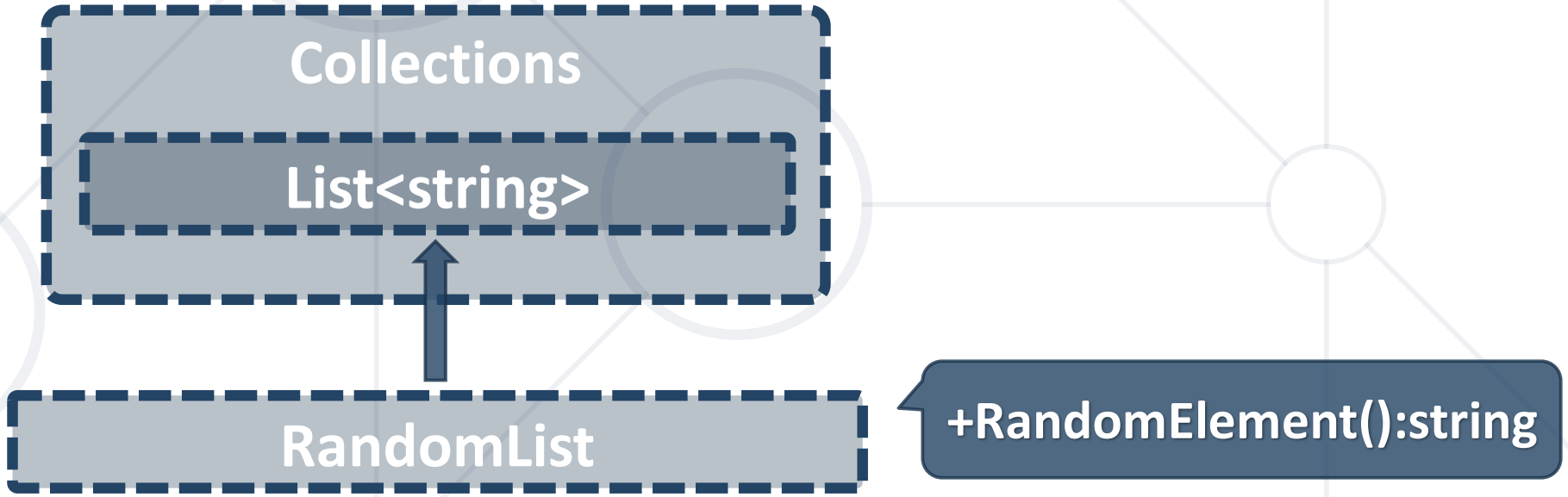
```
class Laptop {
  Monitor monitor;
  Touchpad touchpad;
  Keyboard keyboard;
  …
}
```

Reusing classes



Problem: Random List

- Create a list that has
  - All functionality of a `List<string>`
  - Method that returns and removes a random element



Solution: Random List

```
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;
  }
}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1499#3

Problem: Stack of Strings

- Create a simple **StackOfStrings** class which **inherits** the Stack<string>

StackOfStrings
+IsEmpty(): Boolean +AddRange(): void

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1499#4

Solution: Stack of Strings

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

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1499#4

Summary

- Inheritance is a powerful tool for **code reuse**
- **Subclass inherits** members from **Superclass** and can **override** methods
- Look for classes with the **same role**
- Look for **IS-A** and **IS-A-SUBSTITUTE**
- Consider **Composition** and **Delegation**

Questions?

SoftUni

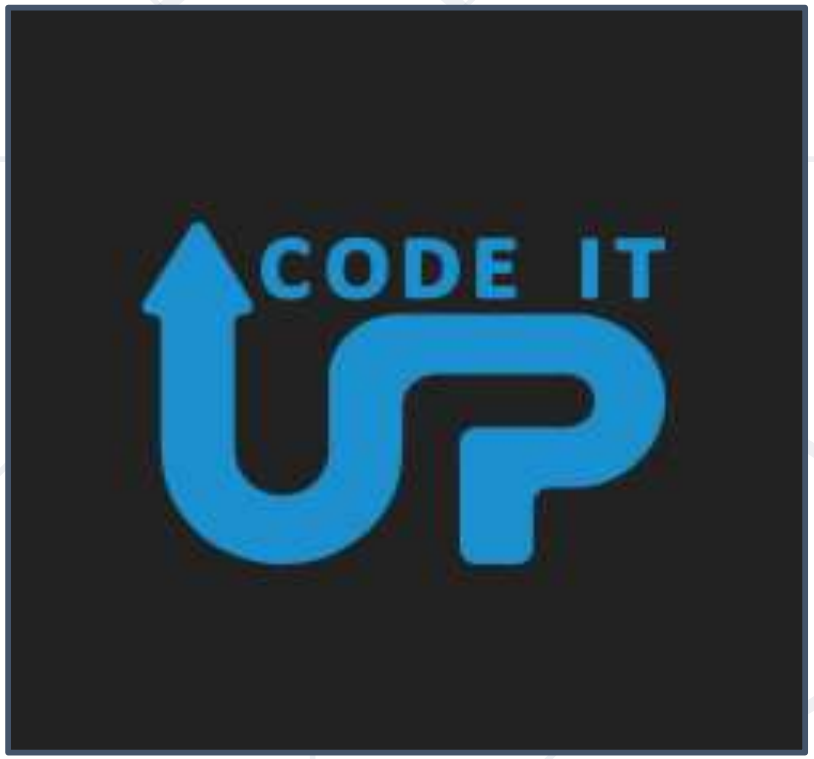
Software University | SoftUni Svetlina | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids

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