Encapsulation and Inheritance

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

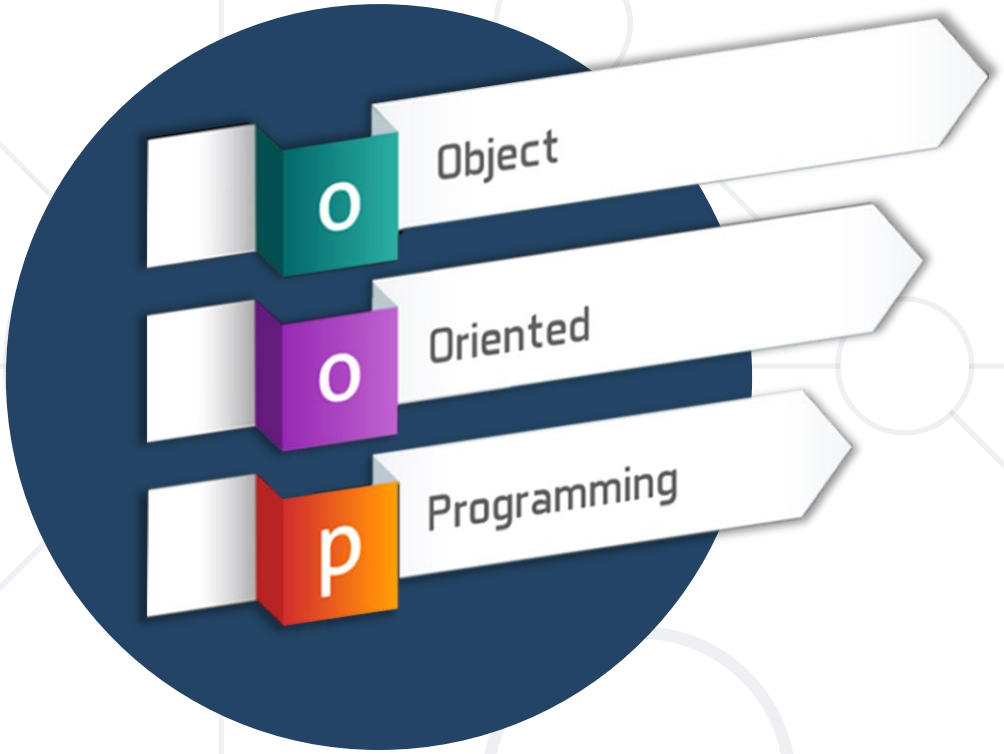
Software University

sli.do

#prgm-for-qa

Table of Contents

1. **OOP** Principles
2. **Encapsulation**
3. **Access** Modifiers
4. **State** Validation
5. **Inheritance**
6. Accessing Base **Class Members**



OOP Principles

Encapsulation, Inheritance, Abstraction, Polymorphism

The Principles of OOP

Software University

- Classical **principles** of the object-oriented programming (OOP):
  - **Encapsulation**: objects keep its state **private** (no direct access)
  - **Inheritance**: child classes **inherit** data + functionality from a parent
  - **Abstraction**: hide complexity behind an **interface** or **abstract class**
  - **Polymorphism**: use subclass objects through their base class

Encapsulation

Hiding Implementation

Encapsulation

- Process of wrapping code and data together into a **single unit**
- Flexibility and extensibility of the code
- Reduces **complexity**
- Structural changes remain **local**
- Allows **validation** and **data binding**

Encapsulation – Example

- Fields should be **private**

Person
-name: string -age: int
+Person(string name, int age) +Name: string +Age: int

- == private

+ == public

- Properties should be **public**

Keyword This

- Reference to the **current object**
- Refers to the **current instance** of the class
- Can be passed as a **parameter to other methods**
- Can be **returned** from method
- Can invoke **current class methods**

Visibility of Class Members

Access Modifiers

Private Access Modifier

- It's the main way to perform encapsulation and hide data from the outside world

```
private string name;
Person (string name) {
  this.name = name;
}
```

- The default field and method modifier is **private**

Public Access Modifier

- The most **permissive** access level
- There are **no restrictions** on accessing public members

```
public class Person {
    public string Name { get; set; }
    public int Age { get; set; }
}
```

- To access class directly from a namespace use the **`using`** keyword to include the namespace

Internal Access Modifier

- `Internal` is the **default** class access modifier

```
class Person {
    internal string Name { get; set; }
    internal int Age { get; set; }
}
```

- Accessible to any other class in the same project

```
Team rm = new Team("Real");
rm.Name = "Real Madrid";
```

Validation

Validation

- Setters are a good place for simple **data validation**

```
public double Salary {
   get { return this.salary }
   set {
      if (value < 650)
         throw new ArgumentException("...");
      this.salary = value; }
}
```

Throw exceptions

- Callers of your methods should take care of **handling** exceptions

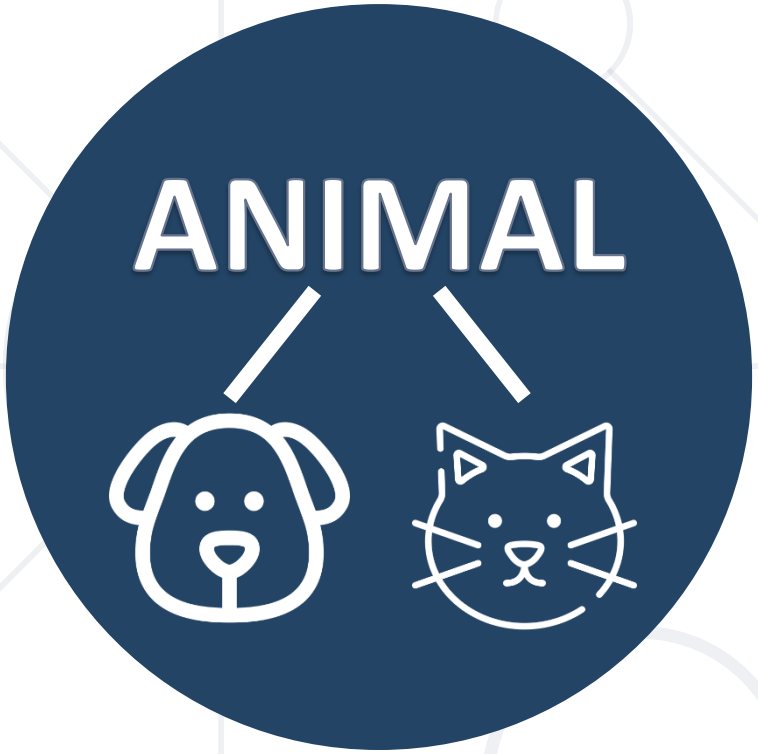
Validation

- Constructors use **private setters** with validation logic

```
public Person(string firstName, string lastName,
              int age, double salary) {
  this.FirstName = firstName;
  this.LastName = lastName;
  this.Age = age;
  this.Salary = salary;
}
```

Validation happens inside the setter

- Guarantee **valid state** of the object after its creation

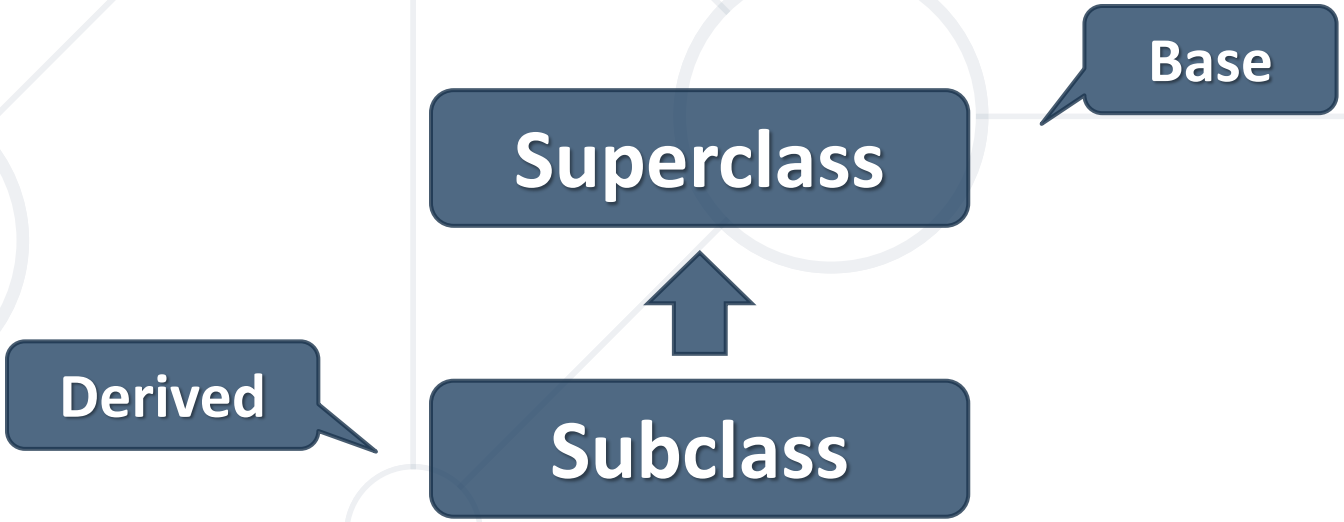


Inheritance

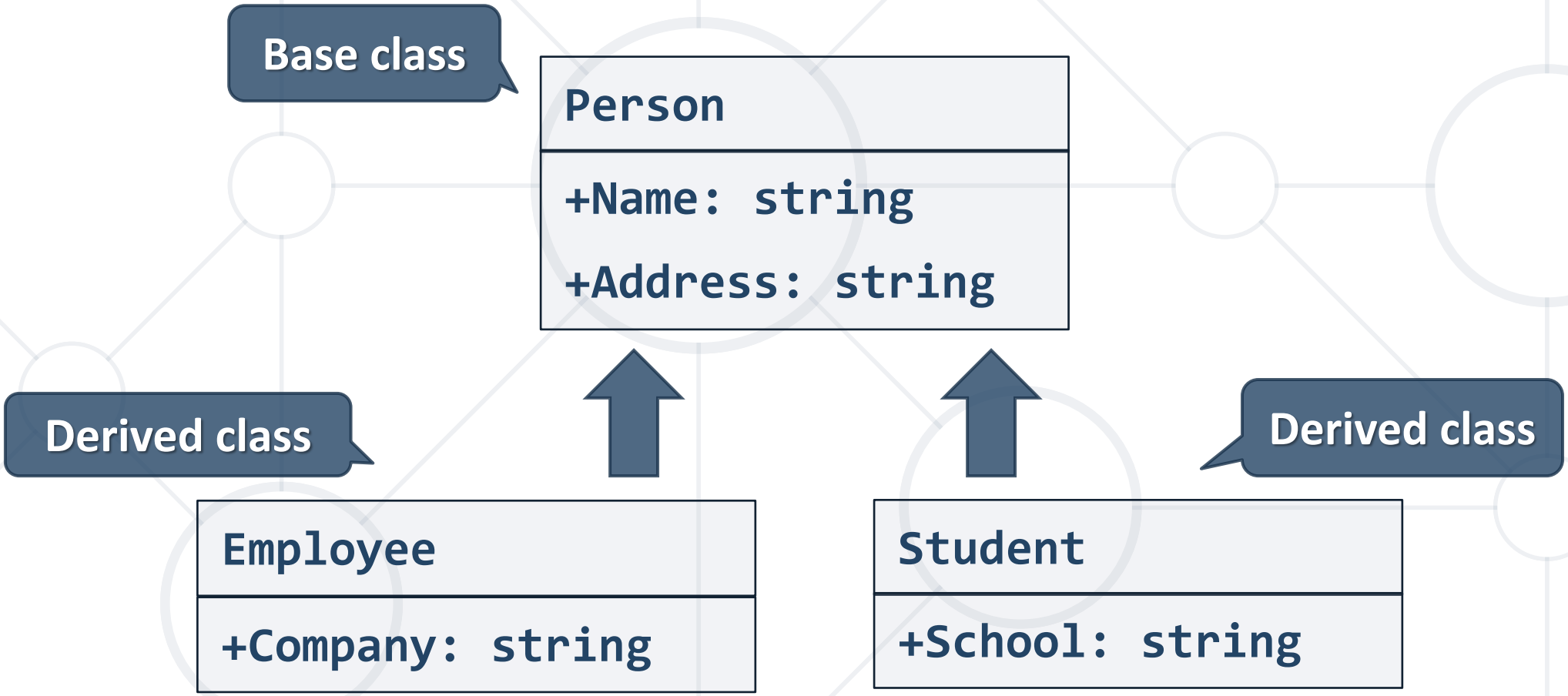
Extending Classes

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



Accessing Base Class Members

Access to Base Class Members

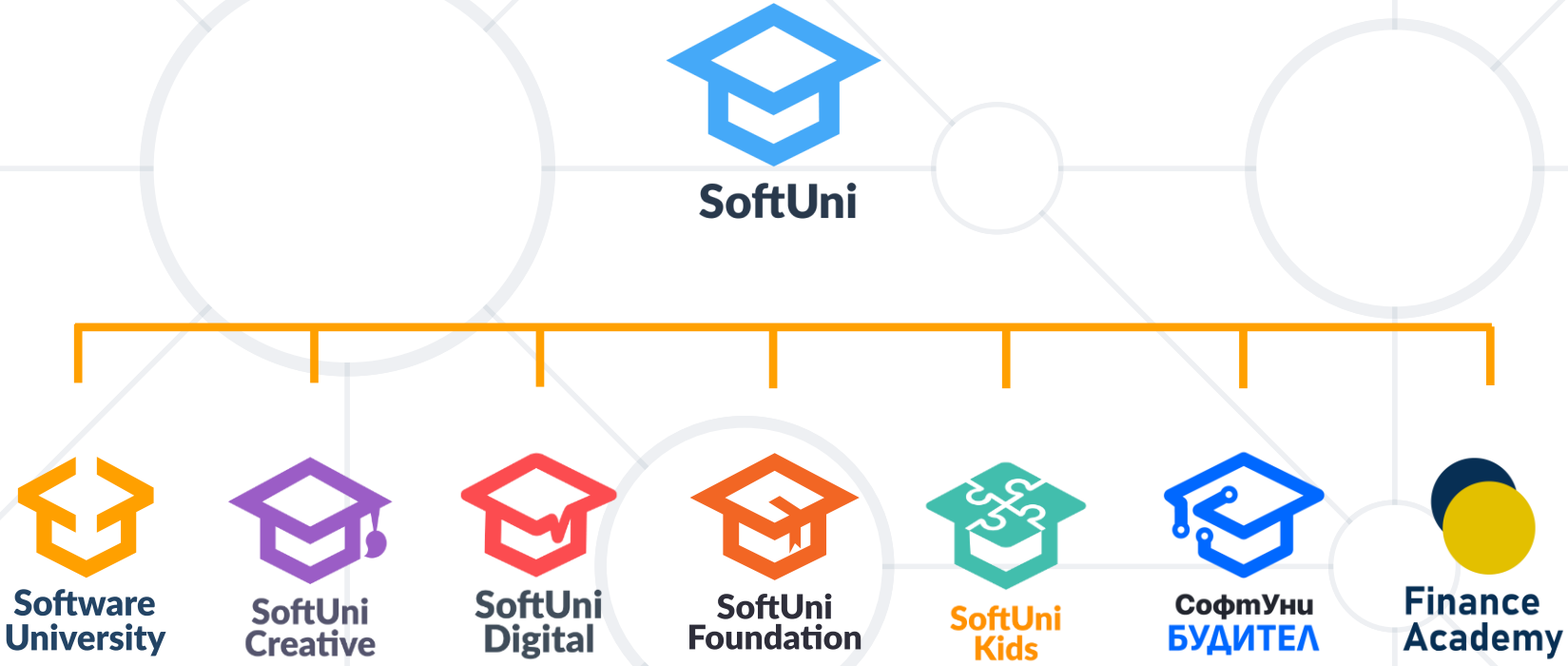
- Use the **base** keyword

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

Summary

- Encapsulation:
  - Hides implementation
  - Reduces complexity
  - Ensures that structural changes remain local
- Inheritance is a powerful tool for code reuse

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

SOFTWARE GROUP

PHAR VISION

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

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