

# Енкапсулация

Ползи от енкапсулацията



SoftUni Team  
Technical Trainers

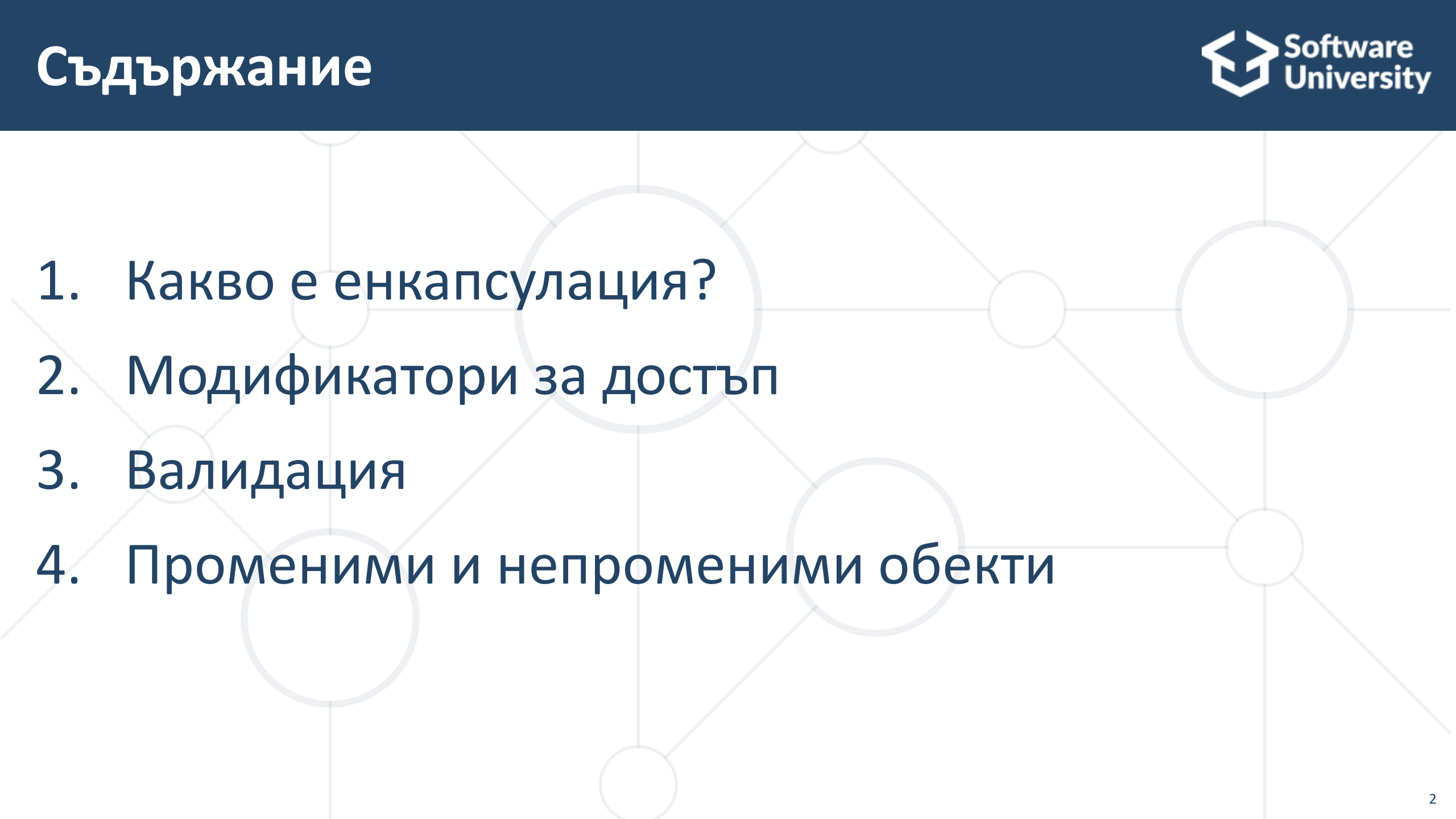


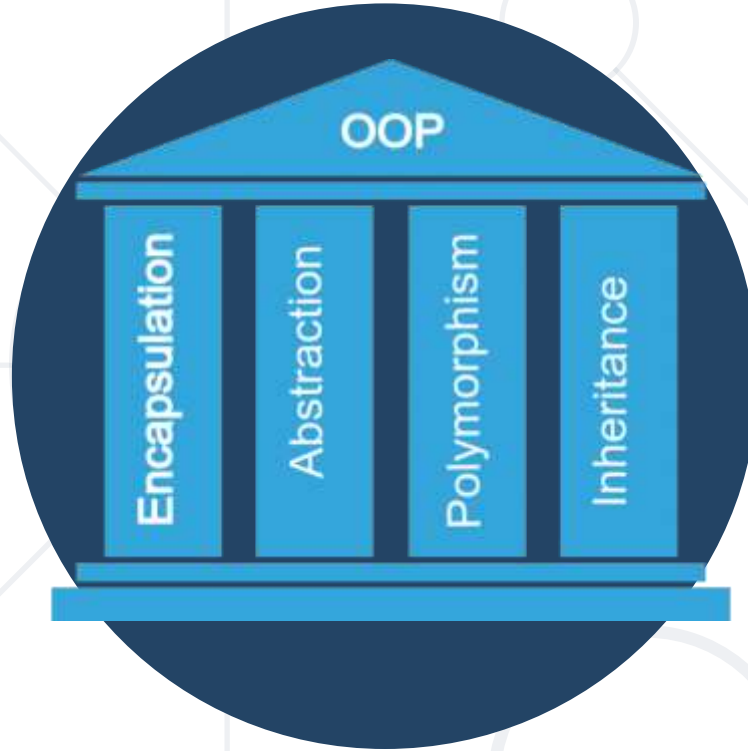
**SoftUni**



Software University

<https://softuni.bg>

- 
1. Какво е енкапсулация?
  2. Модификатори за достъп
  3. Валидация
  4. Променими и непроменими обекти



**Скриване на имплементацията**



- Процесът на обединяване на кода и данните в **едно цяло**
- Позволява **валидация** и **обвързване на данните**
- Структурните промени остават **локални**
- Намалява **комплексността**

Достъпен само за  
публичните  
методи на класа

```
public class Student
{
    private string studentName;
    public string Name
    {
        get { return studentName; }
        set { studentName = value; }
    }
}
```

Accessor-и за достъп и  
промяна на стойността

- Полетата трябва да бъдат **частни**

Person	
-name: string	 - means "private"
-age: int	
+Person(string name, int age)	
+Name: string	 + means "public"
+Age: int	

- Свойствата трябва да бъдат **публични**



**Видимост на членовете на класа**

- Основният начин да осъществим енкапсулация и да **скрием данните** от външния свят

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

- Модификаторът на **полето** и **метода** по подразбиране е **private** (частен)
- Избягвайте декларирането на **частни класове** и **интерфейси**
  - Достъпни са **само** в класа, в който са декларирани

# Модификатор за публичен достъп (1)

- Модификаторът, който дава **най-високо ниво на достъп**
- **Няма ограничения** при достъпване на публични членове

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



# Модификатор за публичен достъп (2)

- За да достъпите класа директно от namespace, използвайте ключовата дума **using**.

```
namespace Mathematical
{
    public class Basic
    {
        public double PI = 3.14;
    }
}
```

```
using System;
using Mathematical;
namespace Distinct
{
    public class Program
    {
        Console.WriteLine(Basic.PI);
    }
}
```

- **internal** е модификаторът по подразбиране на всеки клас

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

- **Достъпен** от всеки друг клас в същия проект

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

# Задача: Сортирайте хора по име и възраст

- Създайте клас **Person**, който трябва да има **публични свойства** с **частни setter-и** за:

**Person**

**+FirstName:string**

**+LastName:string**

**+Age:int**

**+ToString():string**



# Решение: Сортирайте хора по име и възраст (1)

```
public class Person
{
    // TODO: Add a constructor
    public string FirstName { get; private set; }
    public string LastName { get; private set; }
    public int Age { get; private set; }
    public override string ToString()
    {
        return $"{FirstName} {LastName} is {Age} years old.";
    }
}
```

# Решение: Сортирайте хора по име и възраст (2)

```
var lines = int.Parse(Console.ReadLine());  
var people = new List<Person>();  
for (int i = 0; i < lines; i++)  
{  
    var cmdArgs = Console.ReadLine().Split();  
    // Create variables for constructor parameters  
    // Initialize a Person  
    // Add it to the list  
}
```

# Решение: Сортирайте хора по име и възраст (3)

*//continued from previous slide*

```
var sorted = people.OrderBy(p => p.FirstName)
    .ThenBy(p => p.Age).ToList();
```

```
Console.WriteLine(string.Join(Environment.NewLine, sorted));
```

# Задача: Увеличение на заплатата

- Разширете класа **Person** със **Salary** (заплата)
- Добавете getter за **Salary**
- Добавете метод, който увеличава **Salary** с определен процент
- Хора, по-млади от 30, получават половината от стандартното увеличение.

## Person

+FirstName: string

+Age: int

+Salary: decimal

+IncreaseSalary(decimal): void

+ToString(): string

# Решение: Увеличение на заплатата

```
public decimal Salary { get; private set; }  
public void IncreaseSalary(decimal percentage)  
{  
    if (this.Age > 30)  
        this.Salary += this.Salary * percentage / 100;  
    else  
        this.Salary += this.Salary * percentage / 200;  
}
```

Проверете решението си тук: <https://judge.softuni.bg/Contests/Practice/Index/3163#1>





**Валидация в Getter-и и Setter-и**

- Setter-ите са добро място за базова **валидация на данните**

```
public decimal Salary
{
    get { return this.salary; }
    set {
        if (value < 460)
            throw new ArgumentException("...");
        this.salary = value;
    }
}
```

Връща **изключения**  
(exceptions)

- Конструкторите използват **частни setter-и** с логика за валидация

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

Валидацията се  
осъществява в setter-и

- Гарантират **валидно състояние (state)** на обекта при създаването му

# Задача: Валидиране на данни

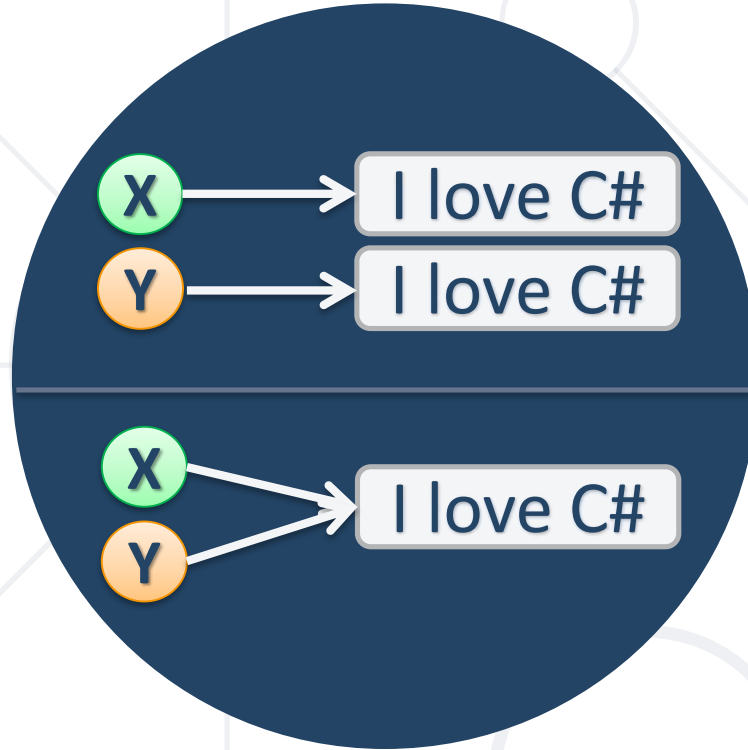
- Разширете класа **Person** с валидация за всяко поле
- Имената трябва да имат поне 3 символа
- Възрастта не може да бъде 0 или отрицателна
- Заплатата не може да бъде по-малко от 460

## Person

```
-firstName: string  
-lastName: string  
-age: int  
-salary: decimal
```

```
+Person()  
+FirstName(string fname)  
+LastName(string lname)  
+Age(int age)  
+Salary(decimal salary)
```

```
public int Age
{
    get => this.age;
    private set
    {
        if (age < 1)
            throw new ArgumentException("...");
        this.age = value;
    }
}
// TODO: Add validation for the rest
```



**Променими и непроменими обекти**

## ■ Променими обекти

- Променими == mutable
- Използват една и съща локация в паметта
- **StringBuilder**
- **List**

## ■ Непроменими обекти

- Непроменими == immutable
- Заделят нова памет всеки път, когато се променят
- **string**
- **int**



- **Private mutable (частните променими)** полета все още **не са енкапсулирани**

```
class Team
{
    private List<Person> players;
    public List<Person> Players { get { return this.players; } }
}
```

- В този пример можете да **достъпите** полетата чрез **getter**



- Можете да използвате **ICollection**, за да енкапсулирате колекции

```
public class Team
{
    private List<Person> players;
    public ICollection<Person> Players
    {
        get { return this.players.AsReadOnly(); }
    }
    public void AddPlayer(Person player)
        => this.players.Add(player);
}
```

- Отборът има два екипа
  - Първи екип и резервен екип
- Прочетете данните на участниците и ги добавете в отбора
- Ако са по-млади от 40 години, ги включете в първи екип
- Отпечатайте размера и на двата екипа

## Team

```
-Name : string  
-FirstTeam: List<Person>  
-ReserveTeam: List<Person>
```

```
+Team(String name)  
+Name(): string  
+FirstTeam(): ReadOnlyList<Person>  
+ReserveTeam: ReadOnlyList<Person>  
+AddPlayer(Person person)
```

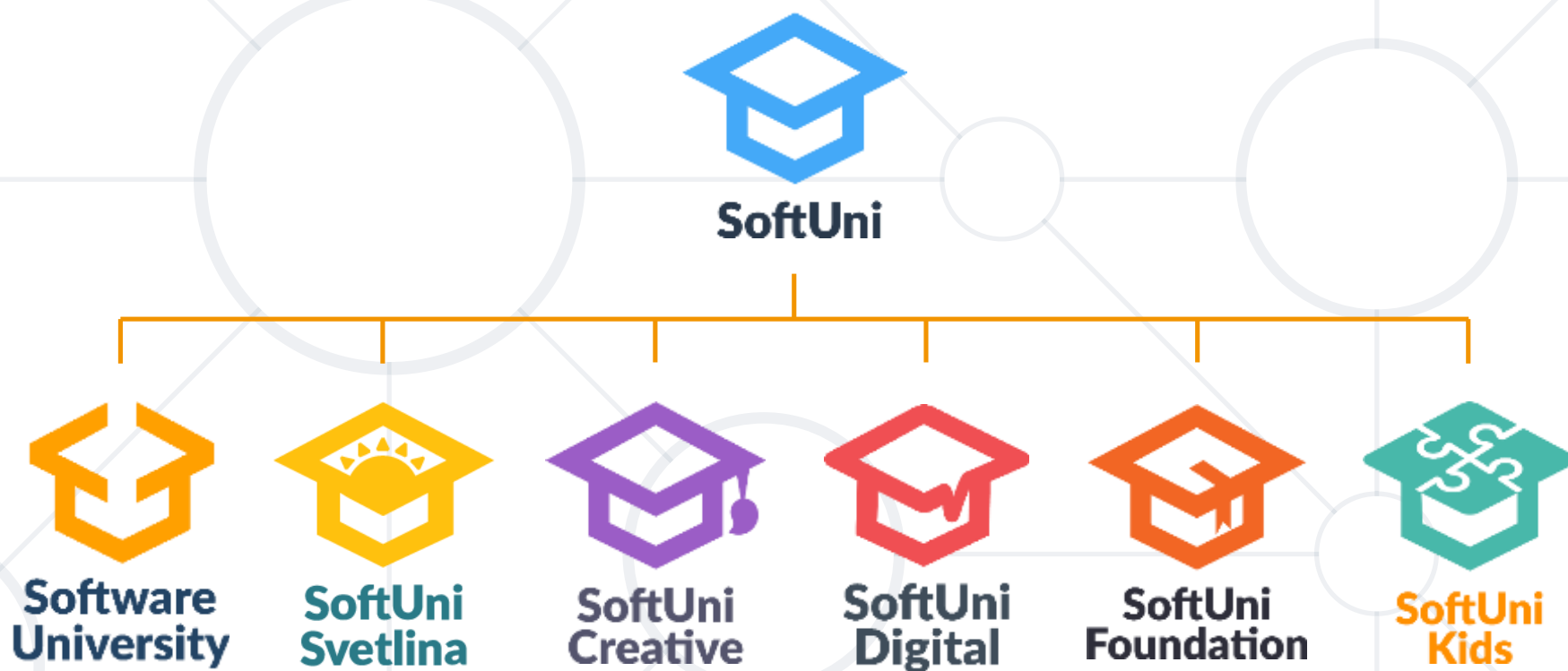
```
private string name;  
private List<Person> firstTeam;  
private List<Person> reserveTeam;  
  
public Team(string name)  
{  
    this.name = name;  
    this.firstTeam = new List<Person>();  
    this.reserveTeam = new List<Person>();  
}  
  
// continues on the next slide
```

```
public IReadOnlyCollection<Person> FirstTeam
{
    get { return this.firstTeam.AsReadOnly(); }
}
// TODO: Implement reserve team getter
public void AddPlayer(Person player)
{
    if (player.Age < 40)
        firstTeam.Add(player);
    else
        reserveTeam.Add(player);
}
```

- Енкапсулация:
  - Скрива **имплементацията**
  - Намалява **комплексността**
  - Гарантира, че структурните промени остават локални
- **Променими** и **непроменими** обекти



# Въпроси?



- 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>

