Encapsulation

Benefits of Encapsulation

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Questions

Software University

sli.do

#csharp-advanced

Table of Contents

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`**
- Avoid declaring private classes and interfaces
  - accessible only within the declared class itself

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";
```

Problem: Sort People by Name and Age

- Create a read-only class **Person**
- Read and sort people by first name and age

Person
+FirstName:string
+LastName:string
+Age:int
+ToString():string

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1497#0

Solution: Sort People by Name and Age (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.";
  }
}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1497#0

Solution: Sort People by Name and Age (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
}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1497#0

Solution: Sort People by Name and Age (3)

```
//continued from previous slide
var sorted = people.OrderBy(p => p.FirstName)
  .ThenBy(p => p.Age).ToList();

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

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1497#0

Problem: Salary Increase

- Expand `Person` with `salary`
- Add getter for `salary`
- Add a method, which updates `salary` with a given percent
- Persons younger than 30 get half of the normal increase

Person
+FirstName: string +Age: int +Salary: decimal +IncreaseSalary(decimal): void +ToString(): string

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1497#1

Solution: Salary Increase

```
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;
}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1497#1

Validation

Validation (1)

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

```
public decimal 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 (2)

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

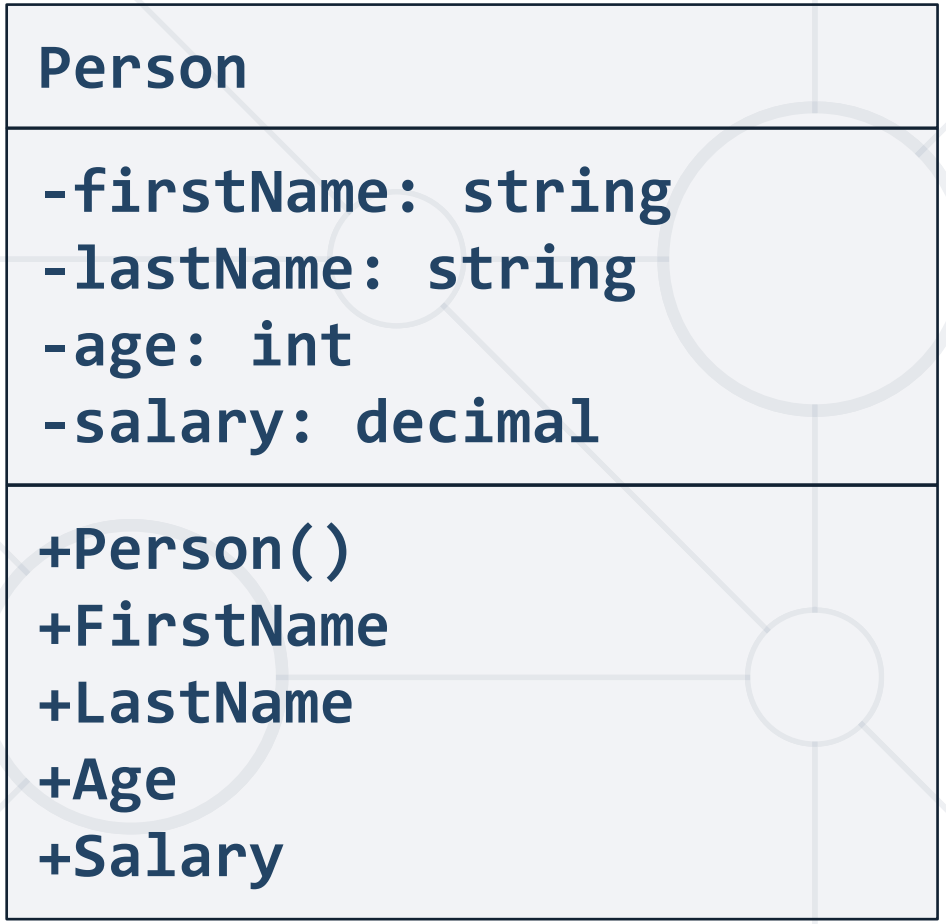
```
public Person(string firstName, string lastName,
              int age, decimal salary) {
  this.FirstName = firstName;
  this.LastName = lastName;
  this.Age = age;
  this.Salary = salary;
}
```

Validation happens inside the setter

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

Problem: Validate Data

- Expand **Person** with validation for every field
- Names must be at least 3 symbols
- Age cannot be zero or negative
- Salary cannot be less than 650



Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1497#2

Solution: Validate Data

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

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1497#2

Mutable vs Immutable Objects

- Mutable Objects
  - Mutable == changeable
  - Use the same memory location
  - `StringBuilder`
  - `List`
- Immutable Objects
  - Immutable == unchangeable (read-only)
  - Create new memory every time they're modified
  - `string`
  - `Tuples`

Mutable Fields

- **Private mutable** fields are still **not encapsulated**

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

- In this case you can **access** the field methods through the **getter**

Immutable Fields

Software University

- You can use `IReadOnlyCollection` to encapsulate collections

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

Problem: Team

- Team have two squads
  - First team & Reserve team
- Read persons from console and add them to team
- If they are younger than 40, they go to first squad
- Print both squad sizes

Team
-name : string -firstTeam: List<Person> -reserveTeam: List<Person>
+Team(string name) +Name: string +FirstTeam: ReadOnlyList<Person> +ReserveTeam: ReadOnlyList<Person> +AddPlayer(Person person)

Check your solution here: https://judge.softuni.bg/Contests/1497/Encapsulation-Lab

Solution: Team (1)

```
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
```

Check your solution here: https://judge.softuni.bg/Contests/1497/Encapsulation-Lab

Solution: Team (2)

```
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); }
```

Check your solution here: https://judge.softuni.bg/Contests/1497/Encapsulation-Lab

Summary

- Encapsulation:
  - Hides **implementation**
  - Reduces **complexity**
  - Ensures that structural changes remain local
- **Mutable** and **Immutable** objects

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond 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

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