SOLID Principles

The Benefits and Potential of Using SOLID Principles

- **S** Single Responsibility
- **O** Open/Closed
- **L** Liskov substitution
- **I** Interface Segregation
- **D** Dependency Inversion

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Questions

Software University

sli.do

#python-advanced

Table of Contents

Single Responsibility

What is Single Responsibility?

- Each class is responsible for **only one thing** and should have only one reason to change
- A class that has many responsibilities is **coupling** these responsibilities together, which leads to **complexity and fragility**

SRP Violations

- SRP states that classes should have one responsibility. Here we have **two**:
  - Student properties management
  - Student database management

```
class Student:
    def __init__(self, name):
        self.name = name

    def get_name(self):
        return self.name

    def register(self, student):
        pass
```

SRP Approaches

- We can avoid the domino effect if the application changes by **splitting the class**:
  - Create another class that will handle the responsibility of storing a student in a database

```
class Student:
    def __init__(self, name):
        self.name = name

    def get_name(self):
        return self.name
```

```
class StudentRecords:
    def get_student(self, id):
        pass

    def register(self, student):
        pass
```

Open / Closed

What is the Open / Closed Principle?

- Software entities like **classes**, **modules**, and **functions** should be **open** for **extension** but **closed** for **modifications**
- This can be achieved through:
  - Abstraction
  - Mix-ins
  - Monkey-Patching
  - Generic functions (using overloading)

OCP Violation

- Let's imagine that we want to make a 40% **discount** on the semester taxes to all students with **grades above 5**

```
class StudentTaxes:
    def __init__(self, name, semester_tax, average_grade):
        self.name = name
        self.semester_tax = semester_tax
        self.average_grade = average_grade

    def get_discount(self):
        if self.average_grade > 5:
            return self.semester_tax * 0.4
```

OCP Violation

- Later we decide that we want to give a 20% **discount** to students with **grades above 4**

```
class StudentTaxes:
    def __init__(self, name, semester_tax, average_grade):
        self.name = name
        self.semester_tax = semester_tax
        self.average_grade = average_grade

    def get_discount(self):
        if self.average_grade > 5:
            return self.semester_tax * 0.4
        elif self.average_grade > 4:
            return self.semester_tax * 0.2
```

OCP Approaches

Software University

```
class StudentTaxes:
    def __init__(self, name, semester_tax, avg_grade):
        self.name = name
        self.semester_tax = semester_tax
        self.average_grade = average_grade

    def get_discount(self):
        if self.average_grade > 5:
            return self.semester_tax * 0.4

class AdditionalDiscount(StudentTaxes):
    def get_discount(self):
        result = super().get_discount()
        if result:
            return result
        if 4 < self.average_grade <= 5:
            return self.semester_tax * 0.2
```

Keep the class unchanged

Extend the base class functionality by adding new class

Liskov Substitution

LSP – Substitutability

- Derived types must be completely **substitutable** for their base types
- Derived classes
  - only **extend** functionalities of the base class
  - must **not** remove **base** class **behavior**

```
Student IS-SUBSTITUTED-FOR Person
```

Remarks on the LSP

- LSP is fundamental to a good object-oriented software design because it emphasizes one of its core traits – **polymorphism**
  - It is about creating **correct hierarchies** so that classes derived from a base one are polymorphic along the parent one
  - Carefully thinking about new classes in the way that LSP suggests helps us to **extend the hierarchy correctly**
  - We could say that **LSP** contributes to the **OCP**

Design Smell – Violations

Software University

- If the code is **checking the type** of class
- Overridden methods **change** their **behavior**
- Override a method of the superclass by an **empty method**
- Base class **depends** on its **subtypes**

Interface Segregation

What is Interface Segregation?

- A client **should not depend** on methods it **does not use**
  - A good way of ensuring this is by **separation** through multiple **inheritance**
  - This is precisely the purpose of the **mix-ins** - to provide multiple clients with **specific behaviors**
- ISP is intended to keep a system **decoupled** and thus easier to refactor, change, and redeploy

ISP issues

- Python **doesn't have** interfaces
- Languages that do have interfaces:
  - Breaking them up too much ends up with **interfaces implementing interfaces**

ISP Violations

- Class **Shape** draws rectangle and circle
- Class Circle or Rectangle implementing the Shape class must define the methods **draw_rectangle() and draw_circle()**

```
class Shape:
    def draw_rectangle(self):
        raise NotImplementedError

    def draw_circle(self):
        raise NotImplementedError
```

ISP Violations

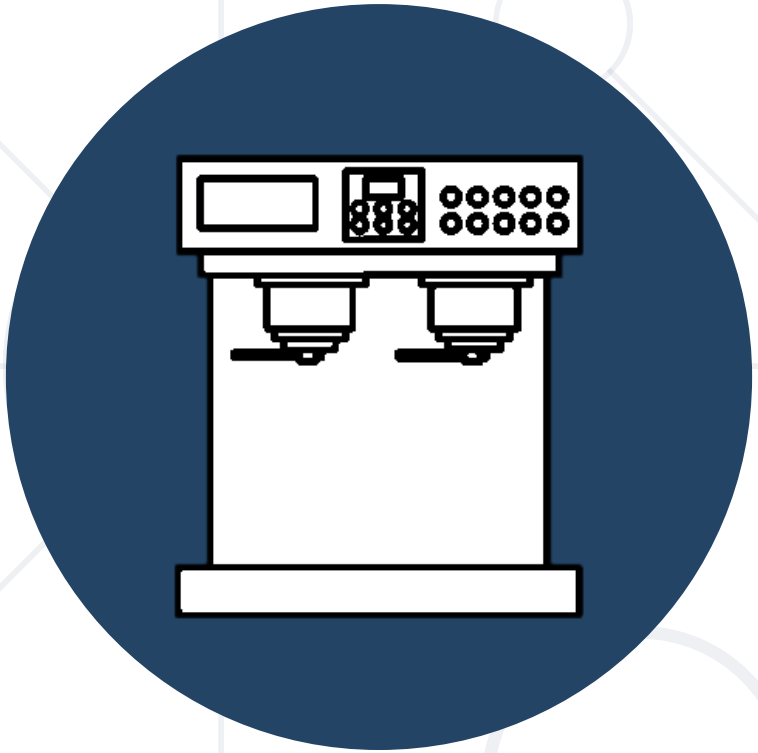
- Class Rectangle implements the method **`draw_circle`** that it has no use of
- Class Circle implements the method **`draw_rectangle`**

```
class Rectangle(Shape):
    def draw_rectangle(self):
        pass
    def draw_circle(self):
        pass
class Circle(Shape):
    def draw_rectangle(self):
        pass
    def draw_circle(self):
        pass
```

ISP Approaches

- To make Shape conform to the ISP principle, we segregate the actions into **different classes**
  - Classes Circle and Rectangle can inherit from class Shape and implement **their own** draw **behavior**

```
class Shape:
    def draw(self):
        raise NotImplementedError
class Rectangle(Shape):
    def draw(self):
        ...
class Circle(Shape):
    def draw(self):
        ...
```



Dependency Inversion

Dependency Inversion

- Interesting design principle by which we protect our code by making it **independent** of things that are fragile, volatile, or out of our control
- Depend on **abstractions**, **not** on **concretions**
  - High-level modules should not depend on low-level modules. **Both** should depend on **abstractions**
  - Abstractions should not depend on details. **Details** should depend on **abstractions**

Dependency Injection

Software University

- Software engineering technique for defining the **dependencies** among objects
- Why use Dependency Injection?
  - **Decreases coupling** between a class and its dependency
  - Can be applied to legacy code as a **refactoring** because it doesn't require any changes in code behavior
  - Allows a client to **remove** all knowledge of a **concrete implementation** that it needs to use

Example

```
class Email:
    def send_email(self):
        pass

class Notification:
    def __init__(self):
        self._email = Email()

    def promotional_notification(self):
        self._email.send_email()
```

Notification depends on Email

Example: Constructor Injection

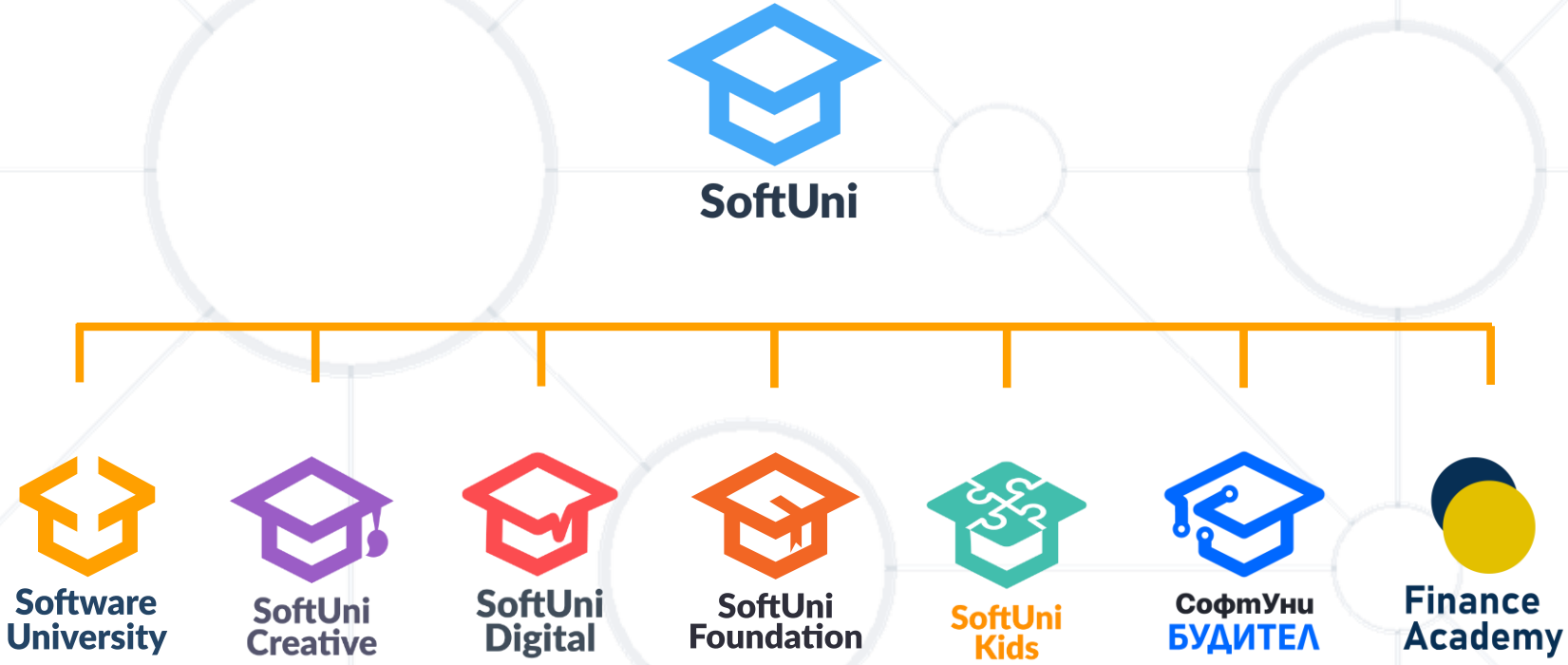
```
class MessageService:
    def send_message(self):
        pass

class Email(MessageService):
    def send_message(self):
        ...

class Notification:
    def __init__(self, service: MessageService):
        self._service = service

    def promotional_notification(self):
        self._service.send_message()
```

Using abstraction

Summary

- SOLID principles make your code more:
  - Extendable
  - Logical
  - Easier to read

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, softuni.org
- 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://softuni.org
- © Software University – https://softuni.bg