Clean Code SOLID

The Single Responsibility Principle

Pre-work

- Video: https://cleancoders.com/episode/clean-code-episode-9
- Exam: https://cleancoders.com/episode/clean-code-episode-9/exam

Chapters

Timetable

Activity	Time
Greetings, Warmup	5 min
Separation of concerns	10 min
Exercise 1	10 min
Group discussion	10 min
SPR overview	10 min
Exercise 2	15 min
Group discussion	10 min
Exercise 3	5 min
Group discussion	10 min
Summary	10 min
Closing	5 min

Warmup

- What are the common challenges that you face when creating unit tests for your code?
  - Type in the meeting chat

Separation of Concerns

- How?
  - Reduce artificial dependencies
  - Simplify change
- What?
  - Split
  - Segregate
  - Extract
- Orthogonality
- Cohesion
- Single Responsibility Principle (SRP)

Exercise 1

- Prompt
  - Rooms 1, 3, 5, 7, 9
    - Describe the method to define responsibilities.
    - What to do if class has multiple functions?
    - Are all of them a separate responsibility?
  - Rooms 2, 4, 6, 8, 10
    - How users/actors/roles/responsibilities correlate?
    - Are they the same? Could they differ?
    - Provide example of same and different combinations of users, actors, roles, and responsibilities.
- Time limit: 10 minutes

Definitions

- Actor - single source of change
  - Roles
  - Separate users of your software from the roles they playing
- Responsibility
  - Responsibility, that your software has, is the responsibility to serve different groups of customers who consumes those services.
  - Responsibility is the source of change.
  - Responsibilities are tied to the actors, not the users
  - To find responsibilities, find families of functions with the similar audience/users requesting changes.
- Two values of software
  - Ability to change frequently - primary value
  - Expected behavior - secondary value

SRP Overview

- Group only those things that truly belong together, and separate everything that does not strictly belong
- How to design software:
  - Identify the actors
  - Responsibilities that serve those actors
  - Allocate those responsibilities in modules
  - Structure the software so that responsibilities became a plugins to the rest of the application
  - Separate source files

Exercise 2

- Code review practice:
  - Use the worksheet
  - Based on the code, suggest code improvements, explain why.
  - Use the worksheet to record your suggestions
- Time limit: 15 min

Exercise 2 code for review

```
def place_order(customer, product, quantity):
    # Calculate the order total
    total = product.price * quantity

    # Check if the customer has enough credit
    if customer.credit >= total:
        # Reduce the customer's credit
        customer.credit -= total

        # Create an order object and add the order to the database
        order = Order(
            customer=customer, product=product, quantity=quantity, total=total
        )
        db.add_order(order)

        # Update the product inventory
        product.inventory -= quantity
        db.update_product(product)

        # Send an email confirmation to the customer
        email.send_confirmation_email(customer.email, order)
        return order
    else:
        raise ValueError("Insufficient credit")
```

Group discussion

- Groups to share their findings

Exercise 2 solution

```
def place_order(customer, product, quantity):
    order_total = calculate_order_total(product, quantity)
    check_customer_credit(customer, order_total)
    reduce_customer_credit(customer, order_total)
    order = create_order(customer, product, quantity, order_total)
    add_order_to_database(order)
    update_product_inventory(product, quantity)
    send_email_confirmation(customer, order)
    return order


def check_customer_credit(customer, order_total):
    if customer.credit < order_total:
        raise ValueError("Insufficient credit")


def reduce_customer_credit(customer, order_total):
    customer.credit -= order_total
```

Exercise 2 solution (cont'd)

```
def calculate_order_total(product, quantity):
    return product.price * quantity


def create_order(customer, product, quantity, order_total):
    return Order(
        customer=customer, product=product, quantity=quantity, total=order_total
    )


def add_order_to_database(order):
    db.add_order(order)


def update_product_inventory(product, quantity):
    product.inventory -= quantity
    db.update_product(product)


def send_email_confirmation(customer, order):
    email.send_confirmation_email(customer.email, order)
```

Exercise 3

- Code review practice:
  - Propose the code changes for the code snippet
- Use the worksheet to record your suggestions
- Time limit: 5 min

Exercise 3 code

```
class Car:
    def __init__(self, engine_size, num_doors):
        self.engine_size = engine_size
        self.num_doors = num_doors

    def start(self):
        # code to start the car's engine

    def lock_doors(self):
        # code to lock the car's doors

    def play_music(self):
        # code to play music in the car
```

Group discussion

- Groups to share their findings

Exercise 3 potential solution

```
class Car:
    def __init__(self, engine_size, num_doors):
        self.engine_size = engine_size
        self.num_doors = num_doors
        self.engine = Engine()
        self.music_player = MusicPlayer()
        self.doors = DoorLocks()


class Engine:
    def start(self):
        # code to start the car's engine


class MusicPlayer:
    def play_music(self):
        # code to play music in the car


class DoorLocks:
    def lock(self):
        # code to lock the car's doors
```

Summary

- Conformance with SRP might require pulling apart code/functions/classes/components
- Potential solutions:
  - Dependency inversion
  - Extract classes
  - Use design patterns (facade)
  - Interface segregation
- None of the solution are perfect
- Carefully allocating responsibilities to classes and modules we keep the primary value of software high
- When module has more than one responsibility, the system tends to become fragile

What is next?

- Coding dojo to practice the Single Responsibility Principle
- Expect an e-mail with instructions for upcoming coding dojo

Final words

Always leave the code better than you found it.
– The Software Craftsmanship Rule