

# 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

---

Chapter	Time
Overview	00:01:01
General Relativity	00:03:54
Responsibility	00:10:05
It's About Users	00:12:50
It's About Roles	00:14:37
Reprise	00:15:50
The Two Values of Software	00:16:19
Friction	00:20:01
CM Collision	00:20:58
Fan Out	00:22:41
Colocation is Coupling	00:24:15
Encroaching Fragility	00:26:49
SRP	00:27:51

---

---

Chapter	Time
Examples	00:30:12
Conclusion	00:39:58
Solutions	00:41:15
Invert Dependencies	00:41:38
Extract Classes	00:42:45
Facade	00:44:05
Interface Segregation	00:45:02
Welcome to Engineering	00:45:53
Case Study	00:48:06
Architecture	00:53:47
Design	00:55:20
Faking it	00:57:01
Conclusion	01:00:14

---

## 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

---

```
1 def place_order(customer, product, quantity):
2     # Calculate the order total
3     total = product.price * quantity
4
5     # Check if the customer has enough credit
6     if customer.credit >= total:
7         # Reduce the customer's credit
8         customer.credit -= total
9
10        # Create an order object and add the order to the database
11        order = Order(customer=customer, product=product, quantity=quantity, total=total)
12        db.add_order(order)
13
14        # Update the product inventory
15        product.inventory -= quantity
16        db.update_product(product)
17
18        # Send an email confirmation to the customer
19        email.send_confirmation_email(customer.email, order)
20        return order
21    else:
22        raise ValueError("Insufficient credit")
```

---

## Group discussion

- Groups to share their findings

## Exercise 2 solution

---

```
1 def place_order(customer, product, quantity):
2     order_total = calculate_order_total(product, quantity)
3     check_customer_credit(customer, order_total)
4     reduce_customer_credit(customer, order_total)
5     order = create_order(customer, product, quantity, order_total)
6     add_order_to_database(order)
7     update_product_inventory(product, quantity)
8     send_email_confirmation(customer, order)
9     return order
10
11 def check_customer_credit(customer, order_total):
12     if customer.credit < order_total:
13         raise ValueError("Insufficient credit")
14
15 def reduce_customer_credit(customer, order_total):
16     customer.credit -= order_total
```

---

## Exercise 2 solution (cont'd)

---

```
1 def calculate_order_total(product, quantity):
2     return product.price * quantity
3
4 def create_order(customer, product, quantity, order_total):
5     return Order(customer=customer, product=product, quantity=quantity, total=order_total)
6
7 def add_order_to_database(order):
8     db.add_order(order)
9
10 def update_product_inventory(product, quantity):
11     product.inventory -= quantity
12     db.update_product(product)
13
14 def send_email_confirmation(customer, order):
15     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

---

```
1 class Car:
2     def __init__(self, engine_size, num_doors):
3         self.engine_size = engine_size
4         self.num_doors = num_doors
5
6     def start(self):
7         # code to start the car's engine
8
9     def lock_doors(self):
10        # code to lock the car's doors
11
12    def play_music(self):
13        # code to play music in the car
```

---



## Group discussion

- Groups to share their findings

## Exercise 3 potential solution

---

```
1 class Car:
2     def __init__(self, engine_size, num_doors):
3         self.engine_size = engine_size
4         self.num_doors = num_doors
5         self.engine = Engine()
6         self.music_player = MusicPlayer()
7         self.doors = DoorLocks()
8
9
10 class Engine:
11     def start(self):
12         # code to start the car's engine
13
14
15 class MusicPlayer:
16     def play_music(self):
17         # code to play music in the car
18
19
20 class DoorLocks:
21     def lock(self):
22         # 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*