Clean Code Fundamentals

Polly want a message

Pre-work

- Video: https://www.youtube.com/watch?v=XXi_FBrZQiU

Chapters

Chapter	Time
Design stamina hypothesis	03:50
Procedures vs. OO	05:59
Churn vs. complexity	07:49
Design star anti-pattern	11:04
Easy is the enemy of simple	15:35
Affordances	21:35

Chapter	Time
Anthropomorphic polymorphic	24:24
Resolution	29:55
Isolate thing you want to vary	32:13
Push conditionals back on stack	33:15
Dependency injection	34:47
But, conditionals?	35:24

Timetable

Activity	Time
Warmup	5 min
Exercise 1	30 min
Exercise 2	30 min
Exercise 3	20 min
Wrap up	5 min

Warmup

- Do you have the “death star” anti-pattern examples from your code?
    - Type in the meeting chat

Exercise 1

- Prompt
  1. How object-oriented design principles can improve (or detract) the success of a project or application?
  2. Are there any similarities or differences between OOP and TDD?
- Time limit: 10 minutes

Discussion

- Groups to share their findings

Possible answers

- Pros
  - Modularity and reusability
    - Can reduce time and effort required to develop and maintain the code
  - Abstraction and encapsulation
    - Hides the implementation details, making the code more readable
  - Flexibility and adaptability
    - Loose coupling and polymorphism makes it easier to change or update individual objects without affecting the rest of the system
  - Improved debugging and testing
    - Makes it easier to isolate and debug individual objects
- Cons
  - If not properly applied, can lead to complex and hard-to-maintain code, or overly complex class hierarchies ("Faux OO")
  - Not always the best fit for every type of problem or project

Exercise 2

- Dependency injection
  - Design pattern that involves passing objects as dependencies to an object, rather than creating these dependencies within the object itself
- Prompt
  - Discuss the benefits and challenges of using dependency injection in a project that involves multiple objects that depend on each other
  - Consider the following questions:
    1. How does dependency injection help decouple the objects in the project?
    2. What are the trade-offs of using dependency injection versus creating dependencies within the objects themselves?
    3. How does dependency injection impact the maintainability and scalability of the project?
    4. Can you think of a real-world example where dependency injection would be useful?
- Time limit: 15 minutes

Discussion

- Groups to share their findings

Exercise 3

- What is polymorphism
  - The quality where different kinds of objects can respond to the same message
  - Objects can take on many forms and behave differently based on context
- Prompt
  1. How does polymorphism make code more expressive and modular?
  2. What are the trade-offs of using polymorphism, and how to mitigate them?
  3. How can you test code with polymorphism, and what are the challenges?
- Time limit: 10 minutes

Discussion

- Groups to share their findings

Possible answers

- How does polymorphism make code more expressive and modular?
  - Creates a more dynamic code, as the same message can have different effects on different objects
  - Leads to more modular code, as different implementations of a message can be separated into different classes; easier to reuse and maintain the code
- What are the trade-offs of using polymorphism, and how to mitigate them?
  - Can make the code harder to understand and maintain since the same message can have different effects on different objects
  - Can lead to runtime errors if the message is sent to an object that doesn't have an implementation of that message
- How can you test code with polymorphism, and what are the challenges?
  - Test the behavior of each implementation of a message
  - Ensure that the code functions correctly when you send messages to objects of different types

Summary

- OOP affords . . . objects
  - Anthropomorphic
    - The attribution of human traits, emotions or intentions to non-human entities
  - Polymorphic
    - The quality where different kinds of objects can respond to the same message
  - Loosely-coupled
    - Objects strive for independence
  - Role-playing
    - Objects are more players of their roles than instances of their types
  - Factory-created
    - Factories hide the rules of picking the right player for a role
  - Message-sending
    - "I know what I want, you know how to do it"

What is next?

- Next session
  - We will do dojo of participants' choice
- 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