Exercise 3 - Software Engineering Design - 2025

2. The GRASP (General Responsibility Assignment Software Patterns) paper by Aldrich provides a structured way to assign responsibilities to classes and objects in object-oriented design. Here's how you can find the responsible class for a method and for the creation of an object, according to the GRASP principles:

Finding the Responsible Class for a Method
Use the "Information Expert" pattern:

Definition: Assign responsibility to the class that has the necessary information to fulfill the method.

How to Apply:
Identify what the method needs to do (what information is required).
Determine which class has the data or knowledge needed.
Assign the method to that class.

Example: If a method calculates a customer's total order cost, the class that has access to line items and prices (e.g., Order) should be responsible for that calculation.

Finding the Responsible Class for Object Creation
Use the "Creator" pattern:

Definition: Assign responsibility for creating an instance of class A to class B if one or more of the following is true:
B contains A
B aggregates A
B records instances of A
B closely uses A
B has the initializing data for A
How to Apply:
Analyze which class logically aggregates or uses the class being created.
Check if it has the data necessary to instantiate the new object.
Delegate the creation responsibility to that class.

Example: If an Order has multiple LineItem objects, then Order should be responsible for creating LineItem instances.

3.

Class WebsiteMonitor

Responsibility	Collaboration
• Add/remove subscription • Periodically check subscribed websites • Detect changes • Trigger notifications	• Subscription • Notification • NotificationChannel

- **Controller principle justification:**
  Coordinates system operations and workflow logic. Centralizes control.
- **Change:**
  The logic to check for updates and trigger notifications was moved from Subscription to WebsiteMonitor to obey the **Controller** pattern.

Class User

Responsibility	Collaboration
• Register to the system • Manage subscriptions (create, modify, cancel)	• Subscription • WebsiteMonitor (delegates processing)

- **Expert justification:**
  User has the subscription data.

Class Subscription

Responsibility	Collaboration
• Store URL, frequency, and notification channel • Modify/cancel details	• User • NotificationChannel • WebsiteMonitor

- **Expert justification:**
  Subscription knows its own configuration.

Class Notification

Responsibility	Collaboration
• Create a message when an update is detected • Delegate sending to the chosen channel	• User • NotificationChannel

- **Expert justification:**
  Notification handles the delivery logic since it owns the message content.

Interface CommunicationChannel

Responsibility	Collaboration
• Define method to send a notification	• Notification

- **Design Note:**
  Allows Notification to be decoupled from communication methods.

Interface EmailChannel / SMSChannel / PushNotificationChannel

Responsibility	Collaboration
• Define method to send a notification	• NotificationChannel

- **Expert justification:**
  These classes know how to deliver notifications via their respective medium.

Enum Frequency

Responsibility	Collaboration
• Define possible values: MONTHLY, DAILY, WEEKLY	• Subscription

Changes made to the responsibilities:
Originally, Subscription might have been responsible for detecting updates. But following the Controller pattern, this was moved to WebsiteMonitor to centralize orchestration.
Also, Notification doesn't send messages directly. It delegates to a NotificationChannel, adhering to the Dependency Inversion Principle.