**SWED Exc.3**

**Aufgabe 2**

**Verantwortliche Klasse für eine Methode finden**

Um die verantwortliche Klasse für eine Methode zu bestimmen, verwendet man das **Information Expert Prinzip** aus GRASP[[1]](file:///C:\\Users\\omidf\\Downloads\\kannst%20du%20die%20aufgabe%20aus%20dem%20screenshot%20lösen.docx" \l "fn1). Dieses Prinzip besagt, dass die Verantwortung der Klasse zugewiesen werden soll, die über die notwendigen Informationen verfügt, um diese Verantwortung zu erfüllen[[1]](file:///C:\\Users\\omidf\\Downloads\\kannst%20du%20die%20aufgabe%20aus%20dem%20screenshot%20lösen.docx" \l "fn1).

**Vorgehen:**

1. **Verantwortung klar definieren** - Was soll die Methode genau tun?
2. **Benötigte Informationen identifizieren** - Welche Daten werden zur Erfüllung der Aufgabe benötigt?
3. **Information Expert finden** - Welche Klasse besitzt diese Informationen oder hat den direktesten Zugang dazu?

**Beispiel aus dem PDF:** Für die Berechnung der Gesamtsumme eines Verkaufs (getTotal()) ist die Klasse Sale der Information Expert, da sie die Verkaufsposten (SalesLineItem) kennt und somit Zugang zu allen notwendigen Informationen hat[[1]](file:///C:\\Users\\omidf\\Downloads\\kannst%20du%20die%20aufgabe%20aus%20dem%20screenshot%20lösen.docx" \l "fn1).

**Verantwortliche Klasse für Objekterstellung finden**

Für die Objekterstellung wird das **Creator Prinzip** angewendet[[1]](file:///C:\\Users\\omidf\\Downloads\\kannst%20du%20die%20aufgabe%20aus%20dem%20screenshot%20lösen.docx" \l "fn1). Eine Klasse B sollte für die Erstellung von Objekten der Klasse A verantwortlich sein, wenn:

* **B aggregiert A-Objekte**
* **B enthält A-Objekte**
* **B registriert Instanzen von A-Objekten**
* **B verwendet A-Objekte häufig**
* **B hat die Initialisierungsdaten für A-Objekte**[[1]](file:///C:\\Users\\omidf\\Downloads\\kannst%20du%20die%20aufgabe%20aus%20dem%20screenshot%20lösen.docx" \l "fn1)

**Bevorzugte Regel:** Wenn eine Wahl besteht, sollte die Klasse gewählt werden, die A-Objekte aggregiert oder enthält[[1]](file:///C:\\Users\\omidf\\Downloads\\kannst%20du%20die%20aufgabe%20aus%20dem%20screenshot%20lösen.docx" \l "fn1).

**Beispiel aus dem PDF:** Für die Erstellung von SalesLineItem-Objekten ist die Klasse Sale verantwortlich, da sie diese Objekte enthält und die notwendigen Daten für deren Erstellung besitzt[[1]](file:///C:\\Users\\omidf\\Downloads\\kannst%20du%20die%20aufgabe%20aus%20dem%20screenshot%20lösen.docx" \l "fn1).

**Vorteile des Creator Prinzips:**

* **Fördert lose Kopplung**, da Klassen die Objekte selbst erstellen, die sie benötigen
* **Vermeidet Abhängigkeiten** zu anderen Klassen für die Objekterstellung[[1]](file:///C:\\Users\\omidf\\Downloads\\kannst%20du%20die%20aufgabe%20aus%20dem%20screenshot%20lösen.docx" \l "fn1)

Diese GRASP-Prinzipien helfen dabei, Verantwortlichkeiten methodisch und nachvollziehbar zu verteilen, was zu besser strukturiertem und wartbarerem Code führt

**Aufgabe 3**

Klassen:

1. User
2. Website
3. Subscription
4. NotificationPreferences
5. Notification

Controller Klassen nach GRASP:

* SubscriptionController (oder allgemeiner WebsiteMonitorController für Benutzerinteraktionen)
* MonitoringService (für den Hintergrundprozess der Webseitenüberprüfung)
* NotificationService (für die Erstellung und den Versand von Benachrichtigungen)
* NotificationDispatcher (als spezialisierter Helfer für den Versand)

**Class Name: User**  
**Responsibilities:**

* Kennt Benutzeridentität (userID, name).
* Kennt Benutzerkontaktinformationen (contactInfo).
* Stellt Informationen für die Erstellung von Abonnements bereit.
* Empfängt Benachrichtigungen (konzeptionell).  
  **Collaborators:**
* SubscriptionController (zur Verwaltung von Abonnements)
* Subscription (um Details seiner Abonnements einzusehen)

**Class Name: Website**  
**Responsibilities:**

* Kennt seine URL (url).
* Kennt den Zeitpunkt der letzten Überprüfung (lastCheckedTimestamp).
* Kennt den Zeitpunkt der letzten erkannten Aktualisierung (lastUpdateDetectedTimestamp).
* **Expert:** Überprüft sich selbst auf Inhaltsänderungen (checkForUpdates()).  
  **Collaborators:**
* MonitoringService (wird von diesem zur Überprüfung aufgefordert)
* (Optional: ContentFetcherService, falls das Abrufen von Inhalten komplex ist)

**Class Name: Subscription**  
**Responsibilities:**

* Verknüpft einen User mit einer Website.
* **Expert:** Kennt die zugehörigen NotificationPreferences.
* Stellt die für eine Benachrichtigung notwendigen Daten bereit (User, Website, Präferenzen).  
  **Collaborators:**
* User
* Website
* NotificationPreferences
* SubscriptionController (für Erstellung, Änderung)
* MonitoringService (wird von diesem verarbeitet)
* NotificationService (liefert Daten für Benachrichtigung)

**Class Name: NotificationPreferences**  
**Responsibilities:**

* **Expert:** Kennt die gewünschte Benachrichtigungsfrequenz (frequency).
* **Expert:** Kennt den bevorzugten Kommunikationskanal (communicationChannel).
* Stellt diese Einstellungen bereit.  
  **Collaborators:**
* Subscription (ist Eigentümer dieser Einstellungen)
* SubscriptionController (für Änderungen)
* NotificationService (um Einstellungen abzufragen)

**Class Name: Notification**  
**Responsibilities:**

* **Expert:** Kennt den Inhalt der Benachrichtigung (message).
* Kennt den Erstellungszeitpunkt (timestamp).
* **Expert:** Kennt und verwaltet seinen Zustellstatus (status: PENDING, SENT, FAILED).  
  **Collaborators:**
* NotificationService (erstellt Instanzen dieser Klasse)
* NotificationDispatcher (wird von diesem versendet)

**Class Name: SubscriptionController**  
**Responsibilities:**

* **Controller:** Bearbeitet Benutzeranfragen zur Registrierung für Website-Updates (erstellt Subscription, verknüpft User, Website, NotificationPreferences).
* **Controller:** Bearbeitet Benutzeranfragen zur Änderung von Abonnement-Einstellungen (aktualisiert NotificationPreferences für ein Subscription).
* **Controller:** Bearbeitet Benutzeranfragen zur Kündigung eines Abonnements.  
  **Collaborators:**
* User (oder UserRepository zum Laden/Finden)
* Website (oder WebsiteRepository zum Laden/Finden/Erstellen)
* Subscription (oder SubscriptionRepository zum Erstellen/Finden/Löschen)
* NotificationPreferences (oder Fabrik zum Erstellen/Ändern)
* (UI-/API-Schicht, die Anfragen weiterleitet)

**Class Name: MonitoringService**  
**Responsibilities:**

* **Controller:** Identifiziert alle aktiven Subscriptions/Websites, die überprüft werden müssen.
* Koordiniert die regelmäßige Überprüfung von Websites auf Aktualisierungen (ruft Website.checkForUpdates() auf).
* Initiiert den Benachrichtigungsprozess, falls eine Aktualisierung erkannt wird.  
  **Collaborators:**
* Subscription (oder SubscriptionRepository - um die Liste der Abonnements zu erhalten)
* Website (um checkForUpdates() aufzurufen)
* NotificationService (um die Erstellung von Benachrichtigungen auszulösen)

**Class Name: NotificationService**  
**Responsibilities:**

* **Controller:** Erstellt Notification-Instanzen basierend auf erkannten Updates und den Subscription-Details (einschließlich NotificationPreferences).
* Orchestriert den Versand von Notifications.  
  **Collaborators:**
* Subscription (um Benutzer- und Präferenzdaten zu erhalten)
* NotificationPreferences (um Kanal etc. zu erhalten)
* Notification (erstellt und verwaltet Instanzen)
* NotificationDispatcher (delegiert den eigentlichen Versand)
* MonitoringService (wird von diesem aufgerufen, wenn eine Benachrichtigung generiert werden soll)

**Class Name: NotificationDispatcher**  
**Responsibilities:**

* **Expert:** Sendet eine formatierte Notification über den spezifizierten Kommunikationskanal (z.B. E-Mail, SMS) an den Benutzer.  
  **Collaborators:**
* Notification (erhält den Inhalt und implizit die Zieldaten)
* Externe Kommunikationsdienste (z.B. E-Mail-Gateway, SMS-Gateway)

**Änderungen der Verantwortlichkeiten durch Anwendung der Prinzipien "Controller" und "Expert"**

Ja, die Verantwortlichkeiten einiger Methoden bzw. Klassen haben sich durch die Anwendung dieser Prinzipien verfeinert oder verschoben:

1. **Einführung expliziter Controller/Services:**
   * Die ursprüngliche, eher vage System-Klasse wurde durch spezifischere Controller (SubscriptionController) und Services (MonitoringService, NotificationService) ersetzt. Dies folgt dem **Controller-Prinzip**, indem die Bearbeitung von Systemereignissen und Benutzerinteraktionen dedizierten Klassen zugewiesen wird.
   * Die Verantwortung für das *Starten des Registrierungsprozesses* oder das *Ändern von Einstellungen* liegt nun klar beim SubscriptionController, der die Anfragen von der Benutzeroberfläche oder API entgegennimmt und die notwendigen Operationen auf den Fachobjekten koordiniert.
   * Der MonitoringService ist nun der Controller für den Hintergrundprozess der Webseitenüberprüfung.
   * Der NotificationService ist der Controller für die Logik rund um die Erstellung und Initiierung des Versands von Benachrichtigungen.
2. **Website.checkForUpdates()**:
   * Diese Methode war bereits gut auf das **Expert-Prinzip** ausgerichtet. Die Website-Klasse besitzt die notwendigen Informationen (URL, ggf. vorheriger Zustand), um festzustellen, ob eine Aktualisierung stattgefunden hat. Diese Verantwortung bleibt zentral bei der Website-Klasse.
3. **Notification.send() (implizit):**
   * Ursprünglich hätte man denken können, dass die Notification-Klasse sich selbst sendet. Durch die Anwendung des Expert-Prinzips und des Single-Responsibility-Prinzips ist die Notification-Klasse der Experte für ihren *Inhalt* und ihren *Status*. Der eigentliche Versandmechanismus (z.B. per E-Mail) wird an den NotificationDispatcher delegiert. Der NotificationService agiert als Controller, der die Notification-Daten an den NotificationDispatcher weitergibt.
4. **Verantwortung für die Erstellung von Objekten:**
   * Die Erstellung von Subscription, NotificationPreferences und Notification wird nun klarer von den Controllern (SubscriptionController, NotificationService) orchestriert. Diese können dann entweder die Konstruktoren der Klassen direkt aufrufen oder, bei komplexerer Logik, Factory-Methoden oder separate Factory-Klassen verwenden.
5. **Zuweisung von Operationen zum Informations-Experten:**
   * Methoden, die Informationen einer bestimmten Klasse benötigen, um ihre Aufgabe zu erfüllen, wurden stärker dieser Klasse zugeordnet (Expert-Prinzip). Beispielsweise bleibt das Subscription der Experte für seine zugehörigen NotificationPreferences.

Zusammenfassend lässt sich sagen, dass durch die Anwendung der Prinzipien "Controller" und "Expert" eine klarere Trennung der Belange erreicht wurde. Controller-Klassen übernehmen die Koordination und Bearbeitung von Anfragen, während die Fachobjekte (Experten) die Logik und Daten verwalten, für die sie zuständig sind. Dies führt zu einem modulareren, besser wartbaren und verständlicheren Design. Die ursprünglichen Operationen, die unter der User-Klasse oder einer allgemeinen System-Klasse angesiedelt waren, sind nun spezifischeren Controllern und Services zugeordnet.