Exercise 4:

Task1 : Implement the system from Exercise 3 task 1 in Java

Habe ich gemacht in IntelliJ und gespusht aug GitHub.

Task 2: Calculate the metrics afferent coupling, efferent coupling and instability for each class.

Metrik

Afferent Coupling (AC) : Anzahl der Klassen, die diese Klasse benutzen. „Wer ruft mich?“

Efferent Coupling (EC) : Anzahl der Klassen, die diese Klasse verwendet. „Wen benutzte ich“

Instability(I) : Formel : I = EC/(EC+AC). Zeigt wie „instabil“ eine Klasse ist. Wert zwischen 0 (stabil) und 1 (instabl)

How to measure?

- **Efferent (outgoing) coupling** (EC) is the number of classes outside of a particular module that depend on classes within that module.
- **Afferent (incoming) coupling** (AC) refers to the number of classes within a particular module that depend on classes outside of that module.
- **Instability** of a module is defined
  - Closer to 1 means unstable
  - Closer to 0 means stable

$$I = \frac{EC}{EC+AC}$$

Class Analysis:

1.) User Class

Afferent Coupling :

Is used by :

- WebsiteMonitorService
- Notification
- > AC = 2

Efferent coupling

- Uses : no other Classes
- > EC = 0

Instability:

- I = 0/ (2+0) = 0(stable)

2.) Notification Class :

Afferent Coupling :

Is used by :

- WebsiteMoitorService
- > AC = 1

Efferent coupling:

Uses:

- User
- NotificationPreference
- > EC= 2

Instability:

- I = 2/ (1+2)= 0,67

3.) NotificationPreference Class :

Afferent Coupling :

Is used by :

- Subscription
- Notification
- > AC = 2

Efferent coupling:

Uses:

- no other classes
- > EC= 0

Instability:

- I = 0/ (2+0)= 0

4.) Subscription Class :

Afferent Coupling :

Is used by :

- WebsiteMonitorService
- > AC = 1

Efferent coupling:

Uses:

- NotificationPreference
- > EC= 1

Instability:

- I = 1/ (1+1)= 0,5

5.) WebsiteMonitorService Class :

Afferent Coupling :

Is used by :

- Main
- > AC = 1

Efferent coupling:

Uses:

- User
- Subscription
- Notification
- > EC= 3

Instability:

- I = 3/ (1+3)= 0,75

6.) Main Class :

Afferent Coupling :

Is used by :

- none
- > AC = 0

Efferent coupling:

Uses:

- User
- WebsitemonitorService
- Subscription
- Notification
- > EC= 4

Instability:

- I = 4/ (0+4)= 1

Interpretation:

- User and NotificationPreference are **very stable** (no efferent coupling!).
- Main is **extremely unstable** – that's normal since it's just the entry point of the app.
- WebsiteMonitorService has high instability because it uses many classes – **also normal**, since it contains the core logic.

Task 3: Suggest a package structure for your implementation.

Main: Contains the Main class, the app's entry point.
- Main.java

Model: Contains the core data structures
(User, Notification, NotificationPreference, Subscription).

- User.java
- Subscription.java
- Notification.java
- NotificationPreferences.java

Service: Contains the WebsiteMonitorService, which houses the core logic and interactions.
- WebsiteMonitorService.java

Task 4: Commit your software to a new github.com repository

1.Initialisiere ein lokales Git-Repository:

git init

2.Füge deine Dateien hinzu:

git add .

3.Mach erster Commit:

git commit -m "Initialer Commit"

4. Dann auf GitHub:

Neuen Repository erstellen (z.B: WebsiteMonitor)

5. Verbinden lokales Repository mit GitHub:

git remote add origin https://github.com/Nisack16/WebsiteMonitor.git

Task 5: Name options to reduce coupling between your packages.

Define Interfaces:
I plan to use interfaces (like INotificationSender) so that my classes, such as Notification, are better abstracted and more easily replaceable.

Dependency Injection:
I want to inject dependencies like Notification or User through the constructor instead of creating them directly inside WebsiteMonitorService. This keeps my code modular and easier to test.

Factory Pattern:
I'm considering using a factory (e.g., NotificationFactory) to provide objects. This reduces direct references between packages and makes my code easier to maintain.