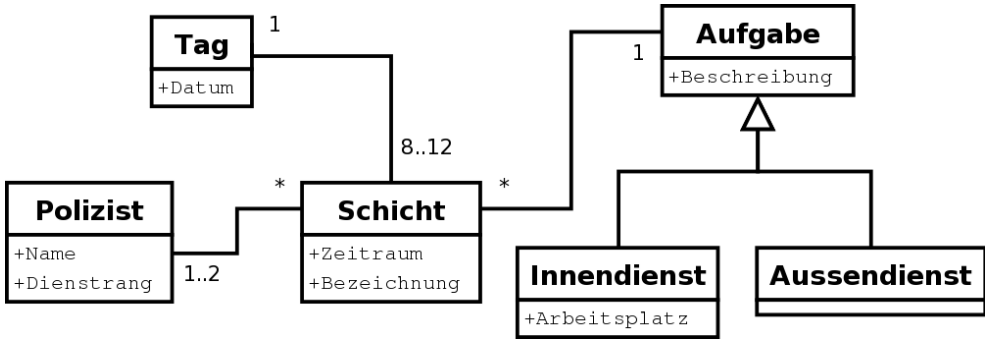
Bitte tragen Sie individuell bis 08:00 Uhr am Vortag des Seminars über den Checklistenbaustein in Opal ein, welche Aufgaben Sie präsentieren können. Für die Präsentation sollten Sie ein Dokument im Querformat vorbereitet haben, das Sie dann entsprechend im Seminar in BigBlueButton hochladen können. Auch müssen Sie den virtuellen Klassenraum mit Mikrofon betreten.

Aufgabe 1: OCL

Eine Polizeiwache möchte ihre Dienstpläne durch eine Software unterstützen. Es liegt bereits ein erster Entwurf für das Klassendiagramm vor.

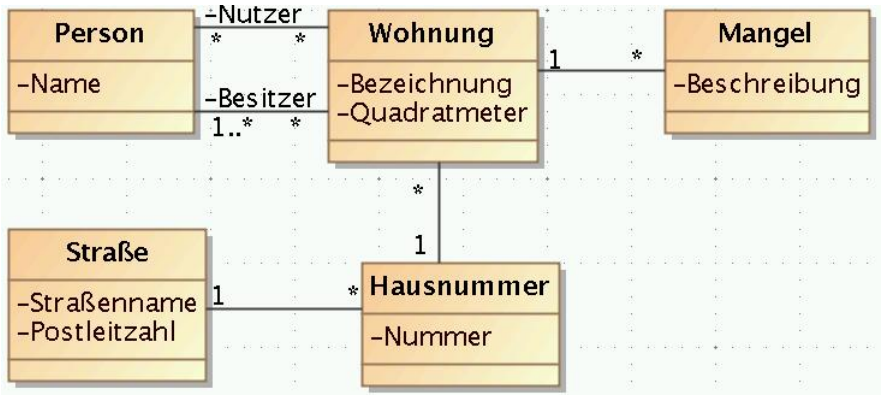


Beschreiben Sie die folgenden Sachverhalte im Klassendiagramm mit OCL.

- Schichten des Aussendiensts sind immer mit 2 Polizisten besetzt.
- Alle Schichten eines Tages decken mindestens 8 verschiedene Aufgaben ab.
- Der Nachtschicht ($\texttt{Zeitraum} = 0$) werden keine Polizisten mit $\texttt{Dienstrang} > 5$ zugeordnet.

Aufgabe 2: Kommunikations- und Sequenzdiagramm

In einer Software sind Daten gemäß des folgenden Klassendiagramms gespeichert.



a) Erstellen Sie ein Kommunikationsdiagramm für die folgenden Anfrageereignisse:

- Wohnraum ermitteln
  Für eine Straße soll die gesamte in dieser Straße vorhandene Größe des Wohnraums (in Quadratmetern) ausgegeben werden.

- Mangelliste ausgeben
  Für eine gegebene Person sollen alle Wohnungen in ihrem Besitz mit genauer Anschrift sowie den gespeicherten Mängeln ausgegeben werden.

b) Erstellen Sie ein Sequenzdiagramm für die folgenden Mutationsereignisse:

- Haus-Mangel erstellen
  Alle Wohnungen in einem Haus (gegeben durch Straße und Hausnummer – Einstieg ist das Hausnummer-Objekt) werden mit einem neuen Mangel (z.B. "Heizung funktioniert nicht") versehen.
- Haus verkauft
  Ein Haus (wie oben: Hausnummer-Objekt ist Einstieg) wird vollständig an einen neuen (noch nicht erfassten) Besitzer verkauft. Der alte Besitzer wird gelöscht.

Aufgabe 3: Zustandsdiagramm

Modellieren Sie als Zustandsdiagramm die folgende Beschreibung:

/1/ Im Normal- und im Debug-Betrieb können Eingaben getätigt werden, was in der Konfiguration nicht möglich ist.

/2/ Der Konfiguration-Betrieb wird durch Eingabe eines Passworts erreicht. Beim Abbruch der Konfiguration ist man im Normal-Betrieb, sonst gelangt man direkt in den Debug-Betrieb.

/3/ Zwischen Normal- und Debug-Betrieb kann gewechselt werden.

/4/ Bei Inbetriebnahme startet das System im Debug-Betrieb.

Aufgabe 4: LCS

Wenden Sie den LCS-Algorithmus auf die folgenden beiden Zeichenketten an:
$x = acccbacc$ und $y = acbbccacb$. Lesen Sie aus dem Ergebnis des Algorithmus eine möglichst kurze Folge von Einfüge-, Lösch- und Ersetzoperationen ab, um x in y zu überführen.

Aufgabe 5: Praktische Übung mit Git

Installieren Sie auf Ihrem Rechner die Software Git – in Linux-Distributionen ist dies üblicherweise ein Paket (z.B. `git` in Ubuntu), in Windows können Sie die Software unter `https://git-scm.com/download/win` und unter Mac OS unter `https://git-scm.com/download/mac` herunterladen.

Starten Sie unter Windows das Programm `GitBash` bzw. unter Linux ein Terminal. In dieser Übungsaufgabe werden die Git-Befehle direkt in der so gestarteten Konsole und nur auf ihrem eigenen lokalen Archiv durchgeführt.

Die Aufgabe ist im Stil eines schrittweisen Tutorials gehalten. Protokollieren Sie die Ausgaben die Sie erhalten und erklären Sie selbige.

1. Erzeugen Sie ein Verzeichnis für die Bearbeitung dieser Übungsaufgabe (mit Unix-Kommando `mkdir`) und wechseln Sie in dieses Verzeichnis (mit `cd`).
2. Einstellungen für das Git-Archiv (Ersetzen Sie die Bezeichner in spitzen Klammern):

```
git config --global user.name "<ihr name>"
git config --global user.email "<ihr.name>@stud.htwk-leipzig.de"
```

3. ggf. Einstellungen für die Arbeit unter Windows:

   ```
   git config --global core.autocrlf true
   git config --global core.safecrlf true
   ```

4. Ein neues leeres Archiv anlegen:

   ```
   git init
   ```

5. Legen Sie mit einem einfachen Editor (`TextPad`, `gedit`, ...) eine Datei `lessonslearnt.txt` in diesem Verzeichnis an, in dem Sie stichpunktartig mit Anstrichen die acht wichtigsten Dinge aufführen, die Sie bisher in Softwaretechnik gelernt haben. Legen Sie ferner eine Datei in einem Unterverzeichnis `punkte/blatt2.txt` an, in der Sie die Anzahl der bearbeiteten und in Opal eingetragenen Aufgaben des Übungsblatts 2 eintragen. Überprüfen Sie den Status des Archivs mit

   ```
   git status
   ```

6. Übergeben Sie die Dateien der Git-Versionskontrolle (Staging):

   ```
   git add .
   ```

   Überprüfen Sie erneut den Status von Git.

7. Übertragen Sie die Dateien in das Archiv:

   ```
   git commit -m "erste Version"
   ```

   (Wird der Kommentar mit `-m` vergessen, öffnet sich ein Editor – i.d.R. ein Editor im Stile der Editor-Legende `vi` aus dem Jahre 1976: dort den Kommentar eintragen, ESC drücken und `:wq` ENTER eingeben.) Führen Sie anschließend `git show` aus.

8. Ändern Sie die Datei `lessonslearnt.txt`, indem Sie für einen Punkt kommentieren, was nicht gut erklärt war. Modifizieren Sie ferner die Datei `punkte/blatt2.txt`, indem Sie die Anzahl der bearbeiteten Aufgaben um 1 erhöhen.

9. Führen Sie die folgenden Kommandos aus:

   ```
   git status
   git diff
   ```

10. Bestimmen Sie, welche Änderungen in das Archiv sollen (Staging):

    ```
    git add lessonslearnt.txt
    ```

    Führen Sie nochmals `git diff` aus. Führen Sie `git diff --cached` aus. Was wird jeweils angezeigt?

11. Übertragen Sie die Änderung in das Archiv:

```
git commit -m "Kritik eingetragen"
```

Geben Sie nochmals die beiden Diff-Kommandos ein.

12. Sie können die aktuelle Version mit einer Markierung versehen, um später wieder darauf zuzugreifen:

    ```
    git tag zweiteversion
    ```

13. Nehmen Sie nochmals Veränderungen an der Datei `lessonslearnt.txt` vor: Zeile löschen, Zeile modifizieren, Zeile einfügen. Legen Sie ferner eine weitere Datei mit beliebigem Inhalt an.

14. Geben Sie das folgende Kommando ein:

    ```
    git commit -a -m "Fehler entfernt"
    ```

    Überprüfen Sie Status und Differenz. Was macht die Option `-a`?

15. Versehen Sie die aktuelle Version mit einem Tag `dritteversion`. Und geben Sie den Unterschied mit

    ```
    git diff zweiteversion..dritteversion
    ```

    aus.

16. Wechseln Sie in die frühere Version mit

    ```
    git checkout zweiteversion
    ```

17. Wechseln Sie zurück in die letzte Version und prüfen Sie die Versionsgeschichte mit

    ```
    git log
    ```

    und

    ```
    git log --pretty=oneline
    ```

Aufgabe 6: Git: Arbeit mit Branches

Diese Aufgabe ist die wichtigere Aufgabe der beiden Git-Aufgaben! Die angesprochenen Programme müssen nicht lauffähig sein - es geht nur um die Arbeit mit Git. Protokollieren Sie auch hier alle Schritte.

1. Legen Sie ein neues Verzeichnis für ein neues Git-Archiv an und initialisieren Sie das Archiv. Legen Sie eine Java-Datei mit einer möglichst einfachen Bubbelsort-Klasse an (ja, auch das geht zur Not im TextPad! – erster Algorithmus von https://de.wikipedia.org/wiki/Bubblesort). Fügen Sie die Datei per add und commit in das Archiv ein.

2. Fügen Sie eine Ausgabe am Ende der Bubblesort-Methode ein und übertragen sie in das Archiv.

3. Erstellen Sie einen neuen Entwicklungszweig mit

```
git branch fix
git checkout fix
git branch
```

4. Ändern Sie die Sortierung von aufsteigend zu absteigend (im Vergleich des If-Befehls). Legen Sie ferner eine Textdatei an, die eine Testeingabe mit vielen Vertauschsoperationen skizziert. Fügen Sie diese Datei hinzu und übertragen alle Änderungen in das Archiv.

5. Wechseln Sie in den Hauptentwicklungszweig

   ```
   git checkout master
   ```

   Was ist geschehen? Sind alle Dateien noch da?

6. Ändern Sie die äußerste For-Schleife von Bubblesort, sodass das letzte Element nicht bei der Sortierung berücksichtigt wird. Übertragen Sie die Änderungen in das Archiv.

7. Führen Sie die Änderungen aus beiden Entwicklungszweigen zusammen:

   ```
   git merge fix
   ```

   Übertragen Sie die Version mit den gemeinsamen Änderungen in den Hauptentwicklungszweig des Archivs.

8. Nehmen Sie in zwei Entwicklungszweigen an der aktuellen Version von Bubblesort Änderungen vor – es sollte aber immernoch ein Algorithmus sein! – , die nicht konfliktfrei durch ein merge zusammengeführt werden können. Führen Sie den Merge durch, lösen Sie den Konflikt auf und übermitteln Sie wieder eine gemeinsame Version in das Archiv.

9. Visualisieren Sie die Branching-Historie des Archivs – in Window können Sie `Git GUI` starten, in Linux geht `git log --graph --decorate` bzw. das Zusatzprogramm `gitk`.