

Herbst 2021/22 – B  
252-0027 – Einführung in die Programmierung

Departement Informatik  
ETH Zürich

27. Januar 2022 – Programmieren

Nachname: \_\_\_\_\_ Vorname: \_\_\_\_\_

Legi-Nummer: \_\_\_\_\_ – \_\_\_\_\_ – \_\_\_\_\_ Computer: slab \_\_\_\_\_

Sie dürfen diese Prüfung oder die schriftliche Prüfung erst öffnen nachdem die Aufsicht die Prüfung gestartet hat. **Wenn Sie diese Dokumente vorher öffnen gilt dies als Täuschungsversuch.**

Mit Ihrer Unterschrift bestätigen Sie, dass Sie die hier aufgeführte Person sind, Sie die Hinweise zur Kenntnis genommen haben, Sie die Aufgaben selbständig bearbeitet haben, Sie Ihre eigene Lösung abgeben, Sie keine Kopie der Prüfung mitnehmen, Sie alle technischen Probleme und etwaige störende äussere Einflüsse gemeldet haben bzw. wissen, dass Sie diese melden sollen, und dass Sie keine gesundheitlichen Probleme hatten, die Ihre Leistungen in dieser Prüfung beeinträchtigten.

Unterschrift: \_\_\_\_\_

## Hinweise

1. Bitte schreiben Sie Ihren Namen und Legi-Nummer sowie die Nummer ihres Computers (finden Sie auf dem Computer) auf diese Seite. Vergessen Sie nicht die Unterschrift am Ende der Prüfung.
2. Während der Programmierprüfung dürfen Sie nicht mehr an der schriftlichen Prüfung weiterarbeiten, auch wenn diese noch nicht eingezogen worden ist. **Dies gilt als Täuschungsversuch.**
3. Beachten Sie bitte während und *nach* der Prüfung unbedingt die Hygienevorschriften.
4. Die Prüfung hat 12 Seiten. Vergewissern Sie sich dass Ihr Exemplar vollständig ist. Die letzten zwei Seiten können Sie für Skizzen o.ä. benutzen, aber diese werden nicht für die Benotung hinzugezogen.
5. Die Programmierprüfung dauert 2 Stunden (120 Minuten). Falls Sie sich durch irgendjemanden oder irgendetwas gestört fühlen, oder technische Probleme an Ihrem Computer auftreten, so melden Sie dies sofort der Aufsicht. (Falls es unerwartete Fehlermeldungen gibt: Lassen Sie solche Fehlermeldungen oder PopUp Nachrichten auf dem Bildschirm und informieren Sie die Aufsicht. Nur so verhindern Sie, dass durch Systemfehler Ihre Programme verändert werden. ) Sollten Sie durch die Behandlung eines technischen Problems Zeit verlieren, so werden Sie die verlorene Zeit nachholen können.

6. Wir beantworten keine inhaltlichen Fragen während der Prüfung.
7. Lesen Sie die Aufgabenstellungen genau durch. Es ist wichtig, dass Ihre Antworten den Anforderungen der Aufgaben *genau* entsprechen. Wenn die Aufgabenstellung etwas nicht spezifiziert dann können Sie frei entscheiden (wir testen nur was wir spezifizieren). `Class.name()` heisst Methode `name()` in Klasse `Class`.
8. Benutzen Sie die Anzahl der Sterne in der Programmierprüfung als *Hinweis*, der ungefähr den Aufwand und die erreichbare Punktzahl der Aufgabe widerspiegelt. Je mehr Sterne, desto aufwändiger. Eine gut gelöste Aufgabe gibt mehr Punkte als zwei halb gelöste Aufgaben mit der selben Anzahl Sterne.
9. Für jede Aufgabe gibt es ein separates Java-Projekt in Ihrem Eclipse-Workspace.
10. Die Programmieraufgaben werden vorwiegend automatisch getestet und bewertet. Programme, welche nicht mindestens teilweise ein korrektes Resultat zurückgeben (oder gar nicht erst kompilieren), erhalten keine Punkte.
11. Stellen Sie regelmässig sicher, dass Ihre Dateien *im Workspace* gespeichert sind. Nur diese Dateien werden von einem Backup-Prozess während der Prüfung gespeichert. Was nicht gespeichert ist, kann nicht bewertet werden.
12. Sollten Sie eine Ihrer Lösungsdateien überschreiben, so kann die Aufsicht Ihnen helfen! Melden Sie sich sofort.
13. Ändern Sie unter keinen Umständen die Signaturen der im Aufgabentext erwähnten Methoden (Name, Typ und Reihenfolge der Parameter), ihren Rückgabetyt, Modifizierer wie `static`, `public` oder gegebenenfalls die Liste der geworfenen Exceptions. Das gleiche gilt für Konstruktoren und Attribute. Auch die Namen der erwähnten Klassen dürfen Sie nicht ändern und auch nicht Interfaces in Klassen umwandeln. Solche Änderungen können dazu führen, dass Sie keine Punkte für die Aufgabe erhalten. Wenn nicht anders vermerkt, dürfen Sie Methoden, Attribute, Interfaces oder Klassen zu den vorhandenen hinzufügen oder Klassen und Interfaces importieren. Die Verwendung von Java Reflection ist nicht erlaubt (und auch nicht von Vorteil).
14. Das Verwenden von `static`-Attributen ist grundsätzlich falsch. Rechnen Sie damit, dass wir das abgegebene Programm mehrfach ausführen (und ein Test selber aus mehreren Methodenaufrufen bestehen kann), ohne dass `static`-Attribute neu initialisiert werden. Lösungen, welche `static`-Attribute verwenden, und nur funktionieren, wenn ein Programm/eine Methode nur ein Mal ausgeführt wird, können potenziell 0 Punkte bekommen.
15. In jedem Projekt gibt es neben dem "src"-Ordner einen "test"-Ordner mit einigen JUnit-Tests. Wir empfehlen, diese mit ihren eigenen Tests zu erweitern. **Tests werden nicht bewertet.**
16. Falls gewisse Tests beim Ausführen scheinbar keine Resultate liefern, könnte es daran liegen, dass Ihre Lösung eine Endlosschleife enthält. Stoppen Sie in diesem Fall die Tests von Hand (siehe weitere Hinweise zu Eclipse weiter unten).
17. Als zusätzliche Sicherheitsmassnahme wird Ihr Bildschirm während der Prüfung aufgezeichnet.
18. Wenn Sie in der IDE Zeichen ersetzen statt einfügen, dann drücken Sie die Insert Taste (über der Delete Taste).
19. Auf einer Schweizer Tastatur schreiben Sie eckige und geschweifte Klammern durch `Alt + Ctrl` + die entsprechende Taste links neben der Enter Taste.
20. Wenn Sie zur Toilette müssen, melden Sie dies einer Aufsichtsperson durch Handzeichen. Es darf zur gleichen Zeit immer nur eine Person zur (Unisex)Toilette.
21. Wenn Sie früher abgeben wollen, sperren Sie bitte Ihren Computer und melden Sie sich bitte lautlos. Die Aufsicht wird Ihnen sagen, wann Sie Ihren Arbeitsplatz verlassen können. Vorzeitige Abgaben sind nur bis 20 Minuten vor Prüfungsende möglich.
22. Wenn die Aufsicht die Prüfung beendet, vergewissern Sie sich, dass alle Dateien gespeichert sind. Nach der Sperrung der Computer können Sie keine weiteren Änderungen mehr vornehmen. Befolgen Sie bitte die Anweisungen der Aufsicht (gestaffeltes Verlassen des Prüfungslokals).
23. Verlassen Sie bitte den Prüfungsraum *leise* nach der Prüfung. Es kann sein, dass andere Studierende noch weiterarbeiten da sie eine Zeitgutschrift bekommen haben. Auch diese Studierenden sollen in Ruhe arbeiten können. Bitte lassen Sie unbedingt die unterschriebene Aufgabenstellung auf Ihrem Tisch - wir sammeln diese später ein.

## Anmelden und Eclipse starten

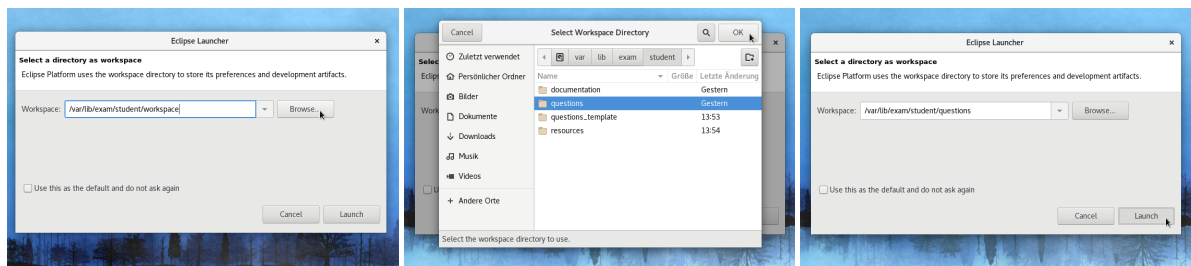
1. Sobald die Programmierprüfung startet, können Sie sich an Ihrem Computer anmelden. Geben Sie zuerst Ihren vollen Namen und im nächsten Schritt Ihren NETHZ-Namen und Ihre Legi-Nummer ein. (Sie brauchen *nicht* Ihr NETHZ-Passwort.) Sie werden auch in einem weiteren Fenster darauf hingewiesen, dass Ihr Computer aufgezeichnet wird, und dass Sie technische Probleme sofort melden müssen. Sobald Sie angemeldet sind, erscheint ein Browsertab mit allgemeinen Hinweisen zur Computer-Prüfung.
2. Starten Sie Eclipse, indem Sie oben links auf “Activities” (oder “Aktivitäten”) klicken und dann im Suchfeld “Eclipse” eingeben. Wählen Sie “Eclipse” (*nicht* “Eclipse C/C++”). Warten Sie, bis Eclipse gestartet ist. Dies kann einige Minuten in Anspruch nehmen.



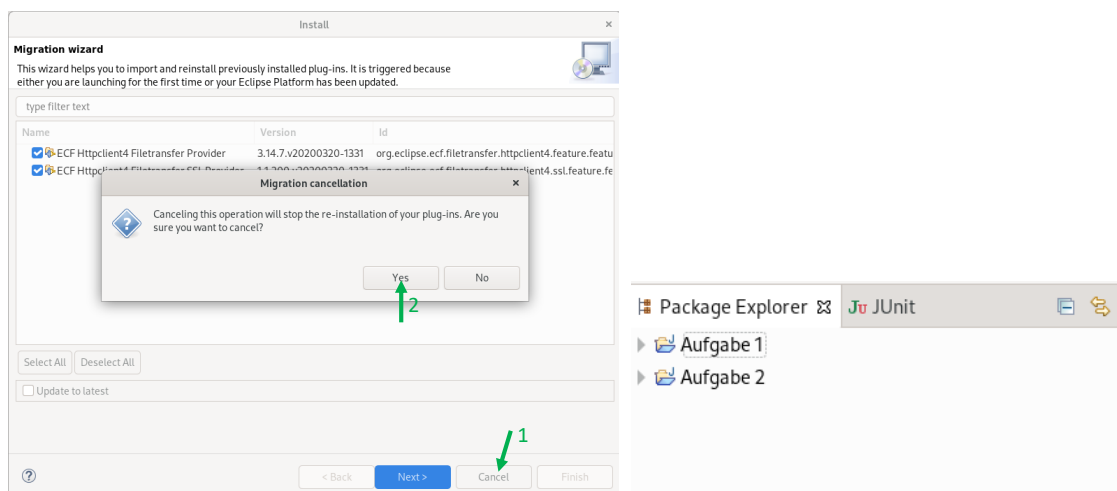
3. Wenn sich das Fenster “Eclipse Launcher” öffnet, stellen Sie sicher, dass der richtige Prüfungs-Workspace ausgewählt ist. Im Feld “Workspace” sollte folgender Pfad stehen, bevor Sie auf “Launch” klicken:

`/var/lib/exam/student/questions`

Falls dies nicht der Fall ist, klicken Sie auf “Browse...” und wählen Sie dann im Auswahldialog den “questions”-Ordner aus. Klicken Sie oben rechts auf “OK” und dann unten auf “Launch”.







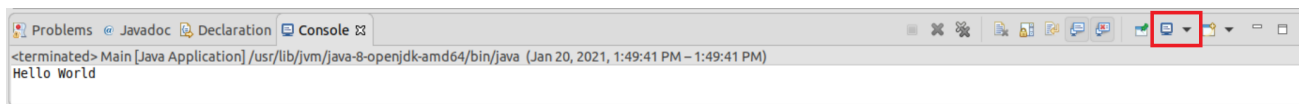
4. Nachdem der Workspace geöffnet wurde, erscheint ein Migration Wizard. Klicken Sie “Cancel” und bestätigen Sie dann mit “Yes”. Wenn Eclipse fertig gestartet ist, sehen Sie den Willkommens-Bildschirm. Klicken Sie wenn nötig oben rechts auf “Workbench”. Nun sollten Sie links die zwei Projekte “Aufgabe 1” und “Aufgabe 2” sehen. **Es kann einige Minuten dauern bis Eclipse alles geladen hat. Warten Sie bis die Ladenachricht “Initializing Java Tooling” (unten rechts in Eclipse) nicht mehr sichtbar ist.** Ausserdem können Sie allgemeine Hinweise zur Computer-Prüfung lesen, indem Sie oben links auf die “Activities” gehen und auf das Informationsicon klicken. Viel Spaß!




# Hinweise zu Eclipse






## Verhindern von Abstürzen

Bevor Sie ein Programm oder einen Test ausführen, achten Sie darauf, dass alle anderen Programme und Tests korrekt terminiert wurden. Wenn zu viele Programme gleichzeitig laufen, dann wird Ihr Computer langsamer, manchmal einfrieren, und im schlimmsten Fall abstürzen. Auch verhält sich dann manchmal Eclipse oder der Debugger unnatürlich. **Das geht von Ihrer Zeit ab.** Ein Problem ist, dass, auch wenn die aktive Konsole mit  terminiert wurde und somit ausgegraut ist () , es noch weitere Konsolen geben kann, welche immer noch laufen. Wenn das Icon  vorhanden und nicht ausgegraut ist () , dann gibt es mehr als eine Konsole, was auch heisst, dass mehrere Programme oder Tests noch nicht terminiert sein können:

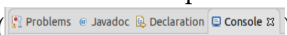



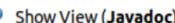


Durch einen Klick auf den Pfeil von , wird eine Liste aller vorhandenen Konsolen angezeigt:

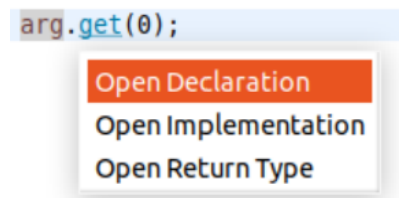


Durch klicken von  werden alle terminierten Konsolen geschlossen. Klicken Sie wiederholt  und  (oder ) bis  ausgegraut oder verschwunden ist, um alle Programme und Tests zu terminieren und alle Konsolen zu schliessen.




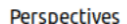


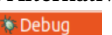
## Javadoc

Es gibt verschiedene Möglichkeiten die Java Dokumentation zu öffnen. Eine komfortable Option ist den Javadoc View zu verwenden. Dieser sollte in einer der Tabs bei der Konsole zu sehen sein (). Falls der Tab nicht vorhanden ist, oder falls Sie den Tab einfach nicht finden, dann können Sie durch das drücken von Alt + Shift + Q und dann J den Tab öffnen. Alternativ können Sie auch im Quick Access Fenster (ganz oben rechts ) "Javadoc" eingeben und dann   oder  drücken.

Wenn Sie den Javadoc View geöffnet haben, dann wird Ihnen die verfügbare Dokumentation von allem gezeigt das Sie anklicken. Zusätzlich, wenn Sie auf etwas zeigen, während Sie die Ctrl Taste gedrückt haben, dann können Sie sich die Declaration anzeigen lassen:



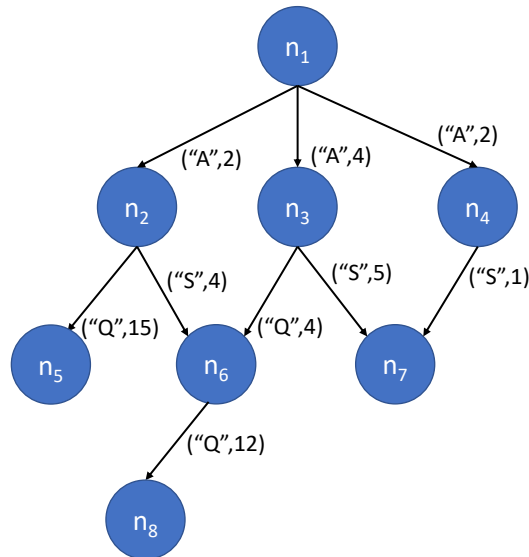
## Debugger

Ob Sie den Debugger verwenden ist Ihre Entscheidung. **Wir werden keine Fragen zum Debugger beantworten.** Die Aufgaben sind auch gut ohne Debugger lösbar. Um in den Debugging Modus zu wechseln, klicken Sie den Debugger Knopf  rechts neben dem Quick Access Fenster (ganz oben rechts ). Falls dieser Knopf nicht vorhanden ist, dann können Sie im Quick Access Fenster (ganz oben rechts ) "Debug" eingeben und dann   drücken. Alternativ können Sie auch den Knopf direkt neben dem Quick Access Fenster drücken  und dann "Debug" wählen .

[Diese Seite ist leer.]

## Aufgabe 1 (★★)

In dieser Aufgabe geht es um Berechnungen in gerichteten Graphen, wobei die Kanten eine Beschriftung (gegeben durch einen String) und ein Gewicht (gegeben durch eine ganze Zahl) haben. Das folgende Bild zeigt ein Beispiel eines solchen Graphen:



In diesem Beispiel hat die einzige Kante von  $n_2$  zu  $n_5$  die Beschriftung "Q" und das Gewicht 15.

Ein Pfad in einem Graphen entspricht einer Sequenz von Kanten  $(e_1, e_2, \dots, e_k)$ , wobei der Zielknoten von  $e_i$  gleich dem Ursprungsknoten von  $e_{i+1}$  entspricht (für  $i \in \{1, 2, \dots, k-1\}$ ). Die *Pfadbeschriftung* entspricht der Sequenz  $(b_1, b_2, \dots, b_k)$ , wobei  $b_i$  der Beschriftung von  $e_i$  entspricht. Das *Pfadgewicht* entspricht der Summe der Kantengewichte des Pfades. Im oberen Beispiel hat der einzige Pfad von  $n_1$  nach  $n_5$  die Pfadbeschriftung ("A", "Q") und das Pfadgewicht 17.

Die Klasse `Node` repräsentiert einen Knoten im Graphen und die Klasse `Edge` repräsentiert eine Kante im Graphen. Die Methode `Node.getNeighbours()` gibt die ausgehenden Kanten von einem Knoten als eine Liste von `Edge`-Objekten zurück. Die Methoden `Edge.getTarget()`, `Edge.getLabel()`, `Edge.getWeight()` geben jeweils den Zielknoten (als `Node`-Objekt), die Beschriftung (als `String`), und das Gewicht einer Kante (als `int`) zurück.

In den folgenden Aufgaben dürfen Sie annehmen, dass die Graphen keine Zyklen aufweisen.

- (a) (★) Implementieren Sie die Methode `Graph.removeEdges(Node origin, String label, int maxWeight)`. Die Methode sollte alle von `origin` erreichbaren Kanten  $e$  entfernen, wenn die Beschriftung von  $e$  gleich `label` ist und zusätzlich das Gewicht von  $e$  höchstens `maxWeight` ist. Der Rest des Graphen soll unverändert bleiben. Zum Beispiel muss `Graph.removeEdges( $n_1$ , "Q", 12)` im oberen Graphen nur die einzige Kante von  $n_3$  zu  $n_6$  und die einzige Kante von  $n_6$  zu  $n_8$  entfernen.
- (b) (★★) Implementieren Sie die Methode `Graph.findNodes(Node origin, List<String> path)`, welche ein `Node`-Objekt `origin` und eine Pfadbeschriftung `path` (wobei das  $i$ -te Element von `path` der Beschriftung der  $i$ -ten Kante im Pfad entspricht) als Input nimmt. Die Methode gibt eine Liste von Knoten (als `Node`-Objekte) zurück. Dabei muss die zurückgegebene Liste  $l$  alle Knoten enthalten, welche von `origin` durch einen Pfad mit Pfadbeschriftung `path` erreicht werden können (keine anderen Knoten sollten in  $l$  enthalten sein). Zusätzlich muss folgendes gelten für  $l$ :
- Jeder Knoten in  $l$  darf nur einmal in  $l$  vorkommen.
  - $l$  muss wie folgt sortiert sein. Seien  $v_1$  und  $v_2$  zwei Knoten in  $l$ . Sei  $d_i$  ( $i \in 1, 2$ ) das kleinste Pfadgewicht aller Pfade mit Pfadbeschriftung `path` von `origin` zu  $v_i$ . Dann muss  $v_1$  vor  $v_2$  in  $l$  vorkommen, wenn eine der folgenden zwei Fälle auftritt: (1)  $d_1 < d_2$ , oder (2)  $d_1 = d_2$  und  $v_1$  hat strikt weniger ausgehende Kanten als  $v_2$ .
  - Wenn gemäss der oberen Sortierbedingung  $v_1$  nicht vor  $v_2$  und  $v_2$  nicht vor  $v_1$  vorkommen muss, dann spielt die Reihenfolge in  $l$  zwischen  $v_1$  und  $v_2$  keine Rolle.

Zum Beispiel muss `Graph.findNodes( $n_1$ , ["A", "S"])` im oberen Graphen eine Liste mit genau zwei Elementen zurückgeben, welche  $n_7$  als erstes Element und  $n_6$  als zweites Element enthält.  $n_7$  muss vor  $n_6$  erscheinen, weil es einen Pfad von  $n_1$  nach  $n_7$  mit Pfadbeschriftung ("A", "S") und Pfadgewicht 3 gibt und das Pfadgewicht jedes Pfades mit Pfadbeschriftung ("A", "S") von  $n_1$  nach  $n_6$  strikt grösser als 3 ist.

Tests finden Sie in der Datei "GraphTest.java".

## Aufgabe 2 (★★★)

In dieser Aufgabe implementieren Sie ein Nachrichtensystem von einem Spital, in welchem Pager miteinander kommunizieren. Jeder Pager gehört zu einem einzigen Spital und kommuniziert nur mit Pagern des selben Spitals. Die Methode `Hospital.createPager(String role)` erstellt einen Pager. Das Argument `role` spezifiziert die Rolle des Pagers, welche das Verhalten vom Pager bestimmt. Konkrete Rollen werden in den Unteraufgaben eingeführt.

Das Interface `Pager` repräsentiert einen Pager und hat 3 Methoden:

- `Pager.register(String name)` registriert einen Pager auf einen Namen. `name` ist nie null.
- `Pager.inbox()` gibt eine neue Liste aller empfangenen Nachrichten des Pagers zurück (wir nennen das die Inbox). Eine Nachricht wird nie von der Inbox entfernt.
- Über `Pager.command(String destination, Message msg)` kann ein Pager einen Befehl entgegennehmen. Ein Befehl besteht aus einem *Zielnamen* `destination` (auf dem potenziell ein Pager registriert ist) und einer Nachricht `msg`. Konkrete Arten von Nachrichten werden in den Unteraufgaben eingeführt.

Das Entgegennehmen eines Befehls für einen Pager `p` besteht konzeptionell aus zwei Schritten: Der erste Schritt ist, dass ein Befehl *zugestellt* wird. Das passiert immer direkt wenn `p.command(destination, msg)` aufgerufen wird. Wir nennen `p` den *Sourcepager*. Der zweite Schritt ist, dass der Sourcepager `p` den Befehl (bestehend aus dem Zielnamen `destination` und der Nachricht `msg`) verarbeitet. Achten Sie beim Lesen der Unteraufgaben darauf, wann genau eine Nachricht verarbeitet wird und ob etwas beim Zustellen oder Verarbeiten passiert.

Ein Aufruf von `Pager.register(name)` wirft eine `IllegalArgumentException`, falls bereits ein anderer Pager auf den Namen `name` im Spital registriert ist. Die Methode `Pager.register` kann mehrmals aufgerufen werden, wobei sich dabei der registrierte Name des Pagers ändern kann. Daher ist es relevant, welche Pager auf welche Namen im aktuellen Zustand des Spitals registriert sind. Beachten Sie, dass nie zwei Pager auf den gleichen Namen registriert sind. Bei einem Aufruf von `p.register(name)` bleibt die Inbox vom Pager `p` unverändert.

In den folgenden Unteraufgaben implementieren Sie verschiedene Rollen und verschiedene Arten von Nachrichten. Dabei müssen Sie auch `Pager.createPager(String role)` implementieren.

- (a) (★) Implementieren Sie die Methoden `register`, `inbox`, und `command` für die Rollen “normal” und “slow”. Die Rollen “normal” und “slow” unterscheiden sich nur dahingehend, wann ein Befehl verarbeitet wird. Bei Pagern mit Rolle “normal” wird ein Befehl direkt verarbeitet, nachdem der Befehl zugestellt wird, das heisst nachdem `command` aufgerufen wird. Ein Pager mit Rolle “slow” hingegen zwischenspeichert die Befehle beim Zustellen und verarbeitet die zwischengespeicherten Befehle erst nach jedem dritten Aufruf von `command`. Die Befehle werden dann in der selben Reihenfolge verarbeitet, in der die entsprechenden Aufrufe von `command` erfolgten. Das heisst, wenn es auf einem Pager `p` mit Rolle “slow” die Aufrufe `p.command(d1, m1)`, `p.command(d2, m2)`, und dann `p.command(d3, m3)` gibt, dann wird, sobald `p.command(d3, m3)` aufgerufen wurde, zuerst der erste Befehl (mit Zielname `d1` und Nachricht `m1`), dann der zweite Befehl (mit `d2` und `m2`), und dann der dritte Befehl (mit `d3` und `m3`) verarbeitet.



Für Unteraufgabe (a) müssen nur Befehle, welche Nachrichten vom Typ `TextMessage` beinhalten, verarbeitet werden. Wenn ein Befehl mit einer `TextMessage` für Rolle “normal” oder “slow” verarbeitet wird, dann gilt: Falls ein Pager  $q$  mit dem Zielnamen des Befehls im aktuellen Zustand vom Spital registriert ist, dann wird die Nachricht der Inbox von  $q$  hinzugefügt. Falls kein Pager auf den Zielnamen registriert ist, dann wird eine `PagerNotRegisteredMessage` der Inbox des `Sourcepagers` hinzugefügt. Das Attribut `destination` von `PagerNotRegisteredMessage` enthält den Zielnamen der fehlgeschlagenen Nachricht. Neue Nachrichten werden immer am Ende der Liste der Nachrichten hinzugefügt, welche beim Aufruf von `Pager.inbox()` zurückgegeben werden.

- (b) (★) Implementieren Sie die Methoden `register`, `inbox`, und `command` für die Rolle “admin”. Für die Rolle “admin” wird beim Aufruf von `command` der Befehl sofort verarbeitet (wie bei “normal”). Wenn ein Pager mit Rolle “admin” einen Befehl mit einer Nachricht vom Typ `TextMessage` verarbeitet, dann wird die Nachricht den Inboxes von allen registrierten Pagern des Spitals am Ende der Inbox hinzugefügt (auch der eigenen). Der Zielname des Befehls ist dabei irrelevant.

Zusätzlich kann ein Pager mit Rolle `admin` auch Befehle verarbeiten, welche Nachrichten vom Typ `PauseMessage` beinhalten. Beim Verarbeiten von einem Exemplar von `PauseMessage` wird der Zielname *pausiert* bzw. die Pausierung wird aufgehoben, wenn der Zielname schon pausiert ist. Wenn ein Befehl von einem Pager verarbeitet wird, welcher im aktuellen Zustand des Spitals auf einen pausierten Namen registriert ist, dann passiert nichts. Zum Beispiel, wenn eine Nachricht vom Typ `TextMessage` verarbeitet wird, dann wird die Nachricht keiner Inbox hinzugefügt und geht unwiderruflich verloren. Das Zustellen eines Befehls wird nicht beeinflusst. Ein Zielname ist nicht mehr pausiert, sobald erneut ein Befehl mit einer `PauseMessage` und dem gleichen Zielnamen verarbeitet wird. Für die Rollen “normal” und “slow” passiert nichts wenn eine `PauseMessage` Nachricht verarbeitet wird.

Beachten Sie, dass ein Name, und kein Pager, pausiert wird. Daraus folgt: Ein Name wird auch pausiert, falls darauf aktuell kein Pager registriert ist. Ein Pager, welcher auf einen pausierten Namen registriert ist, kann sich vom Einfluss des pausierten Namen befreien, indem er auf einen nicht pausierten Namen registriert wird. Für einen Pager, welcher auf einen aktuell pausierten Namen registriert wird, wird die Ausführung von `Pager.command()` beeinflusst (eine `TextMessage` wird nicht zur Inbox hinzugefügt). Beachten Sie, dass bei der Rolle “slow” *alle* Befehle beim Zustellen immer zwischengespeichert werden bevor sie verarbeitet werden (wie in (a) beschrieben). Pager mit der Rolle “admin” können auch pausiert werden.

- (c) (★) Fortsetzung nächste Seite.

- (c) (★) Implementieren Sie für die Rollen “normal”, “slow”, und “admin” das Entgegennehmen von Befehlen, welche Nachrichten vom Typ `QueryMessage` beinhalten. Alle Rollen machen das gleiche, wenn solche Befehle verarbeitet werden (die Rolle “slow” zwischenspeichert die Befehle wie in (a) beschrieben).

Wenn ein Befehl mit `QueryMessage` verarbeitet wird, dann wird der `Inbox` vom `Sourcepager` ein Exemplar von `QueryAnswerMessage` am Ende hinzugefügt. Das Attribut `numberOfMessages` von `QueryAnswerMessage` enthält wie viele Nachrichten vom `Pager q`, welcher auf den Zielnamen des Befehls registriert ist, an den `Sourcepager p` geschickt wurden **seit dem** `Pager p` **und** `Pager q` auf ihren aktuellen Namen registriert worden sind. Falls der letzte Aufruf von `register` auf `p` nach dem letzten Aufruf von `register` auf `q` passiert ist, dann zählen exakt die Exemplare von `TextMessage`, welche von `q` verarbeitet wurden und der `Inbox` von `p` hinzugefügt wurden seit dem letzten Aufruf von `register` auf `p`. Vice versa, falls der letzte Aufruf von `register` auf `q` nach dem letzten Aufruf von `register` auf `p` passiert ist, dann ist der Zeitraum relevant seit dem letzten Aufruf von `register` auf `q`. Das Attribut `destination` von `QueryAnswerMessage` enthält den Namen auf den `q` registriert ist. Falls kein `Pager` auf den Zielnamen registriert ist, dann wird eine `PagerNotRegisteredMessage` der `Inbox` des `Sourcepagers` am Ende hinzugefügt. Das Attribut `destination` von `PagerNotRegisteredMessage` enthält den Zielnamen.

Tests finden Sie in der Datei “`HospitalTest.java`”.

## Notizen

## Notizen