

Frühling 2022
252-0027 – Einführung in die Programmierung

Departement Informatik
ETH Zürich

30. August 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: _____

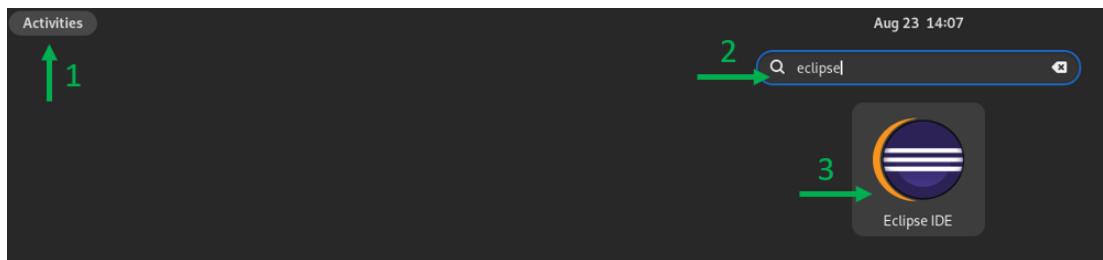
Hinweise

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. 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.
4. 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.
5. Wir beantworten keine inhaltlichen Fragen während der Prüfung.

6. 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`.
7. 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.
8. Für jede Aufgabe gibt es ein separates Java-Projekt in Ihrem Eclipse-Workspace.
9. 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.
10. 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.
11. Sollten Sie eine Ihrer Lösungsdateien überschreiben, so kann die Aufsicht Ihnen helfen! Melden Sie sich sofort.
12. 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).
13. 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.
14. 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.**
15. 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).
16. <https://exam-translate.ethz.ch/> ist ein Übersetzungsservice, den wir ohne Gewähr versuchsweise zur Verfügung stellen.
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 über oder 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 mit der Nummer des Computers auf Ihrem Tisch - wir sammeln diese später ein.

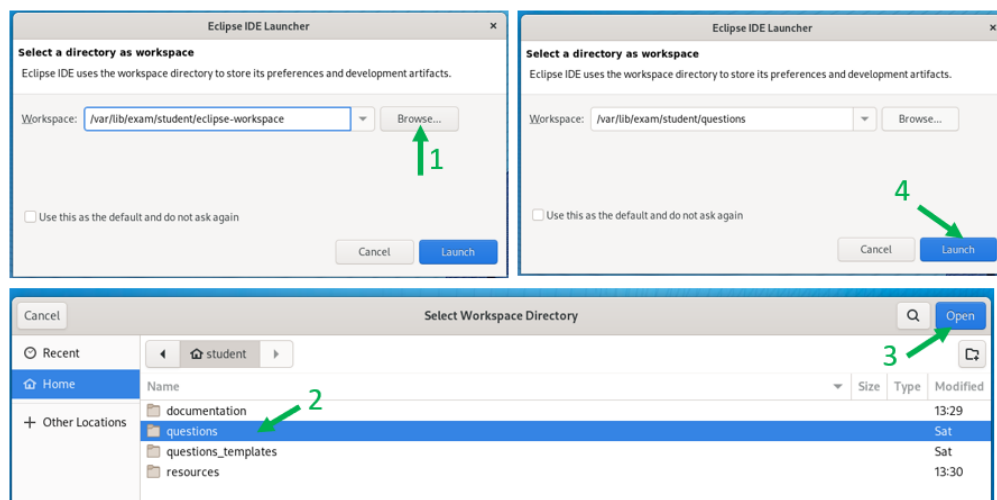
Anmelden und Eclipse starten

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 IDE” (*nicht* “Eclipse C/C++”). Warten Sie, bis Eclipse gestartet ist. Dies kann einige Minuten in Anspruch nehmen.

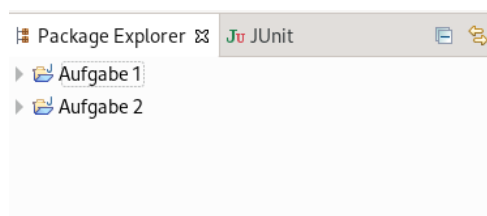


3. Wenn sich das Fenster “Eclipse IDE Launcher” öffnet, müssen Sie zuerst den Eclipse-Workspace Ordner im Feld “Workspace” anpassen (**der Standardpfad, welcher im Fenster automatisch angegeben wird, ist inkorrekt**). Klicken Sie auf “Browse...” und wählen Sie dann im Auswahldialog den “questions”-Ordner aus. Klicken Sie oben rechts auf “Open” und dann unten rechts auf “Launch”. Im Feld “Workspace” sollte folgender Pfad stehen, bevor Sie auf “Launch” klicken:

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







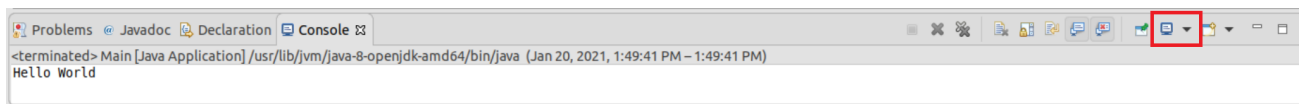
4. Wenn Eclipse fertig gestartet ist, sehen Sie den Willkommens-Bildschirm. Klicken Sie wenn nötig oben rechts auf “Workbench”. Nun sollten Sie links die zwei Projekte “Aufgabe 1” und “Aufgabe 2” sehen. **Es kann einige Minuten dauern bis Eclipse alles geladen hat. Warten Sie bis die Ladenachricht “Initializing Java Tooling” (unten rechts in Eclipse) nicht mehr sichtbar ist.** Ausserdem können Sie über die Webseite <https://exam-translate.ethz.ch/> auf ein Übersetzungstool von Deutsch auf Englisch zugreifen. Zusätzlich 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:

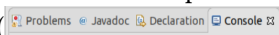






Durch einen Klick auf den Pfeil von , wird eine Liste aller vorhandenen Konsolen angezeigt:

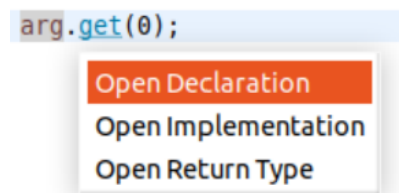


Durch klicken von  werden alle terminierten Konsolen geschlossen. Klicken Sie wiederholt  und  (oder ) bis  ausgegraut oder verschwunden ist, um alle Programme und Tests zu terminieren und alle Konsolen zu schliessen.









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 (★★★)

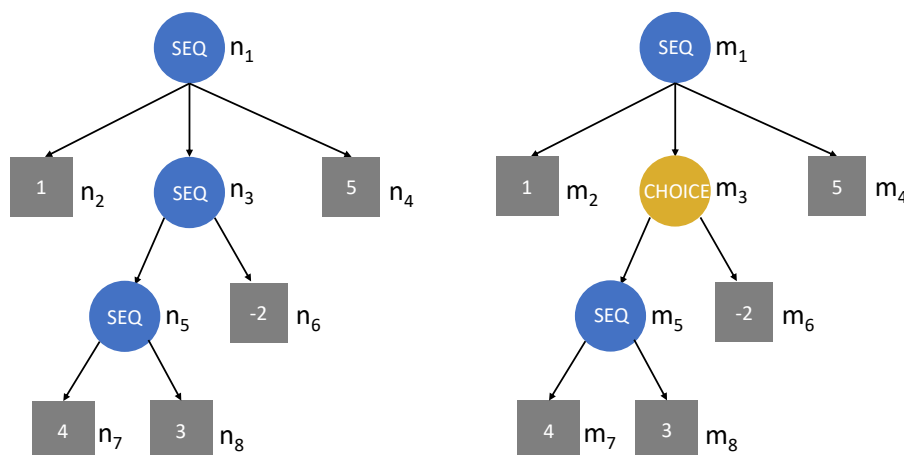
In dieser Aufgabe verwenden wir gerichtete azyklische Graphen, um Programme zu repräsentieren. Der Programmzustand ist dabei immer durch ein Tupel $(sum, counter)$ gegeben, wobei sum und $counter$ ganze Zahlen sind. Programmzustände werden durch `ProgramState`-Objekte modelliert, wobei `ProgramState.getSum()` (bzw. `ProgramState.getCounter()`) dem ersten Element (bzw. dem zweiten Element) des Tupels entspricht.

Eine Ausführung des Programms manipuliert den Programmzustand und das Resultat eines Programms ist gegeben durch den erreichten Programmzustand, nachdem alle Operationen im Programm ausgeführt wurden. Programme können nichtdeterministisch sein: Das heisst, für ein einzelnes Programm kann es fuer den gleichen Startzustand mehrere Programmausführungen geben, welche zu unterschiedlichen Resultaten führen.

Knoten in Graphen werden durch `Node`-Objekte modelliert. `Node.getSubnodes()` gibt die Nachbarknoten als eine Liste zurück. Wir unterscheiden drei Arten von Knoten, wobei `Node.getType()` die Knotenart als String repräsentiert. Um ein Programm, welches durch den Knoten n repräsentiert wird, auszuführen, muss man den "Knoten n ausführen". Wir beschreiben nun die drei Knotenarten und jeweils die Ausführung der Knoten:

1. **Additionsknoten** (`Node.getType()` ist "ADD"): Solche Knoten besitzen einen Additionswert a gegeben durch `Node.getValue()` (eine ganze Zahl) und bei der Ausführung dieses Knotens wird der Programmzustand von $(sum, counter)$ zu $(sum + a, counter + 1)$ aktualisiert. Die Nachbarknoten von solchen Knoten werden bei der Ausführung ignoriert.
2. **Sequenzknoten** (`Node.getType()` ist "SEQ"): Bei der Ausführung eines Sequenzknoten n werden die Nachbarknoten von n nacheinander ausgeführt. Die Reihenfolge in welcher die Nachbarknoten ausgeführt werden spielt keine Rolle, da der erreichte Programmzustand für jede Reihenfolge gleich ist. `Node.getValue()` ist irrelevant.
3. **Auswahlknoten** (`Node.getType()` ist "CHOICE"): Bei der Ausführung eines Auswahlknoten n wird ein beliebiger Nachbarknoten von n ausgewählt und ausgeführt. `Node.getValue()` ist irrelevant. Diese Knoten führen zu Nichtdeterminismus.

Sie dürfen davon ausgehen, dass Sequenz- und Auswahlknoten immer mindestens einen Nachbarknoten haben, und dass es zwischen zwei Knoten immer höchstens einen Pfad gibt. Die folgende Abbildung zeigt zwei Beispielgraphen, wobei Knoten mit der Beschriftung "SEQ" (bzw. "CHOICE") Sequenzknoten (bzw. Auswahlknoten) entsprechen und die Zahlen in Additionsknoten den Additionswerten entsprechen.



Beim linken Graphen in der Abbildung gibt es immer nur eine mögliche Ausführung pro Startzustand. Für den Startzustand $(1, 2)$ ist das Resultat gegeben durch $(1 + 1 + 4 + 3 - 2 + 5, 2 + 5) = (12, 7)$. Beim rechten Graphen gibt es zwei mögliche Ausführungen. Beim Auswahlknoten m_3 wird entweder m_5 oder m_6 ausgeführt (da m_5 und m_6 die Nachbarknoten von m_3 sind). Die beiden möglichen Resultate für den Startzustand $(0, 0)$ sind $(0 + 1 + 4 + 3 + 5, 0 + 4) = (13, 4)$ (wenn m_5 gewählt wird) und $(0 + 1 - 2 + 5, 0 + 3) = (4, 3)$ (wenn m_6 gewählt wird).

- (a) (★) Implementieren Sie die Methode `GraphExecution.allResults(Node n, ProgramState initState)`, welche für den Startzustand `initState` alle möglichen Resultate für das Programm repräsentiert durch `n` zurückgibt. Die Resultate sollten als eine Liste von `ProgramState`-Objekten zurückgegeben werden. Die Reihenfolge der zurückgegebenen Liste spielt keine Rolle. Wenn das gleiche Resultat durch genau k verschiedene Ausführungen generiert werden kann, dann muss das Resultat k Mal in der zurückgegebenen Liste vorkommen. Zwei Ausführungen sind unterschiedlich, wenn es mindestens einen Knoten gibt, der in einer aber nicht in der anderen Ausführung ausgeführt wird.
- (b) (★) Implementieren Sie die Methode `GraphExecution.sort(List< List<ProgramState> > input)`, welche `input` wie unten beschrieben sortiert. Sie dürfen annehmen, dass jede enthaltene Liste in `input` mindestens ein `ProgramState`-Objekt enthält.

Sei für eine Zustandsliste l (d.h. l ist ein Objekt vom Typ `List<ProgramState>`) $\text{maxState}(l)$ ein Zustand in l für welchen gilt: Für alle Zustände s in l gilt, dass (1) $\text{maxState}(l).\text{getSum}() \geq s.\text{getSum}()$ und (2) wenn $\text{maxState}(l).\text{getSum}() = s.\text{getSum}()$, dann gilt $\text{maxState}(l).\text{getCounter}() \geq s.\text{getCounter}()$. Seien l_1 und l_2 zwei Zustandslisten in `input`. Nach dem Aufruf von `GraphExecution.sort(input)` muss l_1 vor l_2 in `input` auftreten, wenn einer der folgenden zwei Fälle auftritt:

- Fall 1: $\text{maxState}(l_1).\text{getSum}() > \text{maxState}(l_2).\text{getSum}()$
- Fall 2: $\text{maxState}(l_1).\text{getSum}() = \text{maxState}(l_2).\text{getSum}()$ und $\text{maxState}(l_1).\text{getCounter}() > \text{maxState}(l_2).\text{getCounter}()$

Wenn gemäss der oberen Regel l_1 nicht vor l_2 und l_2 nicht vor l_1 auftreten muss, dann spielt die Reihenfolge zwischen l_1 und l_2 keine Rolle.

- (c) (★) In dieser Aufgabe können Sie annehmen, dass keine Auswahlknoten vorkommen. Implementieren Sie die Methode `GraphExecution.isSubProgram(Node n1, Node n2)`, welche entscheidet, ob n_2 ein *Unterprogramm* von n_1 ist. Der Knoten n_2 ist genau dann ein Unterprogramm von n_1 , wenn es einen Knoten n_1' gibt, welcher von n_1 erreichbar ist (das heisst, n_1' ist gleich n_1 oder kann durch die gerichteten Kanten im Graphen von n_1 aus erreicht werden), so dass n_1' und n_2 äquivalent sind.

Wir beschreiben nun, wann zwei Knoten äquivalent sind. Zwei Additionsknoten sind genau dann äquivalent, wenn ihre Additionswerte gleich sind. Zwei Sequenzknoten n und n' sind genau dann äquivalent, wenn sie die gleiche Anzahl Nachbarknoten haben und wenn es ein Objekt ns vom Typ `List<Node>` gibt, welches eine Permutation von $n.\text{getSubnodes}()$ ist (das heisst, $ns.\text{size}() == n.\text{getSubnodes}().\text{size}()$ und ns enthält genau die gleichen Objekte wie $n.\text{getSubnodes}()$, aber potenziell in einer anderen Reihenfolge), so dass $ns.\text{get}(i)$ und $n'.\text{getSubnodes}().\text{get}(i)$ äquivalent sind für $0 \leq i < ns.\text{size}()$.

Aufgabe 2 (★★☆)

In dieser Aufgabe implementieren Sie ein Verteilungssystem für Ressourcen eines Bauunternehmens, mit welchem Ressourcen (z.B. Sandsäcke, Kabelrollen, Ziegelpaletten) auf Baustellen verteilt werden. Das Interface `Resource` repräsentiert Ressourcen. Ressourcen haben einen Typ, welchen wir als Integer repräsentieren. Die Methode `Resource.type()` gibt den Typen einer Ressource zurück. Bauunternehmen und Baustellen gehören Ressourcen. Dabei gehört jede einzelne Ressource immer nur maximal entweder einem einzigen Bauunternehmen oder einer einzigen Baustelle. Der Aufgabentext beschreibt wann und wie sich der Besitz von Ressourcen ändert. Gleichermassen gehört jede Baustelle zu einem einzigen Bauunternehmen. Beim Erstellen der Baustelle wird angegeben, welche Ressourcetypen auf der Baustelle verwendet werden und wie viele Ressourcen des gleichen Typs maximal der Baustelle gehören dürfen.

Die Klasse `CCompany` repräsentiert ein Bauunternehmen (construction company) und hat folgende Methoden:

- `CCompany.resources()` gibt ein Set aller Ressourcen zurück, welche dem Bauunternehmen gehören. Zum Zeitpunkt, wenn ein Bauunternehmen erzeugt wird, gehören dem Bauunternehmen noch keine Ressourcen.
- `CCompany.add(Resource resource)` übergibt dem Bauunternehmen die Ressource `resource`, worauf dem Bauunternehmen die Ressource gehört. Sie dürfen annehmen, dass `resource` vor dem Aufruf keinem anderen Bauunternehmen und keiner Baustelle gehört.
- Die drei Methoden `CCompany.createCSite(Set<Integer> types, int limit)`, `CCompany.createCSite(int type)`, und `CCompany.createCSite(Set<Integer> types, int limit, int flowLimit)` erzeugen eine neue Baustelle, welche dem Bauunternehmen gehört, bis die Baustelle geschlossen wird.
- `CCompany.nextDay()` löst das Übergeben der Ressourcen, welche dem Bauunternehmen gehören, an die Baustellen, welche dem Bauunternehmen gehören, aus. Die Unteraufgaben beschreiben, wie Ressourcen an die Baustellen übergeben werden. Eine übergebene Ressource gehört nicht mehr dem Bauunternehmen (sondern der Baustelle).

Das Interface `CSite` repräsentiert eine Baustelle (construction site) und hat 5 Methoden:

- `CSite.resources()` gibt ein Set aller Ressourcen zurück, welche der Baustelle gehören. Zum Zeitpunkt, wenn eine Baustelle erzeugt wird, gehören der Baustelle noch keine Ressourcen.
- `CSite.canAdd(Resource resource)` gibt zurück, ob der Baustelle die Ressource `resource` übergeben werden darf. Die Unteraufgaben beschreiben, wann das der Fall ist.
- `CSite.add(Resource resource)` übergibt der Baustelle die Ressource `resource`, worauf der Baustelle die Ressource gehört. Die Methode wirft eine `IllegalArgumentException`, falls ein Aufruf von `CSite.canAdd(resource)` `false` zurückgibt. Für die Tests dürfen Sie annehmen, dass `resource` vor dem Aufruf keinem anderen Bauunternehmen und keiner Baustelle gehört und dass die Methode nicht mehr aufgerufen wird, nachdem durch `CCompany.nextDay()` für die Baustelle die Übergabe von Ressourcen ausgelöst wurde.

- `CSite.use(Resource resource)` verbraucht die Ressource `resource`, welche der Baustelle gehört. Die Methode wirft eine `IllegalArgumentException`, falls das Argument nicht der Baustelle gehört. Nachdem eine Ressource verbraucht wurde, gehört die Ressource nicht mehr der Baustelle (und die Ressource gehört auch nichts anderem mehr).
- `CSite.close()` schliesst eine Baustelle, das heisst, die Baustelle wird vom Bauunternehmen entfernt. Sie dürfen annehmen, dass keine weiteren Methoden auf einer geschlossenen Baustelle aufgerufen werden. Die Ressourcen, welche der Baustelle gehören, ändern sich nicht.

Sie dürfen annehmen, dass, solange die Aufgabenstellung nichts anderes vorgibt, alle Argumente nie null sind.

Figure 1a zeigt ein Bauunternehmen Q mit 2 Baustellen X und Y, wobei X vor Y erstellt wurde; das Lager vom Bauunternehmen Q enthält 4 Typen von Ressourcen (T_a , T_b , T_c , T_d). Jedes graue Quadrat stellt eine Ressource dar, wobei die Zahlen angeben in welcher Reihenfolge die Ressourcen an das Bauunternehmen Q übergeben wurden. Die grauen Quadrate bei der Baustelle repräsentieren Ressourcen, welche schon der Baustelle gehören. Auf der Baustelle X werden nur Ressourcen vom Typ T_a , T_c und T_d verwendet (bei Baustelle Y sind es T_a , T_b und T_c), wobei pro Typ maximal zwei Ressourcen der Baustelle X gehören dürfen (bei Baustelle Y sind es pro Typ maximal drei).

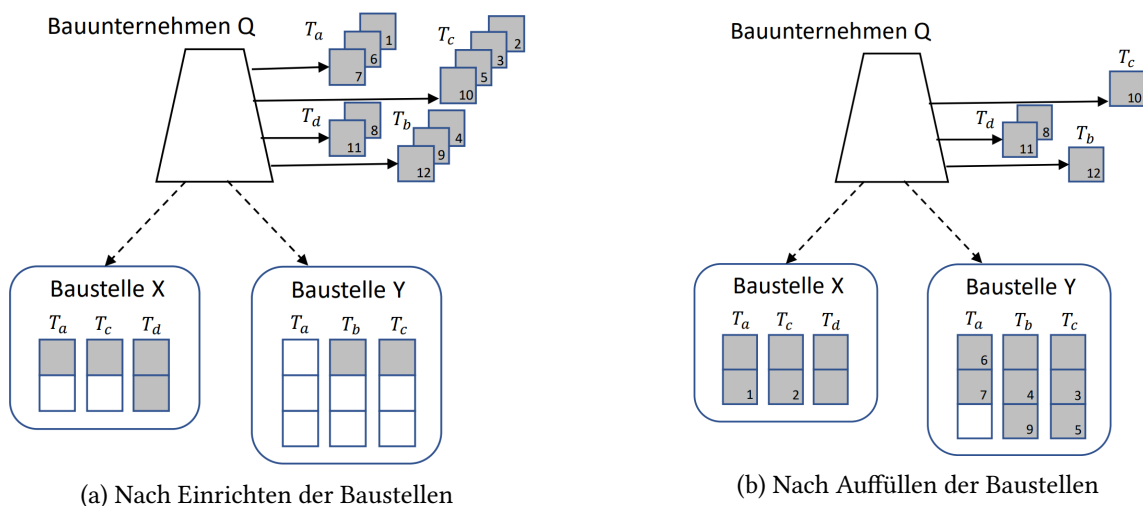


Abbildung 1: Bauunternehmen (Q) mit 2 Baustellen (X und Y) und den 4 Typen (T_a , T_b , T_c , T_d).

(a) (★) Implementieren Sie die Methoden

- `CCompany.createCSite(Set<Integer> types, int limit)` (Sie dürfen annehmen $\text{limit} \geq 0$). Diese Methode gibt eine Baustelle `site` mit Platz für `limit` Ressourcen jedes Typs zurück. Eine Ressource `resource` darf der Baustelle `site` übergeben werden (das heisst `site.canAdd(resource)` gibt `true` zurück) genau dann, wenn:
 - die Ressource `resource` einen Typ hat, welcher im Set `types` enthalten ist, *und*
 - falls die Anzahl der Ressourcen, welche der Baustelle gehören und den gleichen Typ wie die Ressource `resource` haben, strikt kleiner als `limit` ist.
- `CSite.canAdd(Resource resource)`,
- `CSite.add(Resource resource)`,
- `CSite.use(Resource resource)`, und
- `CSite.resources()`.

- (b) (★) Implementieren Sie die Methoden `CSite.close()`, `CCompany.resources()`, `CCompany.add(Resource resource)`, und `CCompany.nextDay()`. Ein Aufruf von `CCompany.nextDay()` sorgt dafür, dass die Ressourcen, welche dem Bauunternehmen gehören, an die Baustellen, welche dem Bauunternehmen gehören, verteilt werden. Ressourcen werden dabei wie folgt an die Baustellen verteilt: Die Ressourcen werden in der Reihenfolge an die Baustellen übergeben, in welcher die Ressourcen an das Bauunternehmen übergeben wurden. In dieser Reihenfolge wird eine Ressource `resource` an die Baustelle `site` übergeben, falls:
- (i) die Ressource `resource` der Baustelle `site` übergeben werden darf, das heisst `site.canAdd(resource)` gibt `true` zurück, *und*
 - (ii) falls die Ressource `resource` an keine andere Baustelle `otherSite`, welche dem Bauunternehmen gehört und **vor** der Baustelle `site` erzeugt wurde, übergeben werden darf, das heisst der Aufruf `otherSite.canAdd(resource)` gibt `false` zurück.

Beachten Sie, dass es sein kann, dass eine Ressource an keine Baustelle übergeben wird (diese Ressource verbleibt im Lager und kann vielleicht später übergeben werden).

Figure 1b zeigt den Zustand nach Ausführen der Methode `Q.nextDay()`, wobei nach Einrichten der Baustellen (Figure 1a) und dem Aufruf von `nextDay()` keine Ressourcen hinzugefügt oder verbraucht wurden.

Falls Sie Unteraufgabe (a) nicht gelöst haben, so können Sie Teilpunkte bekommen. Eine Teilmenge der Tests wird nur die Methode `CCompany.createCSite(int type)` zum Erzeugen von Baustellen verwenden. Diese Methode gibt eine Baustelle zurück, welche eine Ressource nur annehmen darf, falls der Typ der Ressource gleich `type` ist und der Baustelle nicht schon 2 Ressourcen gehören. Um Teilpunkte zu bekommen, können Sie die Methode `CCompany.createCSite(int type)` mit Hilfe der Klasse `TaskBCSite` implementieren, welche bereits alle Methoden der Unteraufgabe (a) implementiert. Beachten Sie, dass Sie die Klasse `TaskBCSite` verändern dürfen.

- (c) (★) Implementieren Sie die Methode `CCompany.createCSite(Set<Integer> types, int limit, int flowLimit)` (Sie dürfen annehmen $\text{limit} \geq 0$ und $\text{flowLimit} \geq 0$). Die Methode gibt, wie bei Unteraufgabe (a), eine Baustelle `site` zurück, welcher eine Ressource `resource` nur übergeben werden darf (das heisst `site.canAdd(resource)` gibt `true` zurück) genau dann, wenn:
- (i) die Ressource `resource` einen Typ hat, welcher im `Set types` enthalten ist,
 - (ii) und falls die Anzahl der Ressourcen, welche der Baustelle gehören und den gleichen Typ wie die Ressource `resource` haben, strikt kleiner als `limit` ist.

Wir nennen die Baustellen, welche von dieser Methode zurückgegeben werden, “dynamische” Baustellen. Eine dynamische Baustelle `site` hat einen “Überfluss” von einem Typ `type`, falls die Anzahl der Ressourcen, welche der Baustelle `site` gehört und Typ `type` hat, strikt grösser als `flowLimit` ist. Dynamische Baustellen erweitern was bei einem Aufruf von `CCompany.nextDay()` passiert: **Bevor die Ressourcen verteilt werden**, übergibt jede dynamische Baustelle `site`, welche dem Bauunternehmen gehört, dem Bauunternehmen die kleinstmögliche Menge an Ressourcen, sodass die Baustelle `site` keinen Überfluss hat. Im Allgemeinen kann es mehrere Mengen geben, welche dieses Kriterium erfüllen und daher gültige Lösungen sind. Eine Ressource, welche dem Bauunternehmen von einer Baustelle übergeben wird, gehört danach dem Bauunternehmen und nicht mehr der Baustelle. Danach werden die Ressourcen verteilt, wie in Unteraufgabe (b) beschrieben.

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

Notizen

Notizen