

Frühjahr 2023
252-0027 – Einführung in die Programmierung

Departement Informatik
ETH Zürich

25. August 2023 – 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: _____

Bitte lassen Sie unbedingt die unterschriebene Aufgabenstellung mit der Nummer des Computers auf Ihrem Tisch - wir sammeln diese später ein.

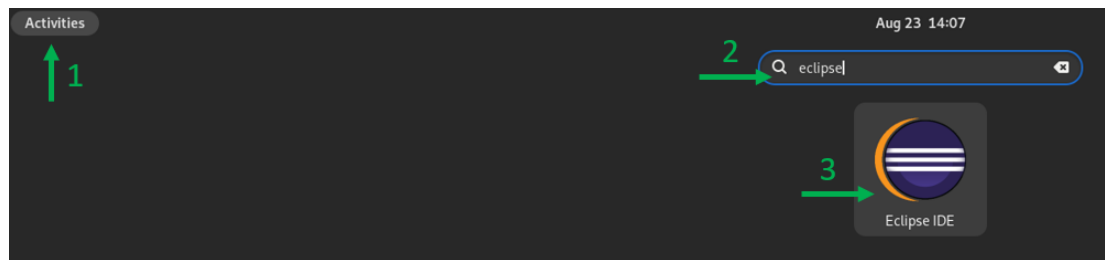
Hinweise

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 14 Seiten. Vergewissern Sie sich dass Ihr Exemplar vollständig ist. Die letzten fünf 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.
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 Rückgabetypp, 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). Java Assertions sind per default nicht aktiviert (auch nicht während der Bewertung).
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 die "Alt Gr" Taste + 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 Toilette.
21. Wenn Sie früher abgeben wollen, sperren Sie bitte Ihren Computer (Windows+L) und melden Sie sich bitte lautlos bei der Aufsicht. 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; auch vor dem Prüfungsraum bitten wir um Ruhe.. 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

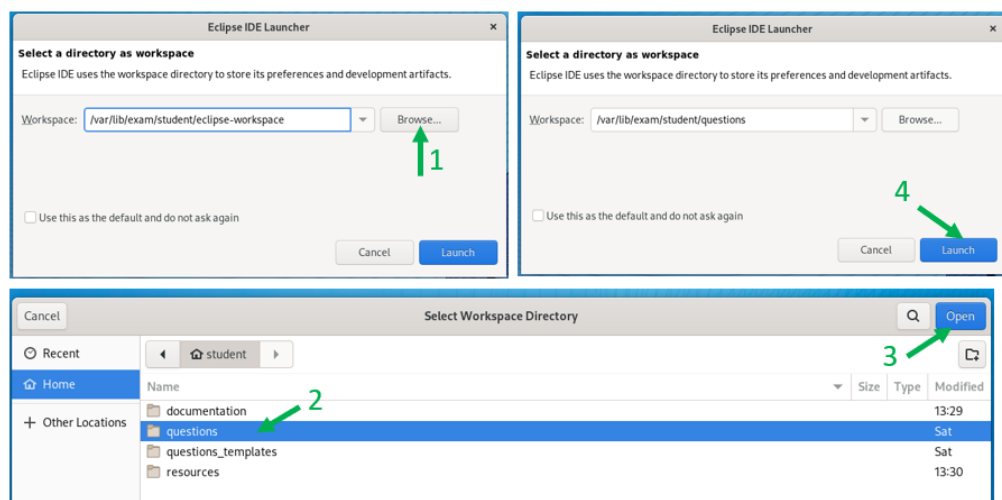
1. Sobald die Programmierprüfung startet, können Sie sich an Ihrem Computer anmelden. Geben Sie zuerst Ihren vollen Namen und im nächsten Schritt Ihren NETHZ-Namen und Ihre Legi-Nummer ein. (Sie brauchen *nicht* Ihr NETHZ-Passwort.) Sie werden auch in einem weiteren Fenster darauf hingewiesen, dass Ihr 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 (1) oben links auf “Activities” (oder “Aktivitäten”) klicken und dann (2) im Suchfeld “Eclipse” eingeben. Wählen Sie (3) “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, 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

Meistens ist dieser Pfad bereits ausgewählt. **Wenn dies nicht der Fall ist**, dann (1) klicken Sie auf “Browse...”, (2) wählen Sie dann im Auswahldialog den “questions”-Ordner aus, (3) klicken Sie oben rechts auf “Open” und (4) klicken Sie unten rechts auf “Launch”.







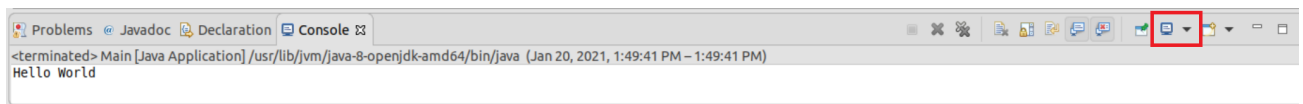
4. Wenn Eclipse fertig gestartet ist, sehen Sie den Willkommens-Bildschirm. Klicken Sie wenn nötig oben rechts auf “Workbench”. Nun sollten Sie links die drei Projekte “Aufgabe 1”, “Aufgabe 2” und “Aufgabe 3” sehen. **Es kann einige Minuten dauern bis Eclipse alles geladen hat. Warten Sie bis die Ladenachricht “Initializing Java Tooling” (unten rechts in Eclipse) nicht mehr sichtbar ist.** 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:

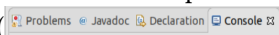





Durch einen Klick auf den Pfeil von , wird eine Liste aller vorhandenen Konsolen angezeigt:

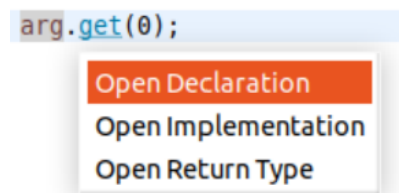


Durch klicken von  werden alle terminierten Konsolen geschlossen. Klicken Sie wiederholt  und  (oder ) bis  ausgegraut oder verschwunden ist, um alle Programme und Tests zu terminieren und alle Konsolen zu schliessen.







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  **Javadoc (Java)** oder  **Show View (Javadoc)** 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  **Debug** drücken. Alternativ können Sie auch den Knopf direkt neben dem Quick Access Fenster drücken  und dann "Debug" wählen .

Aufgabe 1 (★)

Die Klasse `Triangle` erlaubt die Darstellung von $Z \times S$ Dreiecksmatrizen (von `int` Werten). Z und S sind immer strikt grösser als 1 (d.h., > 1). Eine $Z \times S$ Dreiecksmatrix hat Z Zeilen X_0, X_1, \dots, X_{Z-1} , wobei Zeile X_i genau $(i * (S - 1) / (Z - 1)) + 1$ viele Elemente hat. Dieser Ausdruck wird nach den Regeln für `int` Ausdrücke in Java ausgewertet. Für eine Dreiecksmatrix D ist $D_{i,j}$ das $(j + 1)$ -te Element in der $(i + 1)$ -ten Zeile. $D_{0,0}$ ist das erste Element in der ersten Zeile [die immer genau 1 Element hat]. Abbildung 1 zeigt Beispiele von Dreiecksmatrizen. Beachten Sie, dass es möglich ist, dass zwei (aufeinanderfolgende) Zeilen die selbe Anzahl Elemente haben.

0,0		0,0		0,0
1,0 1,1		1,0 1,1		3,0 3,1 3,2 3,3
2,0 2,1 2,2 2,3		2,0 2,1 2,2		0,0
3,0 3,1 3,2 3,3 3,4		3,0 3,1 3,2 3,3		1,0
4,0 4,1 4,2 4,3 4,4 4,5 4,6				2,0 2,1

Abbildung 1: Beispiele von 5×7 , 4×4 , 2×4 und 3×2 Dreiecksmatrizen.

In der Datei `Triangle.java` finden Sie die Klasse `Triangle` mit einem Konstruktor `Triangle(int z, int s)`, der eine $z \times s$ Dreiecksmatrix erstellt. Dieser Konstruktor setzt die Werte aller Elemente auf 0. Vervollständigen Sie diese Klasse, so dass die folgenden Methoden unterstützt werden:

1. `int get(int i, int j)` gibt das Element $D_{i,j}$ zurück. Falls es nicht existiert wird eine `ArrayIndexOutOfBoundsException` Exception geworfen.
2. `void put(int i, int j, int value)` setzt das Element $D_{i,j}$ auf den Wert `value`. Falls das Element nicht existiert wird eine `ArrayIndexOutOfBoundsException` Exception geworfen.
3. `int[] linear()` liefert die Elemente in der kanonischen Reihenfolge (die Elemente jeder Zeile mit steigendem Index, und die Zeilen in steigender Reihenfolge).
4. `void init(int[] data)` setzt die Elemente von D durch die Werte in `data`. Sie dürfen annehmen, dass `data` genauso viele Elemente hat wie D . Die Methode setzt die Elemente von D , so dass die Folge `x.linear(); int[] y = x.linear();` in einen Array `y` resultiert für den `Arrays.equals(y, data)` den Wert `true` ergibt.
5. `void add(Triangle t)` Ein Aufruf `D.add(t)` addiert zu jedem Element $D_{i,j}$ den Wert von $t_{i,j}$, falls $t_{i,j}$ existiert. Falls $t_{i,j}$ nicht existiert, dann bleibt $D_{i,j}$ unverändert.

Aufgabe 2 (★★★)

In dieser Aufgabe implementieren Sie für eine Datenbank von Personengesundheitsdaten das Erheben von Statistiken (Task a), das Deklassifizieren von Einträgen (Task b), und das Verlinken von Einträgen (Task c). Alle Unteraufgaben können separat gelöst werden. Sie können neue Klassen, Konstruktoren, Attribute, Methoden etc. hinzufügen, aber Sie dürfen die vorgegebenen Signaturen nicht ändern.

Die Datenbank selber ist bereits mit der Klasse `Database` implementiert. Die Datenbank hält eine Liste von Einträgen, welche durch die Klasse `Item` repräsentiert werden. Die folgenden 4 Paragraphen erklären alle in der Vorlage gegebenen Klassen im Detail.

Item Die Klasse `Item` repräsentiert einen Datenbankeintrag mit 4 Attributen: eine ID (int), ein Alter (int), einen Gesundheitswert (int), und ein Sicherheitslevel, welches durch die Klasse `Level` repräsentiert wird. Alter und Gesundheitswert sind immer ≥ 0 . Die Methoden `Item.getID()`, `Item.getAge()`, `Item.getHealth()`, `Item.getLevel()` geben jeweils die ID, das Alter, den Gesundheitswert, und das Sicherheitslevel eines Eintrags zurück. Die Methode `Item.setHealth(int newHealth)` setzt den Gesundheitswert auf `newHealth`. Die anderen Attribute können nicht geändert werden.

Level Die Klasse `Level` repräsentiert ein Sicherheitslevel. Ein Sicherheitslevel wird über ein Set von Integern definiert, welches in einem Attribut der Klasse `Level` gespeichert wird und von der Methode `Level.getPoints()` zurückgegeben wird. Ein Level A ist *schwächer* als ein Level B, falls es für jeden Wert in `A.getPoints()` einen grösseren Wert in `B.getPoints()` gibt. Zum Beispiel sind die Level `{2,3}` und `{3,4}` schwächer als das Level `{3,5}`, da 5 grösser als die anderen Werte ist. Die Level `{5}` und `{4,6}` hingegen sind nicht schwächer als `{3,5}`, da es keinen Wert grösser als 5 und 6 in `{3,5}` gibt. Ein Level A ist *verwandt* mit einem Level B, falls die Summe der Werte in `A.getPoints()` gleich der Summe der Werte in `B.getPoints()` ist. Zum Beispiel ist das Level `{1,2,3,4}` verwandt mit den Levels `{10}` und `{4,6}` (die Summe ist überall 10), aber nicht mit dem Level `{4,5}`.

ItemFactory Die Klasse `ItemFactory` wird verwendet, um Datenbankeinträge zu erstellen. Die Methode `ItemFactory.createItem(Level level, int id, int age, int health)` gibt ein Exemplar der Klasse `Item` zurück, deren Attribute mit den Argumenten initialisiert wurden.

Database Die Klasse `Database` repräsentiert eine Datenbank und hat folgende vorgegebene Methoden:

- `Database.getItemFactory()` gibt ein Exemplar der Klasse `ItemFactory` zurück. Die `ItemFactory I` ist assoziiert mit der Datenbank `D`, falls `I` von `D.getItemFactory()` zurückgegeben wird.
- `Database.add(Item item)` fügt der Datenbank den Eintrag `item` hinzu.
- `Database.getItems()` gibt die Liste aller Einträge zurück, welcher der Datenbank hinzugefügt wurden. Sie dürfen annehmen, dass für eine Datenbank `D` alle Einträge in `D.getItems()` eine einzigartige ID haben, über `D.add` hinzugefügt wurden, über `D.getItemFactory()` erstellt wurden, und keiner anderen Datenbank hinzugefügt werden. Ein hinzugefügter Eintrag wird nie wieder entfernt.

- (a) (★) Implementieren Sie die Methode `Database.summary(Level groupLevel)`, welche die durchschnittlichen Alter für Gruppen von Einträgen berechnet. Die Gruppen sind wie folgt definiert: Für jede ganze Zahl k , die ein Vielfaches von 10 ist, besteht die Gruppe für k aus der Menge aller Einträge E , so dass
- (a) das Level von E schwächer ist als das Argument `groupLevel`; *und*
 - (b) der abgerundete Gesundheitswert von E gleich k ist (in Java gilt `E.getHealth() / 10 * 10 == k`).

Die Methode gibt eine Map `<Integer, Integer>` zurück, die für jede wie oben definierte Gruppe das durchschnittliche Alter speichert. Für einen Schlüssel k gibt die Map den Wert `null` zurück, falls k kein Vielfaches von 10 ist oder die Gruppe für k leer ist. Die Map gibt für einen Schlüssel k einen Wert v ungleich `null` zurück, genau dann wenn **alle** folgenden Bedingungen erfüllt sind:

- k ist ein Vielfaches von 10;
- Die Gruppe der gefundenen Einträge für k ist nicht leer;
- v ist das runter-gerundete durchschnittliche Alter aller Einträge in der Gruppe.

ID	Level	Alter	Health
1	{1,2}	20	100
2	{2,3}	31	109
3	{5,6}	50	100
15	{4}	20	133

Beispiel: Für die obige Datenbank mit 4 Einträgen soll `summary` für das Level `{3,5}` die Map $\{100 \mapsto 25, 130 \mapsto 20\}$ zurückgeben. Das Level der Einträge mit ID 1, 2 und 15 ist schwächer als `{3,5}`. Die beiden Einträge 1 und 2 sind in einer Gruppe für $k = 100$. Das gerundete durchschnittliche Alter ist $51/2 = 25$. Der Eintrag mit ID 15 ist allein in der Gruppe für $k = 130$. Die Map hat keine weiteren Einträge. Der Eintrag mit ID 3 hat keinen Einfluss auf das Ergebnis, weil sein Level nicht schwächer als `{3,5}` ist.

- (b) (★) Implementieren Sie die Methode `ItemFactory.createDeclass(Level level, int id, int targetId)`, die einen *Deklassifikationseintrag* zurückgibt. Ein Deklassifikationseintrag ist selber ein Eintrag, also ein Exemplar der Klasse `Item`. Ein Deklassifikationseintrag hat damit auch eine ID, ein Sicherheitslevel, ein Alter, und einen Gesundheitswert, welche von den entsprechenden getter-Methoden zurückgegeben werden. ID und Sicherheitslevel eines Deklassifikationseintrags sind jeweils das `id` und `level` Argument des `createDeclass` Aufrufs, mit welchem der Eintrag erstellt wurde. Das Alter und der Gesundheitswert eines Deklassifikationseintrags sind jeweils das Alter und der Gesundheitswert des *Zieleintrags* vom Deklassifikationseintrag. Der Zieleintrag von einem Deklassifikationseintrag D ist der Eintrag E , so dass
- `E.getID()` gleich dem Parameter `targetId` ist, mit welchem D erstellt wurde; *und*
 - E aus der Datenbank ist, mit welcher die `ItemFactory` assoziiert ist, mit welcher D erstellt wurde.

Falls es keinen Zieleintrag gibt, wirft die Methode `createDeclass` eine `IllegalArgumentException`. Beachten Sie, dass Zieleinträge selber Deklassifikationseinträge sein können. Ein Aufruf der Methode `Item.setHealth(h)` auf einem Deklassifikationseintrag hat keinen Effekt; dies wird nicht in den Tests überprüft.

Ein Deklassifikationseintrag R *erreicht* einen Eintrag A , falls entweder A der Zieleintrag von R ist oder falls der Zieleintrag von R ein Deklassifikationseintrag ist, welcher A erreicht. Die Methode `createDeclass` wirft eine `IllegalArgumentException`, falls der zurückzugebene Deklassifikationseintrag R einen Eintrag erreicht, dessen Level verwandt ist mit dem Level von R . Zur Erinnerung: Der Paragraph über die Klasse `Level` erklärt, wann zwei Level verwandt sind.

Wenn Sie auch Task a) (Methode `summary`) implementieren, so wird die Methode `summary` nur auf Datenbanken ohne Deklassifizierungseinträge aufgerufen.

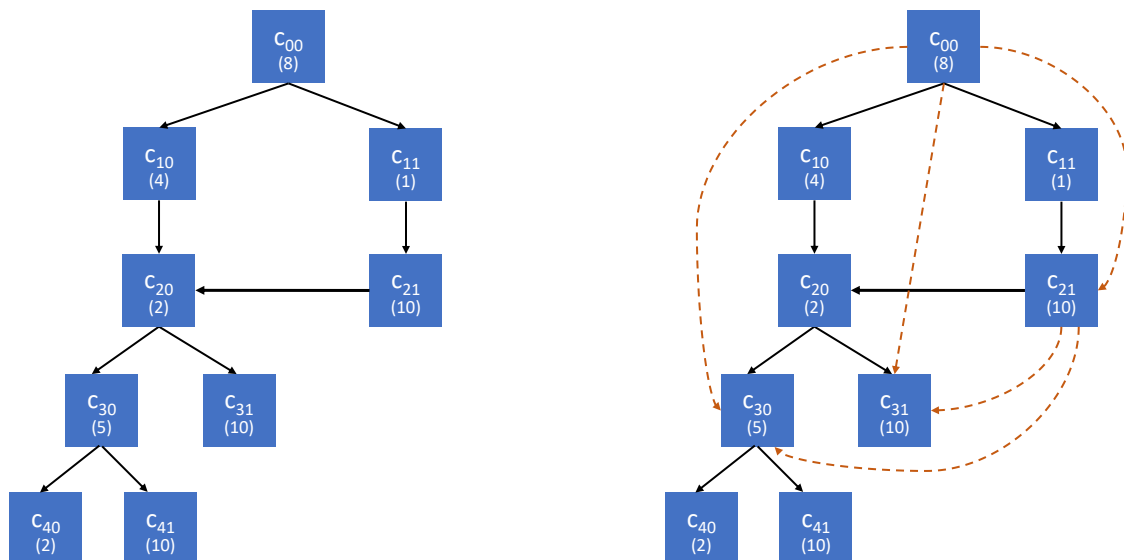
- (c) (★) Implementieren Sie die Methode `Database.createLink(Set<Integer> ids)`. Der Methodenaufruf `D.createLink(ids)` *verlinkt* alle Einträge der Datenbank D miteinander, welche eine ID haben, die im Argument `ids` enthalten ist. Wenn `E.setHealth(h)` auf einem Eintrag E aufgerufen wird, dann wird der Gesundheitswert aller Einträge, welche mit E verlinkt sind, auf das Argument h gesetzt. Einträge können beliebig oft verlinkt werden und verlinken ist transitiv, das heisst, wenn ein Eintrag A mit einem Eintrag B verlinkt ist und B mit einem Eintrag C verlinkt ist, dann ist A auch mit C verlinkt. Verlinken ist auch immer symmetrisch, das heisst, wenn A mit B verlinkt ist, dann ist auch B mit A verlinkt. Zusätzlich ist verlinken reflexiv, das heisst, ein Eintrag ist immer mit sich selber verlinkt.

Der Aufruf `D.createLink(ids)` soll eine `IllegalArgumentException` werfen, falls es eine ID im Argument `ids` gibt, für welche es keinen Eintrag mit der gleichen ID in der Datenbank D gibt.

Aufgabe 3 (★★)

Sie wurden als Stadtplaner eingestellt, um Aufgaben bei der Wartung und Analyse von Stadtnetzwerken zu übernehmen. Stadtnetzwerke bestehen aus Städten und aus Einbahnstrassen, welche Städte verbinden. Die Klasse `City` repräsentiert eine Stadt und hat zwei Attribute: `City.size` definiert die Grösse der Stadt als eine strikt positive ganze Zahl (> 0) und `City.oneWayStreets` enthält Städte, welche von der Stadt durch eine Einbahnstrasse erreicht werden können (`City.oneWayStreets` wird als `Set<City>` repräsentiert). Das heisst, eine Stadt d ist genau dann in $c.oneWayStreets$ enthalten, wenn es eine Einbahnstrasse von c nach d gibt. Eine *Route* (r_1, r_2, \dots, r_N) ist eine Sequenz von Städten, so dass es für alle i mit $1 \leq i \leq N - 1$ eine Einbahnstrasse von r_i nach r_{i+1} gibt. Eine Stadt c ist von einer Stadt c' erreichbar, wenn es eine Route (r_1, \dots, r_N) gibt mit $r_1 = c$ und $r_N = c'$ (das heisst, c ist immer von c erreichbar, da (c) eine Route mit genau einer Stadt ist). Sie dürfen in den folgenden Aufgaben annehmen, dass Stadtnetzwerke keine Zyklen haben (das heisst, es gibt keine Route (r_1, \dots, r_N) mit $r_1 = r_N$ und $N \geq 2$).

Die folgende Abbildung zeigt zwei Strassennetzwerke:



Die Rechtecke repräsentieren Städte und die Zahlen in den Klammern entsprechen den Grössen. Es gibt eine gerichtete Kante von c nach d , genau dann wenn es eine eine Einbahnstrasse von c nach d gibt.

- (a) (★) Das Strassenamt hat beschlossen, dass der Verkehr zwischen grösseren Städten durch neue Einbahnstrassen verbessert werden muss. Für diese Anpassung hat das Strassenamt für jede ganze Zahl k mit $k \geq 0$ die k -Grenze einer Stadt eingeführt, welche einer Menge von Städten entspricht. Eine Stadt d ist genau dann in der k -Grenze der Stadt c enthalten, wenn es eine Route (r_1, r_2, \dots, r_N) gibt, wobei folgende Bedingungen gelten:

- Die Route führt von c nach d (das heisst, $r_1 = c$ und $r_N = d$).
- Die Route enthält mindestens 3 Städte (das heisst, $N \geq 3$).
- d hat mindestens Grösse k
- Alle Städte, welche auf der Route strikt zwischen c und d liegen, sind strikt kleiner als k . Das heisst, für alle i mit $2 \leq i < N$ gilt $r_i.size < k$.

Implementieren Sie die Methode `CityNetwork.transform(City origin, int k)`. Die Methode muss für jede Stadt c , welche mindestens Grösse k hat und von `origin` erreichbar ist, eine Einbahnstrasse von c zu jeder Stadt d in der k -Grenze von c hinzufügen. Sie dürfen annehmen, dass `origin` nie null ist und $k \geq 0$ gilt.

Die obere Abbildung zeigt ein Beispiel für den Effekt von `CityNetwork.transform(c_{00} , 5)`. Das linke Strassennetzwerk zeigt den Zustand vor dem Aufruf und das rechte Strassennetzwerk zeigt den Zustand nach dem Aufruf. Die gestrichelten Kanten (in Orange) entsprechen den neu hinzugefügten Einbahnstrassen. c_{00} und c_{21} sind die einzigen von c_{00} erreichbaren Städte, welche mindestens Grösse 5 haben und vor dem Aufruf eine nicht-leere 5-Grenze haben. Zum Beispiel ist die 5-Grenze von c_{00} vor dem Aufruf durch $\{c_{21}, c_{30}, c_{31}\}$ gegeben (beachten Sie, dass c_{40} und c_{41} *nicht* in der 5-Grenze von c_{00} liegen.)

- (b) (★) Das Strassenamt möchte besser verstehen welche Städte durch eine bestimmte Anzahl von spezifischen Routen erreicht werden können.

Implementieren Sie dafür die Methode `CityNetwork.getCities(City origin, int k, Set<Integer> sizeSet)`, welche eine Menge von Städten (als `Set<City>` repräsentiert) zurückgibt. Die zurückgegebene Menge M muss eine Stadt c genau dann enthalten, wenn es mindestens k unterschiedliche Routen R gibt, wobei jede Route $(r_1, \dots, r_N) \in R$ folgende Bedingungen erfüllen muss:

- Die Route führt von $origin$ nach c . Das heisst, $r_1 = origin$ und $r_N = c$.
- Für jede Grösse s in $sizeSet$ muss es mindestens ein i geben mit $r_i.size = s$.

Zwei Routen r und r' sind unterschiedlich, wenn sie eine unterschiedliche Länge haben, oder wenn für mindestens einen Index i die i -ten Städte von r und r' unterschiedlich sind (also r und r' für Index i unterschiedliche Referenzen enthalten).

Zum Beispiel im linken Strassennetzwerk in der oberen Abbildung muss der Aufruf

`CityNetwork.getCities(c_{00} , k , Set.of(2, 5, 8))`

für $k \in \{1, 2\}$ die Menge $\{c_{30}, c_{40}, c_{41}\}$ zurückgeben.

Notizen

Notizen

Notizen

Notizen

Notizen