

Herbst 2022
252-0027 – Einführung in die Programmierung

Departement Informatik
ETH Zürich

3. Februar 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.

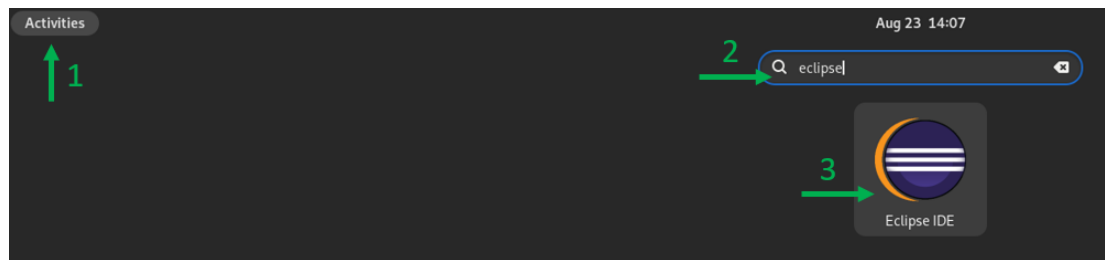
Hinweise

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 vier 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. 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

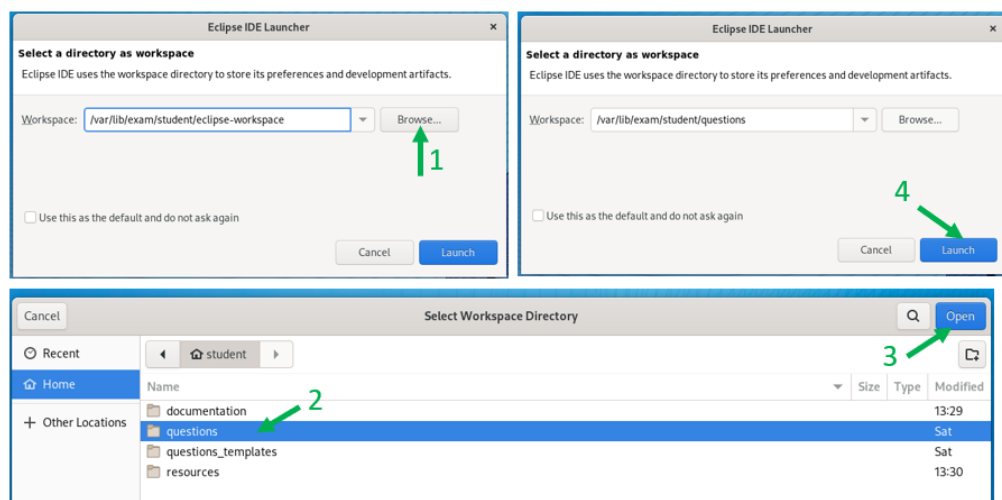
1. Sobald die Programmierprüfung startet, können Sie sich an Ihrem Computer anmelden. Geben Sie zuerst Ihren vollen Namen und im nächsten Schritt Ihren NETHZ-Namen und Ihre Legi-Nummer ein. (Sie brauchen *nicht* Ihr NETHZ-Passwort.) Sie werden auch in einem weiteren Fenster darauf hingewiesen, dass Ihr 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”.







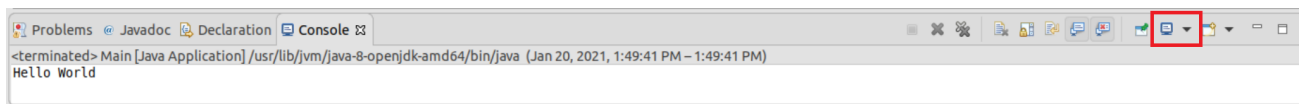
4. Wenn Eclipse fertig gestartet ist, sehen Sie den Willkommens-Bildschirm. Klicken Sie wenn nötig oben rechts auf “Workbench”. Nun sollten Sie links die drei Projekte “Aufgabe 1”, “Aufgabe 2” und “Aufgabe 3” sehen. **Es kann einige Minuten dauern bis Eclipse alles geladen hat. Warten Sie bis die Ladenachricht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

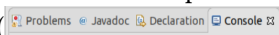





Durch einen Klick auf den Pfeil von , wird eine Liste aller vorhandenen Konsolen angezeigt:

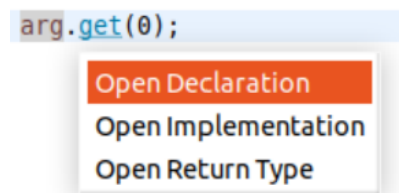


Durch klicken von  werden alle terminierten Konsolen geschlossen. Klicken Sie wiederholt  und  (oder ) bis  ausgegraut oder verschwunden ist, um alle Programme und Tests zu terminieren und alle Konsolen zu schliessen.







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 (★)

Gegeben sei eine $k \times k$ Matrix M von `int` Werten, welche in Java als ein 2D-Array repräsentiert wird. Diese Matrix enthält das Muster, das Sie in einer $n \times n$ Matrix O finden sollen ($n \geq k > 0$). Wir sagen, dass die Matrix O die Matrix M für eine Ursprungsposition (x, y) (mit $0 \leq x < n$, $0 \leq y < n$) enthält wenn für alle i und j mit $0 \leq i < k$, $0 \leq j < k$ gilt: $O[x+i][y+j] = M[i][j]$. Es ist möglich, dass eine Matrix O die Matrix M für verschiedene Ursprungspositionen $(x_1, y_1), (x_2, y_2), \dots$ enthält (oder für keine).

Wenn es keine Ursprungsposition (x, y) gibt, für die eine Matrix O die Matrix M enthält, dann kann durch maximal k^2 Korrekturschritte die Matrix O in eine Matrix O' so verändert werden, so dass danach die Matrix O' die Matrix M enthält. (Es müssen für eine Ursprungsposition (x, y) alle die Elemente $O[x+i][y+j]$ geändert werden, für die $O[x+i][y+j] \neq M[i][j]$. Jeder Korrekturschritt setzt *ein* Element des Arrays.)

Schreiben Sie eine Methode `Pattern.match(int[][] origin, int[][] muster)`, welche die minimale Anzahl Korrekturschritte und eine dazu passende Ursprungsposition findet. Dabei ist `muster` eine $k \times k$ Matrix M von `int` Werten und `origin` eine $n \times n$ Matrix von `int` Werten. Sie können davon ausgehen, dass beide Parameter nicht null sind. Sollte $k > n$ sein, dann sollte die Methode eine `IllegalArgumentException` werfen.

Die Methode `match` gibt das Ergebnis in einem Objekt `record` der Klasse `MatchRecord` zurück. Dabei muss `(record.x, record.y)` eine Ursprungsposition in der Matrix `origin` sein, für welche die Anzahl der Korrekturschritte minimal ist damit die korrigierte Matrix `origin'` die Matrix `muster` enthält. `record.count` muss dabei der minimalen Anzahl Korrekturschritte entsprechen. Wenn die Matrix `origin` die Matrix `muster` an der Position (x, y) ohne Korrekturschritte enthält, dann ist die Anzahl der Korrekturschritte 0.

Wenn es mehrere Positionen $A = \{(x_1, y_1), (x_2, y_2), \dots\}$ gibt, so dass die Matrix `origin` die Matrix `muster` für alle Ursprungspositionen $(x_i, y_i) \in A$ mit q Korrekturschritten enthält und q das Minimum der nötigen Korrekturschritte ist, dann können Sie die Anzahl Korrekturschritte q und eine beliebige Position $(x_i, y_i) \in A$ zurückgeben.

Aufgabe 2 (★★★)

In dieser Aufgabe implementieren Sie eine Contact Tracing Applikation, welche es ermöglichen soll, Kontakte während eines Virus-Ausbruches nachzuverfolgen. Ihre Implementierung soll zunächst Begegnungen zwischen verschiedenen Person Instanzen anonym protokollieren, sodass bei einem positivem Test die Benachrichtigung aller Personen möglich ist, die direkt oder indirekt mit einer positiv getesteten Person in Kontakt standen.

Anonyme Begegnungen. Um Anonymität zu gewährleisten, dürfen zwei Personen *A* und *B* bei einer Begegnung lediglich anonyme Integer IDs austauschen, ohne dabei die Identität der jeweils anderen Person aufzudecken. Beide Personen speichern hierbei sowohl die eigene ID, als auch die ID der anderen Person. Bei der positiven Testung von *A* kann dann mithilfe der anonymen IDs, die *A* genutzt hat, festgestellt werden, ob *B* einer dieser IDs begegnet ist. Um zu vermeiden, dass wiederkehrende IDs die Identifikation einer Person über mehrere Begegnungen hinweg ermöglichen, benutzt jede Person für jede Begegnung frische IDs, welche über eine zentrale Klasse `ContactTracer` vergeben werden. Frisch bedeutet hierbei, dass eine ID zuvor noch nie bei einer Begegnung verwendet wurde.

Direkte und Indirekte Kontakte. Nachdem eine Reihe an Begegnungen protokolliert wurden, wird eine oder mehrere Personen positiv getestet. Mit dem erfassten Netzwerk aus Begegnungen soll Ihre Applikation dann zwei verschiedene Arten an Kontakt-Personen bestimmen:

- Als *direkte Kontakte* gelten alle Personen, die eine Begegnung mit einer positiv getesteten Person hatten.
- Als *indirekte Kontakte* hingegen gelten alle Personen, die zwar selbst keine Begegnung mit einer positiv getesteten Person hatten, jedoch Kontakt mit mindestens einer anderen Person, welche als direkter Kontakt gilt, hatten. Indirekte Kontakte mit mehr als einer Zwischenperson müssen Sie dabei nicht berücksichtigen.

Sie dürfen dabei annehmen, dass zunächst alle Begegnungen erfasst werden und erst dann Personen positiv getestet werden. Nach der ersten positiven Testung finden keine weiteren Begegnungen mehr statt.

Benachrichtigungen. Da nicht alle Personen gleichermassen gefährdet sind, soll Ihre Applikation die Benachrichtigung der Kontakt-Personen vom Alter, der Art des Kontaktes, sowie dem Testergebnis der jeweiligen Kontakt-Person abhängig machen. Dabei soll eine der drei Warnstufen *Keine Benachrichtigung*, *Low-Risk Benachrichtigung* oder *High-Risk Benachrichtigung* ausgesprochen werden. Zu Beginn haben alle Personen die Standard-Warnstufe *Keine Benachrichtigung* und gelten als negativ getestet. Davon ausgehend sollen nach jedem registrierten positiven Test die zugehörigen Kontakt-Personen wie folgt benachrichtigen werden:

Testergebnis der Kontakt-Person	Alter der Kontakt-Person	Direkter Kontakt	Indirekter Kontakt
Positiv	-	Keine Benachr.	Keine Benachr.
Negativ	≤ 60 Jahre alt	High-Risk	Keine Benachr.
Negativ	> 60 Jahre alt	High-Risk	Low-Risk

Eine negativ getestete Person, die höchstens 60 Jahre alt ist und die nur in indirektem Kontakt zu einer positiven Person stand, soll beispielsweise keine Benachrichtigung erhalten (Reihe 2). Eine negativ getestete Person über 60 Jahre hingegen soll als indirekter Kontakt eine Low-Risk Benachrichtigung erhalten (Reihe 3).

Wenn mehrere Personen positiv getestet werden, soll Ihre Applikation immer die höchste, geltende Warnstufe für die anderen, negativ getesteten Personen berechnen. Dabei ist die Ordnung der Warnstufen wie folgt definiert: *Keine Benachrichtigung* < *Low-Risk Benachrichtigung* < *High-Risk Benachrichtigung*. Positiv getestete Personen hingegen sollen immer die Warnstufe *Keine Benachrichtigung* erhalten. Im Allgemeinen dürfen Sie zudem annehmen, dass eine Person, die einmal positiv getestet wurde, für den Rest der Laufzeit Ihrer Applikation als positiv getestet gilt.

Implementierung. Erweitern Sie den vorgegebenen Code für die Klasse `ContactTracer` und das Interface `Person` wie folgt, um die Contact Tracing Applikation umzusetzen:

Implementieren Sie das Interface `Person` mit den folgenden public Methoden:

- `Person.getUsedIds()`. Diese Methode gibt die Liste aller IDs zurück (`List<Integer>`), die für diese Person als frische ID verwendet wurden, um eine Begegnung zu protokollieren. Nach Hinzufügen einer ID in diese Liste muss die selbe ID in die jeweilige `Person.getSeenIds()`-Liste des Gegenübers eingetragen sein.
- `Person.getSeenIds()`. Diese Methode gibt die Liste aller IDs zurück (`List<Integer>`), die diese Person als die frische ID des jeweiligen Gegenübers bei einer Begegnung protokolliert hat. Nach Hinzufügen einer ID in diese Liste muss die selbe ID in die jeweilige `Person.getUsedIds()`-Liste des Gegenübers eingetragen sein.
- `Person.getNotification()`. Diese Methode gibt den aktuellen Benachrichtigungsstatus der Person zurück. Der Rückgabewert soll vom enum-Typ `NotificationType` sein, welcher vorgegeben ist und die drei möglichen Warnstufen modelliert. `NotificationType` ist im Interface `Person` definiert und enthält die drei Werte `NoNotification` (keine Benachrichtigung), `LowRiskNotification` (Low-Risk Benachrichtigung), `HighRiskNotification` (High-Risk-Benachrichtigung).
- `Person.setTestsPositively()`. Diese Methode wird aufgerufen, um eine Person als positiv getestet zu markieren. Nach dem Aufrufen dieser Methode sollen automatisch alle Kontakte von A benachrichtigt worden sein und die entsprechenden Warnstufe per `Person.getNotification()` zurückgeben.

Implementieren Sie zusätzlich die Klasse `ContactTracer`, welche die folgenden public Methoden besitzt:

- `ContactTracer.registerEncounter(Person p1, Person p2)`. Mit dieser Methode wird eine (beidseitige) Begegnung zwischen Person-Objekten `p1` und `p2` protokolliert, indem die beiden Personen anonyme IDs austauschen. Die ausgetauschten IDs müssen dabei unterschiedlich sein. Eine Begegnung zwischen `p1` und `p2` ist beidseitig und muss somit auch als Begegnung zwischen `p2` und `p1` gewertet werden.
- `ContactTracer.createPerson(int age)`. Diese Methode gibt ein `Person` Objekt zurück. Das Alter der Person ist durch den `age` Parameter bestimmt.

Alle `Person`-Objekte werden von der Methode `ContactTracer.createPerson(int age)` erstellt. Der `ContactTracer` wird über den parameterfreien Konstruktor `ContactTracer()` instanziiert. Sie dürfen annehmen, dass nie mehr als 1024 Begegnungen zwischen Personen protokolliert werden.

Implementieren Sie auf Basis dieser Vorlage eine Lösung für das Contact Tracing Problem. Sie dürfen dabei zusätzliche Enum-Typen, Klassen, Methoden und Attribute frei einfügen. In der Datei `ContactTracerTests` finden Sie Beispiel-Szenarien als Test Cases, die Sie nutzen können, um Ihre Implementierung zu testen.

Aufgabe 3 (★★★)

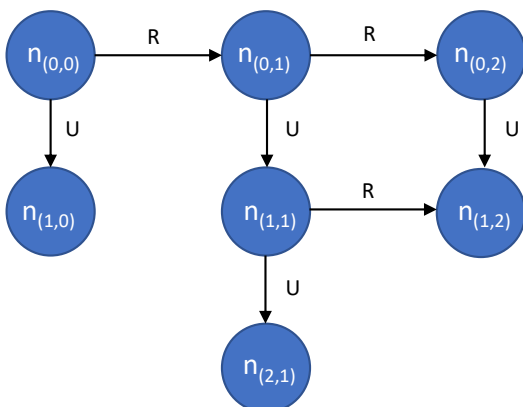
In dieser Aufgabe betrachten wir gerichtete Graphen, wobei es für jeden Knoten g höchstens zwei gerichtete Kanten von g zu anderen Knoten f, h geben kann (f, g, h können gleich sein). Wir unterscheiden dabei zwischen der rechten und unteren Kante (und damit dem rechten und unteren Knoten).

Die Klasse `Node` repräsentiert einen Knoten in einem solchen Graphen. Die Methode `Node.getRight()` (bzw. `Node.getDown()`) gibt den rechten Knoten (bzw. unteren Knoten) zurück (als `Node`-Objekt). Wenn der rechte Knoten von n_0 nicht existiert, dann gibt `Node.getRight()` `null` zurück (analog für den unteren Knoten). Die Methode `Node.setRight(Node r)` (bzw. `Node.setDown(Node d)`) setzt den rechten (bzw. unteren) Knoten.

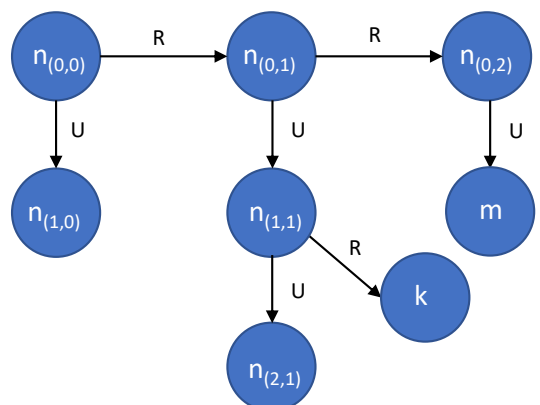
Im ersten Teil der Aufgabe ist das Ziel zu entscheiden, ob der von einem `Node`-Objekt definierte Graph ein *partiell*es Gitter modelliert. Im zweiten Teil der Aufgabe geht es darum zu entscheiden, ob ein partielles Gitter existiert, welches eine bestimmte Eigenschaft erfüllt.

Die folgende Abbildung zeigt ein Beispiel für ein partielles Gitter mit Ursprungsknoten $n_{(0,0)}$ und Koordinaten $\{(0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,1)\}$ (links) und zeigt ein Beispiel, welches keinem partiellen Gitter entspricht (rechts), wobei jeweils R der rechten Kante und U der unteren Kante entspricht:

Partielles Gitter:



Kein partielles Gitter:



Ein Graph G (wie oben definiert) definiert ein partielles Gitter mit Ursprungsknoten u und Koordinaten K (wobei $K \subseteq \mathbb{N}_{\geq 0} \times \mathbb{N}_{\geq 0}$; das heisst, Koordinaten haben keine negativen Komponenten), so dass folgende Bedingungen gelten:

- Jeder Knoten in G ist über die gerichteten Kanten vom Ursprungsknoten u erreichbar.
- Es gibt gleich viele Knoten in G wie Koordinaten in K . Ausserdem wird jede Koordinate $(i, j) \in K$ durch genau einen einzigartigen Knoten $n_{(i,j)}$ in G repräsentiert. Das heisst, wenn $\{(i, j), (i', j')\} \subseteq K$ und $(i, j) \neq (i', j')$, dann gilt $n_{(i,j)} \neq n_{(i',j')}$.
- Sei $(i, j) \in K$. Wenn der untere Knoten von $n_{(i,j)}$ existiert, dann gilt $(i+1, j) \in K$ und der untere Knoten von $n_{(i,j)}$ ist gegeben durch $n_{(i+1,j)}$. Wenn der rechte Knoten von $n_{(i,j)}$ existiert, dann gilt $(i, j+1) \in K$ und der rechte Knoten von $n_{(i,j)}$ ist gegeben durch $n_{(i,j+1)}$.

In der oberen Abbildung ist der rechte Graph kein partielles Gitter, da k und m unterschiedlich sind.

In den folgenden Unteraufgaben können Sie die vorgegebene Klasse `Coordinate` verwenden, welche Koordinaten modelliert. Der Konstruktor `Coordinate(int x, int y)` erstellt ein `Coordinate`-Objekt `c`, welches die Koordinate (x, y) modelliert. `c.getX()` gibt dabei x zurück und `c.getY()` gibt dabei y zurück. Allgemein modelliert ein `Coordinate`-Objekt `c` die Koordinate $(c.getX(), c.getY())$.

- (a) (★★) Implementieren Sie die Methode `PartialGrid.isPartialGrid(Node origin)`, welche entscheidet, ob Koordinaten K existieren, so dass der durch `origin` definierte Graph (d.h. der Graph gegeben durch alle Knoten und Kanten, welche von `origin` über die gerichteten Kanten erreichbar sind, inklusive `origin` selber) ein partielles Gitter mit Ursprungsknoten `origin` und Koordinaten K definiert.
- (b) (★) Implementieren Sie die Methode `PartialGrid.isRepresentable(List<Coordinate> coordinates)`, welche entscheidet, ob es einen Knoten u gibt, so dass ein partielles Gitter mit Ursprungsknoten u und Koordinaten `coordinates` existiert.

Sie dürfen annehmen, dass `coordinates` nicht leer ist und dass jede Koordinate (i, j) höchstens ein Mal in der Inputliste `coordinates` vorkommt.

Notizen

Notizen

Notizen

Notizen

Notizen