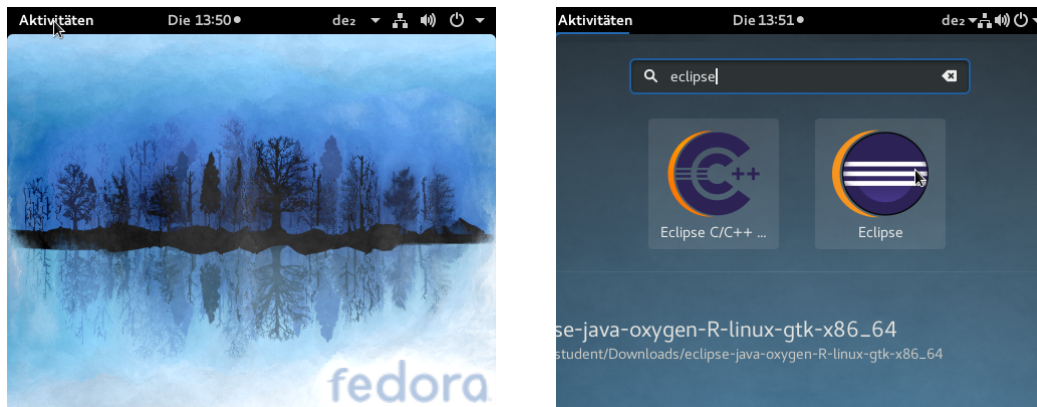


Anmelden und Eclipse starten

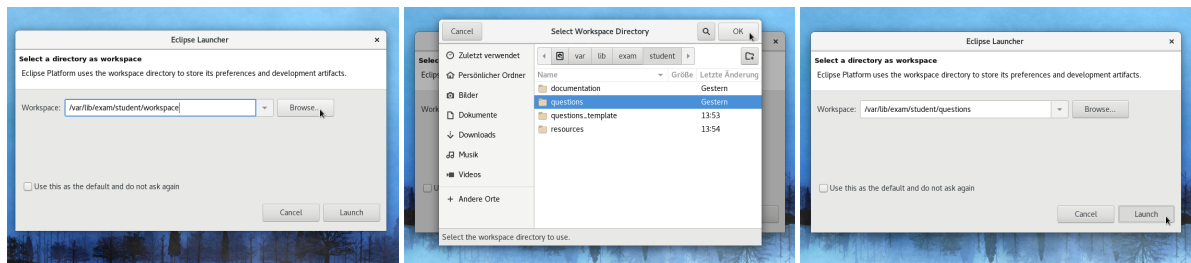
1. Sobald die Programmierprüfung startet, können Sie sich an Ihrem Computer anmelden. Geben Sie zuerst Ihren vollen Namen und im nächsten Schritt Ihren NETHZ-Namen und Ihre Legi-Nummer ein. (Sie brauchen *nicht* Ihr NETHZ-Passwort.) Sie werden auch in einem weiteren Fenster darauf hingewiesen, dass Ihr Computer aufgezeichnet wird, und dass Sie technische Probleme sofort melden müssen.
2. Starten Sie Eclipse, indem Sie oben links auf “Aktivitäten” (oder “Activities”) 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.



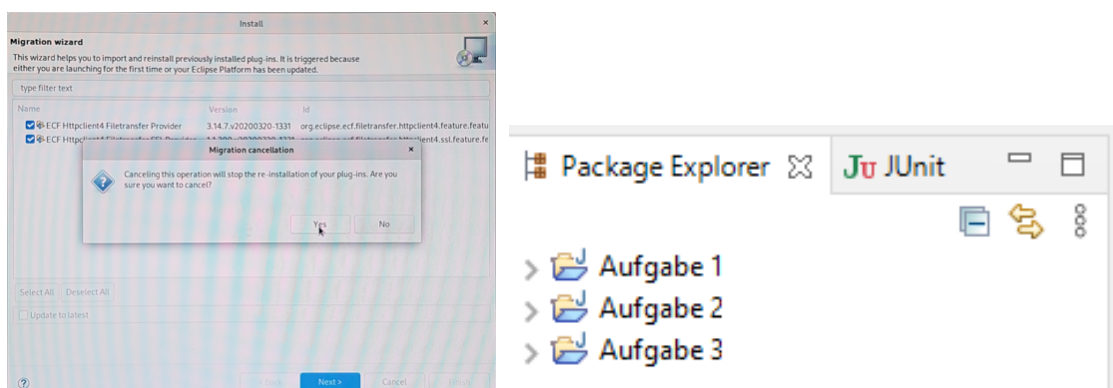
3. Wenn sich das Fenster “Eclipse Launcher” öffnet, stellen Sie sicher, dass der richtige Prüfungs-Workspace ausgewählt ist. Im Feld “Workspace” sollte folgender Pfad stehen, bevor Sie auf “Launch” klicken:

/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”. 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” bis “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.** Ausserdem können Sie die rechtlichen Hinweise zur Computer-Prüfung lesen, indem Sie oben links auf die “Activities” gehen und auf das Informationsicon klicken. Viel Spaß!



Hinweise





Während der Programmierprüfung dürfen Sie nicht mehr an der schriftlichen Prüfung weiterarbeiten, auch wenn diese noch nicht eingezogen worden ist. **Dies gilt als Täuschungsversuch.**

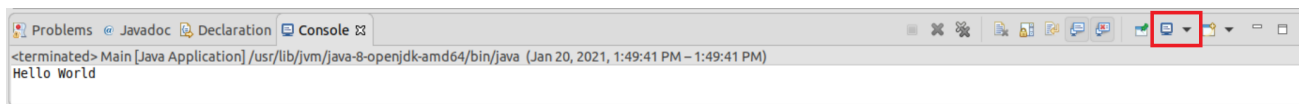
1. Öffnen Sie diese Prüfung erst, wenn die Aufsicht den Beginn der gesamten Prüfung bekannt gibt (Sie dürfen schon vor dem Ende der schriftlichen Prüfung die Programmieraufgaben anschauen).
2. Beachten Sie bitte während und *nach* der Prüfung unbedingt die Hygienevorschriften.
3. Im Prüfungsraum bitte keine Gespräche oder Lärm.
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 3 Stunden (180 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.
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 Rückgabotyp, 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. 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. Die Verwendung von Java Reflection ist nicht erlaubt (und auch nicht von Vorteil). Sonst dürfen Sie, sofern keine Einschränkungen aufgeführt sind, Klassen und Interfaces importieren.
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 Studentin oder ein Student zur Toilette. Da Studierende aus verschiedenen Räumen die selben WC Anlagen benutzen, kann die Koordination etwas Zeit in Anspruch nehmen.
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 30 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. Nehmen Sie bitte keine Aufgabenstellung mit - wir sammeln diese später ein.

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:

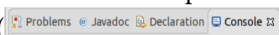


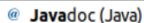
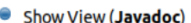


Durch einen Klick auf den Pfeil von , wird eine Liste aller vorhandenen Konsolen angezeigt:

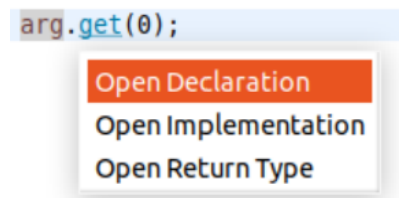


Durch klicken von  werden alle terminierten Konsolen geschlossen. Klicken Sie wiederholt  und  (oder ) bis  ausgegraut oder verschwunden ist, um alle Programme und Tests zu terminieren und alle Konsolen zu schliessen.




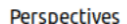


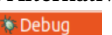
Javadoc

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 .

Aufgabe 1 (★★★)

In dieser Aufgabe schreiben Sie ein Programm, welches Funkfrequenzen zu Mobilfunk-Stationen zuordnet. Jede Mobilfunk-Station kann nur auf einer bestimmten Frequenz senden und muss dabei darauf achten, dass es keine Interferenzen mit anderen Stationen in der Nähe gibt. Die Reichweite einer Mobilfunk-Station ist $d/2$ Kilometer. Es gibt Interferenzen, sobald eine Station auf der gleichen Frequenz sendet wie eine (oder mehrere) andere Stationen und weniger als d Kilometer dazwischenliegen. Der Einfachheit halber sind die Mobilfunk-Stationen auf einer flachen Ebene angesiedelt und die Interferenz wird nur durch die Entfernung bestimmt.

Die Standorte der Mobilfunk-Stationen sind als Koordinaten-Paare (x_i, y_i) gegeben, welche die Position in Kilometern (Länge und Breite) bestimmen. Das Programm-Skelett beinhaltet schon eine Klasse `Coordinate` für die Beschreibung von Koordinaten und eine Funktion (`Coordinate.distance(Coordinate c)`) für die Berechnung der Distanz zwischen zwei Punkten. `Coordinate.getX()` und `Coordinate.getY()` geben die Position in den zwei Dimensionen an.

Desweiteren enthält das Skelett eine Klasse `Frequencies`, welche eine Stationentopologie und einen Minimalabstand d fixiert (diese werden über den Konstruktor von `Frequencies` initialisiert und danach nicht mehr geändert). `Frequencies.stations` enthält die Standorte der Stationen als `ArrayList<Coordinate>` und `Frequencies.minDistance` entspricht dem Minimalabstand d als `double`.

- (a) (★★) Ergänzen Sie nun die Klasse `Frequencies`, indem Sie die Funktion `Frequencies.assignmentPossible(int N)` implementieren, welche als Parameter die Anzahl N ($N \geq 0$) der verfügbaren Funkfrequenzen nimmt. Die Methode muss entscheiden, ob es für die gegebenen Stationen und dem gegebenen Minimalabstand d (gespeichert im `Frequencies`-Objekt) möglich ist, jeder Station eine von N möglichen Frequenzen zuzuordnen, damit keine zwei Stationen, welche weniger als d Kilometer voneinander entfernt sind, auf der gleichen Frequenz senden. Falls eine solche Zuordnung möglich ist, dann muss die Methode `true` und sonst `false` zurückgeben.
Eine Station a und eine andere Station b können auf derselben Frequenz senden, falls $\text{distanz}((x_a, y_a), (x_b, y_b)) \geq d$ gilt.
- (b) (★) Implementieren Sie die Funktion `Frequencies.minFrequencies()`. Die Method muss die minimale Anzahl von Frequenzen bestimmen, welche für die gegebenen Stationen und dem gegebenen Minimalabstand nötig ist. Das heisst, der Rückgabewert muss das kleinste N sein, für welches eine korrekte Implementierung von `Frequencies.assignmentPossible(int N)` `true` zurückgibt.
- (c) (★) Die *Konfliktzahl* einer Station s ist gegeben durch die Anzahl Stationen, welche mit s interferieren könnten, d.h., für welche die Distanz zu s strikt kleiner ist als d . Implementieren Sie die Funktion `Frequencies.mostProblematicStations(int m)`, welche eine nicht-negative ganze Zahl m ($m \geq 0$) als Parameter akzeptiert und die m Stationen mit den grössten Konfliktzahlen als eine Liste vom Typ `List<Coordinate>` zurückgibt. Die zurückgegebene Liste muss absteigend sortiert sein gemäss der Konfliktzahl der Stationen, wobei die Reihenfolge von Stationen mit der gleichen Konfliktzahl keine Rolle spielt. (Wenn s_1 die letzte Station der zurückgegebenen Liste ist, und es eine weitere Station s_2 mit der gleichen Konfliktzahl gibt, welche nicht in der Liste enthalten ist, dann kann s_1 oder s_2 in der Liste zurückgegeben werden.)

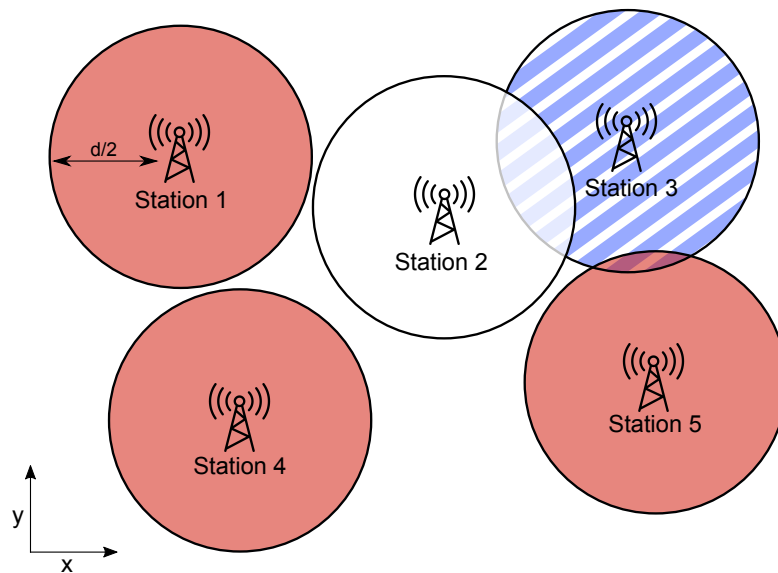
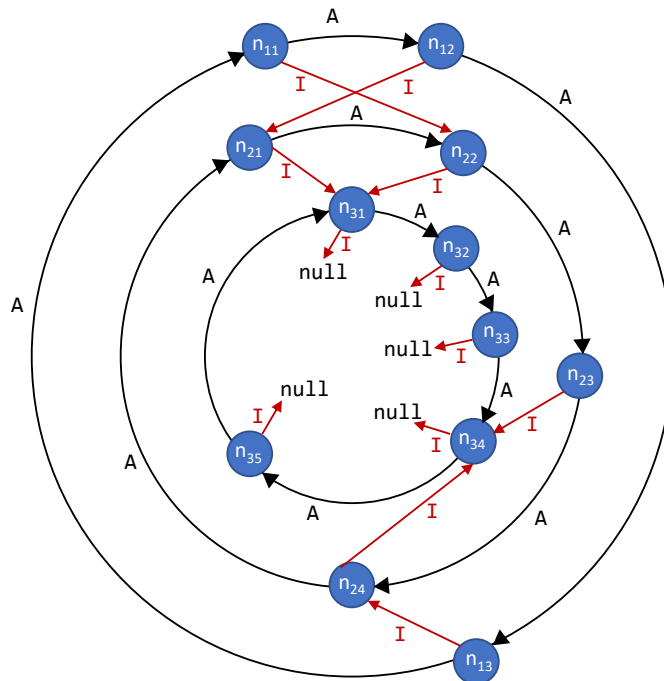


Abbildung 1: Beispiel für eine mögliche Frequenz-Zuordnung mit 3 verfügbaren Frequenzen und 5 Mobilfunk-Stationen. Die Farbe bzw. das Muster gibt die Frequenz an: Station 1, Station 4 und Station 5 senden auf derselben Frequenz. Station 2 und Station 3 senden beide auf einer eigenen Frequenz. *Beachten Sie: In diesem Beispiel wären nicht zwingend 3 Frequenzen nötig. Eine Zuordnung mit nur 2 Frequenzen wäre möglich. Station 2 dürfte auch auf der Frequenz von Station 1, 4 und 5 senden.* Die Stationen 1 und 4 haben Konfliktzahl 0. Die Stationen 2 und 5 haben Konfliktzahl 1. Die Station 3 hat Konfliktzahl 2.

Aufgabe 2 (★★)

Die Klasse `Node` repräsentiert einen Knoten in einem gerichteten Graphen, wobei es für jeden Knoten g höchstens zwei gerichtete Kanten von g zu anderen Knoten f, h geben kann (f, g und h können gleich sein). Wir unterscheiden dabei zwischen der *äusseren* und der *inneren Kante* (und damit dem äusseren und inneren Knoten). Die Methode `Node.getOuter()` (bzw. `Node.getInner()`) gibt den äusseren Knoten (bzw. inneren Knoten) als `Node`-Objekt zurück. Wenn der äussere (bzw. innere) Knoten von n_0 nicht existiert, dann gibt `n_0.getOuter()` (bzw. `n_0.getInner()`) `null` zurück.

In dieser Aufgabe geht es um `Node`-Objekte, welche eine *verschachtelte Kreistopologie* (kurz *VKT*) modellieren. Die folgende Grafik zeigt ein Beispiel für eine VKT, wo die äusseren Kanten (schwarz) mit "A" und die inneren Kanten (rot) mit "I" beschriftet sind.



Eine Menge N von Knoten modelliert eine VKT, wenn N in k ($k > 0$) disjunkte, nicht-leere Teilmengen N_1, \dots, N_k unterteilt werden kann, wobei gilt:

- Die Knoten N_i bilden einen gerichteten Kreis, wenn man nur die äusseren Kanten von N_i in Betracht zieht (für $1 \leq i \leq k$). Das heisst, es gibt einen ausschliesslich aus äusseren Kanten bestehenden Pfad $n_{i1} \rightarrow n_{i2} \dots \rightarrow n_{i|N_i|} \rightarrow n_{i1}$ ($n_{i1}, \dots, n_{i|N_i|} \in N_i$), der n_{i1} genau zweimal und alle anderen Knoten in N_i genau einmal besucht. Für einen Knoten $n_{ij} \in N_i$ nennen wir N_i den *Kreis von n_{ij}* .
- Für $n_{ij} \in N_i$ mit $1 \leq i < k$ gilt, dass der innere Knoten von n_{ij} existiert und in N_{i+1} liegen muss. Knoten in N_k haben keine inneren Knoten.

Die Unterteilung, welche im obigen Beispiel alle Bedingungen für eine VKT erfüllt, ist $N_1 = \{n_{11}, n_{12}, n_{13}\}$, $N_2 = \{n_{21}, n_{22}, n_{23}, n_{24}\}$, $N_3 = \{n_{31}, n_{32}, n_{33}, n_{34}, n_{35}\}$.

- (a) (★) Implementieren Sie die Methode `NestedCycles.cycleDistance(Node n0)`, welche ein `Node`-Objekt einer VKT als Input nimmt und eine Map m von `Node` zu `Integer` zurückgibt. Die Schlüssel von m sollten alle `Node`-Objekte sein, welche auf dem Kreis von n_0 liegen. Für einen Schlüssel n_1 muss $m.get(n_1)$ gleich der Länge des kürzesten Pfades von n_0 zu n_1 sein.

Im Beispiel oben muss `NestedCycles.cycleDistance(n_{12})` eine Map `m` zurückgeben, welche die Schlüssel n_{11} , n_{12} und n_{13} hat und für die gilt: `m.get(n_{11}) = 2`, `m.get(n_{12}) = 0`, `m.get(n_{13}) = 1`.

- (b) (★★) Implementieren Sie die Methode `NestedCycle.isNestedCycle(Node n0)`, welche entscheidet, ob die von `n0` erreichbaren Node-Objekte (`n0` inklusiv) eine VKT modellieren.

Aufgabe 3 (★★★)

In dieser Aufgabe implementieren Sie ein Buchungssystem, in welchem Produkte erstellt, reserviert, ausgeliehen, und nachher zurückgegeben werden können.

Das Buchungssystem verwaltet Produkte X . Wenn ein Produkt ausgeliehen wird, dann wird das Produkt aus X entfernt. Wenn ein über das Buchungssystem ausgeliehenes Produkt zurückgegeben wird, oder wenn ein Produkt für das Buchungssystem neu erstellt wird, dann wird das Produkt zu X hinzugefügt. Produkte werden nicht direkt, sondern über eine Reservierung ausgeliehen. Eine Reservierung nimmt als Argument einen Namen und gibt ein Ticket zurück, über welches ein Produkt mit dem reservierten Namen erhalten werden kann, sobald das Ticket bereit ist. Ein Ticket ist bereit, sobald es ein Produkt mit dem reservierten Namen gibt, dass verfügbar ist (in X enthalten ist) und für welches das Ticket an erster Stelle ist (ein Produkt definiert eine Reihenfolge von Tickets). Dieses existierende Produkt wird dann ausgeliehen. **Wichtig:** Ein Produkt gilt als ausgeliehen (und wird aus X entfernt), sobald das Ticket bereit ist und nicht erst wenn das Produkt über das Ticket erhalten wird. Ein Produkt hat eine Verwaltungsart, welche die Reihenfolge von Tickets definiert. Seien A und B zwei Tickets mit dem gleichen reservierten Namen wie ein Produkt P . Wenn P die Verwaltungsart "fifo" hat (steht für "first in, first out"), dann kommt A vor B , wenn A vor B erstellt wurde.

Wir verwenden eine Klasse, `BookingSystem`, und zwei Interfaces, `Ticket` und `Product`, für jeweils die Buchungssysteme, Reservierungstickets, und Produkte. Die Methode `BookingSystem.createProduct(String productName, String productKind)` erstellt ein Produkt mit entsprechendem Namen und Verwaltungsart, welche als String angegeben wird. Es gibt nur zwei Verwaltungsarten "fifo" und "priority" (definiert in der zweiten Teilaufgabe). Die Methode wirft eine `IllegalArgumentException`, wenn `productKind` weder "fifo" noch "priority" ist. Die Methode `BookingSystem.reserve(String productName, int priority)` erstellt eine Reservierung und gibt ein Ticket zurück. Das Argument `productName` ist der reservierte Name und `priority` ist ein Integer (kann negativ sein), welcher die Dringlichkeit einer Reservierung angibt und ausschliesslich für die Verwaltungsart "priority" relevant ist. Die Methode `Ticket.isReady()` gibt zurück, ob das Ticket bereit ist. Über die Methode `Ticket.getProduct()` wird das Produkt erhalten. `Ticket.getProduct()` soll eine `IllegalArgumentException` werfen, wenn das Ticket noch nicht bereit ist. Nachdem ein Ticket einmal bereit ist, gibt `Ticket.isReady()` immer `true` zurück und `Ticket.getProduct()` immer das reservierte Produkt zurück. Die Methoden `Product.name()` und `Product.kind()` geben Namen und Verwaltungsart von einem Produkt zurück. Die Methode `Product.giveBack()` gibt ein ausgeliehenes Produkt an das Buchungssystem, von welchem das Produkt ausgeliehen wurde, zurück. `Product.giveBack()` wirft eine `IllegalArgumentException`, falls das Produkt nicht gerade ausgeliehen ist.

Die Klassen können wie folgt verwendet werden:

```
BookingSystem booking = new BookingSystem();

// Zwei Reservierungen werden erstellt. Die Produkte sind noch nicht verfuegbar.
Ticket ticket1 = booking.reserve("Tisch", 500);
Ticket ticket2 = booking.reserve("Tisch", 1000);

// Ein Produkt wird dem System zur Verfuegung gestellt.
// Danach ist 'ticket1' bereit und 'ticket2' noch nicht.
booking.createProduct("Tisch", "fifo");
System.out.println("Das erste Ticket ist bereit: " + ticket1.isReady());
System.out.println("Das zweite Ticket ist noch nicht bereit: " + !ticket2.isReady());

// Ein Produkt kann gegen das bereite 'ticket1' eingeloest werden.
Product table = ticket1.getProduct();

// Nachdem das Produkt zurueckgegeben wird, ist 'ticket2' bereit.
table.giveBack();
```

```
System.out.println("Das zweite Ticket ist bereit: " + ticket2.isReady());
```

In der Vorlage finden Sie das Skelett für die eine Klasse und die zwei Interfaces sowie eine Test-Klasse `BookingTest`, welche Sie als Starthilfe für das Testen Ihrer Lösung verwenden können.

- (a) (★) Implementieren Sie alle in der Aufgabenstellung beschriebenen Methoden (`BookingSystem.reserve(String productName, int priority)`, `BookingSystem.createProduct(String productName, String productKind)`, `Ticket.isReady()`, `Ticket.getProduct()`, `Product.name()`, `Product.kind()`, `Product.giveBack()`). Sie dürfen annehmen, dass "fifo" die einzige Verwaltungsart ist und dass es immer nur ein Produkt mit dem gleichen Namen gibt (diese Annahme ist ab der nächsten Teilaufgabe ungültig).
- (b) (★) Ihre Lösung muss damit umgehen können, dass es mehrere Produkte mit dem gleichen Namen gibt. Zusätzlich muss Ihre Lösung mit der Verwaltungsart "priority" umgehen können: Seien A und B zwei Tickets mit dem gleichen reservierten Namen wie ein Produkt P. Wenn P die Verwaltungsart "priority" hat, dann kommt A vor B, wenn das Argument `priority` der Reservierung von A grösser ist als das Argument `priority` der Reservierung von B. Wenn das Argument gleich ist, dann ist A vor B, wenn A zuerst erstellt wurde.
- (c) (★) Implementieren Sie die Methode `BookingSystem.multiReserve(String[] productNames, int priority)`. Die Methode erstellt eine Reservierung für mehrere Produkte, deren Namen durch `productNames` gegeben sind, und gibt dafür ein Ticket zurück. Die Methode wirft eine `IllegalArgumentException`, falls `productNames` null oder eine leere Liste ist.

Wie zuvor wird ein reserviertes Produkt für ein Ticket t ausgeliehen, sobald das Produkt verfügbar ist und t für das Produkt an der Reihe ist. Das von `multiReserve` zurückgegebene Ticket t ist bereit, sobald ein Produkt für jedes Element in `productNames` für t ausgeliehen wurde (dabei spielt der `priority` Parameter die gleiche Rolle für t wie bei Aufgabe b)). **Anders als zuvor, kann bei einer Multi-Reservierung ein Produkt ausgeliehen werden bevor das Ticket bereit ist!** *Beispiel: Angenommen `multiReserve(new String[]{"B", "A"}, 10)` erstellt das Ticket t . Es kann sein, dass t für ein Produkt mit Namen "A" an der Reihe ist (und somit das Produkt für t ausgeliehen wird) bevor t für ein Produkt mit Namen "B" an der Reihe ist.*

Die `Ticket.getProduct()` Methode des zurückgegebenen Tickets iteriert zyklisch über die ausgeliehenen Produkte, wenn das Ticket bereit ist (sonst wird eine `IllegalArgumentException` geworfen wie oben beschrieben): Zum Beispiel, wenn die Reservierung mit `productNames` gleich `new String[]{"B", "A", "B", "C"}` erstellt wurde, dann geben aufeinanderfolgende Aufrufe von `Ticket.getProduct()` die Produkte "B", "A", "B", "C", "B", "A", "B", "C", "B", "A", etc zurück, wobei sich die identischen Produkte nach 4 Aufrufen wiederholen. In diesem Beispiel ist das erste zurückgegebene Produkt unterschiedlich vom dritten, obwohl sie den gleichen Namen "B" haben.

Notizen

Notizen