Basisprüfung Frühjahr 2018
0027 – Einführung in die Programmierung

Departement Informatik
ETH Zürich

9. August 2018 – Programmierprüfung

Nachname:_____ Vorname:_____ Stud.number:_____

Mit Ihrer Unterschrift bestätigen Sie, dass Sie folgende Hinweise zur Kenntnis genommen haben, die Aufgaben selbständig gelöst haben, Sie Ihre eigene Lösung abgeben, Sie keine Kopie der Prüfung mitnehmen, es keine störenden äusseren Einflüsse gab, und Sie keine gesundheitlichen Probleme hatten, die Ihre Leistungen in dieser Prüfung beeinträchtigten.

Signature:_____

Allgemeine Informationen

1. Öffnen Sie diese Prüfung erst, wenn die Aufsicht den Beginn der Prüfung bekannt gibt.
2. Die Prüfung 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 Aufsichtsperson. Sollten Sie durch die Behandlung eines technischen Problems Zeit verlieren, so werden Sie die verlorene Zeit nachholen können.
3. Die Prüfung hat 5 Seiten. Vergewissern Sie sich dass Ihr Exemplar vollständig ist. Sie können die Rückseiten für Skizzen o.ä. benutzen, aber diese werden nicht für die Benotung hinzugezogen.
4. Lesen Sie die Aufgabenstellungen genau durch. Es ist wichtig, dass Ihre Antworten den Anforderungen der Aufgaben *genau* entsprechen.
5. Wir sammeln das unterschriebene Anleitungsblatt und die Aufgabenstellung zum Schluss ein. Wichtig: stellen Sie unbedingt selbst sicher, dass diese Dokumente von einer Aufsichtsperson eingezogen werden. Stecken Sie keine Prüfung ein und lassen Sie Ihre Prüfung nicht einfach am Platz liegen.
6. Nachdem alle Aufgabenstellungen eingezogen worden sind, beginnt der 2. Teil der Prüfung.
7. 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.
8. Wir beantworten keine inhaltlichen Fragen während der Prüfung.

Hinweise zum Programmierteil

1. Eine gut gelöste Aufgabe gibt mehr Punkte als zwei halb gelöste Aufgaben.
2. 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.
3. Stellen Sie regelmässig sicher, dass Ihre Dateien gespeichert sind. Was nicht gespeichert ist, kann nicht bewertet werden.
4. Für jede Aufgabe gibt es ein separates Java-Projekt in Ihrem Eclipse-Workspace.
5. Ändern Sie unter keinen Umständen die Signaturen der im Aufgabentext erwähnten Methoden (Name, Parameter oder Rückgabetyp) oder die Namen der erwähnten Klassen. Solche Änderungen können dazu führen, dass Sie keine Punkte für die entsprechende Aufgabe erhalten. Sie dürfen Methoden oder Klassen zu den vorhandenen hinzufügen.
6. In jedem Projekt gibt es neben dem "src"-Ordner einen "test"-Ordner mit einer einfachen Vorlage für JUnit-Tests. Wenn Sie möchten, können Sie damit Tests für Ihre Lösungen schreiben. **Tests werden nicht bewertet.**
7. Die Prüfungscomputer haben keinen Internet-Zugang. Dadurch kann es in Eclipse zu Fehlermeldungen kommen. Meldungen im Zusammenhang mit fehlendem Internet-Zugang (z.B. "*Code Recommenders cannot download its model repository index*") können Sie ignorieren.

   Die Dokumentation der Java-Klassen ist offline vorhanden. Sie können die "Javadoc"-Ansicht im unteren Teil des Eclipse-Fensters verwenden, um Informationen zu Klassen und Methoden zu erhalten.

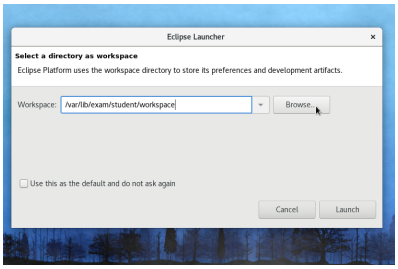
Anmelden und Eclipse starten

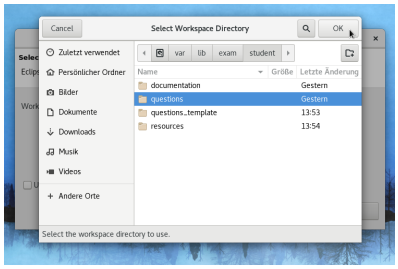
1. Sobald die Prüfung startet, können Sie sich an Ihrem Computer anmelden. Geben Sie zuerst Ihren vollen Namen und im nächsten Schritt Ihren NETHZ-Namen ein.

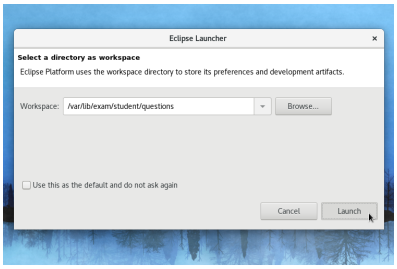
2. Starten Sie Eclipse, indem Sie oben links auf "Aktivitäten" klicken und dann im Suchfeld "Eclipse" eingeben. Wählen Sie "Eclipse" (*nicht* "Eclipse C/C++"). Warten Sie, bis Eclipse gestartet ist. Dies kann einige Minuten in Anspruch nehmen, also nutzen Sie die Zeit, um sich einen Überblick über die Aufgaben zu verschaffen.

3. Wenn sich das Fenster "Eclipse Launcher" öffnet, wählen Sie den Prüfungs-Workspace folgendermassen aus:

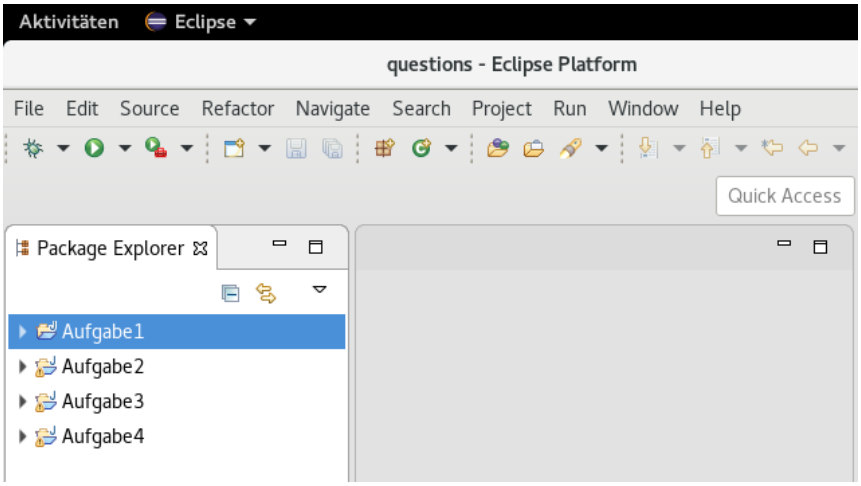
   Klicken Sie auf "Browse..." und wählen Sie dann im Auswahldialog den "questions"-Ordner aus. Klicken Sie oben rechts auf "OK". Im Fenster "Eclipse Launcher" sollte unter "Workspace" jetzt folgender Pfad stehen: "/var/lib/exam/student/questions". Klicken Sie "Launch".







4. Wenn Eclipse fertig gestartet ist, sehen Sie den Willkommens-Bildschirm. Klicken Sie oben rechts auf "Workbench". Nun sollten Sie links die vier Projekte "Aufgabe 1" bis "Aufgabe 4" sehen.



Aufgabe 1

Gegeben seien zwei `int`-Arrays `ms` und `ns` (beide sind nicht `null`). Gesucht ist der Abstand zwischen den beiden Arrays.

Der Abstand ist die Summe der absoluten Differenzen zwischen den zwei Arrays. Um den Abstand zu bestimmen, berechnen Sie also für jede Position i die absolute Differenz der Elemente $|\texttt{ms}[i] - \texttt{ns}[i]|$ und summieren diese auf.

Falls ein Array weniger Elemente hat als das andere, sollen Sie annehmen, dass die fehlenden Elemente 0 sind. Und falls beide Arrays leer sind, ist der Abstand 0.

Beispiel: Für `ms = [1, -2, 0, 3]` und `ns = [1, 4, -5, 3, 7]` sind die absoluten Differenzen $\mathbf{0}$, $|(-2) - 4| = \mathbf{6}$, $|0 - (-5)| = \mathbf{5}$, $\mathbf{0}$ und $|0 - 7| = \mathbf{7}$. Der gesuchte Abstand zwischen `ms` und `ns` ist somit $0 + 6 + 5 + 0 + 7 = \mathbf{18}$.

Vervollständigen Sie die Methode `abstand()` in der Klasse `Abstand` im Projekt "Aufgabe 1", sodass sie den Abstand der beiden Parameter `ms` und `ns` berechnet und als `int` zurückgibt.

Tipp: Das obige Beispiel befindet sich als JUnit-Test in der Datei "AbstandTest.java".

Aufgabe 2

Gegeben sei ein `String S`, der aus n unterschiedlichen Buchstaben besteht (und nicht `null` ist). Das heisst, kein Buchstabe in `S` tritt doppelt oder häufiger auf. Gesucht sind alle 2^n Kombinationen der Buchstaben in `S`.

Beispiel: Für `S = "12a"` sind die 2^3 gesuchten Kombinationen `"" "1" "2" "a" "12" "1a" "2a" "12a"`.

Vervollständigen Sie die Methode `generate()` in der Klasse `Kombinationen` im Projekt "Aufgabe 2", sodass sie die Kombinationen für den `String`-Parameter `input` generiert und diese als `String` zurückgibt.

Der zurückgegebene `String` soll die einzelnen Kombinationen, eingeschlossen in `"` und mit einem Leerzeichen getrennt, aufführen. Die Reihenfolge der Kombinationen ist *nicht wichtig* (jede beliebige Reihenfolge ist akzeptabel), aber die Reihenfolge der Buchstaben innerhalb einer Kombination muss der Reihenfolge in `S` entsprechen. Im obigen Beispiel wird also `"12"` erwartet und `"21"` ist nicht korrekt.

Aufgabe 3

In dieser Aufgabe analysieren Sie die Verkaufsdaten einer Firma. Ein Beispiel für die Verkaufsdaten finden Sie in der Datei "transaktionen.txt". Die Datei enthält eine Liste der Transaktionen, welche die Firma getätigt hat. Für jede Transaktion gibt es genau eine Zeile, in der Sie (durch Leerzeichen getrennt) folgende Informationen finden:

1. Den Namen des Produkts (ein String ohne Leerzeichen, d.h. ein Wort), das in dieser Transaktion verkauft wurde.
2. Den Namen des Kundens (ein String ohne Leerzeichen, d.h. ein Wort), der diese Transaktion tätigte.
3. Die Anzahl der Einheiten, die in dieser Transaktion gehandelt wurden (eine positive ganze Zahl, kann als `int`-Wert gespeichert werden).
4. Der Preis *einer* Einheit in CHF (eine positive reelle Zahl mit 2 Ziffern hinter dem Dezimalpunkt, kann als `double`-Wert gespeichert werden). Der Preis eines Produkts ändert sich kontinuierlich und muss deshalb nicht für alle Transaktionen desselben Produkts gleich sein.

Vervollständigen Sie die Methode `analyse()` in der Klasse `Warenanalyse` im Projekt "Aufgabe 3". Die Methode hat zwei `File`-Parameter. Der erste Parameter bezeichnet die Eingabe-Datei, aus der Sie die Transaktionen lesen sollen, und der zweite die Ausgabe-Datei, in welche Sie die folgenden Informationen schreiben sollen:

1. Den **Namen des Produkts**, welches Teil der grössten Transaktion war, und dessen **Preis** in dieser Transaktion. Die grösste Transaktion ist diejenige, welche für die Firma die grössten Einnahmen erzielte (Anzahl Einheiten $\times$ Preis einer Einheit).
2. Den **Namen des Kunden**, mit dem die Firma die grösste Summe an Einnahmen erzielte, und eben diese **Summe**.
3. Der **Name des Produkts** mit der grössten Spanne zwischen kleinstem und grösstem erzielten Preis (für eine Einheit).

Jede der Informationen 1.–3. soll (in dieser Reihenfolge) auf einer separaten Zeile ausgegeben werden. Trennen Sie Namen und Zahlen innerhalb einer Zeile jeweils durch ein einzelnes Leerzeichen. Zahlen können als ungerundete reelle Zahlen ausgegeben werden. Falls bei einer Information mehrere gleichwertige Ausgaben möglich sind, kann Ihr Programm *eine* beliebige der gleichwertigen ausgeben.

Ihr Programm muss nur korrekt formatierte, nicht-leere Eingabe-Dateien unterstützen. Ein Beispiel einer solchen Datei finden Sie im Projekt unter dem Namen "transaktionen.txt". Exceptions im Zusammenhang mit Ein- und Ausgabe können Sie ignorieren. Für die Beispiel-Datei sollte die Ausgabe wie folgt aussehen:

```
US-Tastatur 24.95
Luigi 2135.05
Cape
```

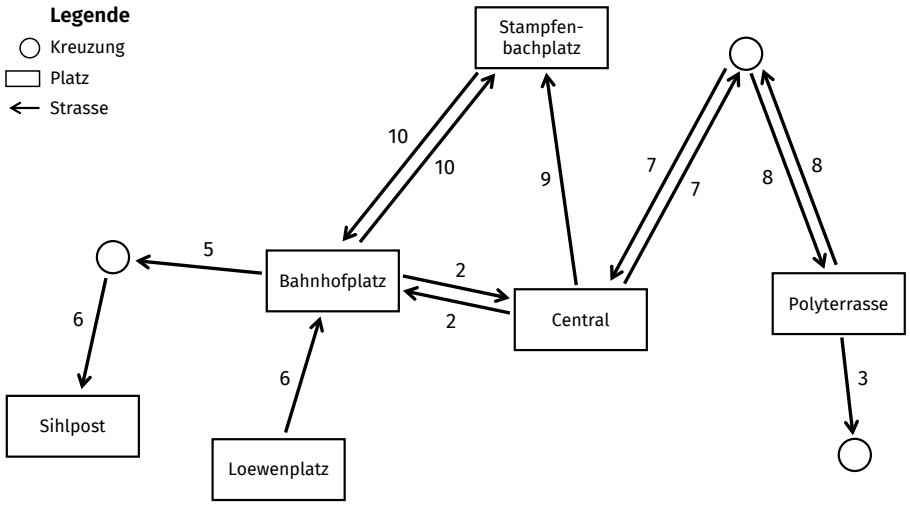


Abbildung 1: Stadt Zürich

Aufgabe 4

In dieser Aufgabe erweitern Sie eine vorgegebene Datenstruktur für eine Stadt. Die gegebenen Klassen befinden sich im Projekt "Aufgabe 4".

Eine Stadt wird durch eine `City`-Instanz repräsentiert. Städte bestehen aus Kreuzungen (siehe Feld `intersections`), die untereinander über Strassen (`streets`) verbunden sind.

Eine Strasse, die von (`from`) einer Kreuzung zu (`to`) einer anderen geht, wird durch eine `Street`-Instanz repräsentiert. Strassen sind nur in der `from`-`to` Richtung befahrbar und sie haben eine positive, ganzzahlige Länge (`length`).

Namenlose Kreuzungen werden durch `Intersection`-Instanzen repräsentiert. Kreuzungen mit Namen, also *Plätze*, werden hingegen durch `Square`-Instanzen repräsentiert. Alle Arten von Kreuzungen kennen die Strassen, die von ihnen wegführen (`outgoingStreets`).

Tipp: Für die nachfolgenden Aufgaben dürfen Sie alle Klassen um weitere Methoden, Konstruktoren und Felder *erweitern*. Vorhandene Methoden, Konstruktoren und Felder dürfen Sie nur ändern, falls ein `TODO`-Kommentar dies erlaubt.

a) In dieser Aufgabe erstellen Sie eine `City`-Instanz von Zürich. Vervollständigen Sie dazu die Methode `build()` in der Klasse `CityBuilder`, sodass sie die `City`-Instanz zurückgibt.

   Erstellen Sie dazu die Instanzen von `City`, `Square`, `Intersection` und `Street`, die für die dargestellte Stadt in Abbildung 1 nötig sind. Wählen Sie für Kreuzungen den passendsten Typ (`Square` oder `Intersection`) und stellen Sie sicher, dass alle Instanzen untereinander richtig verknüpft und den entsprechenden `City`- und `Intersection`-Feldern hinzugefügt wurden.

b) In dieser Aufgabe ermitteln Sie alle Strassen, die Teil einer Sackgasse ("dead end") sind. Eine Strasse ist Teil einer Sackgasse falls die Strasse in einer Kreuzung endet...

   - welche keine wegführende Strassen hat;
   - oder bei welcher alle wegführenden Strassen Teil einer Sackgasse sind.

   Tipp: Dies ist eine rekursive Definition. Sie können die Aufgabe aber auch ohne rekursive Methoden lösen, z.B., indem Sie wiederholt Sackgass-Strassen zu einem `Set` hinzufügen, solange bis Sie keine neuen mehr finden.

Beispiel: In Abbildung 1 sind folgende Strassen Teile von Sackgassen: Die Strasse der Länge 3 von der *Polyterrasse* aus, und die beiden Strassen vom *Bahnhofplatz* über eine Kreuzung nach *Sihlpost*.

Vervollständigen Sie die Methode `deadEnds()` der Klasse `City`, sodass sie ein `Set<Street>` aller solcher Strassen in der Stadt zurückgibt.

c) In dieser Aufgabe ermitteln Sie alle Plätze, die innerhalb einer maximalen Distanz von einem Start-Platz aus erreichbar sind. Vervollständigen Sie dazu die Methode `reachableSquares()` der Klasse `City`, sodass sie ein `Set<Square>` der erreichbaren Plätze zurückgibt. Die Methode nimmt zwei Parameter: `squareName` bestimmt den Start-Platz innerhalb der Stadt; `distance` entspricht der maximalen Entfernung. Der Start-Platz selber ist immer erreichbar.

Ihre Methode soll zuerst sicherstellen, dass ein Platz mit dem gegebenen Namen existiert und dass die gegebene Maximale Entfernung ≥ 0 ist. Andernfalls soll eine `IllegalArgumentException` geworfen werden.

Beispiel: In Abbildung 1 sind die folgenden Plätze vom Start-Platz *Central* innerhalb von Distanz 9 erreichbar: *Central*, *Bahnhofplatz*, *Stampfenbachplatz*. Der *Loewenplatz* ist **nicht** erreichbar, da Strassen nur in eine Richtung befahrbar sind.

Wir wünschen Ihnen alles Gute für den Rest der Prüfungssession und das nächste Semester. Ihr "Einführung in die Programmierung"-Team.