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 6 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. Jede Aufgabe ist mit einer Anzahl von Sternen **(★)** versehen, welche ungefähr den Aufwand und die erreichbare Punktzahl der Aufgabe widerspiegeln. Je mehr Sterne, desto aufwändiger.
3. 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.
4. Stellen Sie regelmässig sicher, dass Ihre Dateien gespeichert sind. Was nicht gespeichert ist, kann nicht bewertet werden.
5. Für jede Aufgabe gibt es ein separates Java-Projekt in Ihrem Eclipse-Workspace.
6. 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 Aufgabe erhalten. Wenn nicht anders vermerkt, dürfen Sie Methoden, Attribute oder Klassen zu den vorhandenen hinzufügen.
7. 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.**
8. Die Prüfungscomputer haben keinen Internet-Zugang. Dadurch kann es in Eclipse zu Fehlermeldungen kommen. Meldungen im Zusammenhang mit fehlendem Internet-Zugang können Sie ignorieren. Ein Beispiel ist die Meldung betreffend “Polling news feeds”, die nach einiger Zeit erscheinen kann.

   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.

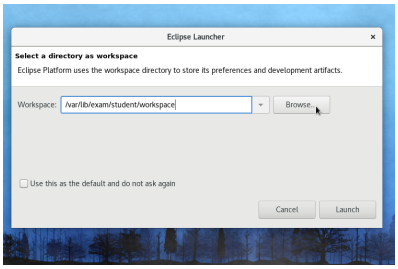
Anmelden und Eclipse starten

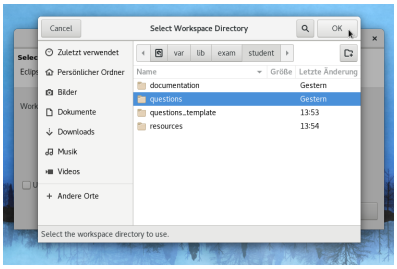
1. Sobald die Prüfung startet, können Sie sich an Ihrem Computer anmelden. Geben Sie zuerst Ihren vollen Namen und im nächsten Schritt Ihren NETHZ-Namen ein.

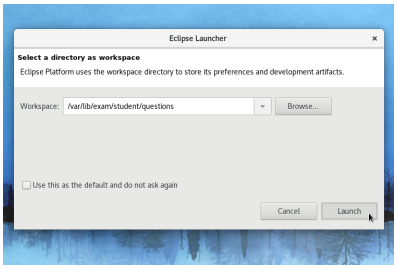
2. Starten Sie Eclipse, indem Sie oben links auf “Aktivitäten” klicken und dann im Suchfeld “Eclipse” eingeben. Wählen Sie “Eclipse” (*nicht* “Eclipse C/C++”). Warten Sie, bis Eclipse gestartet ist. Dies kann einige Minuten in Anspruch nehmen.

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

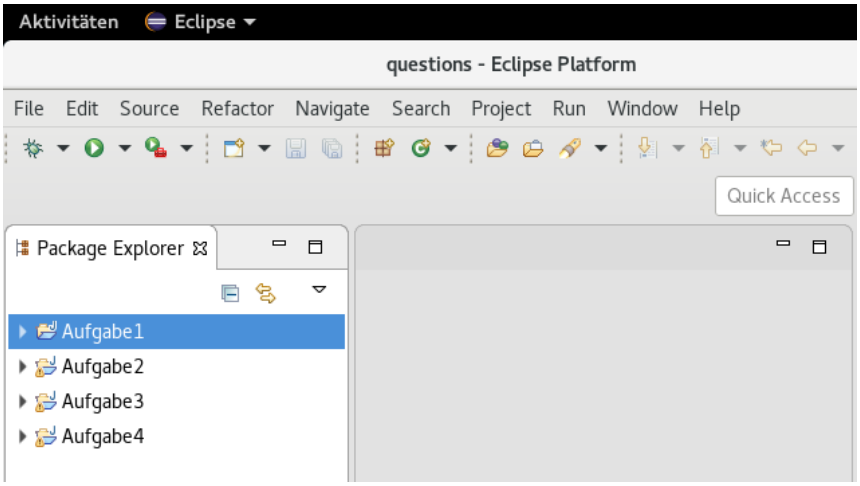
   Klicken Sie auf “Browse...” und wählen Sie dann im Auswahldialog den “questions”-Ordner aus. Klicken Sie oben rechts auf “OK”. Im Fenster “Eclipse Launcher” sollte unter “Workspace” jetzt folgender Pfad stehen: “/var/lib/exam/student/questions”. Klicken Sie “Launch”.







4. Wenn Eclipse fertig gestartet ist, sehen Sie den Willkommens-Bildschirm. Klicken Sie oben rechts auf “Workbench”. Nun sollten Sie links die fünf Projekte “Aufgabe 1” bis “Aufgabe 5” sehen. *Erst wenn alle Studierenden diesen Punkt erreicht haben beginnt die Prüfungszeit. Warten Sie also bis die Prüfung offiziell gestartet wird.*



Aufgabe 1 (★)

Implementieren Sie die Methode `Summe.ohne7()`, die eine positive ganze Zahl N in zwei Summanden $Summand_1$ und $Summand_2$ zerlegt, so dass $Summand_1 + Summand_2 = N$ gilt und die Ziffer 7 weder in $Summand_1$ noch in $Summand_2$ auftritt (in der Dezimaldarstellung).

Die Methode gibt eine Instanz der Klasse `Pair` zurück, deren Attribute entsprechende gesetzt werden müssen.

Beispiel: Für $N = 9734$ ist $Summand_1 = 9634$ und $Summand_2 = 100$ eine mögliche Lösung.

Aufgabe 2 (★★)

Vervollständigen Sie die Methode `Matrix.findWord()`, welche in einer übergebenen Matrix `matrix` von Buchstaben (2-dimensionaler Array von `chars`), ein Wort `word` finden soll. Das Wort `word` kann dabei horizontal in einer Zeile, vertikal in einer Spalte oder auch diagonal vorkommen. Dabei ist die Leserichtung in allen Fällen von links (oben) nach rechts (unten). Der Array `matrix` wird mit Zeile `i` und Spalte `j` indiziert (`matrix[i][j]`, Indizes beginnen links oben mit 0).

Beispiel: In der folgenden Matrix ist das Wort `word="eprog"` genau dreimal zu finden, nämlich an den Startpositionen (0, 1), (0, 2) und (1, 4).

g **e** **e** a p r o g o
o **p** b **p** **e** **p** **r** **o** **g**
r **r** c c **r** c c c e
p **o** d d d **o** d d p
e **g** e e e e **g** e r

`findWord()` soll nun die Positionen der einzelnen Buchstaben in der Matrix als Array von `Position`-Objekten zurückgeben. Die `Position`-Klasse finden Sie im selben Projekt wie die `Matrix`-Klasse. Falls das Wort mehrmals in der Matrix vorkommt, gibt die Methode die Positionen eines beliebigen Fundes zurück. Falls das Wort nicht gefunden werden kann oder `word` leer ist, gibt die Methode `null` zurück. Sie dürfen annehmen, dass keines der Argumente `null` ist.

Für das obige Beispiel kann die Methode also beispielsweise den Array `[(0,2) (1,3) (2,4) (3,5) (4,6)]` zurückgeben. Ein Beispiel finden Sie als JUnit-Test in der Klasse `MatrixTest`.

Aufgabe 3 (★★)

Implementieren Sie die Methode `BestFit.bestFit()`, die aus einer Menge M von Messwerten eine Messreihe R konstruiert, welche einen gewünschten Durchschnittswert D hat. M und D werden der Methode als die Parameter `werte` und `durchschnitt` übergeben. Sie dürfen annehmen, dass `werte` nicht `null` ist. Die Methode soll ein Objekt der Klasse `ArrayList` zurückgeben, welches die Messreihe R enthält. Die Reihenfolge der Messwerte in R spielt keine Rolle.

Gesucht ist eine Messreihe R, konstruiert aus Messwerten von M, deren arithmetisches Mittel exakt dem Wert D entspricht. Die Messwerte dürfen dabei mehrfach in R vorkommen. Allerdings soll die Anzahl *unterschiedlicher* Messwerte minimal sein. Falls mehrere solche R existieren, kann ein beliebiges zurückgegeben werden. Sollte kein R existieren, welches die Bedingungen erfüllt, soll `null` zurückgegeben werden.

Beispiel 1: Für $M = \{3, 4, 7\}$ und $D = 6$ existieren die folgenden Lösungen: $R_1 = [4, 7, 7]$ und $R_2 = [3, 7, 7, 7]$. *Keine* minimale Lösung ist $R = [3, 4, 7, 7, 7, 7, 7]$.

Beispiel 2: Für $M = \{0, 3, 6, 10\}$ und $D = 5$ existieren (unter anderem) folgende Lösungen: $R_1 = [3, 6, 6]$ und $R_2 = [0, 10]$.

Ein Beispiel finden Sie als JUnit-Test in der Klasse `BestFitTest` (falls beim Ausführen des Tests ein “Run As”-Dialog erscheint, wählen Sie dort die Option “JUnit Test”).

Aufgabe 4 (★★★)

Die Klasse `Inspektor` ist zuständig für die Analyse von Logbuch-Einträgen einer grossen Hühnerfarm. Jeder Eintrag beschreibt die Produktivität eines Stalls oder die Tatsache, dass Parasiten in einem Stall festgestellt wurden. Die Log-Einträge werden kontinuierlich (zeilenweise) einer Instanz von `Inspektor` per `eintrag()`-Methode übergeben. Ausserdem können zu jeder Zeit die Methoden `gefahrenStufe()` und `besteStaelle()` aufgerufen werden. `gefahrenStufe()` ermittelt, basierend aus den bisher erhaltenen Einträgen, wie akut das Parasitenproblem auf der Farm ist. Und `besteStaelle()` ermittelt für verschiedene Futterhersteller den produktivsten Stall.

Art der Information	**Datentyp**
Name des Stalls	`String`
Durchschnittliche Anzahl gelegter Eier (pro Huhn & Woche)	`float`
Futtermenge (kg pro Huhn & Woche)	`float`
Futterhersteller	`String`

Abbildung 1: Normaler Log-Eintrag

Ein *normaler* Log-Eintrag ist ein `String` mit den Informationen gezeigt in Abbildung 1 (getrennt durch Leerzeichen, pro Stall höchstens ein solcher Eintrag). Beispiel: `"Stall-1  0.5 0.1667  XYZ"`

Sobald bei einem Stall Parasiten festgestellt werden, wird der `eintrag()`-Methode ein `String` mit dem Stallnamen und der Zahl `-1` übergeben, beispielsweise `"Stall-2 -1"`. Ein solcher Eintrag hat zur Folge, dass dieser Stall definitiv **nicht** für die Berechnung in `besteStaelle()` berücksichtigt wird. Ein solcher *problematischer* Eintrag wird höchstens einmal pro Stall übergeben, aber in beliebiger Reihenfolge mit einem möglichen normalen Eintrag für den selben Stall.

a) (★) Implementieren Sie die Methode `gefahrenStufe()`, welche den Anteil A an parasitenbefallenen Ställen aller gesehenen Ställe berechnet. Darauf basierend gibt die Methode einen `String` zurück, der die Situation einschätzt. Dazu müssen Sie in der `eintrag()`-Methode die nötige Information erfassen (Sie dürfen der `Inspektor`-Klasse Attribute hinzufügen).

   Falls $A \leq 0.2$ gilt oder noch keine Einträge registriert wurden, soll `"OK"` zurückgegeben werden. Andernfalls, wenn $A \leq 0.4$ gilt, ist das Resultat `"EPIDEMIE"`. Ansonsten ist das Resultat `"PANDEMIE"`.

   Das folgende Beispiel zeigt eine mögliche Serie von Methodenaufrufen und die erwarteten Resultate:

```
eintrag("Stall-1  0.5  0.1667 XYZ")
gefahrenStufe() -> "OK"
eintrag("Stall-2  -1")
gefahrenStufe() -> "PANDEMIE"
eintrag("Stall-2  0.7  0.15   ABC")
eintrag("Stall-3  0.7  0.185  ABC")
gefahrenStufe() -> "EPIDEMIE"
```

b) (★★) Implementieren Sie die Methode `besteStaelle()`, welche eine `Map` von Futterhersteller zu Stallname zurückgibt. Die `Map` gibt für alle Hersteller an, welcher Stall mit seinem Futter am produktivsten war (Verhältnis gelegte Eier zu Futtereinsatz ist am grössten). Berücksichtigt werden nur Ställe ohne Parasitenbefall. Falls ein Hersteller nur Ställe mit Parasitenbefall hatte, wird der Hersteller ignoriert.

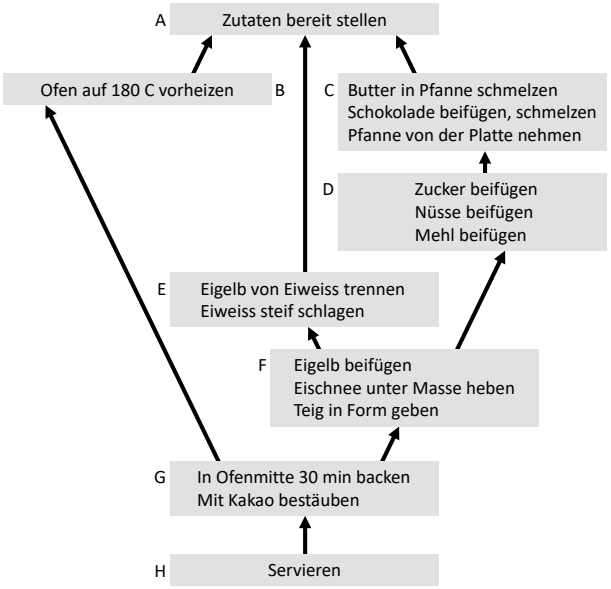
   Beispiel: Eine Serie von Methodenaufrufen und die erwarteten Resultate (`Maps` werden als `{key=value, key=value}` dargestellt):

```
eintrag("Stall-1  0.5  0.1667 XYZ")
besteStaelle() -> {XYZ=Stall-1}
eintrag("Stall-2  0.7  0.15   ABC")
besteStaelle() -> {ABC=Stall-2, XYZ=Stall-1}
eintrag("Stall-2  -1")
eintrag("Stall-3  0.7  0.185  ABC")
besteStaelle() -> {ABC=Stall-3, XYZ=Stall-1}
```

Aufgabe 5 (★★★★)

Die Klasse `Rezept` repräsentiert Kochrezepte, in Form eines gerichteten azyklischen Graphen. Ein Rezept besteht aus einer oder mehreren `Einheiten`, welche jeweils eine Liste von Arbeitsschritten (in Form von `Strings`) und eine Liste von *Vorbedingungen* enthalten. Eine Vorbedingung einer Einheit *e* ist eine andere Einheit, welche vor *e* abgearbeitet werden muss.

Das Bild rechts zeigt ein Rezept für einen Schokokuchen. Jedes Rechteck entspricht einer Einheit und jeder Pfeil einer Vorbedingung. Einheit H hat z.B. genau eine Vorbedingung, nämlich Einheit G. (Die Buchstaben dienen nur der Erklärung und haben keine weitere Bedeutung.)



a) (★) Implementieren Sie die Methode `Rezept.linearisiere(PrintStream)`, welche das Rezept als eine lineare Folge von Einheiten auf den übergebenen `PrintStream` ausgibt, ungefähr so, wie es in einem konventionellen Rezeptbuch stehen könnte. Das heisst, jede Einheit soll erst ausgegeben werden, wenn sämtliche ihrer Vorbedingung bereits ausgegeben worden sind. Ausserdem soll jede Einheit nummeriert sein, wobei die Nummer der Stelle in der linearen Folge entspricht. Zum Beispiel wären für das Schokokuchenrezept u.a. folgende beiden Ausgaben erlaubt:

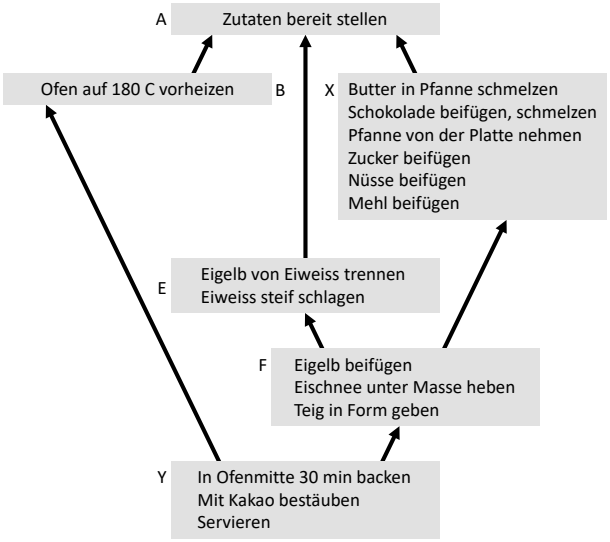
```
1
Zutaten bereit stellen
2
Ofen auf 180 C vorheizen
3
Butter in Pfanne schmelzen
Schokolade beifügen, schmelzen
Pfanne von der Platte nehmen
4
Zucker beifügen
Nüsse beifügen
Mehl beifügen
5
Eigelb von Eiweiss trennen
Eiweiss steif schlagen
6
Eigelb beifügen
Eischnee unter Masse heben
Teig in Form geben
7
In Ofenmitte 30 min backen
Mit Kakao bestäuben
8
Servieren
```

```
1
Zutaten bereit stellen
2
Eigelb von Eiweiss trennen
Eiweiss steif schlagen
3
Butter in Pfanne schmelzen
Schokolade beifügen, schmelzen
Pfanne von der Platte nehmen
4
Zucker beifügen
Nüsse beifügen
Mehl beifügen
5
Ofen auf 180 C vorheizen
6
Eigelb beifügen
Eischnee unter Masse heben
Teig in Form geben
7
In Ofenmitte 30 min backen
Mit Kakao bestäuben
8
Servieren
```

Im Projekt finden Sie eine Klasse `Rezepte`, welche dieses und ein weiteres Rezept konstruiert. Diese können Sie zum Testen verwenden. Achten Sie darauf, dass Sie das Rezept wirklich auf dem übergebenen `PrintStream` und nicht einfach auf `System.out` ausgeben! (Aber für Testzwecke können Sie natürlich `System.out` als Methodenargument übergeben.)

b) (★★★) Implementieren Sie die Methode `Rezept.vereinfache()`. Diese Methode modifiziert das Ziel-Rezept so, dass danach keine Einheiten mehr vorkommen, welche *unnötig voneinander getrennt sind.* Zwei Einheiten e_1 und e_2 sind unnötig voneinander getrennt, wenn e_1 die einzige Vorbedingung von e_2 ist und keine andere Einheit e_1 als Vorbedingung hat. Im Beispiel oben sind einerseits die Einheiten C und D und andererseits G und H unnötig voneinander getrennt.

Die Methode soll also alle Folgen von solchen Einheiten zu einer einzigen Einheit zusammenführen, welche die Schritte aller ursprünglichen Einheiten enthält (in der Reihenfolge entsprechend den Vorbedingungen). Zum Beispiel sollte das Schokokuchenrezept nach dem Aufruf von `vereinfache()` dieser Illustration entsprechen:



Das heisst, Einheiten C und D wurden zur Einheit X zusammengeführt und G und H wurden zu Y zusammengeführt. Alle anderen Einheiten sind unverändert.

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