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. 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. Diese enthalten u.a. die Begriffe "Subversion" und "Code Recommenders".

   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.

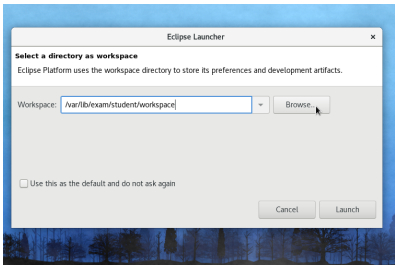
Anmelden und Eclipse starten

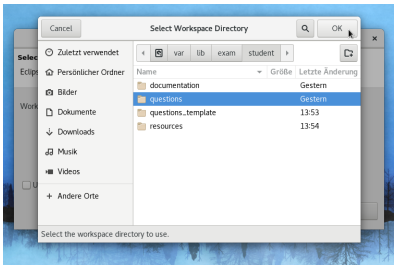
1. Sobald die Prüfung startet, können Sie sich an Ihrem Computer anmelden. Geben Sie zuerst Ihren vollen Namen und im nächsten Schritt Ihren NETHZ-Namen ein.

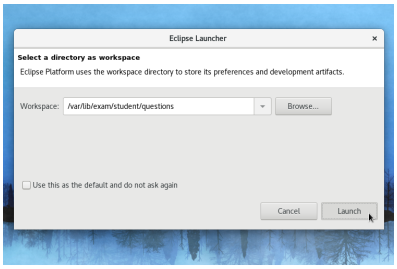
2. Starten Sie Eclipse, indem Sie oben links auf "Aktivitäten" klicken und dann im Suchfeld "Eclipse" eingeben. Wählen Sie "Eclipse" (*nicht* "Eclipse C/C++"). Warten Sie, bis Eclipse gestartet ist. Dies kann einige Minuten in Anspruch nehmen, also nutzen Sie die Zeit, um sich einen Überblick über die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

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

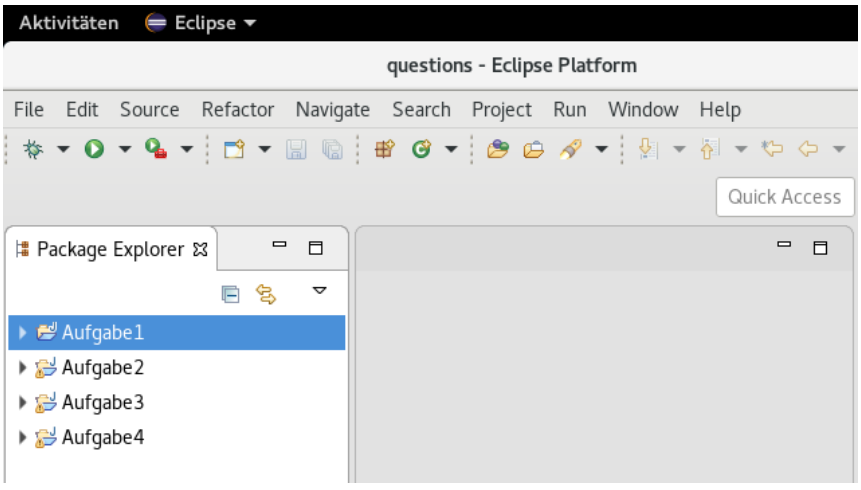
   Klicken Sie auf "Browse..." und wählen Sie dann im Auswahldialog den "questions"-Ordner aus. Klicken Sie oben rechts auf "OK". Im Fenster "Eclipse Launcher" sollte unter "Workspace" jetzt folgender Pfad stehen: "/var/lib/exam/student/questions". Klicken Sie "Launch".







4. Wenn Eclipse fertig gestartet ist, sehen Sie den Willkommens-Bildschirm. Klicken Sie oben rechts auf "Workbench". Nun sollten Sie links die fünf Projekte "Aufgabe 1" bis "Aufgabe 5" sehen.



Aufgabe 1 (★)

Implementieren Sie die Methode `zeroSum()` in der Klasse `ZeroSum`. Diese Methode soll alle Zahlen aus dem übergebenen `Scanner` auslesen und feststellen, ob ein Paar von Zahlen mit den Indizes i und j (mit $i < j$) existiert, für welches die Summe den Wert 0 ergibt. Falls ein solches Paar existiert, soll die Methode ein Objekt der Klasse `Pair` zurückgeben, welches die Indizes i und j der beiden Zahlen enthält; andernfalls soll `null` zurückgegeben werden. Falls es mehrere Paare gibt, welche die Bedingung erfüllen, darf die Methode ein beliebiges solches Paar zurückgeben.

Beispiel: Der Scanner enthält die Zahlen `1 30 42 -7 100 -1`. Das erwartete Resultat ist ein Paar mit $i = 0$ und $j = 5$ (Indizes beginnen wie bei Arrays mit 0).

Sie können davon ausgehen, dass der übergebene `Scanner` nicht `null` ist und die zugrundeliegende Zeichenkette aussschliesslich durch Leerzeichen getrennte ganze Zahlen enthält.

Als Starthilfe für das Testen der Aufgabe finden Sie das obere Beispiel als JUnit-Test in der Klasse `ZeroSumTest`.

Aufgabe 2 (★★)

Vervollständigen Sie die Methode `Blocks.largestBlock()`, welche aus einem String eine $N \times N$-Matrix einliest. Der String enthält $N^2 + 1$ ganze Zahlen (durch Leerzeichen und Zeilen-umbrüche getrennt), wobei die erste Zahl der Grösse N der Matrix entspricht. Die restlichen Zahlen sind die Werte der Matrix, wobei jede Zeile in der Zeichenkette einer Zeile der Matrix entspricht. Die Methode soll in der Matrix die Seitenlänge des grössten quadratischen Blocks finden, welcher aus lauter gleichen Zahlen besteht.

Beispiel: Für den folgenden String lautet das erwartete Resultat 3 (der grösste Block befindet sich in der rechten oberen Ecke und besteht aus lauter 2):

```
7
1 2 3 1 2 2 2
4 5 6 2 2 2 2
8 8 7 6 2 2 2
8 8 9 2 3 0 9
7 8 9 2 4 9 5
4 5 6 2 2 3 2
9 9 9 9 0 3 1
```

Dieses Beispiel finden Sie als JUnit-Test in der Klasse `BlocksTest`.

Aufgabe 3 (★★)

In der Klasse `Nesting` finden Sie die Methode `isCorrectlyNested()`, welche einen `String`-Parameter `s` hat. Die Methode soll prüfen, ob `s` einen korrekten Klammerausdruck enthält. Falls dies so ist, soll die Methode `true` zurückgeben, andernfalls `false`. Ob ein Klammerausdruck korrekt ist, wird durch folgende EBNF-Beschreibung vorgegeben:

$$\textit{nested} \Leftarrow \left\{ \boxed{[}\ \textit{nested}\ \boxed{]} \;\middle|\; \boxed{(}\ \textit{nested}\ \boxed{)} \;\middle|\; \boxed{<}\ \textit{nested}\ \boxed{>} \right\}$$

Falls `s` ein legales Symbol für die Beschreibung *nested* ist, so ist `s` ein korrekter Klammerausdruck.

Beispiel: Für `s="(([]))()"` soll die Methode `true` zurückgeben, für `s="([]"` und `s="["` hingegen `false`.

Implementieren Sie die Methode `isCorrectlyNested()`. Sie können dabei annehmen, dass `s` nicht `null` ist und keine Zeichen ausser `(`, `)`, `[`, `]`, < und > enthält. Als Starthilfe finden Sie einen ersten kleinen JUnit-Test in der Klasse `NestingTest`.

Aufgabe 4 (★★★)

Die Klasse `Service` stellt verschiedene Analysen für Prüfungsergebnisse von S Studierenden zur Verfügung. Die Liste von Ergebnissen besteht aus S Einträgen, also jeweils ein Eintrag pro Student/in. Jeder Eintrag besteht aus einer Zeile und enthält (in dieser Reihenfolge):

1. die Immatrikulationsnummer des Studierenden (ein identifizierender `int`-Wert)
2. drei Noten (drei reelle Zahlen im Bereich von 1.0 bis 6.0, getrennt durch Leerzeichen)

Die drei Noten gehören zu den Fächern *Fach 1*, *Fach 2* und *Fach 3*. Zusätzliche Leerzeilen und -zeichen sollen ignoriert werden. Eine Beispiel für eine Liste für 3 Studierende ist:

```
111111004  5.0  5.0  6.0
111111005  3.75 3.0  4.0
111111006  4.5  2.25 4.0
```

Ihre Aufgabe ist es nun, die `Service`-Klasse und ihre Analysen zu implementieren. Die `Service`-Klasse hat einen Konstruktor, welcher alle Prüfungsergebnisse aus einem `Scanner` auslesen und damit das `Service`-Objekt initialisieren soll. Das Objekt soll so initialisiert werden, dass die vorgegebenen Methoden ihre Analysen durchführen können. Sie dürfen dabei Attribute und zusätzliche Methoden frei bestimmen.

a) **(★)** Implementieren Sie nun die Methode `critical()`, welche die zwei Argumente `bound1` und `bound2` erwartet. Die Methode sucht alle "kritischen" Fälle und gibt eine Liste dieser Studierenden zurück. Die zurückgegebene Liste besteht aus den Immatrikulationsnummern dieser Studierenden (in beliebiger Reihenfolge).

   Ein/e Student/in gilt als kritisch, wenn die Note in *Fach 1* $\leq$ `bound1` und die Summe der Noten für *Fach 2* und *Fach 3* kleiner als `bound2` ist.

   Für das obige Beispiel gäbe `critical(4, 8)` eine Liste mit dem Element `111111005` zurück.

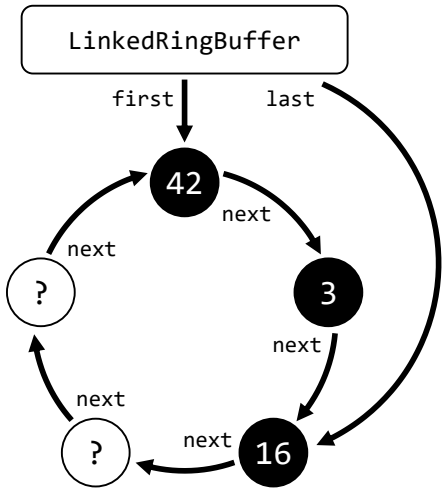
b) **(★★)** Implementieren Sie nun die Methode `top()`, welche die Studierenden mit den besten Ergebnissen zurückgeben soll. Der Parameter `limit` bestimmt die maximale Anzahl der zurückzugebenden Studierenden. Falls weniger Ergebnisse als `limit` existieren, sollen einfach alle gefundenen zurückgegeben werden.

   Der Rückgabewert der Methode ist wieder eine Liste der Immatrikulationsnummern. Diese Liste soll absteigend nach der Leistung sortiert sein (der/die Student/in mit dem besten Ergebnis zuerst). Dabei gilt, dass ein Ergebnis A besser ist als ein Ergebnis B, wenn die Summe aller Noten von A grösser ist als die Summe der Noten von B. Sind die Summen gleich, sind die Ergebnisse gleich gut (und die Reihenfolge in der Liste somit egal).

   Für das obige Beispiel gäbe `top(2)` entweder die Liste `[111111004, 111111006]` oder die Liste `[111111004, 111111005]` zurück (beide wären richtig).

In der Klasse `ServiceTest` finden Sie einen ersten kleinen JUnit-Test als Starthilfe. Ausserdem dürfen Sie folgende Annahmen machen: Der Parameter `limit` ist immer grösser als 0 und die beiden Parameter für `critical()` sind immer im Bereich von 0.0 bis 100.0.

Aufgabe 5 (★★★★)



Hinweis: Zu dieser Aufgabe gibt es eine grössere Sammlung von Unit-Tests, mit welcher Sie Ihre Implementation (und Ihr Verständnis der Aufgabe) zu Beginn testen können. Die Tests decken allerdings nicht die gesamte Funktionalität ab.

In dieser Aufgabe sollen Sie eine verlinkte Datenstruktur schreiben, die eine Warteschlange mit einer festgelegten Kapazität implementiert. Es gibt zwei Hauptmethoden: `enqueue()`, welche ein Element einreiht; und `dequeue()`, welche das *älteste* Element aus der Schlange entfernt und zurückgibt. Die Warteschlange soll als verlinkter Ringspeicher (*ring buffer*) implementiert werden, welcher wie rechts gezeigt aufgebaut ist.

Ein Ringspeicher besteht aus einem `LinkedRingBuffer`-Objekt und aus mehreren `Node`-Objekten, welche die Werte in der Warteschlange speichern und ringförmig verlinkt sind. (Durch die ringförmige Verlinkung werden immer die selben `Node`-Objekte wiederverwendet und niemals neue erstellt.) Das `LinkedRingBuffer`-Objekt speichert zwei Referenzen: `first` zeigt auf den Knoten, der das "erste", d.h. das älteste Element in der Schlange speichert, und `last` zeigt auf den Knoten mit dem "letzten", d.h. dem neuesten Element. Die dargestellte Warteschlange hat eine Kapazität von 5 und enthält die Elemente 42, 3, 16, wobei 42 als erstes Element zurückgegeben würde. Die "?" bedeuten, dass in diesen Knoten zurzeit kein Element gespeichert ist.

Bei `enqueue()` wandert die `last`-Referenz um einen Knoten im Ring weiter und das neue Element wird dort gespeichert. Bei `dequeue()` wird das Element bei `first` zurückgegeben und `first` wandert um einen Knoten weiter. Falls die Schlange voll ist, d.h. die Anzahl der Elemente gleich der Kapazität ist, schlägt `enqueue()` fehl; ebenso `dequeue()`, falls die Schlange leer ist.

Achtung: In dieser Aufgabe sind folgende Änderungen nicht erlaubt: das Hinzufügen von Feldern (Attributen) oder Konstruktoren in `LinkedRingBuffer` oder in `Node`, das Ändern der Superklasse (`Object`) dieser beiden Klassen und das Erstellen von zusätzlichen Klassen oder Interfaces.

a) **(★)** Implementieren Sie den Konstruktor `LinkedRingBuffer(int)`, welcher die Kapazität als Argument entgegen nimmt und die Ringstruktur aufbaut. Der Konstruktor soll eine Exception vom Typ `java.lang.IllegalArgumentException` werfen, falls eine Kapazität < 1 übergeben wird. (Ihre Implementation soll beliebige Kapazitäten ≥ 1 unterstützen.)

   Vervollständigen Sie ebenfalls die `capacity()`-Methode, welche anhand der Ringstruktur die Kapazität berechnet und zurückgibt.

   Hinweis: Am Ende des Konstruktors ist die Warteschlange initialisiert und leer. Finden Sie einen Weg, eine leere Warteschlange zu repräsentieren.

b) **(★★★)** Implementieren Sie `enqueue()` und `dequeue()` wie oben beschrieben. In einem Ausnahmefall sollen beide Methoden eine `java.lang.IllegalStateException` werfen.

   Füllen Sie zusätzlich die `length()`-Methode aus, welche die momentane Länge der Warteschlange zurückgibt, d.h. die Anzahl Elemente, die sich in der Schlange befinden.

Hinweis: Sie können für b) nur die volle Punktzahl erhalten, wenn Sie Teile von a) gelöst haben.

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