Lab-Exam #i215: P1 (bzw. **PT**) Shf 21/07/05

Sie dürfen diesen Aufgabenzettel am Ende der Prüfung mitnehmen.

Formales:

Hinweise:

- Dauer der Prüfung: 120 Minuten.
- Legen Sie Lichtbild- und Studentenausweis gut sichtbar auf den Tisch.
- Sie müssen alle Aufgaben alleine lösen. Sie dürfen während der gesamten Prüfung ausschließlich mit der Aufsicht kommunizieren. Kommunikation zu anderen - in welcher Form auch immer ist untersagt. Die Rechner werden überwacht und die Zugriffe protokolliert.
  Alle Zugriffe werden aufgezeichnet. Verschiedene Sniffer sind aktiv! Eine automatisierte Überwachung ist während der Prüfungsdauer aktiviert.
- Beachten Sie insbesondere, dass Ihr Programm bei der Abgabe funktionstüchtig sein muss. Programme, die nicht laufen, werden als ungenügend gewertet. Die Punkte werden entsprechend des korrekt implementierten Funktionsumfanges vergeben.
  Tipp: Sichern Sie daher lauffähige Zwischenstände.
- Die eingeforderten Klassen müssen vorhanden sein und die jeweils geforderten Variablen und Methoden darin enthalten. Sie können weitere eigene Klassen, Methoden und Variablen hinzufügen – vielleicht ist dies sogar nötig.
- Lösen Sie Ihre Aufgaben in den voreingestellten Eclipse-Projekten/Directories. Legen Sie **keine(!)** eigenen Projekte an(!). Eclipse finden Sie in der aus den Labor-Übungen gewohnten Standard-Einstellung der HAW vor.

Achtung
Das "Abgabetool" mag keine Sonderzeichen (also z.B. Umlaute, Leerzeichen oder "ß") in den Dateinamen. Meiden Sie diese Zeichen insbesondere bei allen Arten von Dateien bzw. konvertieren Sie diese entsprechend. Für Buchstaben dürfen Sie nur die 26 Buchstaben 'a' bis 'z' bzw. 'A' bis 'Z' (also die Buchstaben des modernen lateinischen Alphabets) verwenden.

Wenn Sie in Aufgabe a1 dazu aufgefordert werden Ihre Nach- und Vor-Namen zu schreiben, zu nutzen oder als Ergebnis abzuliefern, dann müssen Sie dies immer in konsistenter Weise tun. Ferner muss sowohl Ihr Nach- als auch Vor-Name ausschließlich aus Kleinbuchstaben(!) bestehen und diese Kleinbuchstaben sind dem "modernen" lateinischen Alphabet zu entnehmen - also nur 'a' bis 'z'. Sollten Sie mehr als einen Vornamen oder Nachnamen haben, nehmen Sie denjenigen mit dem Sie auch bei der HAW (zuerst) geführt werden. Ferner sind mögliche Namenszusätze (wie z.B. "von", "van", "ten" oder "Mc") hinten anzustellen. Falls der volle Name gefordert ist, dann ist die korrekte Darstellung der Nachname gefolgt von einem Underscore und dann gefolgt vom Vornamen.

Beispiele:

Sie heißen Max Mustermann.
Max sei Ihr Vorname / First Name / Given Name und wird als **`max`** dargestellt.
Mustermann sei Ihr Nachname / Familienname / Surname / Family Name / Last Name und wird als **`mustermann`** dargestellt.
Dann ist die korrekte Darstellung des vollen Namens: **`mustermann_max`**

Sie heißen André-Ëvïno von Croÿ-Gräßler.
André-Ëvïno sei Ihr Vorname / First Name / Given Name und wird als **`andreevino`** dargestellt.
von Croÿ-Gräßler sei Ihr Nachname / Familienname / Surname / Family Name / Last Name und wird als **`croygraesslervon`** dargestellt.
Dann ist die korrekte Darstellung des vollen Namens: **`croygraesslervon_andreevino`**

Tipp
Sollten Sie irgendwelche Unsicherheiten haben, wie Ihr Vorname oder wie Ihr Nachname lautet, dann schauen Sie doch einfach auf Ihren Studentenausweis, den Sie ja zur Prüfung mitbringen müssen.

Allgemeingültige Anforderungen für alle Aufgaben:

- Es ist bereits ein Eclipse-Projekt voreingestellt. Die Prüfung ist in diesem voreingestellten Projekt zu absolvieren.
- Die Aufgabe a1 ist zwingend zuerst zu lösen. Nachfolgende Aufgaben dürfen erst nach erfolgreicher Lösung von a1 begonnen werden.
- Sofern nicht ausdrücklich anders eingefordert,
  - sind die Accessmodifier für Methoden, Konstruktoren (und alle anderen Java-"Dinge") sinnvoll zu wählen
  - müssen alle Zustandsvariablen private sein
- Sie dürfen von Ihnen zu implementierende Klassen oder Interfaces um zusätzliche Bestandteile ergänzen (alles Eingeforderte muss weiterhin exakt erfüllt sein). U.U. ist das Ergänzen eigener "Dinge" sogar zur Lösung zwingend erforderlich.
- Die Reihenfolge der Parameter in den eingeforderten Parameterlisten ist unbedingt zu beachten. Der jeweilige TestFrame und die gegebenen Interfaces/Klassen fordern dies auch so ein.
- Sofern eine **`toString()`**-Methode eingefordert wird und es nicht ausdrücklich anders verlangt wird, dürfen die Ergebnis-Strings der Methode **`toString()`** keinen Zeilenumbruch enthalten und müssen die Werte aller Variablen und den Klassennamen des konkreten Objekts enthalten. Ferner muss klar erkennbar sein zu welcher Klasse die jeweiligen Variablen bzw. deren Wert gehören.
- Sofern Sie Daten als Ergebnisse "abliefern" müssen, reicht es die Originale abzuliefern. *Sie müssen die Ergebnis-Daten nicht klonen. Dies ist eine Vereinfachung, da die zugehörige Thematik nicht Inhalt der Veranstaltung war - wer dennoch gerne klonen möchte, der darf es gern tun.*

Kurzübersicht über die Aufgaben:

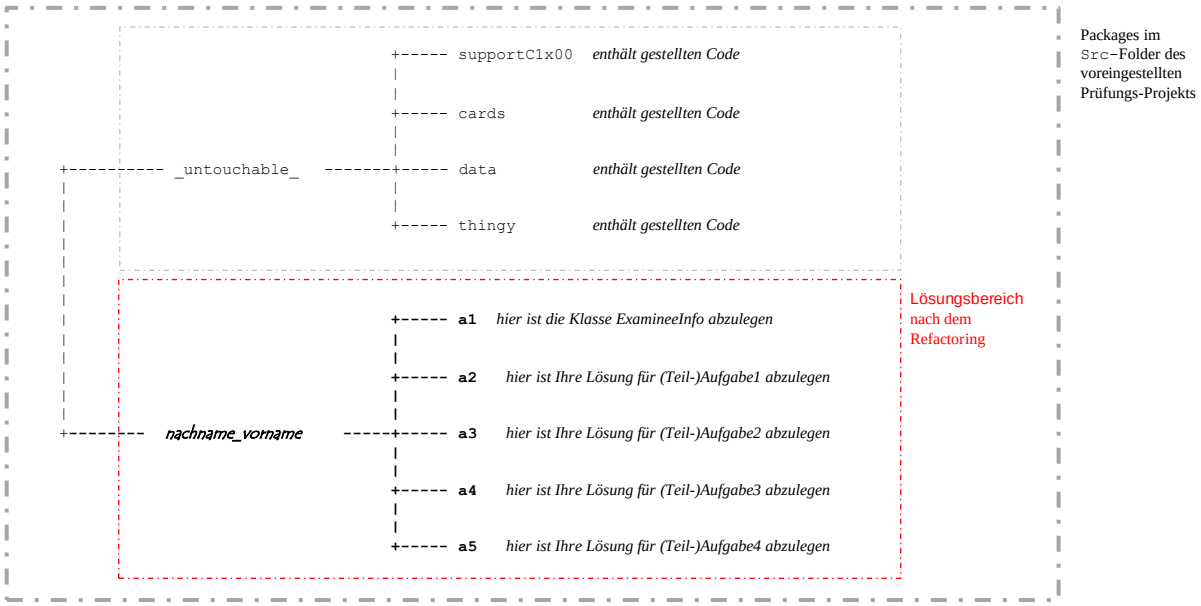
a1	ExamineeInfo - Grundvoraussetzung für alle anderen/nachfolgenden Aufgaben	
a2	WordProcessor	(U.a. Thema: String-Verarbeitung)
a3	NumberPalindromeChecker	(U.a. Thema: BigInteger, Algorithmus)
a4	PrimeFactorComputer	(U.a. Thema: BigInteger, Algorithmus)
a5	ItemProcessor	(U.a. Thema: Collections)

Die Aufgaben a2, a3, a4 und a5 werden unabhängig gewertet. Sie dürfen diese in beliebiger Reihenfolge bearbeiten. Die Idee ist, dass für 15NP alles bzw. jede Aufgabe vollständig und erfolgreich gelöst werden muss.

Grundvoraussetzung für alle nachfolgenden Aufgaben ist das Lösen der Aufgabe a1 !

Aufgabe a1:

Erzeugen Sie ein Package, das Ihren Namen trägt und dem Schema *nachname_vorname* genügt.
Entweder benennen Sie hierfür das gestellte Package **`template_raw`** mit Refactoring entsprechend um oder Sie erzeugen ein neues Package gemäß den Anforderungen und kopieren die Inhalte von `template_raw` dort hinein.
Der Package-Name hat ausschließlich aus Kleinbuchstaben ("a"-"z") und dem Unterstrich ("_"), der Nachname und Vorname trennt, zu bestehen. Sollten Ihr Name beispielsweise "André-Ëvïno von Croÿ-Gräßler" lauten, dann muss der Package-Name **`croygraesslervon_andreevino`** lauten.
Dieses Package dient als Lösungsbereich. **Alle folgenden (Teil-)Aufgaben sind im Lösungsbereich in unterschiedlichen Sub-Packages (konkret a1 bis a5) zu lösen.**
Das folgende Beispiel zeigt einen exemplarischen Aufbau des Projekts
(nach einem exemplarischen Refactoring von **`template_raw`**):



- Im Package _untouchable_ finden Sie gegebenen Code vor. Der dort abgelegte Code ist in Sub-Packages organisiert und darf nicht verändert werden. Der im Package _untouchable_ abgelegte Code wird bei der Korrektur wieder durch den original Code überschrieben.

Auf der nächsten Seite folgt der Auftrag **`ExamineeInfo_I`** zu implementieren als Teil der Aufgabe a1.

Lösen Sie nun die folgende Aufgabe im vorgegebenen Sub-Package **a1**.
Die im vorgegebenen **TestFrameC1x00** enthaltene Sammlung von Tests soll Ihnen nur die Sicherheit vermitteln, dass Sie die Aufgabe richtig verstanden haben. Dass von den Tests dieser Testsammlung keine Fehler gefunden wurden, kann nicht als Beweis dienen, dass Ihre Lösung fehlerfrei ist. Es liegt in Ihrer Verantwortung sicher zu stellen, dass Sie fehlerfreien Code geschrieben haben. Bei der Bewertung werden u.U. andere - konkret: modifizierte und "härtere" Tests - verwendet.

Start der eigentlichen Aufgabe:

Implementieren Sie eine Klasse **ExamineeInfo**, die sich von **ExamineeInfo_I** ableitet und die

- einen parameterlosen Konstruktor unterstützt.
- (u.a. auch) die folgenden Elemente aufweist:

  String getExamineeSurName()

  liefert **Ihren** Nachnamen in Kleinbuchstaben. *Dieser Nachname muss den im Formal-Teil vorgestellten Anforderungen genügen. (D.h. es dürfen nur Kleinbuchstaben verwendet werden und diese dürfen nur dem "modernen" lateinischen Alphabet entnommen werden).*

  String getExamineeFirstName()

  liefert **Ihren** Vornamen in Kleinbuchstaben. *Dieser Vorname muss den im Formal-Teil vorgestellten Anforderungen genügen. (D.h. es dürfen nur Kleinbuchstaben verwendet werden und diese dürfen nur dem "modernen" lateinischen Alphabet entnommen werden).*

Aufgabe a2:

Lösen Sie diese Aufgabe im vorgegebenen Sub-Package **a2**.
Die im vorgegebenen **TestFrameC1x00** enthaltene Sammlung von Tests soll Ihnen nur die Sicherheit vermitteln, dass Sie die Aufgabe richtig verstanden haben. Das von den Tests dieser Testsammlung keine Fehler gefunden wurden, kann nicht als Beweis dienen, dass Ihre Lösung fehlerfrei ist. Es liegt in Ihrer Verantwortung sicherzustellen, dass Sie fehlerfreien Code geschrieben haben. Bei der Bewertung werden u.U. andere - konkret: modifizierte und "härtere" Tests - verwendet.

Start der eigentlichen Aufgabe:

Implementieren Sie eine Klasse **WordProcessor**, die sich von **WordProcessor_I** ableitet und die Analyse von Wörtern unterstützt.
Die Klasse **WordProcessor** soll u.a. die folgenden Elemente aufweisen:

WordProcessor(String)
erzeugt einen WordProzessor. Das als Parameter übergebene Wort wird als zu untersuchendes Wort gesetzt.
Der Konstruktor darf nur "gutartige" Werte als Parameter akzeptieren. Die akzeptierten Parameter müssen der folgenden Eigenschaft genügen:
- Der Parameter darf nicht null sein.

WordProcessor()
erzeugt einen WordProzessor.
Das zu untersuchende Wort ist vom parameterlosen Konstruktor auf einen leeren String zu setzen.

String getOriginalWord()
liefert das zu untersuchende Wort in seinem Original-Zustand. (ToDo: Tests)

String getCurrentWord()
liefert das zu untersuchende Wort in seinem aktuellen Zustand.

void setWord(String)
setzt das zu untersuchende Wort auf das als Parameter übergebene Wort.
Die Methode darf nur "gutartige" Werte als Parameter akzeptieren. Die akzeptierten Parameter müssen der folgenden Eigenschaft genügen:
- Der Parameter darf nicht null sein.

boolean isNeoLatin()
prüft, ob das zu untersuchende Wort nur aus Buchstaben des modernen lateinischen Alphabets besteht.
Als Rückgabewert soll **true** geliefert werden, falls das zu untersuchende Wort ausschließlich aus Buchstaben des modernen lateinischen Alphabets besteht, sonst **false**.

int countVocals()
liefert die Anzahl der im Wort enthaltenen Vokale (keine Umlaute, nur Vokale des modernen lateinischen Alphabets)

void removeVocals()
entfernt alle Vokale aus dem zu untersuchenden Wort.

Tipp:
Sie haben Zugriff auf die Java-API.
U.U. haben Sie die benötigten "Dinge" nicht konkret im Gedächtnis. Aber Sie sollten diese grundsätzlich kennen. Schauen Sie für nötige Details in der Java-API nach ;-)

Aufgabe a3:

Lösen Sie diese Aufgabe im vorgegebenen Sub-Package **a3**.
Die im vorgegebenen **TestFrame**C1x00 enthaltene Sammlung von Tests soll Ihnen nur die Sicherheit vermitteln, dass Sie die Aufgabe richtig verstanden haben. Das von den Tests dieser Testsammlung keine Fehler gefunden wurden, kann nicht als Beweis dienen, dass Ihre Lösung fehlerfrei ist. Es liegt in Ihrer Verantwortung sicherzustellen, dass Sie fehlerfreien Code geschrieben haben. Bei der Bewertung werden u.U. andere - konkret: modifizierte und "härtere" Tests - verwendet.

Start der eigentlichen Aufgabe:

Implementieren Sie eine Klasse **NumberPalindromeChecker**, die sich von **NumberPalindromeChecker_I** ableitet und das Erkennen von Zahlenpalindromen unterstützt.
Vergleichbar einem Wortpalindrom, dass unabhängig von der Leserichtung (links nach rechts oder rechts nach links) das gleiche Wort ist, gilt für ein Zahlenpalindrom, dass die Zahl (im Dezimalsystem dargestellt) unabhängig von der Leserichtung die gleiche Zahl ist.

Beispiele:
- 8 ist ein Zahlenpalindrom.
- 99 ist ein Zahlenpalindrom.
- 121 ist ein Zahlenpalindrom.
- 7557 ist ein Zahlenpalindrom.
- 42 ist kein Zahlenpalindrom.
- 123 ist kein Zahlenpalindrom.
- 1223 ist kein Zahlenpalindrom.

Die Klasse **NumberPalindromeChecker** soll u.a. die folgenden Elemente aufweisen:

```
NumberPalindromChecker()
```
erzeugt einen NumberPalindromChecker.

```
boolean isNumberPalindrom( BigInteger )
```
überprüft, ob die als Parameter übergebene Zahl ein Zahlenpalindrom ist. **null** als Parameter oder eine negative Zahl ist unzulässig. Falls ein Zahlenpalindrom vorliegt ist der Rückgabewert **true**, andernfalls **false**.

Aufgabe a4:

Lösen Sie diese Aufgabe im vorgegebenen Sub-Package **a4**.
Die im vorgegebenen **TestFrame**C1x00 enthaltene Sammlung von Tests soll Ihnen nur die Sicherheit vermitteln, dass Sie die Aufgabe richtig verstanden haben. Das von den Tests dieser Testsammlung keine Fehler gefunden wurden, kann nicht als Beweis dienen, dass Ihre Lösung fehlerfrei ist. Es liegt in Ihrer Verantwortung sicherzustellen, dass Sie fehlerfreien Code geschrieben haben. Bei der Bewertung werden u.U. andere - konkret: modifizierte und "härtere" Tests - verwendet.

Start der eigentlichen Aufgabe:

Implementieren Sie eine Klasse **PrimeFactorComputer**, die sich von **PrimeFactorComputer_I** ableitet und das Zerlegen einer Zahl in seine Primfaktoren unterstützt.

```
PrimeFactorComputer()
```
erzeugt einen PrimeFactorComputer.

```
List<BigInteger> compute( BigInteger )
```
zerlegt die als Parameter übergebene Zahl in seine Primfaktoren. **null** als Parameter oder eine Zahl kleiner 2 ist unzulässig. Die als Ergebnis abgelieferte Liste von Primfaktoren ist in aufsteigender Reihenfolge sortiert.
(Achtung! Der zugehörige Performance-Test `tol_4s_performance_check_no1_about10secondExpected()` könnte länger laufen – je nach Rechner bis zu 10 Sekunden)

Tipp:
Sie haben Zugriff auf die Java-API.
U.U. haben Sie die benötigten "Dinge" nicht konkret im Gedächtnis. Aber Sie sollten diese grundsätzlich kennen. Schauen Sie für nötige Details in der Java-API nach.

Aufgabe a5:

Lösen Sie diese Aufgabe im vorgegebenen Sub-Package **a5**.
Die im vorgegebenen **TestFrameC1x00** enthaltene Sammlung von Tests soll Ihnen nur die Sicherheit vermitteln, dass Sie die Aufgabe richtig verstanden haben. Das von den Tests dieser Testsammlung keine Fehler gefunden wurden, kann nicht als Beweis dienen, dass Ihre Lösung fehlerfrei ist. Es liegt in Ihrer Verantwortung sicherzustellen, dass Sie fehlerfreien Code geschrieben haben. Bei der Bewertung werden u.U. andere - konkret: modifizierte und "härtere" Tests - verwendet.

Die Klasse **Item** sowie die zugehörigen enums (sofern benötigt) importieren Sie von/aus: **_untouchable_.thingy**

Start der eigentlichen Aufgabe:

Implementieren Sie eine Klasse **ItemProcessor**, die sich von **ItemProcessor_I** ableitet und die **Items** verarbeitet. Die Idee ist: Items gemäß den vorgegebenen Anforderungen zu sammeln. Die Items haben Eigenschaften. Diese Eigenschaften sind:

- Farbe — Sie haben Zugriff auf dieses Eigenschaft mit: **Color getColor()**
- Größe — Sie haben Zugriff auf dieses Eigenschaft mit: **Size getSize()**
- Gewicht — Sie haben Zugriff auf dieses Eigenschaft mit: **Weight getWeight()**
- Wert — Sie haben Zugriff auf dieses Eigenschaft mit: **Long getValue()**

Alles weitere müssen **Sie selbst** den gestellten Referenztypen `Color`, `Item`, `Size`, `Weight` entnehmen.

Ein **ItemProcessor** soll einkommende Items sammeln. Immer sobald fünf Items gleichen Werts vorliegen, sollen diese als Quintet "abgeliefert" werden (wobei die jeweilige Eintreffreihenfolge der Items im Bestand des ItemProcessors erhalten bleiben soll).

Das Interface **Quintet_I** beschreibt eine Zusammenstellung von fünf Items. Auf die zugehörigen Items kann mit **getItem()** zugegriffen werden. In einem Quintet sind fünf Items gleichen Werts entsprechend der jeweiligen Eintreffreihenfolge im Bestand des ItemProcessors zusammengefasst.

Item getItem(int)
liefert das zugehörige Item. Der Parameter von getItem() ist ein "Appearance related Index". In der Konsequenz liefert getItem(0) das älteste[1] Item des Quintets, getItem(1) das zweitälteste Item des Quintets, … und schließlich getItem(4) des jüngste Item des Quintets. Die jeweiligen Items verbleiben im Quintet.

[1] Das Alter bezieht auf das jeweilige Eintreffen in der Sammlung bzw. der jeweiligen Aufnahme in den Bestand.

Die Klasse **`ItemProcessor`** soll das Interface **`ItemProcessor_I`** unterstützen und die folgenden Elemente aufweisen:

`ItemProcessor()`
erzeugt einen ItemProcessor.

`Quintet_I process( Item )`
verarbeitet ein Item. Das als Parameter übergebene Item wird dem Bestand (also den "bisher übergebenen" Items) hinzugefügt.
Immer wenn fünf Items gleichen Werts (einschließlich des gerade als Parameter übergebenen Items) im Bestand sind, sollen diese fünf Items als Rückgabewert der Methode (in Form eines Quintets) abgeliefert und aus dem Bestand entfernt werden – andernfalls soll **`null`** zurückgegeben werden.
Die Methode process() darf nur "gutartige" Werte als Parameter akzeptieren. Die akzeptierten Parameter müssen der folgenden Eigenschaft genügen:

- Der Parameter darf nicht null sein.

Bemerkung:

- **Es wird zugesichert**, dass ein und dasselbe Item solange es korrekter Weise im Bestand ist, nicht noch einmal mit process() an den ItemProcessor übergeben wird.
- Ein und dasselbe Item, das zwar im Bestand war, aber korrekter Weise nicht mehr im aktuellen Bestand ist, kann mit process() dem ItemProcessor übergeben werden.
- Doppelte bzw. gleiche Items können jedoch beliebig oft auftreten.

`int itemsProcessed()`
liefert die Anzahl der Items, die vom ItemProcessor verarbeitet wurden (oder in anderen Worten: Die Anzahl der Aufrufe von process()), seit dem letzten Aufruf von reset() bzw. sofern kein reset() aufgerufen wurde, seit dem Programmstart.

`int quintetsFound()`
liefert die Anzahl der "Treffer" (oder in anderen Worten: Wie oft der ItemProcessor eine Zusammenstellung von fünf Items gleichen Werts abliefern konnte) , seit dem letzten Aufruf von reset() bzw. sofern kein reset() aufgerufen wurde, seit dem Programmstart.

`void reset()`
setzt den ItemProcessor auf seinen Startzustand zurück (und löscht insbesondere den Bestand).
(Achtung! Der zugehörige Test `tol_4s_behavior_reset_no1()` könnte länger laufen – je nach Rechner bis zu 10 Sekunden)