Lab-Exam #1317: P1 (bzw. **PT**) Shf 23/07/05

Sie dürfen diesen Aufgabenzettel am Ende der Prüfung mitnehmen.

Formales:

Hinweise:

- Dauer der Prüfung: 180 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 dies nicht ausdrücklich anders verlangt wird,
  - müssen alle eingeforderten Referenztypen, Konstruktoren und Methoden **`public`** sein
  - müssen alle Zustandsvariablen **`private`** sein
  - sind die Accessmodifier für alle anderen Java-"Dinge" sinnvoll zu wählen
- 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	ArrayProcessor	(U.a. Thema: Array & Referenzen)
a3	ResultReporter	(U.a. Thema: Collections)
a4	ItemProcessor	(U.a. Thema: Collections)
a5	Stack	(U.a. Thema: Dynamische Datenstrukturen / "Knoten")

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 **`TestFrameAndStarter`**`C1x00` 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 **TestFrameAndStarterC1x00** 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 **ArrayProcessor**,

- die sich von **ArrayProcessor_I** ableitet und
- ein als Parameter an den Konstruktor übergebenes "quasi 2-dimensionales Array" entgegennimmt und
- die kontinuierliche Erweiterung des Arrays um Summen der intern Grundwerte unterstützt (s.u.), wobei das jeweilige Ergebnis zur Grundlage für weitere Operationen wird (außer natürlich bei getOriginalArray()) sowie
- das Array in seinem aktuellen als auch ursprünglichen Original-/Ausgangs-Zustand abliefern kann (s.u.).

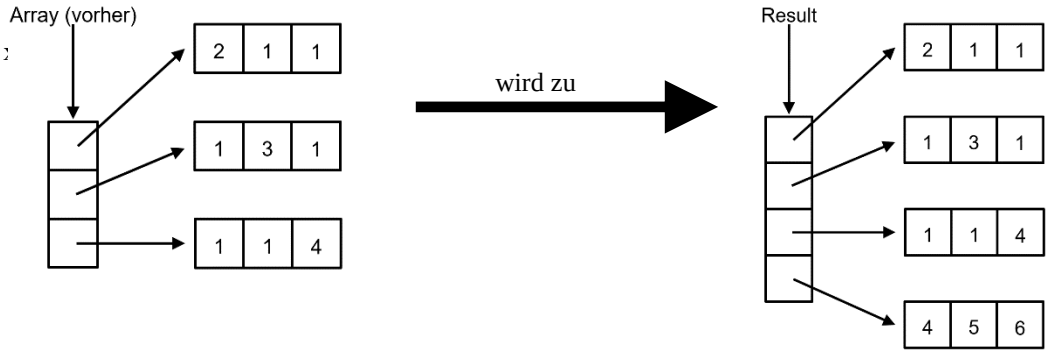
Die Klasse **ArrayProcessor** soll u.a. die folgenden Elemente aufweisen:

ArrayProcessor(long[][])
erzeugt einen ArrayProzessor. Der Konstruktor darf nur "gutartige quasi 2-dimensionale Arrays" als Parameter akzeptieren. Die akzeptierten Parameter müssen den folgenden Eigenschaften genügen um als "gutartig" zu gelten:

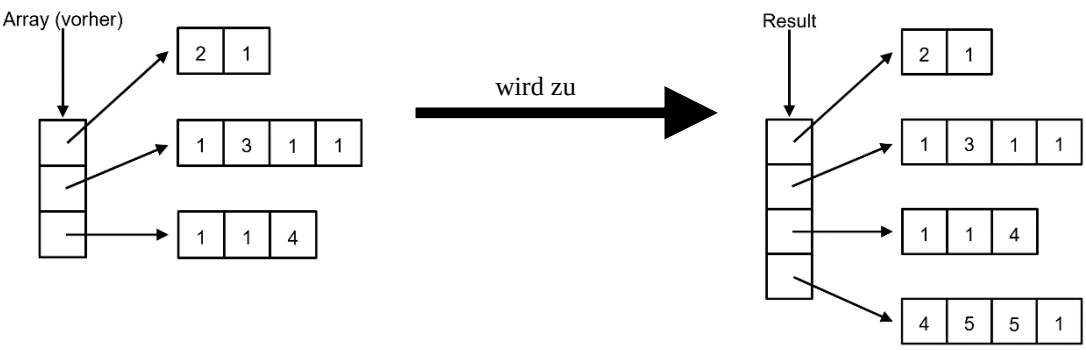
- Die als Parameter übergebene Array-Referenz darf nicht `null` sein.
- Das als Parameter übergebene Array darf nirgends `null` enthalten.
- Das als Parameter übergebene Array darf weder leere Arrays enthalten, noch (selbst) leer sein.

void expand1stDimBySum()
"erweitert" das aktuell im Zustand befindliche Array vom Typ long[][] in der 1.Dimension um die Summe der jeweiligen (Grund-)Werte mit gleicher Position in der 2.Dimension. Das Ergebnis/Resultat wird als neues aktuelles Array in den Zustand des ItemProcessors übernommen.

Bis einschließlich Level C bzw. tol_3n reicht es "quasi echt 2-dimensionale Arrays" (bzw. n×m-Matrizen) zu unterstützen.
Beispiel:



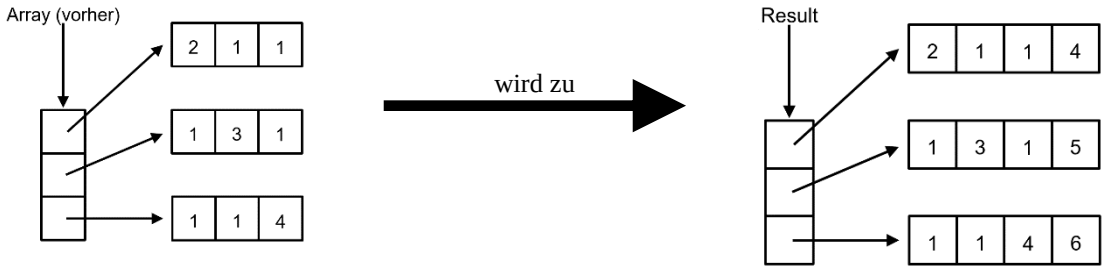
Ab Level D bzw. tol_4s müssen auch nicht "quasi echt 2-dimensionale Arrays" unterstützt werden.
Beispiel:



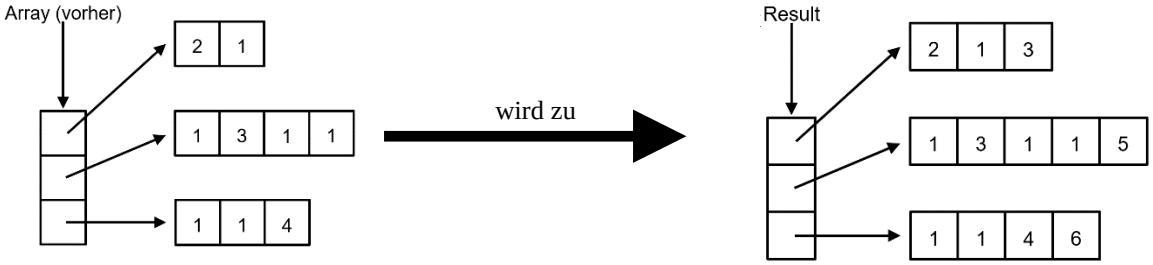
`void expand2ndDimBySum()`

"erweitert" das aktuell im Zustand befindliche Array vom Typ long[][] in der 2.Dimension um die Summe der jeweiligen (Grund-)Werte mit gleicher Position in der 1.Dimension. Das Ergebnis/Resultat wird als neues aktuelles Array in den Zustand des ItemProcessors übernommen.

Bis einschließlich Level C bzw. tol_3n reicht es "quasi echt 2-dimensionale Arrays" (bzw. n×m-Matrizen) zu unterstützen.
Beispiel:



Ab Level D bzw. tol_4s müssen auch nicht "quasi echt 2-dimensionale Arrays" unterstützt werden.
Beispiel:



`long[][] getCurrentArray()`

liefert das aktuelle Array (bzw. das aktuelle Array aus dem aktuellen Zustand des ItemProcessors).

`long[][] getOriginalArray()`

liefert das ursprünglich als Parameter an dem ItemProcessor übergebene Array in seinem Original-Zustand.

Aufgabe a3:

Lösen Sie diese Aufgabe im vorgegebenen Sub-Package **a3**.
Die im vorgegebenen **TestFrameAndStarter**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 generische Klasse **ListAnalyzer<T>**, die sich von **ListAnalyzer_I<T>** ableitet und das Auswerten von Listen unterstützt (**T** ist der generische Datentyp der (Listen-)Elemente). Für die Liste soll die Anzahl des Auftretens der jeweiligen (Listen-)Elemente, die Anzahl unterschiedlicher (Listen-)Elemente innerhalb der Liste und die Menge der unterschiedlichen (Listen-)Elemente bestimmt werden. Diese Informationen/Ergebnisse sollen in Form eines generischen **ResultReporter<T>** für den Klienten bereit gestellt werden. Dieser **ResultReporter<T>** muss das Interface **ResultReporter_I<T>** unterstüzen.

Die Klasse **ListAnalyzer<T>** muss das Interface **ListAnalyzer_I<T>** unterstützen und u.a. die folgenden Elemente aufweisen:

ListAnalyzer(List<T>)
erzeugt einen ListAnalyzer. Der Konstruktor darf nur "gutartige Listen" als Parameter akzeptieren. Die akzeptierten Parameter müssen den folgenden Eigenschaften genügen um als "gutartige Liste" zu gelten:
- Die Liste darf nicht `null` sein.
- Die Liste darf nicht `null` enthalten.

ResultReporter_I<T> computeResultReporter()
bestimmt für die als Parameter übergeben Liste
- die Anzahl des Auftretens der jeweiligen (Listen-)Elemente in der Liste
- die Anzahl unterschiedlicher (Listen-)Elemente in der Liste und
- die Menge der unterschiedlichen (Listen-)Elemente

Diese Informationen/Ergebnisse sollen in Form eines **ResultReporter<T>**, der das Interface **ResultReporter_I<T>** unterstützt, für den Klienten bereit gestellt werden.

List<T> getList()
liefert die Original-Liste, die als Konstruktor-Parameter an den ListAnalyzer übergeben wurde.

Die von Ihnen zu implementierende Klasse **ResultReporter<T>** muss das Interface **ResultReporter_I<T>** unterstützen und soll das "Rückmelden" der Ergebnisse an den Klienten unterstüzen. Die Klasse soll u.a. die folgenden Elemente aufweisen:

int getFrequency(T)
liefert das Berechnungsergebnis: Die Anzahl des Auftretens des jeweiligen (Listen-)Elements in der Liste.
Da diese Information(en)/Ergebnis(se) in Form eines Objekts vom Typ `ResultReporter<T>` für den Klienten bereitgestellt werden soll, ist getFrequency() auch eine eingeforderte Operation für Ihren ResultReporter 😉.

int getAmountOfUniqueItems()
liefert das Berechnungsergebnis: Die Anzahl unterschiedlicher (Listen-)Elemente in der Liste.
Da diese Information/Ergebnis in Form eines Objekts vom Typ `ResultReporter<T>` für den Klienten bereitgestellt werden soll, ist getAmountOfUniqueItems() auch eine eingeforderte Operation für Ihren ResultReporter 😉.

Set<T> knownElements()
liefert das Berechnungsergebnis: Die Menge der unterschiedlichen (Listen-)Elemente.
Da diese Information/Ergebnis in Form eines Objekts vom Typ `ResultReporter<T>` für den Klienten bereitgestellt werden soll, ist knownElements() auch eine eingeforderte Operation für Ihren ResultReporter 😉.

Aufgabe a4:

Lösen Sie diese Aufgabe im vorgegebenen Sub-Package **a4**.
Die im vorgegebenen **TestFrameAndStarterC1x00** 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 **Item**s verarbeitet. Die Idee ist: Items gemäß den vorgegebenen Anforderungen zu sammeln. Die Items haben Eigenschaften. Diese Eigenschaften sind:

- Farbe — Sie haben Zugriff auf diese Eigenschaft mit: `Color getColor()`
- Größe — Sie haben Zugriff auf diese Eigenschaft mit: `Size getSize()`
- Gewicht — Sie haben Zugriff auf diese Eigenschaft mit: `Weight getWeight()`
- Wert — Sie haben Zugriff auf diese 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 drei Items gleichen Gewichts vorliegen, sollen diese als Triple "abgeliefert" werden (wobei die jeweilige Eintreffreihenfolge der Items im Bestand des ItemProcessors erhalten bleiben soll).

Das Interface **Triple_I** beschreibt eine Zusammenstellung von drei Items. Auf die zugehörigen Items kann mit **getItem()** zugegriffen werden. In einem Triple sind drei Items gleichen Gewichts 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 Triples, getItem(1) das zweitälteste Item des Triples und schließlich getItem(2) des jüngste Item des Triples. Die jeweiligen Items verbleiben im Triple.

Informationen zum **ItemProcessor** folgen auf der nächsten Seite.

[1] Das Alter bezieht auf das jeweilige Eintreffen im ItemProcessors bzw. der jeweiligen Aufnahme in den internen Bestand des ItemProcessors.

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

`ItemProcessor()`
erzeugt einen ItemProcessor.

`Triple_I process( Item )`
verarbeitet ein Item. Das als Parameter übergebene Item wird dem Bestand (also den "bisher übergebenen" Items) hinzugefügt.
Immer wenn drei Items gleichen Gewichts (einschließlich des gerade als Parameter übergebenen Items) im Bestand sind, sollen diese drei Items als Rückgabewert der Methode (in Form einesTriples) 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 um als "gutartige Werte/Parameter" zu gelten:

- 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 sein sollte, kann mit process() dem ItemProcessor wieder 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() mit gültigem Parameter), seit dem letzten Aufruf von reset() bzw. sofern kein reset() aufgerufen wurde, seit dem Programmstart.

`int triplesFound()`
liefert die Anzahl der "Treffer"/abgeliefertenTriple (oder in anderen Worten: Wie oft der ItemProcessor eine Zusammenstellung von drei Items gleichen Gewichts 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).

Aufgabe a5:

Lösen Sie diese Aufgabe im vorgegebenen Sub-Package **a5**.
Die im vorgegebenen **TestFrameAndStarterC1x00** 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.

Bemerkung:
Für diese Aufgabe dürfen Sie **keine Collections** verwenden. Sie müssen den Stack mit eigenen Mitteln selbst implementieren und zwar mit einer dynamischen Datenstruktur bzw. konkret einer verketteten Liste. Implementierungen, die dies umgehen, werden als falsch gewertet.

Start der eigentlichen Aufgabe:

Implementieren Sie eine generische Klasse **Stack<T>**, die sich von **Stack_I<T>** ableitet und die typischen Funktionalitäten eines Stacks implementiert. **T** ist der Datentyp der Datensätze, die auf auf dem Stack abgelegt werden. Die von Ihnen zu implementierende Klasse Stack muss sich wie ein Stack verhalten. Dies war Thema der Programmierenvorlesung. Stichwort: LIFO
Gegeben ist ein Klassen-Skelett **Node<T>** für eine Knoten-Klasse um "lästige" Fehlermeldungen im TestFrameAndStarter zu vermeiden. Das Klassen-Skelett ist unvollständig. Sie müssen das Klassen-Skelett geeignet ergänzen.
Der Stack ist als dynamische Datenstruktur bzw. konkret verkettete Liste zu implementieren und für die "Verkettung" müssen Knoten genutzt werden, die durch eine geeignete Vervollständigung des bereits genannte Klassenskeletts **Node<T>** zu realisieren sind.
Die Klasse **Stack<T>** soll u.a. die folgenden Elemente aufweisen:

Stack()
erzeugt einen Stack.

void push(T)
legt einen Datensatz vom generischen Typ T auf dem Stack ab. Die LIFO-Ordnung darf nicht verletzt werden. `null` darf nicht als Parameter akzeptiert werden. Reagieren Sie in der erlernten Weise auf diesen Sonderfall.

T pop()
liefert (von den im Stack enthaltenen Datensätzen) den zuletzt auf dem Stack abgelegten Datensatz (vom generischen Typ T) und entfernt ihn vom Stack. Die LIFO-Ordnung darf nicht verletzt werden. Reagieren Sie in der erlernten Weise auf den Sonderfall, dass ein leerer Stack von der Methode pop() vorgefunden wird.

T top()
liefert (von den im Stack enthaltenen Datensätzen) den zuletzt auf dem Stack abgelegten Datensatz (vom generischen Typ T). Der Datensatz verbleibt im Stack. Die LIFO-Ordnung darf nicht verletzt werden. Reagieren Sie in der erlernten Weise auf den Sonderfall, dass ein leerer Stack von der Methode top() vorgefunden wird.

boolean isEmpty()
prüft, ob der Stack leer ist. Falls der Stack leer ist, muss **true** zurückgegeben werden und andernfalls **false**.

void clear()
leert den Stack und versetzt ihn in den Ausgangszustand zurück (Stichwort "reset").