Algorithmen und Datenstrukturen
Übungsblatt 02

Prof. Karsten Weihe

TECHNISCHE UNIVERSITÄT DARMSTADT

SoSe 2022	v1.0.1
Themen:	Liste mit Hilfe von Arrays
Relevante Foliensätze:	Kapitel 07 aus der FOP (Collections) – finden Sie im AuD-Kurs in moodle
Abgabe der Hausübung:	bis 06.05.2022, 23:50 Uhr

Hausübung 02 — Gesamt: 32 Punkte

Liste mit Hilfe von Arrays

Verbindliche Anforderungen für alle Hausübungen:

Das Dokumentieren und Kommentieren Ihres Quelltextes ist nicht verbindlich, wird zum besseren Verständnis Ihrer Lösung jedoch empfohlen. Alle zur Bewertung dieser Hausübung relevanten Deklarationen von Klassen, Methoden (hierzu zählen auch Konstruktoren) und Attributen sind bereits in der Quelltext-Vorlage enthalten und dürfen nicht modifiziert oder entfernt werden. Ihnen steht aber frei, Hilfskonstrukte in Form von weiteren Klassen, Methoden und Attributen zu erstellen, sofern dies nicht explizit auf dem Übungsblatt verboten wurde und Ihre Hilfskonstrukte nicht gegen verbindliche Anforderungen verstoßen. Datenstrukturen und Hilfsmethoden aus der Java-Standardbibliothek sowie Arrays sind nicht erlaubt, sofern dies nicht explizit auf dem Übungsblatt gefordert oder erlaubt wurde. Ihre Methoden müssen auch dann funktionieren, wenn Aufrufe von in der Vorlage deklarierten Methoden (auch von solchen, welche von Ihnen implementiert werden) durch andere, korrekte Implementationen ersetzt werden.

Der Verstoß gegen verbindliche Anforderungen führt zu Punktabzügen und kann die korrekte Bewertung Ihrer Abgabe unter Umständen beeinflussen. Die Implementation einer in der Quelltext-Vorlage deklarierten Methode wird nur bewertet, wenn der mit `TODO` markierte Exception-Wurf entfernt wird.

Hinweise für alle Hausübungen:

Die zu verwendenen Zugriffsmodifizierer sind in der Vorlage bereits gegeben und werden auf dem Übungsblatt nicht immer angegeben. Beachten Sie die Informationen im Moodle-Abschnitt *Technisches und Probe-Übungsblatt*.

Bei Fragen stehen wir Ihnen vorzugsweise im Moodle-Kurs und in den Sprechstunden zur Verfügung.

Die für diese Hausübung in der Vorlage relevanten Verzeichnisse sind `src/main/h02` und `src/test/h02`.

Hinweis:

Das Verwenden von Arrays ist für alle Aufgaben dieses Übungsblatts erlaubt.

Einleitung

Die Aufgabe auf diesem Übungsblatt besteht darin, die Datenstruktur von Kapitel 07, Folien 193-223 der FOP mit ausgewählten Methoden zu implementieren (die Namen der Attribute eines Listenknotens sind auf diesem Übungsblatt allerdings – im Gegensatz zu den oben genannten Folien – an die Namenskonventionen von Java angepasst).

In der Vorlesung wird ein Verfahren zum Aufteilen eines Arrays vorgestellt, welches angewendet wird, wenn das Array voll ist. Auf diesem Übungsblatt dürfen Sie entweder so vorgehen, wie dort vorgestellt oder wie folgt: Wenn ein Array gefüllt ist, wird bei Bedarf ein nächstes `ListOfArraysItem`-Objekt erstellt und dort wird das nächste Element regulär eingefügt – ganz ohne Aufteilen.

Um Sprachverwirrung zu vermeiden, müssen wir im Folgenden penibel unterscheiden: Mit *Element* bezeichnen wir die Elemente einer abstrakten Sequenz, die als Array, lineare Liste, Liste von Arrays wie unten, Dateiinhalt oder sonstwie realisiert sein kann. Zum Beispiel enthält die abstrakte Sequenz $(1, 2, 3)$ die ganzzahligen Elemente 1, 2 und 3. Werden sie in einem Array gespeichert, sind das die *Komponenten* des Arrays an den *Positionen* (=*Indizes*) 0, 1 und 2. Objekte von einem Typ wie `ListItem` in Kapitel 07 der FOP oder `ListOfArraysItem` in H1 unten sind hingegen *Listenknoten* oder kurz *Knoten*; wenn die Sequenz $(1, 2, 3)$ in einer solchen Liste gespeichert wird, dann ist 1 der Schlüsselwert des Knotens an Position 0 usw.

H1: Liste von Arrays — 5 Punkte

Analog zur generischen Klasse `ListItem` in Kapitel 07, ab Folie 116 der FOP, finden Sie eine generische Klasse mit Namen `ListOfArraysItem` und generischem Typparameter `T`. Ein Objekt von `ListOfArraysItem` hat ein Objektattribut `currentNumber` vom Typ `int`, ein Objektattribut `array` vom Typ „Array von `T`“ und ein Objektattribut `next`, das wie üblich auf den nächsten Listenknoten (bzw. `null` am Listenende) verweist.

Zudem finden Sie eine generische Klasse `ListOfArrays` mit generischem Typparameter `T`. Ein Objekt von `ListOfArrays` hat eine Klassenkonstante `ARRAY_LENGTH` vom Typ `int` sowie zwei Objektvariablen `head` und `tail` vom Typ „`ListOfArraysItem` von `T`“, die auf den ersten bzw. letzten Knoten der Liste verweisen (`null` bei leerer Liste). Die Klassenkonstante `ARRAY_LENGTH` wird in einem `static`-Block mit 256 initialisiert (siehe Kapitel 03c, Folien 125-139 der FOP) und beschreibt die maximale Länge der Arrays, welche von einem `ListOfArraysItem` referenziert wird. In der Vorlage wird `ARRAY_LENGTH` mit 0 initialisiert, was Sie natürlich ändern dürfen.

Der Konstruktor von Klasse `ListOfArrays` hat einen Parameter vom Typ `T[]`. Er erstellt das `ListOfArrays`-Objekt so, dass die Sequenz dieselbe wie im aktualen Parameter ist - Also in der gleichen Reihenfolge und mit den gleichen Objekten. Falls der aktuale Parameter gleich `null` oder ein Array mit Länge 0 ist, wird entsprechend eine leere Sequenz erstellt. Die im Array referenzierten Objekte werden direkt übernommen.

> **Verbindliche Anforderung:**
>
> Die Aufrufe von Operator `new` im Konstruktor sind auf das absolut notwendige Minimum zu beschränken. Sie speichern in jedem einzelnen Array also möglichst viele Elemente, bevor Sie das nächste Array einrichten. Allen Attributen der Objekte sollen ihre korrekten beziehungsweise korrespondierenden Werte zugewiesen werden.

Hinweis:

In Kapitel 06, Folien 138-145 der FOP haben Sie gesehen, dass man in Java keine Arrays von generischen Typparametern mit Operator `new` erzeugen kann. In der spezifischen Situation dieser Hausübung können Sie die Arrays in `ListOfArraysItem<T>` aber einrichten, indem Sie ein „Array von `java.lang.Object`“ erzeugen und mit Downcast auf den Typ `T[]` der Referenz `array` zuweisen. Dabei werden Sie vermutlich gewarnt, dass es sich um einen „unsicheren Cast“ handelt. Diese Warnung können Sie ignorieren und/oder unterdrücken.

Bemerkung:

In der Praxis würde man Klasse `ListOfArrays` das Interface `java.util.List` implementieren lassen. Aber das würde zu erheblichen Komplikationen führen, die für die AuD irrelevant sind.

Verständnisfragen am Rande (0 Punkte)**:**

1. Wir hatten doch immer gesagt, dass Attribute aus gutem Grund eher `private` sein sollten. Bei `ListOfArraysItem` weichen wir davon völlig ab!?
2. Welche Information finden Sie zusätzlich im Namen `currentNumber` statt `number` oder nur `n`? Finden Sie diese Zusatzinformation wichtig? Wie sieht das bei den anderen mehr oder weniger sprechenden Namen oben aus?
3. Zu Beginn wurden die Packages der Standardbibliothek genannt, aus denen Klassen und Interfaces verwendet werden. Bei `List` wurde aber in der Bemerkung oben das Package `java.util` oben angegeben – überflüssig?
4. *Zum Weiterdenken* (*bzw. Weiterrecherchieren*): Erhält man mit dem „Trick“ im Hinweis einen vollwertigen Ersatz für den in Java nicht gültigen Ausdruck „`new T[size]`“, wenn `T` ein generischer Typparameter ist? Falls ja, warum? Falls nicht, was sind die semantischen Unterschiede im Vergleich zum Ergebnis, das Sie intuitiv von „`new T[]`“ erwarten würden? Warum funktioniert der Trick im obigen Hinweis in dieser Hausübung, aber nicht allgemein?

H2: Iterator-Klasse 3 Punkte

Für diese Aufgabe finden Sie in der Vorlage eine Klasse `ListOfArraysIterator`, die wie Interface `Iterator` die Methoden `next` und `hasNext` mit derselben Signatur jeweils hat und dasselbe leistet: Durchlauf durch die einzelnen Elemente der Sequenz im `ListOfArrays`-Objekt.

In Klasse `ListOfArrays` existiert eine Methode `iterator`, die keinen Parameter hat und einen Verweis auf ein `ListOfArraysIterator`-Objekt zurückliefert, das wie üblich so initialisiert ist, dass der erste Aufruf von `next` das erste Element der Sequenz zurückliefert (bzw. `hasNext` liefert sofort `false` zurück, falls die Sequenz leer ist).

Bemerkung:

In der Praxis würde man Klasse `ListOfArraysIterator` das Interface `ListIterator` implementieren lassen. Das würde hier aber zu weit führen.

H3: Methode zum Einfügen einer Sequenz im Block in eine Arrayliste — 6 Punkte

Implementieren Sie in Klasse `ListOfArrays` die Methode `insert`, die keine Rückgabe hat. Diese Methode `insert` hat einen Parameter vom formalen Typ `Collection<T>` sowie einen Parameter vom formalen Typ `int`. Sie darf ohne Überprüfung davon ausgehen, dass der erste aktuale Parameter ungleich `null` ist.

Sei $(s_0, \ldots, s_{n-1})$ die momentane Sequenz des `ListOfArrays`-Objektes unmittelbar bevor Methode `insert` aufgerufen wird. Sei zudem i der zweite aktuale Parameter. Schließlich sei $(t_0, \ldots, t_{k-1})$ die Sequenz, die ein Iterator über den ersten aktualen Parameter von `insert` in diesem Moment zurückliefern würde. Falls $i < 0$ oder $i > n$, wird eine `IndexOutOfBoundsException` mit i als Botschaft geworfen. Ansonsten wird $(t_0, \ldots, t_{k-1})$ an Position i in $(s_0, \ldots, s_{n-1})$ eingefügt, das heißt, das Ergebnis ist $(s_0, \ldots, s_{i-1}, t_0, \ldots, t_{k-1}, s_i, \ldots, s_{n-1})$. In den beiden Randfällen $i = 0$ und $i = n$ läuft das natürlich auf $(t_0, \ldots, t_{k-1}, s_0, \ldots, s_{n-1})$ bzw. $(s_0, \ldots, s_{n-1}, t_0, \ldots, t_{k-1})$ hinaus.

Verbindliche Anforderungen:

1. Das Objekt, auf das der erste aktuale Parameter verweist, darf nicht verändert werden.
2. In dem `ListOfArrays`-Objekt, mit dem `insert` aufgerufen wird, bleiben alle Arrays unverändert außer dem einen oder den beiden, die s_{i-1} und s_i enthalten (bei $i = 0$ bzw. $i = n$ existiert natürlich nur s_i bzw. nur s_{i-1}). Möglichst viele der Werte $t_0, \ldots, t_{k-1}$ werden weitgehend in dieses Array bzw. diese beiden Arrays eingefügt. Falls nicht alle dort hineinpassen, sollen in jedes mit `new` erzeugte Array möglichst viele der Werte, die bis dahin noch nicht in anderen Arrays eingefügt wurden, eingefügt werden. Das heißt, die Anzahl der Aufrufe von Operator `new` wird minimiert unter der Bedingung, dass kein Array außer diesem einen bzw. diesen beiden verändert wird.

Hinweis:

Sowohl die `ListOfArrays`, mit der `insert` aufgerufen wird, als auch der erste aktuale Parameter dürfen auch leer sein.

Unbewertete Verständnisfrage:

Warum wird $(t_0, \ldots, t_{k-1})$ oben nicht einfach als die momentane Sequenz des ersten aktualen Parameters bezeichnet, warum die kompliziertere Formulierung mit Rückgriff auf den Begriff „Iterator“?

H4: Methode zum Extrahieren ganzer Blöcke — 8 Punkte

Implementieren Sie in Klasse `ListOfArrays` die Methode `extract`, die zwei Parameter vom formalen Typ `int` und Rückgabetyp `ListOfArrays<T>` hat.

Sei wieder $(s_0, \ldots, s_{n-1})$ die momentane Sequenz des `ListOfArrays`-Objektes unmittelbar bevor Methode `extract` aufgerufen wird. Seien zudem i und j der erste und der zweite aktuale Parameterwert. Falls $i < 0$, soll analog zu `insert` in H3 eine `IndexOutOfBoundsException` mit i als Botschaft geworfen werden. Falls $i \geq 0$ und $j > n - 1$, soll hingegen eine `IndexOutOfBoundsException` mit j als Botschaft geworfen werden. Falls $i \geq 0$, $j \leq n - 1$ und $i > j$, soll eine `IndexOutOfBoundsException` mit der Botschaft `"<i> is greater than <j>"` geworfen werden, wobei `"<i>"` und `"<j>"` durch die entsprechenden Variablen ersetzt werden.

In allen anderen Fällen wird keine Exception geworfen, sondern Folgendes soll geschehen:

1. Die Rückgabe enthält die Sequenz $(s_i, \ldots, s_j)$.

2. Diese Sequenz wird aus dem `ListOfArrays`-Objekt, mit dem `extract` aufgerufen wird, entfernt.

Letzteres heißt, dass das Objekt, mit dem `extract` aufgerufen wird, unmittelbar nach Beendigung von `extract` die Sequenz $(s_0, \ldots, s_{i-1}, s_{j+1}, \ldots, s_{n-1})$ enthält.

Die `ListOfArrays<T>`-Rückgabeliste wird erstellt, indem die extrahierten Objekte einem Array hinzugefügt werden, welches am Ende dem Konstruktor von `ListOfArrays` übergeben wird, welcher damit die Rückgabe erstellt.

Verbindliche Anforderungen:

1. Neben `ListOfArrays` sind keine anderen Datentypen, die Sequenzen speichern, erlaubt – außer Arrays.
2. Die Aufrufe von Operator `new` sind in H3 auf das absolut notwendige Minimum zu beschränken.

H5: Methode zum Einfügen mehrerer einzelner neuer Elemente 10 Punkte

Schreiben Sie den Inhalt der generischen Klasse `ElementWithIndex` mit Typparameter `T`. Diese Klasse hat zwei Attribute: das Element von Typ `T` und den Index von Typ `int`. Der Konstruktor hat entsprechend zwei Parameter und initialisiert die beiden Attribute mit den aktualen Parameterwerten. Zu beiden Attributen hat `ElementWithIndex` jeweils eine `get`-Methode der üblichen Form (vgl. z.B. Kapitel , ab Folie 79 der FOP, oder auch Kapitel , ab Folie 46 der FOP).

Implementieren Sie in Klasse `ListOfArrays` eine weitere Methode `insert`, Sie überladen `insert` also (Erinnerung: Kapitel , ab Folie 83 der FOP). Diese zweite Methode `insert` hat einen Parameter vom formalen Typ `Iterator<ElementWithIndex<T>>` und wirft potenziell eine `IndexOutOfBoundsException`. Diese Methode `insert` holt sich `ElementWithIndex`-Objekte aus dem Iterator, bis mindestens eine von drei Abbruchbedingungen erfüllt ist: Eine der drei Abbruchbedingungen ist, dass der Iterator am Ende seiner Sequenz ist; die anderen beiden werden im Folgenden spezifiziert.

Die Methode interpretiert jeden Index als eine Art *Offset*, genauer: als Anzahl der ursprünglichen Elemente dazwischen. Beim ersten vom Iterator zurückgelieferten Objekt interpretiert sie den Index als Offset von 0, also als tatsächliche Position. Jeder weitere Index aus dem Iterator wird als Anzahl Zwischenelemente seit dem vorhergehenden Index interpretiert (also Differenz der Indizes$+1$). An den daraus resultierenden Positionen werden die einzelnen Elemente aus den Objekten, die der Iterator zurückliefert, eingefügt. Die zweite, oben angekündigte Abbruchbedingung ist, dass eine so berechnete Position größer als die Länge dieser Sequenz ist; dieses Element wird schon nicht mehr eingefügt.

Um sich die einzelnen Elemente, die überschrieben oder verschoben werden müssen, merken zu können, da die Liste nur ein Mal durchlaufen werden soll, richten Sie ein Array vom Typ T mit Länge `ARRAY_LENGTH` ein, in dem Sie solche Elemente zwischenspeichern, bis sie wieder an ihrer richtigen Position der Liste eingefügt worden sind - Dann werden sie aus dem Array entfernt und der Platz ist wieder nutzbar. Die dritte, oben angekündigte Abbruchbedingung ist, dass das Array keinen weiteren Platz mehr bietet und ein weiteres Element zwischengespeichert werden müsste.

Beispiel:

Sei $(a, b, c, d, e, f, g, h, i, j, k, l, m, n)$ die Sequenz unmittelbar vor Aufruf von `insert` und seien $(r, 1), (s, 2), (t, 1), (u, 2), (v, 0), (w, 3), (x, 4), (y, 2), (z, 1)$ die Paare, die der Iterator in dieser Reihenfolge zurückliefert. Dann ist die Sequenz unmittelbar nach Aufruf von `insert`: $(a, r, b, c, s, d, t, e, f, u, v, g, h, i, w, j, k, l, m, x, n)$.

Eine Exception wird geworfen, wenn ein Offset negativ ist oder das Array zum Zwischenspeichern überfüllt werden würde. Bei einer solchen Exception sind Änderungen bis zum Objekt, durch das die Exception geworfen wird, kein Problem. Je nach auftretender Situation, ist der Offset beziehungsweise `"array could not hold elements to be moved"` die Botschaft.

Verbindliche Anforderung:

Die Liste in dem `ListOfArrays`-Objekt soll nur einmal durchlaufen werden.

H6: JUnit-Tests

Verbindliche Anforderungen für alle Hausübungen:

Auch wenn Sie für das Erstellen von Tests keine Punkte erhalten: Das Teilen jeglicher Tests zu Hausübungen ist *nicht* erlaubt. Unser Ziel ist, die Verbreitung fehlerhafter Tests in Ihrem Sinne zu verhindern. Alle auf diesem Übungsblatt genannten verbindlichen Anforderungen gelten für diese Aufgabe *nicht*. Insbesondere empfiehlt es sich sogar, zum Testen Datenstrukturen und Hilfsmethoden aus der Java-Standardbibliothek zu verwenden.

Hinweise für alle Hausübungen:

Die von uns bereitgestellten Public Tests überprüfen nur einen kleinen Teil Ihrer Implementation. Erfüllt Ihre Lösung nicht alle Public Tests, erhalten Sie auf keinen Fall die volle Punktzahl. Im Umkehrschluss bedeutet dies aber nicht, dass Sie die volle Punktzahl erhalten, wenn Ihre Lösung alle Public Tests besteht.

Der folgende Leitfaden dient als Unterstützung zum Aufbau Ihrer eigenen Tests. Sie können vom Leitfaden abweichen und dabei mindestens genauso aussagekräftige Testergebnisse erzeugen.

Schreiben Sie JUnit-Tests, in denen Sie `ListOfArrays` einmal mit `Double` und einmal mit `String` instanziieren. Alle Tests sollen völlig parallel für beide Instanziierungen dieselben Schritte machen. Wenn im Folgenden von N Instanzen die Rede ist, sind damit durchgängig N Instanzen mit `Double` und N Instanzen mit `String` gemeint.

Eine *Instanz* für eine der beiden Methoden `insert` bzw. `extract` besteht im Folgenden aus einem `ListOfArrays`-Objekt plus aktualen Parameterwerten für die jeweilige Methode.

Wir unterscheiden zwischen Sonderfällen und anderen Fällen. Ein Sonderfall liegt vor, wenn mindestens ein Sondertatbestand vorliegt. Sondertatbestände sind: Die Sequenz des `ListOfArrays`-Objektes oder die Sequenz, die jeweils durch die Parameter definiert ist[1], ist leer, oder eine Exception wird geworfen. In einem Fall, der kein Sonderfall ist, sind dementsprechend beide Listen nicht leer, und die Methode wird ohne Wurf einer Exception regulär beendet.

Weiter unterscheiden wir zwischen Grenzfällen und anderen Fällen. Ein Grenzfall liegt bei `insert` aus H3 vor, wenn am Beginn oder am Ende der Sequenz einzufügen ist, analog wenn bei `extract` am Beginn oder am Ende zu extrahieren ist. Dabei sind Beginn und Ende zwei verschiedene Grenzfälle, bei `insert` aus H3 und bei `extract` müssen also jeweils zwei Grenzfälle unterschieden werden (beide Grenzfälle können aber in derselben Instanz auftreten, nämlich wann?). Bei `insert` aus H5 sind drei Grenzfälle zu unterscheiden: wenn ein Element zu Beginn einzufügen ist, wenn am Ende einzufügen, und drittens, wenn der Offset zwischen zwei Elementen 0 ist (bei `insert` aus H5 können sich die drei Grenzfälle offenbar beliebig in derselben Instanz häufen).

[1] Also durch die `Collection` bei `insert` aus H3, durch das Intervall $i, \ldots, j$ bei `extract` sowie durch die Rückgaben des Iterators bei `insert` aus H5.

Der allgemeine Fall liegt genau dann vor, wenn weder ein Sonderfall noch ein Grenzfall vorliegt. Eine Instanz kann sowohl Sonderfall als auch Grenzfall zugleich sein.

Sie testen die beiden Methoden `insert` sowie `extract` jeweils mit mindestens 100 Instanzen, die den Allgemeinfall erfüllen. Weiter testen Sie für jede der drei Methoden und jeden einzelnen Grenzfall, den diese Methode haben kann, jeweils diesen Grenzfall für diese Methode mit mindestens 100 Instanzen. Bei `insert` aus H3 und `extract` sind das, wie oben gesagt, die beiden Fälle „am Beginn“ und „am Ende“. Bei `insert` aus H5 testen Sie den Grenzfall „Offset 0“ mit Instanzen, die jeweils mindestens 100x Offset 0 haben, davon mindestens 10x am Beginn, mindestens 10x am Ende, und an mindestens einer weiteren Position zwischendrin sollen mindestens 11 Elemente unmittelbar aufeinanderfolgend eingefügt werden (also mindestens 10x nacheinander Offset 0). In allen bis hier geforderten Instanzen ist die Länge der Sequenz im `ListOfArrays`-Objekt mindestens 100.

Für jede der beiden Methoden `insert` sowie für `extract` testen Sie zudem jeden der Sondertatbestände, die bei dieser Methode auftreten können, mit jeweils mindestens 10 Instanzen. Das heißt, sie testen einerseits jeweils den Fall, dass die Liste im `ListOfArrays`-Objekt leer ist, andererseits den Fall, dass die durch die Parameter spezifizierte Liste leer ist, und mit jeweils 10 Instanzen pro Methode auch den Fall, dass beides zugleich eintritt. Schlussendlich testen Sie auch die Exceptionwürfe, das heißt, ob jeweils die erwartete Exception mit der erwarteten Botschaft geworfen wird. Bei `insert` testen Sie mit jeweils mindestens 10 Instanzen den Fall, dass der Index zu klein bzw. zu groß ist. Analog testen Sie mit jeweils mindestens 10 Instanzen bei `extract`, dass der erste Index zu klein, der zweite Index zu groß oder der erste Index größer als der zweite ist.

Für jede Instanz kreieren Sie das jeweilige `ListOfArrays`-Objekt, indem Sie die Sequenz zuerst in einem Array aufbauen und damit den Konstruktor von `ListOfArrays` aufrufen. Bei allen Instanzen, die keine Exception werfen sollen, kopieren Sie diese Sequenz zudem in ein `LinkedList`-Objekt mit derselben generischen Typinstanziierung. Die testweisen Aufrufe einer der beiden Methoden `insert` bzw. von `extract` vollziehen Sie jeweils parallel mit dem `LinkedList`-Objekt nach, und danach vergleichen Ihre Tests die resultierende Sequenz in der `ListOfArrays` mit der resultierenden Sequenz in der `LinkedList`, (wobei Sie gleich `ListOfArraysIterator` mit testen). Sie müssen also die Funktionalität der beiden Methoden `insert` und von `extract` auch für `LinkedList` noch einmal schreiben; dafür haben Sie aber keine Vorgaben zu erfüllen.

Für die Instanzen mit Typinstanziierung `Double` verwenden Sie einfach den Zufallsgenerator für `Double`]-Werte in Klasse `Random`. Als String nehmen Sie solche, die jeweils drei Zeichen haben, und zwar drei zufällige Kleinbuchstaben aus `'a'...'z'`.

Hinweis:

Es bietet sich an, dass Sie für das Debuggen zusätzlich ein paar fest in die JUnit-Tests geschriebene Instanzen verwenden, die Sie auch drin lassen. Diese wählen Sie möglichst klein und übersichtlich, aber dennoch so, dass eigentlich jeder Fehler hierin schon auftreten müsste.

Unbewertete Verständnisfragen:

1. Was ist Ihre Einschätzung – reicht es aus, wie oben gefordert jeden Sondertatbestand separat zu testen oder sollten nicht doch eher alle Kombinationen von Sondertatbeständen getestet werden?
2. Glauben Sie, dass Sie die obige Spezifikation zum Testen von Sonderfällen, Grenzfällen usw. ohne inhaltliche Verfälschungen und ohne unpräzise, missverständliche Formulierungen substantiell einfacher als oben hinbekommen? (Die redundanten Formulierungen oben müssen Sie natürlich beim Vergleich herausrechnen, wenn Sie selbst keine Redundanz verwenden.)