Algorithmen und Datenstrukturen
Übungsblatt 06

TECHNISCHE UNIVERSITÄT DARMSTADT

Rückfragen zu diesem Übungsblatt vorzugsweise im moodle-Forum zu diesem Blatt!

Sommersemester 2021	v03
Themen:	Hashtabellen
Relevante Foliensätze:	Foliensatz/Video zu Hashtabellen
Abgabe der Hausübung:	02.07.2021 bis 23:50 Uhr

In diesem Übungsblatt werden Sie Hashtabellen realisieren und die verschiedenen Möglichkeiten dazu bezüglich Effizienz miteinander vergleichen. Dazu schreiben Sie ein Interface `MyMap`, das sich an `java.util.Map` anlehnt, aber viel einfacher ist (siehe auch Kapitel 07 der FOP, Folien 100 ff.).

***Anmerkung*:** Per Konvention wird im Englischen die mit einem Schlüsselwert assoziierte Information mit dem eher unspezifischen Wort *value* bezeichnet. Im deutschsprachigen Fließtext wird im Folgenden das Wort *Information* verwendet, im Code und in Kommentaren zum Code verwenden wir und auch Sie stattdessen das Wort `value`.

H1 Interfaces und Klassen für Hashfunktionen — 6 Punkte

Bitte legen Sie alle in dieser Aufgabe angelegte Dateien im package `h06.hashFunctions` ab.

***Wichtiger Hinweis*:** Verwenden Sie **nicht** den Java Operator `%` für Modulo-Berechnungen. Dieser wird zwar meist für den meist für den Modulo Operator gehalten, ist aber ein Restwertoperator (siehe `https://docs.oracle.com/javase/specs/jls/se16/html/jls-15.html#jls-15.17.3`). Stattdessen sollte die Methode `Math.floorMod` verwendet werden.

Schreiben Sie zwei generische `public`-Interfaces, `OtherToIntFunction` und `OtherAndIntToIntFunction`, beide mit generischem Typparameter `T`. Analog zu den verschiedenen `Function`-Interfaces in `java.util.function` haben beide Interfaces eine Objektmethode `apply`. In beiden Interfaces hat diese Methode `apply` einen Parameter vom formalen Typ `T` und liefert `int` zurück. Bei `OtherAndIntToIntFunction` hat sie noch einen zweiten Parameter, und der ist vom formalen Typ `int`.

Beide Interfaces, `OtherToIntFunction` und `OtherAndIntToIntFunction`, haben jeweils noch das übliche `get/set`-Methodenpaar mit Namen `getTableSize` und `setTableSize` für ein Attribut *table size* vom formalen Typ `int`, das in den implementierenden Klassen geeignet zu realisieren ist.

Schreiben Sie eine generische `public`-Klasse `HashCodeTableIndexFct` mit generischem Typparameter `T`, die das Interface `OtherToIntFunction<T>` implementiert. Der `public`-Konstruktor hat als ersten Parameter `initialTableSize` vom formalenTyp `int` und als zweiten Parameter `offset`, ebenfalls vom formalen Typ `int`. Der aktuale Wert von `offset` wird vom Konstruktor in einer geeigneten `private`-Objektkonstanten gespeichert. Für die *table size* richten Sie eine `private`-Objektvariable ein und initialisieren diese im Konstruktor mit `initialTableSize`. Auf diese Objektvariable greifen also `get-/setTableSize` zu. Da `T` eine Klasse ist, hat `T` die in `java.lang.Object` eingeführte Methode `hashCode`. Die Methode `apply` liefert den **Rest** einer ganzzahligen Division zurück: Dividend ist der Hashcode des aktualen Parameterwertes, der mit der mit der Objektkonstante aufaddiert wird, die durch den Parameter `offset` gesetzt wurde; Divisor ist der momentane Wert der *table size*. (Die Rückgabe ist also im Indexbereich eines Arrays der Größe *table size*).

Schreiben Sie eine generische `public`-Klasse `LinearProbingTableIndexFct` mit generischem Typparameter `T`, die das Interface `OtherAndIntToIntFunction<T>` implementiert. Die Klasse `LinearProbingTableIndexFct` hat eine `private`-Objektkonstante vom Typ `OtherToIntFunction<T>`, die wie üblich durch einen entsprechenden Parameter des `public`-Konstruktors von `LinearProbingTableIndexFct` initialisiert wird. Die Methoden `get-/setTableSize` rufen die entsprechenden Methoden des Attributs auf. Die Rückgabe der Methode `apply` von `LinearProbingTableIndexFct` ist mathematisch so defi-

niert: Seien x und i die aktualen Parameterwerte beim Aufruf der Methode `apply` von `LinearProbingTableIndexFct` und sei a die Rückgabe der Methode `apply` des Attributs bei Aufruf mit x; dann ist $(a + i)$ mod `getTableSize` die Rückgabe der Methode `apply` von `LinearProbingTableIndexFct`. Aber:

***Verbindliche Anforderung*:** Kein einziges Zwischergebnis in `apply` von `LinearProbingTableIndexFct` darf betragsmäßig größer als das Maximum aus i und *table size* sein. Insbesondere dürfen Sie a und i nicht addieren, sondern müssen einen anderen Rechenweg wählen.[1]

Schreiben Sie eine generische `public`-Klasse namens `DoubleHashingTableIndexFct` mit generischem Typparameter `T`, die das Interface `OtherAndIntToIntFunction<T>` implementiert. Die Klasse `DoubleHashingTableIndexFct` hat zwei `private`-Objektkonstanten vom Typ `HashCodeTableIndexFct`, `fct1` und `fct2`, die wie üblich durch zwei aktuale Parameterwerte im `public`-Konstruktor zu initialisieren sind (`fct1` durch den ersten, `fct2` durch den zweiten aktualen Parameter). Sie hat ebenfalls Methoden `get-/setTableSize`. Methode `get` ruft die entsprechende Methode von einem der beiden Attribute auf, Methode `set` die entsprechende Methode von ***beiden*** Attributen. Vorbedingung, die nicht abgeprüft wird, ist, dass die *table size* in beiden Konstruktorparametern identisch ist.

Die Rückgabe der Methode `apply` von `DoubleHashingTableIndexFct` ist mathematisch so definiert: Seien x und i die aktualen Parameterwerte der Methode `apply` von `DoubleHashingTable- IndexFct` und seien a bzw. b die Rückgaben der Methode `apply` von `fct1` bzw. `fct2` bei Aufruf mit x; dann ist $(a + i \cdot b)$ mod *table size* die Rückgabe der Methode `apply` von `DoubleHashingTableIndexFct`. Aber:

***Verbindliche Anforderung*:** Kein einziges Zwischergebnis in `apply` von `DoubleHashingTableIndexFct` darf größer als das Maximum aus $i \cdot b$ und *table size* sein.

Verständnisfrage am Rande (0 Punke): Unter welchen Bedingungen ist die in DoubleHashingTableIndexFct implementierte Hashfunktion für den Einsatz in einer Hashtabelle geeignet? (Durchläuft die Hashfunktion immer alle Indizes?)

H2 Interface für Hashtabellen — 1 Punkt

Bitte legen Sie alle in dieser Aufgabe angelegte Dateien im package `h06.hashTables` ab.

Schreiben Sie ein generisches `public`-Interface `MyMap` mit generischen Typparametern `K` und `V` (`K` = *key type* = Schlüsselwerttyp und `V` = *value type* = Informationstyp). Im Folgenden wird ein Objekt einer Klasse, die `MyMap` implementiert, kurz `MyMap`-*Objekt* genannt. Die Darstellungsinvariante ist:

1. Ein `MyMap`-Objekt repräsentiert zu jedem Zeitpunkt seiner Lebenszeit eine endliche Menge von Paaren (`key`,`value`), die auch leer sein kann. Diese Menge kann sich über die Zeit hinweg beliebig häufig, aber ausschließlich durch Aufruf der Methoden von `MyMap` ändern.
2. Jeder *Schlüsselwert* `key` ist vom formalen Typ `K` und ungleich `null`.
3. Kein Schlüsselwert kommt in einem `MyMap`-Objekt zu irgendeinem Zeitpunkt zweimal vor, das heißt, für keine zwei im selben Moment im selben `MyMap`-Objekt repräsentierten Schlüsselwerte `key1` und `key2` gilt `key1.equals(key2)`.
4. Der mit `key` assoziierte `value` ist vom formalen Typ `V` und ungleich `null`.

Interface `MyMap` hat folgende Methoden:

- `containsKey`: hat einen Parameter `key` vom formalen Typ `K` und Rückgabetyp `boolean`; liefert genau dann `true` zurück, wenn es ein Paar (`key`,`value`) in dem `MyMap`-Objekt, mit dem `containsKey` aufgerufen wird, gibt.
- `getValue`: hat einen Parameter `key` vom formalen Typ `K` und Rückgabetyp `V`; falls `key` momentan im `MyMap`-Objekt enthalten, also irgendein Wert `value` vom formalen Typ `V` momentan mit `key` assoziiert ist, wird `value` zurückgeliefert; ansonsten wird `null` zurückgeliefert.
- `put`: der erste Parameter, `key`, ist vom formalen Typ `K`, der zweite Parameter ist vom formalen Typ `V`; Rückgabetyp ist `V`; falls `key` unmittelbar vor Aufruf von `put` im `MyMap`-Objekt enthalten war, wird die bisher mit `key` assoziierte Information zurückgeliefert, und der zweite aktuale Parameter von `put` wird die neue mit `key` assoziierte Information; andernfalls wird ein neues (`K`,`V`)-Paar bestehend aus den beiden aktualen Parametern von `put` in das `MyMap`-Objekt eingefügt und `null` zurückgeliefert.

[1] Das mathematische Stichwort dazu, nach dem Sie bei Bedarf suchen können, lautet *Modulare Arithmetik*.

- `remove`: hat einen Parameter `key` vom formalen Typ `K` und Rückgabetyp `V`; falls `key` in dem `MyMap`-Objekt, mit dem `remove` aufgerufen wird, enthalten ist, werden `key` und die damit assoziierte Information aus dem `MyMap`-Objekt entfernt und letztere zurückgeliefert; andernfalls wird der Inhalt des `MyMap`-Objektes nicht verändert und `null` zurückgeliefert.

***Verständnisfrage am Rande* (0 Punkte):** Was wäre das Problem, wenn erlaubt würde, dass die von einem `MyMap`-Objekt repräsentierte Menge von Paaren auch anders als durch die Methoden von `MyMap` geändert wird? Wie könnte so etwas überhaupt gehen?

H3 Hashtabelle mit mehrfacher Sondierung — 7 Punkte

Bitte legen Sie alle in dieser Aufgabe angelegte Dateien im package `h06.hashTables` ab.

Schreiben Sie eine generische `public`-Klasse `MyIndexHoppingHashMap` mit generischen Typparametern `K` und `V`, die das Interface `MyMap<K,V>` implementiert.

Die Klasse `MyIndexHoppingHashMap` hat drei `private`-Objektvariable: `theKeys` vom Typ „Array von `K`“, `theValues` vom Typ „Array von `V`“ und `occupiedSinceLastRehash` vom Typ „Array von `boolean`“. Wesentliche Implementationsinvariante ist, dass ein Objekt von Klasse `MyIndexHoppingHashMap` zu jedem Zeitpunkt nach Beendigung des Konstruktors die folgenden Bedingungen erfüllt:

1. Die drei Arrays `theKeys`, `theValues` und `occupiedSinceLastRehash` sind nicht `null` und haben dieselbe Länge.
2. Die vom `MyIndexHoppingHashMap`-Objekt repräsentierten Schlüsselwerte sind genau die in `theKeys`$[i]$ gespeicherten Werte, also $i \in \{0,\ldots,$`theKeys.length`$-1\}$ und `theKeys`$[i]$ `!= null`. Die zu einem Schlüsselwert `theKeys`$[i]$ gespeichete Information ist `theValues`$[i]$.
3. Für $i \in \{0,\ldots,$`theKeys.length`$-1\}$ gilt: falls `theValues`$[i]$ `!= null`, dann auch `theKeys`$[i]$ `!= null`. Mit anderen Worten: Es gibt keine Information in `theValues`, die nicht mit einem Schlüsselwert in `theKeys` assoziiert ist.
4. Für $i \in \{0,\ldots,$`theKeys.length`$-1\}$ ist der Wert in `occupiedSinceLastRehash`$[i]$ genau dann `true`, wenn `theKeys`$[i]$ `!= null` momentan *ist* oder[2] `theKeys`$[i]$ `!= null` zumindest *war* zu mindestens einem Zeitpunkt seit dem letzten Aufruf von `rehash` bzw. – falls es noch keinen Aufruf von `rehash` gab – nach Beendigung des Konstruktors (Details von `rehash` weiter unten).

Die Klasse `MyIndexHoppingHashMap` protokolliert in einer geeigneten `private`-Objektvariablen mit, wie viele Komponenten von `occupiedSinceLastRehash` momentan `true` sind.

Weitere geeignete `private`-Objektkonstanten von `MyIndexHoppingHashMap` werden durch entsprechende Parameter des `public`-Konstruktors initialisiert: (i) die initiale Länge der drei Arrays als `int` > 0, der Konstruktor richtet also alle drei Arrays mit dieser Größe ein; (ii) der Faktor als `double`-Zahl > 1, um den die Länge der drei Arrays bei jedem Aufruf von `rehash` (auf `int` abgerundet) wächst; (iii) der Schwellwert für den Füll- und Fragmentierungsgrad als eine `double`-Zahl im halboffenen Intervall $(0\ldots1]$. Dass diese drei Parameter in diesen Bereichen liegen, darf der Konstruktor ungeprüft voraussetzen.

Die Methode `rehash` (Details unten) wird in `put` vor dem Einfügen des neuen `(key,value)`-Paares aufgerufen, falls durch das Einfügen des neuen `(key,value)`-Paares der Anteil derjenigen Indizes $i \in \{0,\ldots,$`theKeys.length`$-1\}$, für die `occupiedSinceLastRehash`$[i]$ `== true` gilt, diesen Schwellwert überschreiten würde.

Die Klasse `MyIndexHoppingHashMap` hat eine weitere `private`-Objektvariable, und zwar vom Typ `OtherAndIntToIntFunction<K>`; diese wird durch einen vierten Parameter des Konstruktors initialisiert. Dieses Objekt wird verwendet, um in den Methoden von `MyMap` die einzelnen Indizes zu berechnen, an denen gemäß Video/Folien zu Hashtabellen nachgeschaut wird, um den ersten aktualen Parameter bzw. eine unbesetzte Arraykomponente zu finden. Der zweite Parameter der Methode `apply` von `OtherAndIntToIntFunction<K>` ist beim ersten Aufruf 0, beim zweiten 1, beim dritten 2 usw. Ist eine Arraykomponente i momentan unbesetzt, aber `occupiedSinceLastRehash`$[i]$ ist `true`,dann suchen alle vier Methoden weiter, bis entweder der erste aktuale Parameter oder eine unbesetzte Arraykomponente j mit `occupiedSinceLastRehash`$[j]$`==false` erreicht ist. Falls Parameter `key` speziell bei Methode `put` nicht auf diesem Weg

[2] Dieses „oder“ ist natürlich zu verstehen als ein inklusiv-oder.

in `theKeys` gefunden wird, soll das Paar(`key`,`value`) im ersten gefundene leeren Index i abgespeichert werden, egal ob `occupiedSinceLastRehash[`i`] == true` oder `== false` ist.

Die parameterlose `private`-Objektmethode `rehash` kreiert drei neue Arrays `theKeys`, `theValues` und `occupiedSinceLastRehash`, und zwar um so viel größer als die momentanen Arrays wie die entsprechende Objektvariable gemäß (ii) oben besagt. Alle Werte ungleich `null` in `this.theKeys` und `this.theValues` werden durch `rehash` mittels Zuweisungsoperator „=“ in die beiden neuen Arrays `theKeys` bzw. `theValues` kopiert, aber nicht einfach nur an denselben Indizes, an denen sie an den alten Arrays stehen. Stattdessen ersetzen Sie die alten Arrays durch die neuen (müssen aber natürlich kurzzeitig Verweise auf die alten speichern) und rufen `put` mit allen (`key`,`value`)-Paaren auf, die in den alten Arrays momentan gespeichert sind. Die Werte im neuen Array `occupiedSinceLastRehash` setzen Sie so, dass die obige Implementationsinvariante unmittelbar nach Beendigung des Aufrufs von `rehash` erfüllt ist. Die *table size* im Attribut von Typ `OtherAndIntToIntFunction<K>` muss dann natürlich ebenfalls aktualisiert werden.

Hinweis (auch für H4): Eigentlich kann man ja in Java kein Array eines generischen Typs erzeugen (siehe Kapitel 06 der FOP, Folien 138 ff.). Bei Übungsblatt 02 hatten wir aber einen Trick gesehen, mit dem das doch geht.

***Verständnisfrage am Rande* (0 Punkte):** Halten Sie es für eine gute Idee, die drei Arrays `public` zu machen, um Nutzern der Klasse `MyIndexHoppingMap` möglichst viel Flexibilität beim Programmieren ihrer auf `MyIndexHoppingMap` basierenden Anwendungen zu erlauben?

H4 Hashtabelle von Listen — 6 Punkte

Bitte legen Sie alle in dieser Aufgabe angelegte Dateien im package `h06.hashTables` ab.

Schreiben Sie eine generische `public`-Klasse `MyListsHashMap` mit generischen Typparametern `K` und `V`, die Interface `MyMap<K,V>` implementiert.

Klasse `MyListsHashMap` hat eine `private`-Objektkonstante eines Arraytyps mit Namen `table`. Und zwar ist der Komponententyp dieses Arrays vom Typ „Liste von Paaren mit ersten Element aus `K` und zweitem Element aus `V`“. Als Listentyp verwenden Sie `java.util.LinkedList`.

Zur Speicherung dieser Paare schreiben Sie eine weitere (nicht interne) `public`-Klasse `KeyValuePair` mit generischen Typparametern K und V. Diese hat einen Konstruktor mit einem Parametern vom formalen Typ `K` und einem zweiten Parameter vom formalen Typ `V`. Der Konstruktor speichert diese Parameter ab und zwar den Parameter vom formalen Typ `K` in einer privaten Objektkonstante und der Parameter vom formalen Typ `V` in einer privaten Objektvariable. Schreiben Sie zusätzlich `public` Get und Set Methoden für den Parameter vom Typ `V` mit Name `getValue` bzw. `setValue` sowie eine `public` Get Methode für den Parameter vom formalen Typ `K` mit Namen `getKey`.

Die wesentliche Implementationsinvariante von `MyListsHashMap` ist, dass zu jedem Zeitpunkt die Menge der in einem Objekt von `MyListsHashMap` gespeicherten (`key`,`value`)-Paare genau die Vereinigung der Mengen von Paaren in den einzelnen Komponenten des Arrays ist und dass kein (`key`,`value`)-Paar zweimal in dieser Vereinigung enthalten ist.

Klasse `MyListsHashMap` hat eine `private`-Objektkonstante vom Typ `OtherToIntFunction`, die wie üblich durch einen Parameter des `public`-Konstruktors vom selben formalen Typ initialisiert wird. Aus diesem Attribut kommt die Größe des Arrays (*table size*). Methode `apply` dieses Attributs liefert für einen Schlüsselwert den Index im Array zurück, an dem dieser Schlüsselwert in der dortigen Liste zu speichern ist bzw. nach der Speicherung wiedergefunden werden kann. Zusätzlich werden im Konstruktor alle Listen der Objektkonstanten `table` im Voraus initialisiert, so das sich die Anzahl der Spezialfälle in den anderen Methoden verringert. Jedes Paar (`key`,`value`) wird beim Einfügen an Position 0 der Liste gespeichert.

H5 Eigene `hashCode`-Implementation — 2 Punkte

Bitte legen Sie alle in dieser Aufgabe angelegte Dateien im package `h06.hashFunctions` ab.

Schreiben Sie eine `public`-Klasse `MyDate` mit fünf `private`-Objektkonstanten vom Typ `int`: Jahr $1970 \ldots 2021$, Monat, Tag im Monat, Stunde und Minute. Der `public`-Konstruktor hat einen Parameter vom formalen Typ `java.util.Calendar`

und initialisiert diese fünf Attribute durch die entsprechenden Werte im aktualen Parameter. Für jedes der fünf Attribute implementieren Sie eine `get`-Methode: `getYear`, `getMonth`, `getDay`, `getHour` und `getMinute`. Sie dürfen die fünf Objektkonstanten intern auch als Array realisieren, wenn sie das bevorzugen.

Ihre Klasse `MyDate` hat eine boolesche Objektkonstante sowie neben den oben genannten noch sechs weitere `private`-Objektkonstanten, alle sechs vom Typ `long`. Das boolesche Attribut wird durch einen zweiten, und zwar booleschen Parameter des Konstruktors initialisiert. Die sechs `long`-Attribute werden im Folgenden *Koeffizienten* genannt: fünf Koeffizienten für die fünf aus dem `Calendar`-Objekt gezogenen `int`-Attribute und ein Koeffizient für die Summe dieser fünf `int`-Attribute. Diese sechs `long`-Attribute werden ebenfalls im Konstruktor gesetzt, aber nicht auf Basis von Parametern des Konstruktors; Genaueres in der verbindlichen Anforderung unten.

Überschreiben Sie die in `java.lang.Object` definierte Methode `hashCode` für `MyDate` so, dass die Rückgabe von `hashCode` im Prinzip wie folgt ist: Falls das boolesche Attribut `true` ist, soll jedes aus dem `Calendar`-Objekt gezogene Attribut mit seinem zugehörigen Koeffizienten multipliziert werden, und diese fünf Produkte werden zusammenaddiert. Ist hingegen das boolesche Attribut `false`, so werden die aus dem `Calendar`-Objekt gezogenen Attribute aufaddiert und die Summe mit dem sechsten Koeffizienten multipliziert. Da Methode `hashCode` Rückgabetyp `int` hat, müssen Sie das Ergebnis in jedem der beiden Fälle noch durch Modulo-Bildung mit dem größten durch `int` darstellbaren Wert (siehe Folien 16 ff. in Kapitel 11 der FOP) ermitteln. Aber:

***Verbindliche Anforderung*:** In Methode `hashCode` von `MyDate` sind alle arithmetischen Operationen im Datentyp `long` auszuführen. Keine dieser arithmetischen Operationen darf einen arithmetischen Überlauf erzeugen. Der Koeffizient für die Minuten lautet: 99991, der Koeffizient für die Stunden lautet: 1234, der Koeffizient für die Tage lautet: 3, der Koeffizient für die Monate lautet: 83231, der Koeffizient für die Jahre lautet: 4563766470487200 und der Koeffizient für die Summe lautet: 98927.

***Verständnisfrage* (0 Punkte):** Die Konversion von `long` auf `int` könnte man auch mit dem Typecast Operator „`(int)`“ oder mit Methode `intValue` von `java.lang.Long` erreichen, aber dann müsste auf das Vorzeichen des Ergebnisses geachtet werden. Warum?

H6 Tests — 4 Punkte

Bitte legen Sie alle in dieser Aufgabe angelegte Dateien im package `h06.test` ab.

***Wichtige Verständnisfrage*:** Weshalb kann der Testfall für DoubleHashingTableIndexFct (k=2) unter Umständen nicht funktionieren? Siehe dazu auch Verständnissfrage in H1! Tests für den Fall k=2 finden deshalb **nicht** statt. Sie sollten für diesen Fall auch selbst keine ausführen - es kann je nach Implementierung zu Endlosschleifen kommen! Auch müssen Sie den Fall natürlich nicht zwangsläufig implementieren ‿

In dieser Aufgabe schreiben Sie einen Test, der die Laufzeit Ihrer Implementierung in bestimmten Szenarien ermitteln soll. Erstellen Sie dazu eine `public`-Klasse `RuntimeTest`. Diese Klasse enthält die `public` Methode `Test` mit Parametern i, j, k und ℓ vom formalen Typ `int`, sowie einen parameterlosen `public`-Konstruktor.

Im Konstruktor wird ein Testdatensatz generiert: Sie generieren einzelne Testdaten, indem Sie zufällige `long`-Werte aus einem `long`-Stream (siehe bspw. `java.util.Random`) lesen und mit `Calendar.setTimeInMillis` aus jedem positiven `long`-Wert ein `Calendar`-Objekt erstellen, wobei dass das Jahr der damit erstellten `Calendar`-Objekte nicht größer als 2021 sein soll. Ihre Testsuite soll dabei insgesamt einen Satz von 60.000 Testdaten, jeweils bestehend aus einem einzelnen `Calendar`-Objekt, erstellen. WICHTIG: Generieren Sie bitte nicht stumpf Zufallszahlen und verwerfen Sie alle Zufallszahlen < 0 oder Calendar-Objekte mit Jahr > 2021! Mit dieser Herangehensweise werden im Schnitt ungefähr 36.000.000.000 Zufallszahlen generiert. Verwenden Sie stattdessen alle Zufallszahlen, indem Sie diese bspw. durch Modulo-Bildung auf ein entsprechendes Intervall begrenzen!

Aus jedem dieser `Calendar`-Objekte erstellen Sie dann zwei `MyDate`-Objekte, je eines mit Wert `true` bzw. `false` als zweiten aktualen Parameterwert des Konstruktors, dies ist nun ihr Testdatensatz, auf dem Sie den folgenden Test durchführen.

Der Test wird über die `public` Methode `Test` gestartet. Dabei haben die Parameter $i, j, k, \ell \in \{1, 2\}$ (in dieser Reihenfolge und vom formalen Typ `int`) Einfluss auf die Art des Tests, konkret sieht das Setup dann wie folgt aus:

- $i = 1$ heißt, diejenigen `MyDate`-Objekte bilden die Testmenge, bei deren Konstruktion der zweite aktuale Parameter `true` war; bei $i = 2$ entsprechend `false`.
- $j = 1$ heißt Hashtabelle mit Klasse `MyIndexHoppingHashMap`, initialisiert mit Resize-Faktor(ii) von 2 und Resize-Schwellwert(iii) von 0,75 ; $j = 2$ heißt entsprechend `MyListsHashMap`. Die Hashtabelle akzeptiert in beiden Fällen nur Key und Value Werte vom formalen Typ `MyDate`, das heißt die Typparamter K und V sind beide `MyDate`.
- Für die dazugehörige passende Hashfunktion gilt: Bei MyIndexHoppingHashMap ($j = 1$) heißt $k = 1$ das LinearProbingTableIndexFct mit interner Hashfunktion HashCodeTableIndexFct mit Offset 0 verwendet wird; $k = 2$ heißt das DoubleHashingTableIndexFct mit interner Hashfunktion 1 HashCodeTableIndexFct mit Offset 0 und interner Hashfunktion 2 HashCodeTableIndexFct mit Offset 42 verwendet wird. Für MyListsHashMap ($j = 2$) ist k irrelevant und es wird HashCodeTableIndexFct mit Offset 0 verwendet.
- $\ell = 1$ bzw. $\ell = 2$ heißt, die Hashtabelle ist **initial** sehr großzügig bzw. sehr klein im Verhältnis zur Größe der Testmenge, konkret:
  - bei $j = 1$: die Anzahl der Komponenten der drei internen Arrays ist bei $\ell = 1$ dreimal so groß wie die Testmenge, bei $\ell = 2$ ein Zehntel der Größe der Testmenge;
  - bei $j = 2$: die Anzahl der Komponenten in dem einen internen Array ist bei $\ell = 1$ dreimal so groß wie die Testmenge, bei $\ell = 2$ ein Zehntel der Größe der Testmenge.

Nach dem Setup basierend auf den Parametern, ist eine voll initialisierte Hashtabelle, auf der dann der eigentlich Test ausgeführt wird, vorhanden. Dieser Test umfasst folgendes, wobei die Ergebnisse der einzelnen Operationen keine Relevanz haben und auch nicht ausgewertet oder gespeichert werden sollen:

1. Fügen sie die ersten 45.000 Elemente aus der Testmenge in die Hashtabelle ein (Key = Value = Element).
2. Überprüfen Sie für alle Elemente aus der Testmenge, ob diese in der Hashtabelle vorkommen.
3. Versuchen Sie für alle Elemente aus der Testmenge, den in der Hashtabelle gespeicherten Wert zu ermitteln.
4. Versuchen Sie für alle Elemente der Testmenge, diese aus der Hashtabelle zu löschen.

Die Ermittlung der Laufzeit der Tests und damit der Methode erfolgt extern und ist hier nicht weiter relevant. Die Geschwindigkeit ist aber kein Bewertungskriterium, Sie brauchen also keine besonderen Geschwindigkeitsoptimierungen an Ihrem Code durchzuführen.

H7 Überprüfen — 0 Punkte

Gehen Sie Ihre Abgabe nochmal durch und überprüfen Sie sie auf Tippfehler in der Namensgebung! Insbesondere sollen alle Interfaces und Klassen `public` sein.