F06: 👕 Kleiderschrank

> Diese Aufgabe ist Teil der freiwilligen inoffiziellen Zusatzaufgaben von Eric Jacob und Jonas Wende, erstellt im WS 23/24 für *IN0002: Grundlagenpraktikum Programmierung*.
> Weder sind sie durch die ÜL überprüft, noch unbedingt vollständig richtig.
> Fehler gerne melden: eric.jacob.2003@gmail.com

F06: 👕 Kleiderschrank

- 🎯 Lernziele
- 📜 Backstory
- 📝 Aufgabenbeschreibung
  - 🧦 Kleiderschrank 1: Kleiderstangen
    - Allgemeine Kleiderstange
    - Rekursive Kleiderstangen
    - Iterative Kleiderstangen
    - Zusätzliche Methoden
  - 👖 Kleiderschrank 2: Kleiderstapel
    - Kleiderstapelkonzept
    - Annis Kleiderstapel
  - 🧣 Kleiderschrank 3: Wäscheleinen-Graph
    - Erste Methoden
  - 🧤 Kleiderschrank 4: Kleiderstangen-Deque
    - Aufbau
    - Deque-Methoden
- 🪜 Aufgaben
- 🧿 Lösungsvorschlag

🎯 Lernziele

Diese Aufgabe dient der Wiederholung folgender Konzepte:

- Datenstrukturen
  - Listen, Stacks, Graphen, Deques

📜 Backstory

Die Pinguine sind sehr ordentliche Tiere. Deswegen verwalten sie auch ihre Kleiderschränke mit Struktur. Verschiedene Pinguine bevorzugen aber verschiedene Datenstrukturen um ihren Schrank zu verwalten. Ebenso sind die Pinguine begeistert von den neusten Techniktrends und würden entsprechend ihres aktuellen Drangs zur Digitalisierung gerne auch ihre Kleiderschränke digital verwalten wollen (irgendwo müssen die NFT-Pullover und -Jeans ja gelagert werden). Kannst du den Pinguinen helfen, ihre Kleiderschränke in Java zu organisieren?

📝 Aufgabenbeschreibung

🧦 Kleiderschrank 1: Kleiderstangen

Pinguin Rick bevorzugt es, seine Kleider an einer Kleiderstange aufzuhängen. Diese wird in Form einer einfach verketteten Liste verwaltet:
Er kann auf den ersten und letzten Kleiderhaken zugreifen, zudem wird für jeden Kleiderhaken gespeichert, welcher Haken dahinter auf der Kleiderstange hängt.
Ebenso sollen verschiedene Methoden implementiert werden, etwa um neue Kleidung an bestimmten Stellen der Stange zu hängen.
Um euch Listenexperten zu fordern, möchte Rick diese Methoden sowohl rekursiv, als auch iterativ implementiert bekommen — so hat er auch mehr Zeit, seinen Lieblingsfilm *Der Polarexpress* fertig zu schauen, bevor ihr ihn daran erinnert, die digitale Kleiderstange mit seinen Hosen, Shirts und Socken zu testen.

Allgemeine Kleiderstange

Implementiere eine einfach verkettete Liste, um die Kleiderstange zu verwalten.
Diese hat außerdem eine Methode `add(T element)`, welche einen neuen Kleiderhaken mit `element` als Inhalt an den Anfang der Kleiderstange hängt.

Rekursive Kleiderstangen

Implementiere die folgenden Methoden rekursiv:

- `removeNthRec(int n)` entfernt das `n`-te Element der Liste und gibt den an der Stelle in der Liste gespeicherten Wert zurück (oder `null` wenn nicht vorhanden).
- `filterSocksRec()` filtert alle Elemente, die vom Typ Socke sind.
  Hinweis: mit `instanceof` könnt ihr in Java prüfen, ob ein Objekt einer bestimmten Klasse angehört.
- Die Methode `insertAtRec(int n, T element)` fügt einen neuen Kleiderhaken mit Wert `element` an `n`-ter Stelle ein.
- `containsRec(T element)` überprüft, ob das übergebene Element in der Liste enthalten ist.

Iterative Kleiderstangen

Nein, nein, nein! Pinguin Rick ist gerade eingefallen, dass er die Methoden doch lieber iterativ möchte! Hilf ihm und implementiere die Methoden aus *Rekursive Kleiderstangen* nochmals, allerdings ohne Rekursion zu verwenden!

Zusätzliche Methoden

Außerdem soll die Kleiderstange noch ein wenig zusätzliche Funktionalität bieten, je nachdem welche Objekte in dieser gespeichert werden. Füge die folgende Methode zum Template hinzu:

- `filterSingleSocks` nimmt eine Liste an Socken als Parameter. Alle Socken, deren Gegenstück nicht auch in der Liste liegen, sollen aus der Liste entfernt werden. Zuletzt wird die neue, gefilterte Liste zurückgegeben.

Kleiderschrank 2: Kleiderstapel

Manche Pinguine bevorzugen es, ihre Kleidung einfach in einer Ecke ihres Iglus zu stapeln — zu ihnen gehört auch Pinguin Anni. Unglücklicherweise können sie so jedoch stets nur auf das oberste Element ihres Kleiderstapels zugreifen und entweder dieses vom Stapel nehmen, oder ein neues Element oben auf den Stapel legen.
Dem kann jedoch Abhilfe geschaffen werden, indem die Pinguine alle Elemente nacheinander von oben auf einen zweiten Stapel legen. Dazu gibt es die Klasse `Kleiderstapel`, welche zwei Stacks verwaltet; einen Stack um Kleidung zu lagern und einen "Zwischenspeicher", auf den die obersten Elemente des ersten Stacks gelegt werden, um auf Elemente in der Mitte des ersten Stacks zugreifen zu können. Es können jedoch keine neuen Elemente für den zweiten Stapel erstellt werden!

Kleiderstapelkonzept

Implementiere die Klasse `Stapel`, welche einen Stack darstellt. Es soll hier nur auf das oberste Element zugegriffen werden können. Wie gewohnt kann man mit `push(T element)` ein neues Element oben auf den Stack legen. Die Methode `pop()` entfernt das oberste Element des Stacks und gibt dieses zurück. `reverse(Stapel<S> stapel)` dreht den übergebenen Stapel um und gibt den neuen Stapel zurück.

Annis Kleiderstapel

Die Klasse Kleiderstapel verwaltet zwei Stapel. Es sollen die folgenden Methoden implementiert werden:

- `push(T element)` pusht auf den ersten Stapel
- `pop()` entfernt das oberste Element vom ersten Stapel und gibt dieses zurück (`null` für einen leeren Stapel).
- `remove(T element)` entfernt das erste Vorkommen von `element` aus dem Stack. Hierfür wird geprüft, ob das oberste Element des ersten Stacks das gesuchte ist. Wenn ja, so wird dieses entfernt. Wenn nein, wird das oberste Element auf den zweiten Stack gelegt und der erste Stack weiter durchsucht.
- `removeAll(T element)` entfernt nach dem selben Prinzip wie `remove` alle Auftreten von `element` im ersten Stack.
- `mergeStacks()` verbindet den ersten und zweiten Stack so, dass das oberste Element vom ersten Stapel auf das oberste Element vom zweiten Stapel folgt, das zweite des ersten Stapels auf das zweite des zweiten Stapels, usw. Ist ein Stack leer, so folgen alle Elemente des verbleibenden anderen Stapels direkt aufeinander.

Kleiderschrank 3: Wäscheleinen-Graph

Der Pinguin Rollo bevorzugt es, seine Kleidung durch ein elaboriertes System von Wäscheleinen zu verwalten. Da man im Wäscheleinengewirr jedoch schnell den Überblick verliert, bittet er dich darum, diese mittels eines Graphen darzustellen. Die einzelnen Knoten stehen dabei für einzelne Kleidungsstücke, während die Kanten des Graphen die Wäscheleinen darstellen sollen. Ebenso sollen die Kanten gerichtet sein (d.h. nur in eine Richtung zeigen), was beim Aufstellen der Wäscheleinen helfen soll.

Hierfür wird die generische Klasse `Waescheleinengraph<T>` verwendet. `T` soll hierbei der Typ der Knoten sein. Der Graph wird durch eine `Map` dargestellt, welche Knoten als Keys besitzt und in den Values alle Knoten speichert, zu denen von diesem Knoten eine Leine führt.

Erste Methoden

- `addVertex(T element)` fügt einen Knoten zur `Map` mit einer leeren Liste als Value hinzu. Ist `element` bereits als Knoten vorhanden, so ändert sich nichts.
- `addEdge(T from, T to)` fügt eine neue Kante zwischen den Kleidungsstücken `from` und `to` hinzu.
- `removeVertex(T element)` entfernt einen Knoten, sofern dieser vorhanden ist. Es gibt keine Änderungen, wenn der Knoten nicht vorhanden ist.
- `removeEdge(T from, T to)` entfernt die Kante zwischen `from` und `to` aus dem Graphen. Auch hier gibt es keine Änderungen, wenn diese nicht vorhanden ist.
- `existsEdge(T from, T to)` prüft, ob die Kante Teil des Graphen ist.

🧤 Kleiderschrank 4: Kleiderstangen-Deque

Pinguin Rick bittet dich nochmals um Hilfe — er ist der Meinung, dass es sinnvoller wäre, seine Kleiderstange durch eine Deque darzustellen, wobei auf das erste und letzte Element je sofort zugegriffen werden kann. Ebenso hat jedes Element eine Referenz auf seinen Vorgänger bzw. Nachfolger.

Aufbau

Implementiere die Klassen `KleiderstangenDequeue` und `DequeueElement`.

Deque-Methoden

Implementiere die folgenden Methoden

- `addFirst(T element)` fügt `element` an den Anfang der Deque ein.
- `addFirst(Collection<T> elements)` fügt alle Elemente einer beliebigen Collection nacheinander an den Anfang der Liste hinzu.
  Hinweis: `Collection` erbt von `Iterable`.
- `removeFirstOccurrence(T element)` entfernt das erste Auftreten von `element`. Ist dieses nicht in der Deque enthalten, soll nichts verändert werden.
- `addLast(T element)` fügt `element` an das Ende der Liste an.
- `addLast(Collection<T> elements)` fügt alle Elemente der Collection an das Ende der Liste an.
- `removeLastOccurrence(T element)` entfernt das letzte Auftreten von `element` aus der Liste. Ist dieses nicht enthalten, so ändert sich nichts.
- `contains(T element)` prüft, ob `element` in der Liste enthalten ist.
- `containsAll(Collection<T> elements)` prüft, ob alle Elemente der Collection enthalten sind.

🪜 Aufgaben

Implementiere die oben genannten Kleiderschränke in `KleiderschrankTemplate`.

🔍 Lösungsvorschlag

Einen Lösungsvorschlag findest du in `Solution - Kleiderschrank`.