252-0027

Einführung in die Programmierung

Übungen

Woche 13: Sets, Maps, Collections & Klassen

Henrik Pätzold
Departement Informatik
ETH Zürich

Weihnachtskahoot

Sets - Überblick

- **Intuition**
  - Eine Instanz von einem Set soll sich in Java genauso verhalten eine mathematische Menge
  - Keine doppelten Elemente, keine zwingende Sortierung, usw.
- **Interface**
  - Sets sind Interfaces!
  - Das Set Interface gibt vor, welche Methoden wir auf einer Set-Instanz aufrufen können **müssen**
- **Was haben Sets, was Listen nicht haben?**
  - **Listen erlauben Duplikate**, Sets nicht.
  - **Indizes**: Listen haben eine feste Reihenfolge mit zugreifbaren Indizes (z. B. Index 0). Sets nicht.

HashSet

- **Intuition**
  - Eine Instanz von einem HashSet verhält sich wie eine **ungeordnete Menge ohne Duplikate**.
  - Nutzt **Hashing**, um Elemente effizient zu speichern und zu finden.
- **Besonderheiten von HashSet**
  - **Keine Reihenfolge**: Die Elemente haben keine feste Reihenfolge.
  - **Schnelle Operationen**: add, remove und contains haben in der Regel eine Zeitkomplexität von **O(1)**.
  - **Einzigartigkeit**: Duplikate werden automatisch entfernt.
  - **Null erlaubt**: Ein null-Element ist zulässig.
  - **HashSet** ist für jeden Typ benutzbar, weil jedes Objekt einen **HashCode** hat

TreeSet

- **Sortierung:**
  - Die Elemente im TreeSet sind immer **sortiert**.
  - Ein TreeSet sortiert basierend auf einem Binärbaum
- **Spezielle Methoden vom TreeSet:**
  - Methoden wie subSet(), headSet() und tailSet() erlauben es, Teilmengen basierend auf einem Bereich zu erstellen.

Wie iterieren wir über ein Set

- **Iterator – wenn man sich den Index merken will**
  - Sei s1: Ein Beispiel-HashSet.
  - s1.iterator() gibt ein Objekt zurück, das den Zustand der Datenstruktur bei Erstellung repräsentiert
  - Zugriff auf die Elemente über den **Iterator**:
    - hasNext() prüft, ob weitere Elemente vorhanden sind.
    - next() liefert das nächste Element.
    - remove() entfernt das zuletzt zurückgegebene Element (optional)
- **Ansonsten auch einfacher mit einer foreach loop**
  - **for(String s:s1)** nutzt hasNext() und next() des Iterators

```java
TreeSet<Integer> t1 = new TreeSet<>();
Iterator<Integer> i = t1.iterator();
while(i.hasNext()) {
    Integer curr = i.next();
    // mach etwas mit i
}
```

```java
TreeSet<Integer> t1 = new TreeSet<>();
for(Integer i : t1) {
    // mach etwas mit i
}
```

```java
TreeSet<Integer> t1 = new TreeSet<>();


t1.add(1);
t1.add(2);
t1.add(3);
t1.add(4);
Iterator<Integer> i = t1.iterator();
t1.add(5);

while(i.hasNext()) {
    Integer curr = i.next();
    System.out.println(curr);
}
```

Comparator

- **Wie weiß Java für jede Klasse, wie es zu sortieren hat?**
  - Gar nicht!
  - Wir müssen für eigene Klassen, definieren wie man die Objekte untereinander vergleicht
- **Was ist ein Comparator?**
  - Ein **Comparator** ist ein Funktionsobjekt, das verwendet wird, um die Reihenfolge von Objekten zu definieren.
- **Warum Comparator verwenden?**
  - Wenn die Klasse nicht die **natürliche Ordnung** (**Comparable**) implementiert.
  - Um mehrere oder flexible Vergleichslogiken zu ermöglichen.

Maps

Maps - Überblick

- **Intuition**
  - Eine Map verknüpft **Schlüssel** mit **Werten**.
  - Jeder Schlüssel ist eindeutig, aber Werte dürfen dupliziert werden.
- **Interface**
  - Maps sind Interfaces!
  - Das Map-Interface definiert Methoden für das Hinzufügen, Entfernen und Abfragen von Schlüssel-Wert-Paaren.
- **Was haben Maps, was Sets oder Listen nicht haben?**
  - **Schlüssel-Wert-Paare:** Listen und Sets speichern nur einzelne Elemente, Maps speichern Zuordnungen.
  - **Effiziente Schlüssel-basierte Abfragen:** Direkter Zugriff auf Werte über Schlüssel (z. B. map.get(key)).

Maps – Konkrete Implementationen

- **HashMap**
  - **Schlüssel-Wert-Paare:**
    Speichert Schlüssel-Wert-Paare basierend auf dem Hashcode des Schlüssels.
  - Analog zu HashSet, nur das wir mit den Keys auf die Buckets zugreifen können.
- **TreeMap**
  - **Schlüssel-Wert-Paare:**
    Speichert Schlüssel-Wert-Paare in **natürlicher Ordnung** oder nach einem **benutzerdefinierten Comparator**.
  - **Sortiert:**
    Die Einträge sind sortiert basierend auf der Ordnung der Schlüssel.

Maps – Wichtig

- **Option 1:** HashMap
  - Die Methoden equals und hashcode der Objekt-Klasse überschreiben.
- **Option 2:** TreeMap
  - Comparable-Interface implementieren und compareTo-Methode überschreiben.

Klassenaufgaben

Klassenaufgaben

- **Oft komplett lösbar ohne Vererbung**
  - Bedeutet aber mehr redundanter Code -> potenzielle Gefahr für Bugs
  - Wird meist schwierig mit den letzten Teilaufgaben.
- **Meist hierarchisch aufgeteilt**
  - Großes System verwaltet viele Objekte, die deterministisch Aufgaben erfüllen aber dadurch aufeinander Einfluss haben können -> sie müssen Ihren zustand dem System kommunizieren können.
- 🤫 🤫 **Cheat** 🤫 🤫**:** Objektreferenz der oberen Managment-Klasse an die „kleinen“ Objekte weitergeben.

ItemFactory*

Graphenaufgaben

Trick zum lösen *vieler* Graphenaufgaben

- **Rekursion**
  - Gegebene Daten (oft einzelner Knoten) rekursiv abarbeiten und wichtige Merkmale speichern. **Wieso rekursiv?**
- **Daten – wenn nötig – in richtige Form bringen**
  - Graph in einer Matrix o.Ä. speichern
- **Bedingungen iterativ überprüfen**
  - Die zu überprüfenden Bedingungen stehen in der Aufgabenbeschreibung
  - Iterativ kann ist das jetzt einfacher zu lösen durch konsequentes Überprüfen der Bedingungen

Schaut gerne im Spick ☺