equals + hashCode
Hash-Tabelle

`boolean equals(Object obj)`

Implementierung in `Object`: `{ return this == obj; }`

meist unterscheidet sich Gleichheit von Identität → Überschreiben

Bedingungen: auch alle daraus ableitbaren Bedingungen

- `x != null` → `!x.equals(null)` — keine Gleichheit mit `null`
- `x != null` → `x.equals(x)` — Identität ⇒ Gleichheit
- `x != null && y != null` → `x.equals(y) == y.equals(x)` — symmetrisch
- `x.equals(y) && y.equals(z)` → `x.equals(z)` — transitiv
- `x` und `y` unverändert → `x.equals(y) == x.equals(y)` — keine Seiteneffekte

Typische Implementierung von `equals`

```
public boolean equals(Object obj) {
    if (this == obj) {
        return true;   // Optimierung, garantiert x.equals(x)
    }
    if (obj == null || getClass() != obj.getClass()) {
        return false;  // dyn. Typen gleich, garantiert !x.equals(null)
    }
    NameOfThisClass that = (NameOfThisClass) obj; // access through that
    ...                // type-specific comparison of this and that
```

Aufgabe: Typ des formalen Parameters von `equals`

Wir benötigen in `equals` „unsaubere“ Techniken wie Typabfragen (`getClass`) und Casts, weil der formale Parameter vom Typ `Object` ist.

Warum ist der Typ des Parameters nicht die Klasse, in der `equals` implementiert ist?

Was „Gleichheit“ bedeutet

entscheidende Fragen:

Unter welchen Bedingungen werden zwei Objekte als gleich angesehen?
Welche Objektvariablen in Vergleich einbeziehen?

Antworten nicht allgemeingültig, von Abstraktion abhängig

häufig werden alle Variablen einbezogen, beim automatischen Generieren

deren Werte (direkt oder indirekt) nach außen sichtbar werden könnten
wobei die Strukturen übereinstimmen (z.B. gleiche Reihenfolge in Liste)

aber eigene Ansätze nötig wenn Gleichheit keine gleiche Struktur voraussetzt

z.B. zwei Suchbäume gleich wenn gleiche Einträge vorhanden,
aber Reihenfolge des Einfügens (= Struktur des Baums) nicht relevant sein soll

Aufgabe: Gleichheit trotz unterschiedlicher Struktur

Listen wurden nur als gleich betrachtet, wenn die Reihenfolgen der Einträge gleich waren. Für die Gleichheit von Suchbäumen spielte die Struktur aber keine Rolle.

Worin könnte die Ursache für diesen Unterschied liegen?

`int hashCode()`

gibt fast beliebige Zahl zurück, z.B. in Hash-Tabellen benötigt

Implementierung in `Object`: Zahl von Speicheradresse abgeleitet

Bedingungen:

`x.equals(y)` $\rightarrow$ `x.hashCode() == y.hashCode()`

`x` unverändert $\rightarrow$ `x.hashCode() == x.hashCode()`

gewünscht, aber nicht garantierbar:

`!x.equals(y)` $\rightarrow$ `x.hashCode()` möglichst verschieden von `y.hashCode()`

wenn `equals` überschrieben, dann muss auch `hashCode` überschrieben werden;
Bedingungen für Gleichheit beachten, keine anderen Variablen als in `equals` einbeziehen

Richtige und falsche Annahmen

Folgende Bedingung immer erfüllt:

```
x.hashCode() != y.hashCode()  →  !x.equals(y)
```

Folgende Bedingung manchmal erfüllt (`toString` liefert nur für Gleichheit relevante Info):

```
x.equals(y)  →  x.toString().equals(y.toString())
```

Folgende Bedingungen **dürfen nicht angenommen werden**:

```
x.hashCode() == y.hashCode()  →  x.equals(y)
x.toString().equals(y.toString())  →  x.equals(y)
```

`x.hashCode()` ist die Speicheradresse von `x`

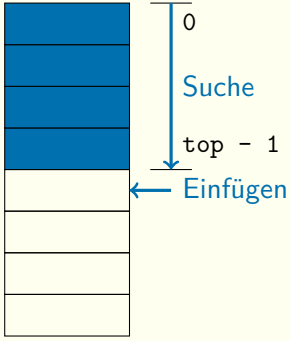
Aufgabe: Generierte Methoden

Methoden wie `toString()`, `equals()` und `hashCode()` sind einfach generierbar.

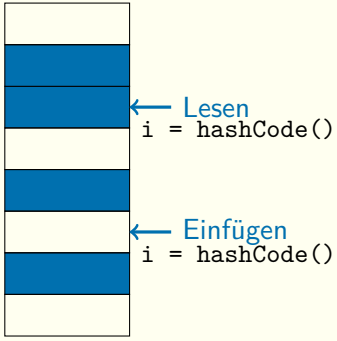
Warum ist es dennoch nötig zu wissen, welche Eigenschaften sie haben müssen?

Funktionsweise von Hash-Tabellen

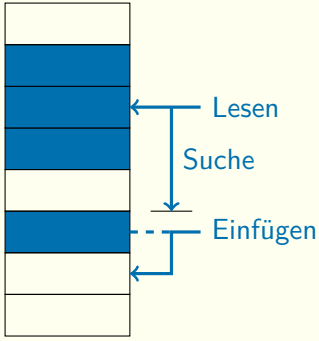
SimpleAssoc



HashTab (Idee)



HashTab (Kollision)



Aufgabe: Re-Hashing

Warum wird beim Vergrößern des Arrays (`HashTab`) alles neu eingetragen statt bestehende Einträge zu kopieren?

Aufgabe: Wo liegt der Fehler?

Warum wurden in der Hash-Tabelle keine Einträge gefunden, im Suchbaum aber schon?

Vergleich

	lineare Liste	Suchbaum	Hash-Tabelle
Reihenfolge der Daten bleibt erhalten	+	−	−
Daten in sortierter Reihenfolge zugreifbar	−	+	−
Suche meist effizient	−	+	+
sicher vor Ausreißern (Laufzeit)	+	−	−
sicher vor Ausreißern bei üblichen Daten	+	−	+
wenig anfällig für Fehler	+	−	−