Collections

openHPI-Java-Team

Hasso-Plattner-Institut

Collections (Sammlungen)

- Organisieren Objekte beliebiger (Objekt-)Datentypen
- Collections sind selbst auch Objektdatentypen
  → besitzen Methoden und Attribute
- Verschiedene Collections haben unterschiedliche Eigenschaften:
  - Array
  - List
  - Map
  - Set

- Elementdatentyp in spitzen Klammern <>:
  - Definiert den Typ der in der Collection erlaubten Elemente
  - Bsp: `ArrayList<String> words = new ArrayList<>();`

Wiederholung Primitive Arrays

```
1  int[] numbers = new int[5];
2  numbers[0] = 6;
3  numbers[1] = 33;
4  numbers[2] = 9;
5  numbers[3] = 0;
6  numbers[4] = 503;
```

Index	Wert
0	6
1	33
2	9
3	0
4	503

- Primitive Arrays können ihre Größe **nicht** verändern → zusätzliche Elemente können nur unter sehr großem Aufwand angefügt werden
- `length` gibt mir nur die Größe, aber nicht die Anzahl der real existierenden Elemente
- Soll ein Element vorne eingefügt werden, müssen die folgenden Elemente manuell verschoben werden
  → Was geschieht mit dem letzten Element?

Array (1/3)

- Array mit veränderlicher Größe
- Schneller Zugriff auf Elemente an beliebiger Position
- Ist ein Array voll wird automatisch ein neues angehängt

Index	Wert
0	6
1	33
2	9
3	0
4	503

5	22
6	31
7	333
8	6
9	5603

ArrayList (2/3)
ein Codebespiel

Hasso Plattner Institut

```
1 ArrayList<String> list = new ArrayList<>();
2 list.add("Hallo");
3 list.add("Huhu");
4 list.size();      → 2
5 list.get(0);      → Hallo
6 list.remove(0);
7 list.get(0);      → Huhu
```

Neue Elemente anfügen

Anzahl der Elemente

Erstes Element ausgeben

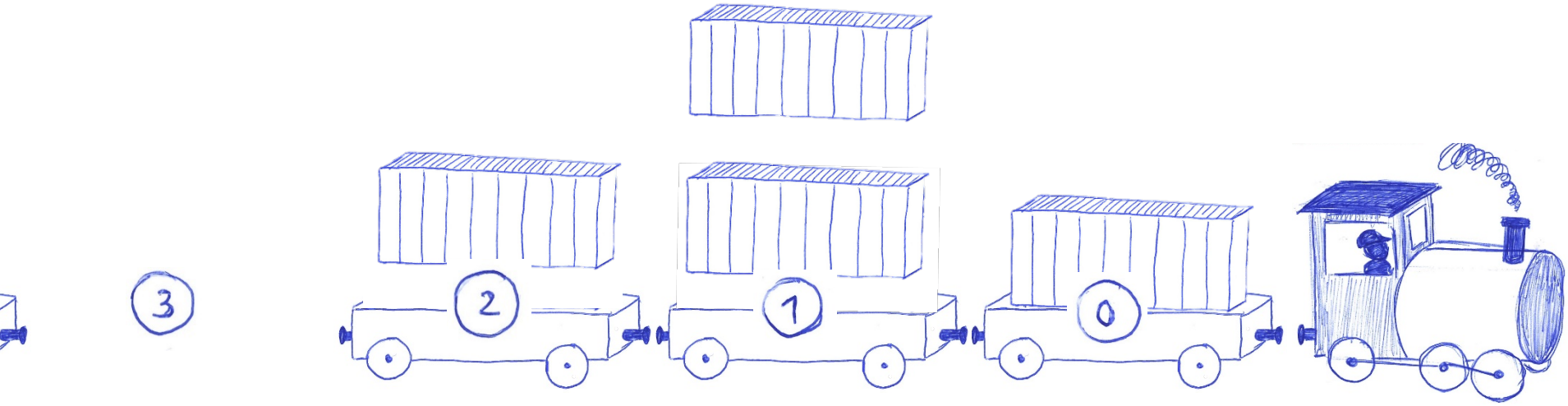
Erstes Element löschen

Erstes Element ausgeben

ArrayList (3/3)

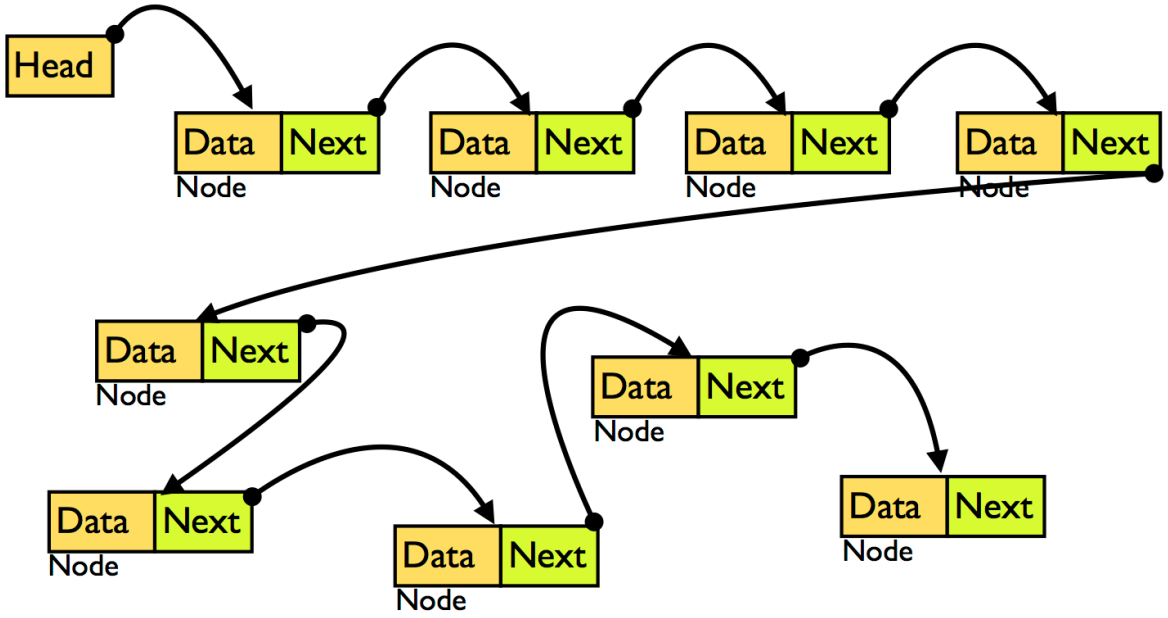
- Elemente vorn einfügen kann „teuer“ werden
- Nachfolgende Elemente werden automatisch nach hinten verschoben

➔ Dieses Problem wurde sozusagen automatisiert aber nicht behoben



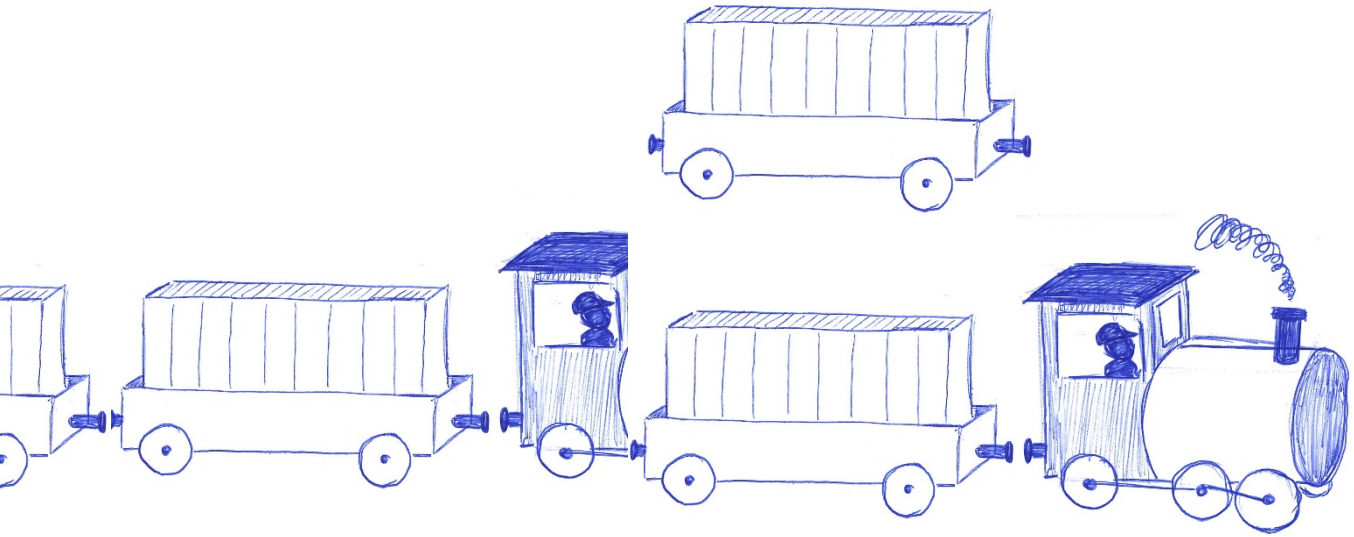
List (1/3)

- Verkettet **Nodes** miteinander
- Nodes enthalten **Referenz auf folgenden Node** (Next) und Referenz auf das enthaltene **Datenobjekt** (Data)
- Nachteil: Kein Random Access → um zu einem Element zu gelangen muss über alle vorhergehenden Elemente iteriert werden



List (2/3)

- Vorteil: Einfaches Einfügen und Entfernen von Elementen auch am Anfang oder in der Mitte der Liste
- Node einfügen/löschen → Referenz auf nächsten Node „umbiegen“



LinkedList (3/3)
ein Codebeispiel

HPI Hasso Plattner Institut

```
1 LinkedList<String> list = new LinkedList<>();
2 list.add("Ronja");
3 list.add("Robin");
4
5 list.size();
6 list.getFirst();
7 list.removeFirst();
```

Neue Elemente einfügen

Anzahl der Elemente

Erstes Element ausgeben

Erstes Element löschen

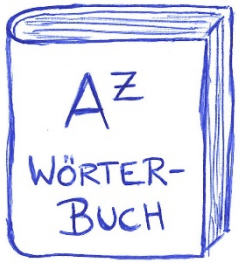
Spezielle Listen:

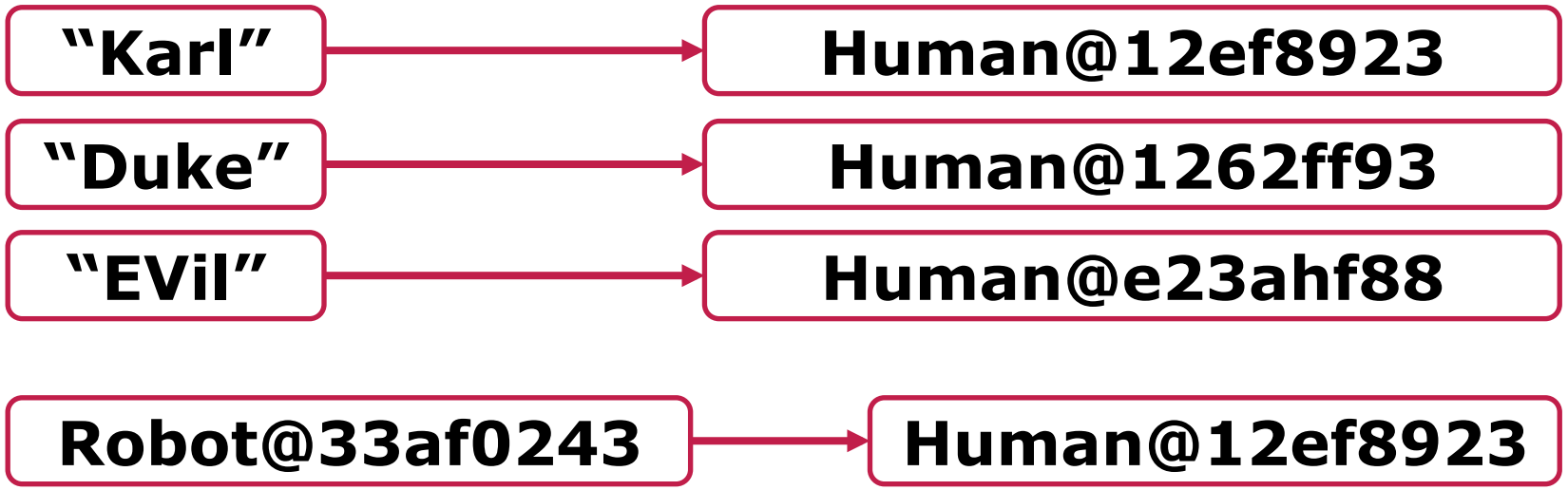
- Queue: First In First Out (FIFO)
  - Schlange

- Stack: Last In First Out (LIFO)
  - Stapel

Map (1/2)

- Jedem Wert wird ein Schlüssel zugeordnet (KEY-VALUE Paar)
  - Schlüssel (KEY) ermöglicht Zugriff auf Wert (VALUE).
- Im Gegensatz zum Array ist der Schlüssel (in der Regel) nicht-numerisch
- Elemente sind „unsortiert“





HashMap (2/2)

```
1 HashMap<Customer, Robot> orders = new HashMap<>();
2 orders.put(duke, new Robot("Ronja"));
3 orders.put(eVil, new Robot("Robur"));
4 orders.size();
5 orders.get(duke);
6
7 orders.remove(eVil);
8
9 orders.containsKey(eVil);
```

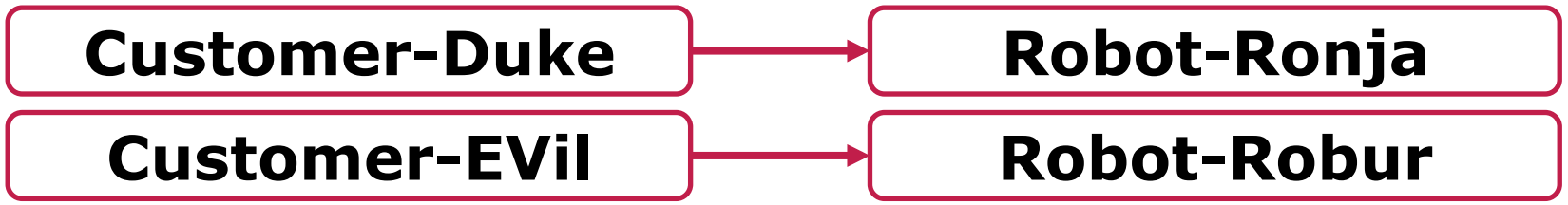
Neue Elemente einfügen

Anzahl der Elemente

Element mit dem KEY duke ausgeben

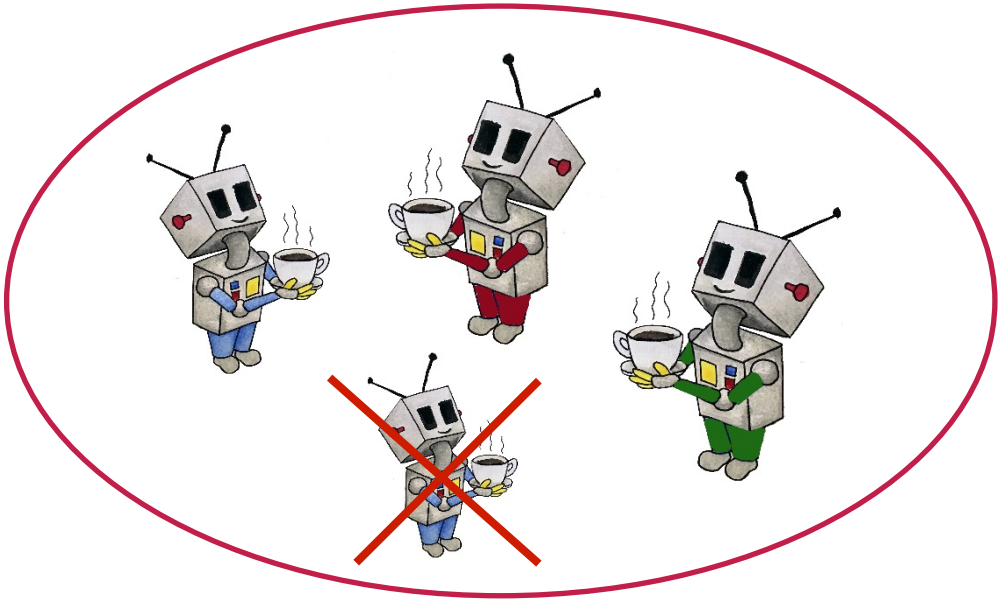
Element mit dem KEY eVil löschen

Prüfen ob ein KEY existiert



Set (1/1)

- Menge
- Jedes Element kann nur **einmal** enthalten sein ➔ keine Duplikate



- Verschiedene Implementierungen in Java, z. Bsp.:
  - HashSet: unsortiert

Collections in der Java API

- Collections sind in der Regel die bessere Wahl als primitive Arrays
- Automatisierung vieler House-Keeping Funktionen
- Collections können nur Objektdatentypen beinhalten
  - Wrapper-Klassen für primitive Datentypen
- **ArrayList** – für Random Access mittels durchgängiger numerischer Schlüssel
- **LinkedList** – wenn häufig Elemente im „vorderen" Bereich eingefügt werden müssen
- **HashMap** – wenn der Zugriff über nicht-numerische Schlüssel wichtig ist
- **HashSet** – wenn sicher gestellt werden soll, dass es keine Duplikate gibt

Collections in der Java API

- JavaAPI
  https://docs.oracle.com/javase/8/docs/api/
- ArrayList
  https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
- LinkedList
  https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html
- HashMap
  https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html
- HashSet
  https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html