

Generische Klassen&Interfaces

Lukas Abelt

`lukas.abelt@airbus.com`

DHBW Ravensburg
Wirtschaftsinformatik

Ravensburg
20. März 2019

Inhalt

1 Java-Klassenbibliothek

2 Generische Klassen

- Motivation
- Syntax&Eigenschaften
- Generische Interfaces
- Wildcards
- Einschränkungen

Inhalt

1 Java-Klassenbibliothek

2 Generische Klassen

- Motivation
- Syntax&Eigenschaften
- Generische Interfaces
- Wildcards
- Einschränkungen

Java-Klassenbibliothek

Kurzüberblick

- Java bietet Vielzahl an „fertigen“ Klassen
- Zusammengefasst in Packages
- Diese implementieren Standardfunktionalitäten wie z.B.
 - Ein- und Ausgabefunktionalitäten
 - Grafische Oberflächen
 - Netzwerkkommunikation
 - Datum- und Zeit, Internationalisierung
 - Und viele mehr...

Wichtige Packages I

... der Klassenbibliothek

Die wichtigsten Standardpackages im schnellen Überblick:

- ▣ *java.lang*: Integriert die fundamentalen Klassen, die in der Regel immer zur Java Entwicklung benötigt werden wie zum Beispiel String Object oder auch die Wrapper Klassen der primitiven Datentypen (Integer, Boolean, Double usw.). **Muss nicht explizit importiert werden!**
- ▣ *java.util*: Häufig benötigte Klassen, wie Listenstrukturen (List, Stack), Klassen zur Verarbeitung von Datum und Uhrzeit (Calendar) oder Zufallszahlengeneratoren (Random)
- ▣ *java.io*: Klassen zur Ein- und Ausgabe über Streams
- ▣ *java.net*: Klasse zur Implementierung von Netzwerkkommunikation

Wichtige Packages II

... der Klassenbibliothek

- *java.rmi*: Klassen zur Entwicklung verteilter Programme unter Nutzung von Remote Method Invocation
- *java.awt*: Grundlegendes Package für die Entwicklung grafischer Oberflächen
- *java.swing*: Erweiterte Komponente zur Entwicklung von grafischen Oberflächen. Baut auf *java.awt* auf, bietet jedoch mehr Funktionalität
- *javax.crypto* und *java.security*: Klassen zur Umsetzung von sicherheitsrelevanten Aspekte (Zugriffsschutz, Rechteverwaltung etc.)
- *java.sql*: Package zur Interaktion mit SQL Datenbanken

Arbeiten mit der Klassenbibliothek

- Packages über `import` in Klasse einbinden
- Oracle bietet umfangreiche Dokumentation zu allen Klassen der Standard-API
- Schnellster Weg zur Doku:
 - In den meisten IDEs sowieso integriert
 - Sonst Google: „Java 8/9/10 API *PackageName*“

API-Dokumentation

Links

Java 8 API

<https://docs.oracle.com/javase/8/docs/api/index.html>

Java 9 API

<https://docs.oracle.com/javase/9/docs/api/index.html>

Java 10 API

<https://docs.oracle.com/javase/10/docs/api/index.html>

Inhalt

1 Java-Klassenbibliothek

2 Generische Klassen

- Motivation
- Syntax&Eigenschaften
- Generische Interfaces
- Wildcards
- Einschränkungen

Inhalt

1 Java-Klassenbibliothek

2 Generische Klassen

- Motivation
- Syntax&Eigenschaften
- Generische Interfaces
- Wildcards
- Einschränkungen

Generische Klassen

Was ist das und Warum?

- Ist eine Methode Klassen deutlich versatiler zu machen
 - Und dadurch wiederverwenbar
 - Bei geringerem Implementierungsaufwand
- Bisher: Festlegen von Datentypen bei Design der Klasse
- Jetzt: Festlegen von Datentypen bei Verwendung der Klasse (Zumindest für einige)
- Funktioniert für:
 - Member Variablen
 - Funktionsparameter
 - Rückgabewerte

Generische Klassen

Beispiel: Benannte Werte

- Angenommen man erhält folgende Anforderungen für eine Klasse
 - Die Klasse soll einen Wert speichern
 - Dieser soll vom Typ `Integer`, `String` oder `Boolean` sein
 - Die Klasse soll einen Namen für diesen Wert als `String` speichern können
- Mögliche Ansätze ohne generische Klassen:
 - Implementierung einer Klasse `NamedValue`, die drei Member der entsprechenden Typen hat
 - Implementierung einzelner Klassen `NamedInteger`, `NamedString` und `NamedBoolean`

Variante 1: One class for all!

Implementierung

```
1  class NamedValue{
2      private Integer intValue;
3      private String stringValue;
4      private Boolean boolValue;
5      private String name;
6
7      void set(Integer newInt);
8      void set(String newString);
9      void set(Boolean newBool);
10     void setName(String newName);
11
12     Integer getIntegerValue();
13     String getStringValue();
14     Boolean getBooleanValue();
15     String getName();
16 }
```

Variante 1: One class for all!

Probleme

- Es sollte **ein** Wert gespeichert werden
 - Unsere Klasse speichert (theoretisch) bis zu drei verschiedene Werte
 - *Könnte* man abfangen
 - Erhöht jedoch weiter den Implementierungsaufwand
- Keine einheitliche Schnittstelle um Wert abzurufen
- Erhöhter Aufwand bei Erweiterung der Klasse
 - Wert soll jetzt auch von Typ `Color` sein
 - Hinzufügen neuer Member Variable `colorValue`
 - Hinzufügen neuer set-/get-Methoden

Variante 2: Viel hilft viel!

Implementierung NamedInteger

```
1  class NamedInteger{
2      private Integer value;
3      private String name;
4
5      void set(Integer newValue);
6      Integer get();
7
8      void setName(String newName);
9      String getName();
10 }
```

Variante 2: Viel hilft viel!

Implementierung NamedString

```
1  class NamedString{
2      private String value;
3      private String name;
4
5      void set(String newValue);
6      String get();
7
8      void setName(String newName);
9      String getName();
10 }
```


Variante 2: Viel hilft viel!

Implementierung NamedBoolean

```
1  class NamedString{
2      private Boolean value;
3      private String name;
4
5      void set(Boolean newValue);
6      Boolean get();
7
8      void setName(String newName);
9      String getName();
10 }
```

Variante 2: Viel hilft viel!

Probleme

- ❑ Löst einige Probleme der ersten Variante...
 - ❑ Tatsächlich nur ein Wert gespeichert
 - ❑ Einheitliche Schnittstelle
- ❑ ...Aber eben nicht alle
- ❑ Copy-Paste-Code → Nach Möglichkeit zu vermeiden
- ❑ Problem bei Erweiterung bleibt ähnlich
 - ❑ Würde hier neue Klasse erfordern

Generische Klassen

Die Lösung des Problems!

- Angabe von „Platzhaltern“ bei Definition der Klasse
 - Namen sind theoretisch beliebig wählbar
 - ...Es gibt jedoch Naming conventions dazu
- Diese repräsentieren den Datentypen
- Die Spezifizierung des Typs erfolgt erst bei Deklaration einer Variable vom Typ der Klasse

Generische Klassen

Syntax

```
1  class NamedValue<T>{
2      private T value;
3      private String name;
4
5      void set(T newValue);
6      T get();
7
8      void setName(String newName);
9      String getName();
10 }
11 //Verwendung:
12 NamedValue<Integer> namedInteger;
13 NamedValue<String> namedString;
14 NamedValue<Boolean> namedBoolean;
```

Eigenschaften von generischen Klassen I

- Schnittstellen bleiben einheitlich (Im Rahmen des spezifizierten Typs)
- Kein Problem mit Anpassungen an neue Typen → Keine Änderung notwendig
- Definieren mehrerer generischer Typen möglich:

```
1 class name<T1, T2, ..., Tn> {/*Klasseninhalt*/}
```

Eigenschaften von generischen Klassen II

- Naming Conventions für Typen:
 - In der Regel ein Buchstabe
 - T - Type
 - E - Element
 - N - Number
 - K - Key
 - V - Value

Kontakt

- E-Mail: lukas.abelt@airbus.com
- GitHub: <https://www.github.com/LuAbelt>
- GitLab: <https://www.gitlab.com/LuAbelt>
- Telefon(Firma): 07545 - 8 8895
- Telegram: LuAbelt