



Bibliotheksklas- sen nutzen

Buch, Kapitel 6

J. Kleimann, H.-W. Sehring, D. Versick, F. Zimmermann

Studiengang Wirtschaftsinformatik

Vorlesungsinhalt

- 1 Lerninhalte
- 2 Bibliotheken
- 3 Pakete und Import
- 4 Wichtige Bibliotheksklassen
- 5 Dokumentation von Klassen
- 6 Klassendokumentationen erstellen
- 7 Zugriffsmodifikatoren
- 8 Quelltext schreiben: Code completion
- 9 Klassenvariablen, -methoden und Konstanten
- 10 Polymorphe Collections

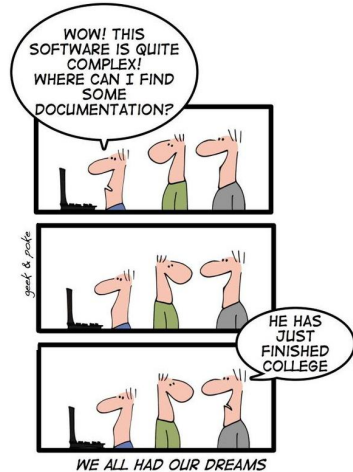


Abbildung: CC-BY-3.0 Oliver Widder

Lernziele

Nach Beenden dieser Lektion werden Sie

- mit Bibliotheksklassen umgehen können, die Sie vorher nicht gekannt haben.
- Java Dokumentation lesen und verfassen können.
- die Klasse `String` nutzen können.
- (pseudo) zufälliges Verhalten mit der Klasse `Random` erzeugen können.
- weitere `Collection` Klassen nutzen lernen: `HashMap`, `HashSet`.
- Autoboxing und Wrapper-Klassen nutzen können.

Die Java Klassenbibliothek

Java zu lernen bedeutet nicht nur die Sprachkonzepte zu kennen, sondern auch sich in der Java Klassenbibliothek zurecht zu finden. Diese enthält vordefinierte Klassen, die in verschiedenen fachlichen Kontexten wiederverwendet werden können und daher die Programmierer entlasten. Die Java Klassenbibliothek, das sogenannte JDK,

- besteht aus tausenden von Klassen.
- besteht aus zehntausenden Methoden.
- verhindert, dass Programmierer das Rad neu erfinden müssen.
- stellt Klassen zur Verfügung, die eine Sache gut erledigen.
- stellt Klassen zur Verfügung, die wie Lego Steine kombinierbar sind, um flexibel einsetzbar zu sein.
- ist in Paketen (packages) organisiert.

Das Arbeiten mit Bibliotheksklassen

- Ein Java Programmierer muss sich in den Java Klassenbibliotheken zurecht finden. Was heißt das?
- Er sollte
 - **wichtige Klassen** mit Namen **kennen**.
 - wissen, wie man etwas über andere **unbekannte Klassen** erfährt.
 - diese Klassen in eigenen Projekten **nutzen** können.
 - sich **nicht** darum kümmern, wie die Klassen umgesetzt sind.
- Ein erfahrener Programmierer sollte auch mit Klassenbibliotheken von Dritten, also z.B. von Google, Apache, Eclipse foundation, SAP usw. umgehen können. Dabei sind die Methoden, das Vorgehen und die Fertigkeiten dieselben, wie bei den JDK Bibliotheken.

Das Übungsbeispiel

Ein IT-System für den technischen Support.

- ein textbasiertes interaktives Dialogsystem.
- inspiriert durch **Eliza**¹ von Joseph Weizenbaum² (MIT, 1960er).
- Probieren Sie das Projekt *Technischer-Kundendienst* aus ...
- ...Das Programm scheint auf den ersten Blick intelligent auf die Nutzereingaben zu reagieren.

¹<https://de.wikipedia.org/wiki/ELIZA>

²Namensgeber des Weizenbaumgebäudes der Nordakademie; bekannt für seinen kritischen Umgang mit dem Thema Informatik und Gesellschaft **6**

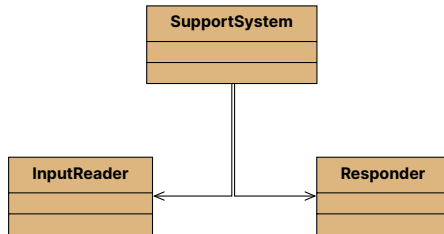
Hauptschleife in start() Methode

Eine häufige Schleifenkonstruktion für textorientierte Dialoge

```
public void start() {  
    boolean finished = false;  
  
    while (!finished) {  
        //do something  
        if (exit condition) {  
            finished = true;  
        } else {  
            //do something more  
        }  
    }  
}
```

Modularisierung

Das Supportsystem verwendet folgende Klassen.



Rumpf der Hauptschleife im SupportSystem

Der input wird in der Antwort nicht berücksichtigt

```
boolean finished = false;  
  
while (!finished) {  
    String input = reader.getInput();  
    if (Abbruchbedingung) {  
        finished = true;  
    } else {  
        String response = responder.generateResponse();  
        System.out.println(response);  
    }  
}
```

Die Abbruchbedingung

ein wenig Fehlertoleranz

```
String input = reader.getInput();  
if (input.startsWith("bye")) {  
    finished = true;  
} else {  
    ...  
}
```

- Wo kommt 'startsWith' her?
- Was ist das? Was macht es?
- Wie kann man das herausfinden?

Nutzung von Klassen aus Bibliotheken (Libraries)

- Klassen sind in Paketen (was nur ein anderer Name für ein Verzeichnis ist) organisiert.
- Pakete (Packages) haben Namen und können ihrerseits in Packages organisiert werden.
- Typische Packages sind `java.util`, `java.io`, `java.sql`, `java.lang`, ...
- voll qualifizierter Klassenname ist `paketname.klassenname` : `java.util.ArrayList`
- Klassen aus einem fremden **package** müssen importiert werden.
- Importieren einer Klasse wird über eine **import** Anweisung gemacht. Beispiel: **import** `java.util.ArrayList`;
- Einzige Ausnahme: Klassen aus `java.lang` wie `java.lang.String` brauchen nicht importiert werden.
- Importierte Klassen können wie Klassen aus dem Projekt benutzt werden.

Packages und import

Eine Klasse zu importieren, bedeutet nicht, dass man den Code der Klasse einbindet. (Für Besserwisser: Importieren bedeutet, dass die Klasse in den aktuellen Namensraum eingebunden wird, und man deshalb die Klasse unter nur dem Klassennamen ansprechen kann und nicht den voll qualifizierten Klassennamen angeben muss.)

Importieren einer einzelnen Klasse

```
import java.util.ArrayList;
```

Importieren aller Klassen eines Packages

```
import java.util.*;
```

Erzeugen von zufälligem Verhalten

- Mit Objekten der Bibliotheksklasse Random kann man Folgen von Zufallszahlen erzeugen.
- Die Folgen sind

Erzeugen einer Zufallszahl mit Werten zwischen 1 und 100

```
import java.util.Random;  
...  
Random rand = new Random();  
...  
int num = rand.nextInt();  
int value = 1 + rand.nextInt(100);  
int index = rand.nextInt(list.size());
```

Zufällige Antworten auswählen

Klasse Responder

```
public Responder() {  
    randomGenerator = new Random();  
    responses = new ArrayList<>();  
    fillResponses();  
}  
  
public void fillResponses() {  
    //fill responses with a selection  
    of response strings  
}  
  
public String generateResponse( ){  
    int index = randomGenerator.nextInt  
        (responses.size());  
    return responses.get(index);  
}
```

Wiederholung

- Java hat eine umfangreiche Klassenbibliothek.
- Ein Programmierer muss sich in der Bibliothek auskennen.
- Die Dokumentation der Bibliotheksklassen liegt im HTML Format vor und enthält alle notwendigen Informationen über die Benutzung einer Klasse (im Kern das Interface).
- Einige Klassen haben Typparameter.
 - Ein anderer Name für eine parametrisierte Klasse ist generische Klasse oder generischer Typ.

- Weitere Klassen um Sammlungen von Objekten zu verarbeiten:
 - **Set** – doppelte vermeiden
 - **Map** – Schlüssel-Wert Paare
- Dokumentation schreiben:
 - javadoc

Nutzen von Mengen (sets)

Set ist eine Collection

```
import java.util.HashSet;
...
HashSet<String> mySet = new HashSet<>();

mySet.add("one");
mySet.add("two");
mySet.add("three");

for (String element : mySet) {
    do something with element
}
```

Bitte beachten Sie, dass die Verarbeitung nicht in der Reihenfolge passiert, in der die Elemente angelegt werden. Sets sind in der Regel nicht sortiert.

Strings in Bestandteile zerlegen

```
public HashSet<String> getInput() {  
    System.out.print("> ");  
  
    String inputLine =  
        reader.nextLine().trim().toLowerCase();  
    String[] wordArray = inputLine.split(" ");  
    HashSet<String> words = new HashSet<>();  
    for (String word : wordArray) {  
        words.add(word);  
    }  
    return words;  
}
```

Maps

- Maps sind Sammlungen von Objekten.
- Maps haben zwei Typparameter K und V.
- Maps enthalten Paare aus K (Key) und V (Value) Objekten.
- Zu einem Key Objekt kann das zugehörige Value Objekt gesucht werden.
- Beispiel: Eine Liste mit Telefonnummern zu Kontakten.

Key-Value Paare

- Eine Map mit Strings als Key und Strings als Values

:HashMap

"Erika Musterfrau"	"(44) 2384 4850"
"Peter Mustermann"	"(43) 5634 7850"
"Maria Musterperson"	"(49) 3454 6657"

HashMap ist ein Beispiel für eine Map

Beispiel suchen in einer HashMap

```
HashMap <String, String> contacts = new HashMap<>();  
  
contacts.put("Erika Musterfrau", "(44) 2384 4850");  
contacts.put("Peter Mustermann", "(43) 5634 7850");  
contacts.put("Maria Musterperson", "(49) 3454 6657");  
  
String number = contacts.get("Maria Musterperson");  
System.out.println(number);
```

HashMap in der Klasse Responder

Antwort zu Stichwort

```
private HashMap <String, String> responseMap;  
...  
responseMap.put("crash", "Which version is this?");  
...  
String response = responseMap.get(word);  
if (response != null) {  
    ...  
}
```

List, Map **und** Set

- Unterschiedliche Arten Sammlungen von Objekten zu verwalten.
- Java bietet unterschiedliche Implementierungen mit unterschiedlichen Vor- und Nachteilen.
- Das Modul Algorithmen und Datenstrukturen beschäftigt sich mit der Frage wann welche Organisationsform besser ist.
 - ArrayList
 - LinkedList
 - HashSet
 - TreeSet
- Der zweite Wortbestandteil beschreibt die Organisationsform, der erste die Implementierung.

Collections und Primitive Typen

- Die Klassen des Collection Frameworks können mit allen Objekttypen verwendet werden.
- Sie funktionieren aber nicht mit den primitiven Datentypen: **int**, **boolean**, etc.
- Gibt es eine Möglichkeit bspw. **int** Werte trotzdem in einer ArrayList zu speichern?

Collections und Primitive Typen

- Die Klassen des Collection Frameworks können mit allen Objekttypen verwendet werden.
- Sie funktionieren aber nicht mit den primitiven Datentypen: **int**, **boolean**, etc.
- Gibt es eine Möglichkeit bspw. **int** Werte trotzdem in einer ArrayList zu speichern?
- **Ja.** Man muss aber einen kleinen Trick anwenden.

Wrapper Klassen

- Primitive Typen sind keine Objekt Typen. Deshalb müssen die Werte primitiver Typen in ein Objekt eingepackt (wrap) werden, bevor man sie in einer Collection ablegen kann.
- Wrapper Klassen gibt es für alle primitiven Typen:

primitiver Typ	Wrapper Klasse
int	Integer
float	Float
char	Character
...	...

- Die korrespondierenden Namen sind intuitiv aber leider nicht sehr systematisch gewählt.

Autoboxing und unboxing

Prinzipiell ist es möglich das (Un-)Boxing manuell durchzuführen:

Manuelles Boxing und Unboxing

```
int i = 18;  
Integer iwrap = new Integer(i);  
...  
int value = iwrap.intValue();
```

In der Praxis ist manuelles Boxing und Unboxing meist nicht nötig:

Autoboxing und -unboxing

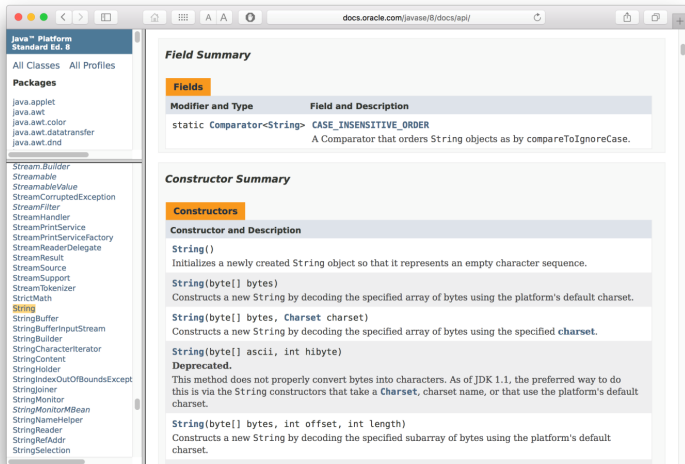
```
ArrayList<Integer> importantIntegers  
    = new ArrayList<>();  
  
importantIntegers.add(42);  
int importantInteger =  
    importantIntegers.get(0);
```

Lesen der Klassendokumentation

- Die Dokumentation von Java Klassen des JDK und anderer Bibliotheken liegt im HTML Format vor.
- API: Application Program Interface (auch Schnittstelle genannt)
- Für JDK Version 8: <https://docs.oracle.com/javase/8/docs/api>
- Die Dokumentation kann zusammen mit dem JDK auch lokal installiert werden. Ein Vorgehen, das insbesondere Anfängern zu empfehlen ist. Experten machen das sowieso.
- Sie kann mit dem Web Browser gelesen werden und Links führen zu verwandten Themen.
- Die Dokumentation enthält die API Beschreibung für alle Klassen der Bibliothek.

API Reference

<https://docs.oracle.com/en/java/javase/index.html/>



The screenshot shows the Java API Reference for the `String` class. The left sidebar lists various packages and classes, with `String` selected. The main content area is divided into two sections: **Field Summary** and **Constructor Summary**.

Field Summary

Modifier and Type	Field and Description
static <code>Comparator<String></code>	<code>CASE_INSENSITIVE_ORDER</code> A <code>Comparator</code> that orders <code>String</code> objects as by <code>compareToIgnoreCase</code> .

Constructor Summary

Constructors

Constructor and Description
<code>String()</code> Initializes a newly created <code>String</code> object so that it represents an empty character sequence.
<code>String(byte[] bytes)</code> Constructs a new <code>String</code> by decoding the specified array of bytes using the platform's default charset.
<code>String(byte[] bytes, Charset charset)</code> Constructs a new <code>String</code> by decoding the specified array of bytes using the specified <code>charset</code> .
<code>String(byte[] ascii, int hibyte)</code> Deprecated. This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the <code>String</code> constructors that take a <code>Charset</code> , charset name, or that use the platform's default charset.
<code>String(byte[] bytes, int offset, int length)</code> Constructs a new <code>String</code> by decoding the specified subarray of bytes using the platform's default charset.

Interface / Schnittstelle

Die Dokumentation enthält

- den Namen einer Klasse und
- eine Beschreibung der Klasse aus der Sicht des Nutzers der Klasse und
- die Konstruktoren der Klasse
- Rückgabewerte und Parameter der Konstruktoren und Methoden
- die Beschreibung der Konstruktoren und der Methoden der Klasse

Das alles macht die Beschreibung des **Interfaces** (der Schnittstelle) der Klasse aus. Dabei wird nur das Verhalten nach außen beschrieben.

Interface vs Implementation

Die Dokumentation enthält **nicht**

- private Exemplarvariablen (die meisten - eigentlich alle - der Exemplarvariablen sollten **private** sein)
- private Methoden
- die Funktionsweise der Implementierung der Methoden

Dies betrifft also Aussagen über die **Implementation** der Klasse, die nicht das Verhalten nach außen hin betreffen.

Dokumentation von `startsWith`

- `startsWith`

public boolean `startsWith(String prefix)`

- Tests if this string starts with the specified prefix.
- Parameters:
 - prefix - the prefix.
 - Returns:
 - **true** if the ... ;
 - **false** otherwise

Andere Methoden der Klasse String

Die Klasse String stellt viele Methoden zur Manipulation zur Verfügung. Wichtige davon sind:

- `contains(String s)`
- `endsWith(String suffix)`
- `indexOf(String str)`
- `substring(int b, int e)`
- `toUpperCase()`
- `trim()`

Achtung: Strings sind (**unveränderlich (immutable)**): jede Manipulation eines Strings erzeugt ein neues Objekt! Das alte Objekt bleibt, wie es war.

Dokumentation lesen

Schlagen Sie die Dokumentation folgender Methoden der Klasse String nach:

- `contains(String s)`
- `endsWith(String suffix)`
- `indexOf(String str)`
- `substring(int b, int e)`
- `toUpperCase()`
- `trim()`

Nutzen Sie die Methoden um folgende

Überprüfungen an der Eingabe „Fischers Fritz fischt frische Fische.“ vorzunehmen:

- Enthält die Eingabe den Namen Fritz?
- Endet die Eingabe auf „Fische“?
- An welcher Stelle taucht die Buchstabenkombination „fi“ auf?
- Ermitteln Sie den Teilstring zwischen 10 und dem Ende des Strings
- Wandeln Sie die Eingabe in Großbuchstaben um
- Verändert `trim()` die Länge unserer Eingabe? Wenn nein: Wie müssen Sie diese verändern so dass sich diese ändert?

Sie können dazu die Direkteingabe in BlueJ verwenden.

Klassendokumentation schreiben

- Eigene Klassen sollten in derselben Weise dokumentiert werden, wie die Klassen der Java Klassenbibliotheken.
- Dritte sollten mit Ihrer Klasse arbeiten können, ohne dass Sie Ihre Implementation zu analysieren und zu lesen.
- Dritte sollten in der Lage sein, die Klasse warten zu können.
- Machen Sie aus Ihrer Klasse eine 'Bibliotheksklasse'!

Wiederholung: Was gehört in die Klassendokumentation?

Die Dokumentation einer Klasse umfasst:

- den Klassennamen
- einen Kommentar, der die Aufgabe der Klasse beschreibt
- eine Versionsnummer
- die Namen der Autoren³
- Dokumentation für jede **public**-Methode und jeden **public**-Konstruktor

³Dies gilt ebenfalls längst nicht in allen Projekten

Mehoden und Konstruktorkommentare

Die Dokumentation einer Methode bzw. eines Konstruktors enthält:

- den Namen der Methode
- den Ergebnistyp
- die Parameternamen und -typen
 - eine Beschreibung jedes Parameters
 - eine Beschreibung des Rückgabewertes
- eine Beschreibung der durch die Methode bzw. den Konstruktor übernommenen Verantwortlichkeit

Beispiel Klassenkommentar

```
/**  
 * The Responder class represents a response  
 * generator object. It is used to generate an  
 * automatic response.  
 * @author      Michael Kölling and David J. Barnes  
 * @version     1.0  (2016.02.29)  
 */
```

Beispiel Methodenkommentar

```
/**
 * Read a line of text from standard input (the text
 * terminal), and return it as a set of words.
 * @param prompt A prompt to print to screen.
 * @return A set of strings, where each String is
 * one of the words typed by the user
 */
public HashSet<String> getInput(String prompt){
    ...
}
```

public vs private

Zugriffsmodifikatoren

- **public** (öffentliche) Elemente sind für Objekte anderer Klassen zugreifbar:
 - Exemplarvariablen
 - Konstruktoren
 - Methoden
- Exemplarvariablen sollten niemals **public** sein.
- Private (**private**) Elemente können nur innerhalb der Klasse verwendet werden.
- Nur solche Methoden **public** machen, die von anderen Klassen benötigt werden.

Information hiding

auch das Geheimnisprinzip

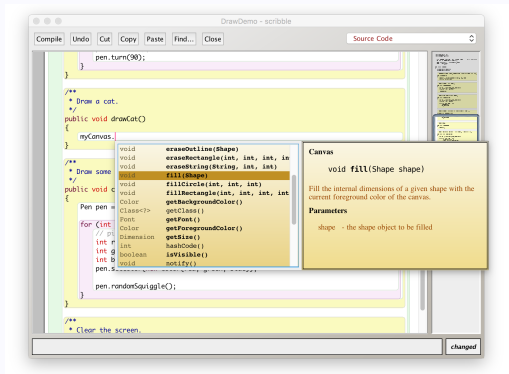
- Der Zustand eines Objekts sollte Objekten anderer Klassen verborgen sein.
 - Software Engineering Prinzip
 - D. L. Parnas (1971). "Information Distribution Aspects of Design Methodology". In: *Information Processing, Proceedings of IFIP Congress 1971, Volume 1 - Foundations and Systems, Ljubljana, Yugoslavia, August 23-28, 1971*. Hrsg. von C. V. Freiman u. a. North-Holland, S. 339–344. URL: <https://cseweb.ucsd.edu/~wgg/CSE218/Parnas-IFIP71-information-distribution.PDF>
 - Geheimnisprinzip: Was anderen bekannt ist, kann nicht geändert werden. Was andere nicht wissen, kann man jederzeit ändern.
- Wissen was ein Objekt leisten kann, nicht wie es das leistet.
- Information hiding reduziert die Abhängigkeit von Klassen.
- Die Unabhängigkeit von Softwareartefakten ist wichtig für die Handhabbarkeit großer Softwaresysteme, für Wiederverwendung und für den Wartungsprozess.

Nützliche Features von IDEs

Code completion

Quelltextvervollständigung:

- IDEs, wie bspw. der BlueJ Editor, Eclipse oder IntelliJ, unterstützten das Auffinden von Methoden.
- Ctrl-space nach einem Punkt für den Methodenaufruf zeigt in einem pop-up Fenster alle zur Verfügung stehenden Methoden. (BlueJ)
- Durch Return wird eine ausgewählte/markierte Methode ausgewählt.



Code Completion in BlueJ

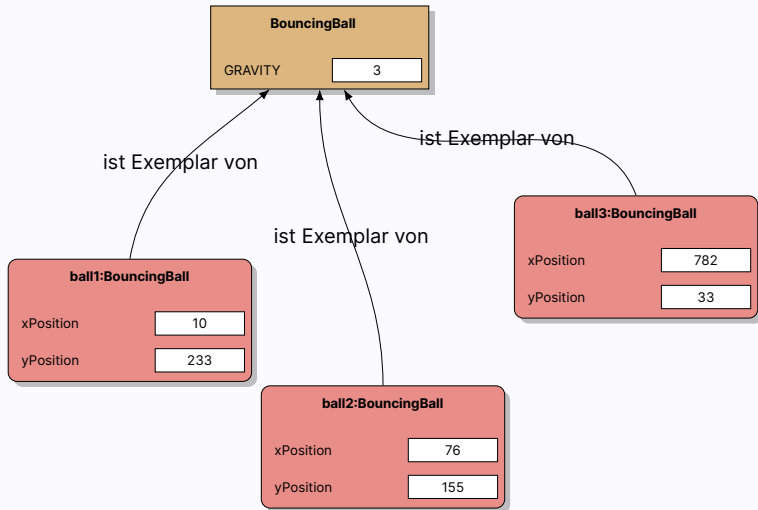
Wiederholung

- Java bringt eine umfangreiche Klassenbibliothek mit.
- Ein Programmierer sollte sich in der Bibliothek zurechtfinden können.
- Die Dokumentation richtet sich an den Nutzer der Klasse und beschreibt die Schnittstelle der Klasse.
- Die Implementation (Umsetzung) bleibt verborgen. Das ist eine wichtige Technik des Software Engineering: Information hiding.
- Wir kommentieren die Klasse so, dass die Texte verständlich sind und für sich allein verstanden werden können. (Klassenkommentare und Methodenkommentare).

Klassenvariablen, statische Variablen

- Eine Klassenvariable hat für alle Exemplare der Klasse denselben Wert.
- Sie gehört zur Klasse und ist unabhängig von Exemplaren. Sie existiert sogar, wenn gar kein Exemplar der Klasse existiert.
- Klassenvariablen werden durch das Schlüsselwort **static** vereinbart.
- **public static** Variablen können über den Klassennamen zugegriffen werden. Es ist kein Objekt erforderlich.
- Beispiel:
 - `Thermometer.boilingPoint`
- Klassenvariablen haben aus Sicht des Software Engineering einige Nachteile und sind deshalb generell zu vermeiden.

Klassenvariablen



Konstanten

- Eine Variable, die einen einmal gesetzten Wert nicht mehr Ändern kann heißt Konstante.
- Konstanten sind Variablen, die mit dem **final** Schlüsselwort deklariert werden.
 - **final int** SIZE = 10;
- **final** Variablen müssen im Konstruktor oder direkt bei Deklaration einen Wert zugewiesen bekommen.
- **static** wird häufig mit **final** und **public** kombiniert. Die Kombination ist vereinbar mit guter Programmierpraxis.

Klassenkonstanten

- **static**: Klassenvariable
- **final**: kann nicht verändert werden
- **public**: kann von allen verwendet werden
- **private static final int gravity = 3;**
- Die Namenskonventionen für **public final static** Variablen sind von denen anderer Variablen abweichend:

```
public static final int BOILING_POINT = 100;
```

Klassmethoden

- Eine **static** Methode gehört zur Klasse, und nicht zu den Exemplaren der Klasse:

```
public static int getDaysThisMonth()
```

- **static** Methoden werden über den Klassennamen angesprochen:

```
int days = Calendar.getDaysThisMonth();
```

- Eine **static** Methode existiert unabhängig von den Exemplaren.
Deshalb:
 - kann man in **static** Methoden nicht auf Exemplarvariablen zugreifen.
 - kann man in **static** Methoden nicht auf Exemplarmethoden zugreifen.
- Die Nutzung von **static** Methoden ist generell zu vermeiden.

Wiederholung I

- Eine Klassenvariable gehört zur Klasse und nicht zu den Exemplaren.
- Eine Klassenmethode gehört zur Klasse und nicht zu den Exemplaren.
- Klassenvariablen können kann man verwenden, wenn sich die Objekte einer Klasse Informationen teilen sollen.
- Klassenmethoden können nicht auf Exemplarvariablen und Exemplarmethoden zugreifen. Die entsprechende Fehlermeldung des Compilers wird jedoch von Anfängern häufig nicht verstanden.
- Sowohl Klassenvariablen und Klassenmethoden werden von Anfängern häufig falsch und unnötig eingesetzt. Deshalb sind sie von Anfängern generell zu vermeiden.
- Ausnahme: **public static final** Variablen sind Konstanten.
- Die Werte von finalen Variablen, sind einmal zugewiesen, unveränderlich.
- Die Wertzuweisung muss direkt bei der Deklaration der Variable oder aber im Konstruktor (für Felder) erfolgen.
- Finale und statische Aspekte sind unterschiedliche Konzepte, werden aber häufig zusammen verwendet.

Polymorphe Collection Typen

Noch eine wichtige Bemerkung:

- Unterschiedliche Collection Typen haben häufig ähnliche Schnittstellen, z.B.:
 - ArrayList und LinkedList
 - HashSet und TreeSet
- Es gibt Typen, die diese Ähnlichkeiten erfassen:
 - List
 - Set

Polymorphe Collection Typen

- Polymorphismus erlaubt es, in vielen Fällen von den konkreten Implementationen wegzukommen.
- Objekte werden zu einer konkreten Klasse erzeugt, aber ...

... die Variablen gehören zu einem generelleren Typ:

Example

```
List<Track> tracks = new LinkedList<>();  
Map<String, String> responseMap = new HashMap<>();
```

-
- Dies ermöglicht es, Programme flexibler zu schreiben. Auch wenn der Nutzen jetzt nicht sichtbar ist, ist dieses vorgehen ein wesentlicher Aspekt der objektorientierten Programmierung und wird uns immer wieder begegnen.
- Mehr dazu in einem späterem Kapitel.

Fragen?

Für weitere Fragen im
Nachgang können Sie mich
gerne über Moodle oder via
E-Mail kontaktieren!

Konzepte: Zusammenfassung I

- **Set** Eine Menge ist eine Sammlung, in der jedes Element nur maximal einmal enthalten ist. Die Elemente einer Menge haben keine spezifische Ordnung.
- **Map** Eine Map ist eine Sammlung, die Schlüssel-Wert-Paare als Einträge erhält. Ein Wert kann ausgelesen werden, indem ein Schlüssel angegeben wird.
- **Autoboxing** Autoboxing wird automatisch durchgeführt, wenn der Wert eines primitiven Typs in einem Kontext verwendet wird, in dem ein Wrapper-Typ erwartet wird.
- **Schnittstelle** Die Schnittstelle einer Klasse beschreibt, was eine Klasse leistet und wie sie benutzt werden kann, ohne dass ihre Implementierung sichtbar wird.
- **Implementation** Der komplette Quelltext, der eine Klasse definiert, wird als die Implementierung der Klasse bezeichnet.
- **unveränderlich (immutable)** Ein Objekt wird als unveränderlich bezeichnet, wenn sein Inhalt oder Zustand nicht mehr geändert werden kann, nachdem es erzeugt wurde. **String** ist ein Beispiel für eine Klasse, die unveränderliche Objekte definiert.
- **Zugriffsmodifikatoren** Zugriffsmodifikatoren definieren die Sichtbarkeit eines Datenfelds, eines Konstruktors oder einer Methode. Öffentliche (**public**) Elemente sind sowohl innerhalb der definierenden Klasse zugreifbar als auch von anderen Klassen aus; private (**private**) Elemente sind ausschließlich innerhalb der definierenden Klasse zugreifbar.

Konzepte: Zusammenfassung II

- **Geheimnisprinzip** Das Geheimnisprinzip besagt, dass die internen Details der Implementierung einer Klasse vor anderen Klassen verborgen sein sollten. Dies unterstützt eine bessere Modularisierung von Anwendungen.
- **Klassenvariablen, statische Variablen** Auch Klassen können Datenfelder haben. Diese werden Klassenvariablen oder auch statische Variablen genannt. Von einer Klassenvariablen existiert immer nur genau eine Kopie, unabhängig von der Anzahl der erzeugten Instanzen (Schlüsselwort: **static**).

Set

Eine Menge ist eine Sammlung, in der jedes Element nur maximal einmal enthalten ist. Die Elemente einer Menge haben keine spezifische Ordnung.

Map

Eine Map ist eine Sammlung, die Schlüssel-Wert-Paare als Einträge erhält. Ein Wert kann ausgelesen werden, indem ein Schlüssel angegeben wird.

Autoboxing

Autoboxing wird automatisch durchgeführt, wenn der Wert eines primitiven Typs in einem Kontext verwendet wird, in dem ein Wrapper-Typ erwartet wird.

Schnittstelle

Die Schnittstelle einer Klasse beschreibt, was eine Klasse leistet und wie sie benutzt werden kann, ohne dass ihre Implementierung sichtbar wird.

Implementation

Der komplette Quelltext, der eine Klasse definiert, wird als die Implementierung der Klasse bezeichnet.

unveränderlich (immutable)

Ein Objekt wird als unveränderlich bezeichnet, wenn sein Inhalt oder Zustand nicht mehr geändert werden kann, nachdem es erzeugt wurde. String ist ein Beispiel für eine Klasse, die unveränderliche Objekte definiert.

Zugriffsmodifikatoren

Zugriffsmodifikatoren definieren die Sichtbarkeit eines Datenfelds, eines Konstruktors oder einer Methode. Öffentliche (`public`) Elemente sind sowohl innerhalb der definierenden Klasse zugreifbar als auch von anderen Klassen aus; private (`private`) Elemente sind ausschließlich innerhalb der definierenden Klasse zugreifbar.

Geheimnisprinzip

Das Geheimnisprinzip besagt, dass die internen Details der Implementierung einer Klasse vor anderen Klassen verborgen sein sollten. Dies unterstützt eine bessere Modularisierung von Anwendungen.

Klassenvariablen, statische Variablen

Auch Klassen können Datenfelder haben. Diese werden Klassenvariablen oder auch statische Variablen genannt. Von einer Klassenvariablen existiert immer nur genau eine Kopie, unabhängig von der Anzahl der erzeugten Instanzen (Schlüsselwort: `static`).