Modul 404

Objektbasiert programmieren nach Vorgabe

Daniel Senften

22. Februar 2019

Inhaltsverzeichnis

1 Modulidentifikation **3**

2 Bereitsstellen der Software **3**
2.1 Installation der Umgebung . . . 3
2.2 Interaktive Entwicklungsumgebung . . . 5

3 Objekte und Klassen **6**
3.1 Objekte und Klassen . . . 6
3.2 Instanzen erzeugen . . . 9
3.3 Methoden aufrufen . . . 9
3.4 Parameter . . . 9
3.5 Datentypen . . . 9

4 Klassendefinitionen **10**

5 Objektinteraktion **11**
5.1 Wie werden die Daten übergeben . . . 11

6 Objektsammlungen **12**
6.1 Generische Klassen . . . 12
6.2 Sammlungen | *Collections* . . . 13

7 Klassendesign durch Vererbung **15**
7.1 Vererbung und Zugriffsrechte . . . 16
7.2 Initialisierung von Subklassen . . . 16
7.3 Weitere Techniken zur Abstraktion . . . 17

A Übungen und Programmbeispiele **18**

Abbildungsverzeichnis

1 UML Diagramm vs Implementation . . . 6
2 Willkommensbildschirm . . . 8
3 Projektstruktur . . . 8
4 Collection API . . . 14
5 Klassen Hierarchie . . . 16

Tabellenverzeichnis

1 Sichtbarkeit von Objekten . . . 7
2 Einfache Datentypen in Java . . . 10

1 Modulidentifikation

Für unsere Lernziele richten wir uns nach den Vorgaben des Verbandes ICT Berufsbildung Bern und hier im Spezifischen nach den Kompetenzanforderungen für das Modul 404.

1. Aufgrund einer Vorgabe den Ablauf darstellen.
2. Eine Benutzerschnittstelle entwerfen und implementieren.
3. Erforderliche Daten bestimmen und Datentypen festlegen.
4. Programmvorgabe unter Nutzung vorhandener Komponenten mit deren Eigenschaften und Methoden, sowie Operatoren und Kontrollstrukturen implementieren.
5. Beim Programmieren vorgegebene Standards und Richtlinien einhalten, das Programm inline dokumentieren und dabei auf Wartbarkeit und Nachvollziehbarkeit achten.
6. Programm auf Einhaltung der Funktionalität testen, Fehler erkennen und beheben.

2 Bereitsstellen der Software

Java ist eine Programmiersprache und Computerplattform, die erstmals 1995 von Sun Microsystems veröffentlicht wurde. Es gibt viele Anwendungen und Websites, die nicht funktionieren, es sei denn, Sie haben Java installiert, und jeden Tag werden weitere erstellt. Java ist schnell, sicher und zuverlässig. Von Laptops bis hin zu Rechenzentren, Spielkonsolen bis hin zu wissenschaftlichen Supercomputern, Handys bis hin zum Internet.

Am 20. April 2009 kündigte Oracle die Übernahme von Sun Microsystems für 7,4 Milliarden US-Dollar an, welche in den Folgemonaten durch verschiedene Behörden überprüft und anschliessend genehmigt wurde.

Java als Entwicklungs- und Laufzeitumgebung kann in verschiedenen Varianten von den folgenden Seiten heruntergeladen und installiert werden:

2.1 Installation der Umgebung

Wir unterscheiden im Wesentlichen zwischen den beiden Installationstypen JRE (*Java Runtime Environment*) und JDK (*Java Development Kit*).

Die wichtigsten *Links* zu diesen Umgebungen finden wir hier:

- java.com
- openjdk.java.net

In der Softwareentwicklung konzentrieren wir uns ausschliesslich auf das JDK, da wir auf die vielen zusätzlichen *Tools* in der Entwicklung angewiesen sind.

Die JDK-Tools und ihre Befehle ermöglichen es uns, Entwicklungsaufgaben wie das Kompilieren und Ausführen eines Programms, das Paketieren von Quelldateien und vieles mehr zu erledigen.

- javac—Kompilieren von Klassen- und Schnittstellendefinitionen in Bytecode.
- java—Starten/Interpretieren eines Java Programms.
- jar—Erstellen eines Java Archives.
- jshell—Interaktives Auswerten von Deklarationen, Anweisungen und Ausdrücken der Programmiersprache Java.
- javap—Befehl, um eine oder mehrere Klassendateien zu disassemblieren.
- javadoc—Erstellen der Java Dokumentation
- keytool—Befehl und Optionen zum Verwalten eines kryptografischer Schlüssel.
- jarsigner—Signieren und verifizieren von Java Archiven

Diese Liste stellt keinen Anspruch auf Vollständigkeit. Die komplette Dokumentation zu den einzelnen Umgebungen und deren *Tools* kann jederzeit unter docs.oracle.com eingesehen werden.

Vorsicht ist allerdings geboten, wenn es um den Einsatz und die Lizenzierung von Java geht. Oracle wird nach Januar 2019 keine weiteren Updates von Java SE 8 zur gewerblichen Nutzung auf seinen Download-Sites veröffentlichen. Kunden, die weiterhin Zugriff auf kritische Fehlerkorrekturen und Sicherheitslücken sowie eine allgemeine Wartung von Java SE 8 bzw. früheren Versionen benötigen, können einen langfristigen Support über Oracle Java SE Advanced erhalten. Oracle Java SE Advanced Desktop oder Oracle Java SE Suite. Für weitere Informationen und Details, wie Sie einen längerfristigen Support für Oracle JDK 8 erhalten, finden Sie in der Oracle Java SE Support-Roadmap.

Die offizielle Preisliste hilft, sich mit dem Oracle Java SE Abonnement für On Premise, Enterprise, Desktop, Server und Cloud Workloads vertraut zu machen.

- **Finch Robot**—Dieser kleine Roboter von Bird Technology nutzt verschiedene Sensoren, die via Java angesteuert werden können.
- **Oracle Academy**—Die Oracle Academy bietet verschiedene Online-Kurse und Anleitungen
- **Scratch**—Dies ist eine sehr einfache, am MIT entwickelte Programmierumgebung. Typischerweise wird diese im Vorschulalter eingesetzt.
- **BlueJ**—Als sehr professionelle und stark vereinfachte (Entwicklungs-) Umgebung wird BlueJ auch im späteren Alter noch für Schulungszwecke verwendet.

Übung 1 Installation der Software

Bevor wir mit unseren Übungen starten müssen wir sicherstellen, dass Java und seine *Tools* komplett und korrekt installiert sind.

Bitte installieren Sie das *Java Development Kit* (OpenJDK) gemäss Anleitung.

Anschliessend überprüfen wir die korrekte Installation mit Hilfe der Konsole wie folgt:

```
# java -version
openjdk version "11.0.1" 2018-10-16
OpenJDK Runtime Environment 18.9 (build 11.0.1+13)
OpenJDK 64-Bit Server VM 18.9 (build 11.0.1+13, mixed mode)
```

2.2 Interaktive Entwicklungsumgebung

Jeder Java-Entwickler benötigt einen Programmiereditor oder eine IDE, der bei den grungierigeren Teilen des Schreibens von Java und der Verwendung von Klassenbibliotheken und Frameworks helfen kann. Die Entscheidung, welche IDE zum Einsatz kommt, hängt von mehreren Faktoren ab, darunter der Art der zu entwickelnden Projekte, dem vom Entwicklungsteam verwendeten Prozess sowie dem Niveau und den Fähigkeiten des Entwicklers. Weitere Überlegungen sind, ob das Team die Tools und Ihre persönlichen Präferenzen standardisiert hat.

Die drei am häufigsten für die Java-Entwicklung gewählten IDEs sind

- IntelliJ IDEA
- Eclipse
- NetBeans

Ein guter Vergleich dieser drei Umgebungen ist auch hier zu finden. Persönlich habe ich Eclipse in den Anfängen der IDE, später NetBeans in den Schulungen und zu guter Letzt IntelliJ IDEA verwendet. Nach jedem Wechsel spürte ich, dass sich meine Produktivität verbessert hatte.

Ich habe mich für IntelliJ IDEA Ultimate entschieden. Obwohl es nicht kostenlos wie Eclipse oder NetBeans ist, glaube ich, dass der Produktivitätsgewinn den Preis wert ist. Für unsere Schulung ist es allerdings unerheblich, welches Produkt wir einsetzten, auch wenn ich bestimmt mit IntelliJ IDEA die bessere Unterstützung bieten kann.

Übung 2 Installation der Entwicklungsumgebung

Für die Entwicklung und die Zusammenarbeit mit dem vorhandenen Git *Repository* und für die Generierung der Unterlagen (Skript und Folien) werden wir noch weitere Werkzeuge nutzen, die wir vorgängig installieren müssen.

1. IntelliJ IDEA
2. Git für Mac, Windows oder Linux.
3. TEX Live für Mac, Windows oder Linux.
4. Python v7.3.2
5. Pygments

3 Objekte und Klassen

- Klassen
- Objekte
- Eigenschaften | Attribute
- Methoden
- Parameter
- *Daten Typen*

3.1 Objekte und Klassen

Wenn Sie ein Computerprogramm in einer objektorientierten Sprache schreiben, dann erstellen Sie im Computer ein Modell eines Auschnitts der realen Welt. Dieser Ausschnitt setzt sich aus den *Objekten* zusammen, die im Anwendungsbereich (*Problem Domain*) vorkommen.

In der Abbildung 1 sehen wir eine einfache Klasse, welche im vorliegenden Fall nur aus dem Namen selbst besteht. Der Name der Klasse, sowie alle weiteren Definitionen wie Eigenschaften (Attribute), Konstanten und Methoden folgen einer klaren Namenskonvention, auch bekannt unter CamelCase.

Abbildung 1: UML Diagramm vs Implementation

Doch was ist eigentlich der Untersdchied zwischen Klassen und Objekten?

Objekte stellen *Dinge* aus der realen Welt oder aus einer Problemdomäne dar (Beispiel: "Das weisse Auto da unten auf dem Parkplatz").

Klassen repräsentieren Objekte einer bestimmten Art (Beispiel: "Auto").

Objekte können klassifiziert werden als alle Objekte einer bestimmten Art und eine Klasse beschreibt genau dies auf abstrakte Art.

Betrachten wir unsere erste Klasse—in diesem Fall unser erstes Programm—in seiner einfachsten Form (s. Listing 1, Seite 7).

```
1  package academy;
2
3  public class HelloWorld {
4
5      public static void main(String[] args) {
6          System.out.println("Hello, World!");
7      }
8  }
```

Listing 1: Erstes Programm: *HelloWorld*

Typischerweise wir eine Klasse in Java in einer Datei unter dem selben Namen abgelegt. Hierbei ist wichtig die Gross- und Kleinschreibung zu beachten.

Der Aufbau einer Java Datei kann sehr einfach durch eine Metasprache, der sogenannten Erweiterte Backus-Naur-Form (EBNF) dargestellt werden.

```
41 grammar Java;
42
43 // starting point for parsing a java file
44 compilationUnit
45     : packageDeclaration? importDeclaration* typeDeclaration* EOF
46     ;
```

Listing 2: Syntaktischer Aufbau einer Java-Datei

Es existiern vier unterschiedliche Zugriffsstufen für die Objekte (Daten und Methoden).

Modifier	Class	Package	Subclass	Universe
private	x			
default	x	x		
protected	x	x	x	
public	x	x	x	x

Tabelle 1: Sichtbarkeit von Objekten

Übung 3 Ein einfacher Taschenrechner

Erstellen eines neuen Projektes

Wenn wir mit IntelliJ arbeiten geschieht alles im Rahmen eines Projekts. Ein Projekt ist eine Darstellung einer entwickelten Komplettlösung, die aus Quellcode, Bibliotheken und

Konfigurationsdateien besteht.

Falls nicht bereits erledigt, müssen wir an dieser Stelle unser IntelliJ IDEA zuerst noch installieren.

Wenn wir IntelliJ nach der Installation öffnen, werden wir vom Willkommensbildschirm begrüsst:

Abbildung 2: Willkommensbildschirm

An dieser Stelle können wir ein neue Projekt erstellen, ein bestehendes öffnen oder in unserem Fall ein bestehendes Projekt aus einem *Repository*, hier GitHub, klonen.

Versucht diese Installation des Projektes vorzunehmen und anschliessend das Programm academy.calculator.Calculator zu starten. Wenn alles korrekt installiert wurde, dann sieht die Projektstruktur wie folgt aus:

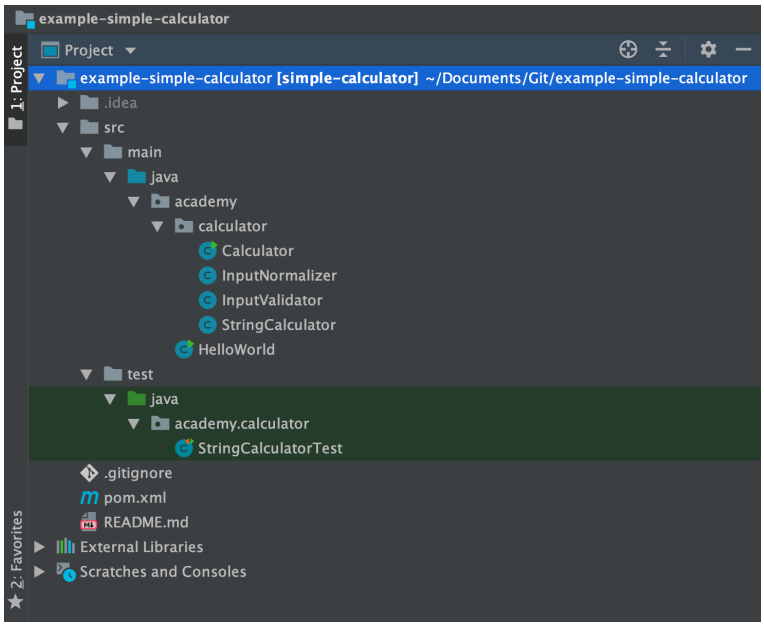

Abbildung 3: Projektstruktur

3.2 Instanzen erzeugen

Anhand des Beispieles aus Übung 3 können wir sehr schön aufzeigen, wie ein Objekt einer Bestimmten Klasse erzeugt wird.

```
1  package academy.calculator;
2
3  public class Calculator {
4
5      public static void main(String[] args) {
6          String numbers = "1,2,3,4";
7          System.out.println(" numbers = " + numbers);
8
9          StringCalculator calculator = new StringCalculator();
10         int output = calculator.add(numbers);
11         System.out.println(" output = " + output);
12     }
13 }
```

Listing 3: Einfacher Additions-Rechner

3.3 Methoden aufrufen

Für die nächsten Beispiele verwenden wir ein anderes Projekt, welches wir auch unter GitHub finden: example-figures.

Basierend auf den installierten Klassen wollen wir nun die zur Verfügung gestellten Klassen wie als Objekte erzeugen und darstellen.

Übung

Erzeugen und darstellen eines Kreis.

```
Circle circle = new Circle();
circle.setVisible();
```

3.4 Parameter

3.5 Datentypen

Die hier aufgeführten Gleitkommazahlen float und double sind nach dem Standard IEEE 754 abgelegt.

Datentyp	Grösse	Wertebereich
boolean		**true**/**false**
char	16 bit	$0 \ldots 65'535$
byte	8 bit	$-128 \ldots 127$
short	16 bit	$-32'768 \ldots 32'767$
int	32 bit	$-2'147'483'648 \ldots 2'147'483'647$
long	64 bit	$-9'223'372'036'854'775'808 \ldots 9'223'372'036'854'775'807$
float	32 bit	$\pm 1.402'398 \times 10^{-45} \ldots 3.402'823 \times 10^{38}$
double	64 bit	$\pm 4.922'656 \times 10^{-324} \ldots 1.797'693 \times 10^{308}$

Tabelle 2: Einfache Datentypen in Java

4 Klassendefinitionen

In Listing 2 (Seite 7) haben wir mit Hilfe der Metasprache EBNF den Aufbau einer Java-Datei erläutert.

An dieser Stelle wollen wir uns nun dem syntaktischen Aufbau einer Klasse widmen. Die komplette Syntax von Java ist in der Datei ’./src/main/tex/Java.g4’ zu finden.[1]

```
90  classDeclaration
91      : 'class' Identifier typeParameters?
92        ('extends' type)?
93        ('implements' typeList)?
94        classBody
95      ;
96
97  typeParameters
98      :   '<' typeParameter (',' typeParameter)* '>'
99      ;
100
101 typeParameter
102     :   Identifier ('extends' typeBound)?
103     ;
```

Listing 4: Syntaktischer Aufbau einer Klasse

Die zentralen Konzepte einer Klasse bestehen aus folgenden Elementen:

- Eigenschaften
- Methoden
- Konstruktoren

[1] Für die korrekte Darstellung dieser Datei muss allenfalls noch das ANTLR v4 Plugin via *Plugin Manager* installiert werden.

- Parameter
- Zuweisungen

5 Objektinteraktion

5.1 Wie werden die Daten übergeben

Bei der Objektinteraktion und der damit verbundenen Parameterübergabe stellt sich immer wieder wir Frage: ”Wie werden die Parameter übergeben?”(*pass-by reference or pass-by value?*)

Java manipuliert Objekte nach Referenz, und alle Objektvariablen sind Referenzen. Java übergibt jedoch keine Methodenargumente als Referenz, sondern übergibt sie als Wert.

Hierzu das folgende Beispiel:

```
7  public void badSwap(int var1, int var2) {
8     int temp = var1;
9     var1 = var2;
10    var2 = temp;
11 }
```

Wenn badSwap() zurückkehrt, behalten die als Argumente übergebenen Variablen ihre ursprünglichen Werte bei. Die Methode wird auch fehlschlagen, wenn wir den Argumenttyp von int auf Object ändern, da Java auch Objektreferenzen nach Wert übergibt.

Wie sieht es hiermit aus:

Wenn wir main() ausführen, sehen wir die folgende Ausgabe:

X: 0 Y: 0
X: 0 Y: 0
X: 100 Y:100
X: 0 Y: 0

Die Methode ändert erfolgreich den Wert von pnt1, obwohl er vom als Wert (*pass-by value*) übergeben wird; ein Austausch von pnt1 und pnt2 schlägt jedoch fehl! Das ist die Hauptquelle der Verwirrung. In der main() Methode sind pnt1 und pnt2 nichts anderes als Objektreferenzen. Wenn Sie pnt1 und pnt2 an die tricky()-Methode übergeben, übergibt Java die Referenzen wie jeder andere Parameter nach Wert. Das bedeutet, dass die an die Methode übergebenen Referenzen tatsächlich Kopien der ursprünglichen Referenzen sind.

Übung 4 Verwalten von Studenten und Kursbelegungen

Ein einfaches Beispiel mit Kursen und Studenten. Es zeigt einmal mehr auf, wie wir Objekte, Attribute und Methoden verwenden. In einem späteren Abschnitt werden wir an dieser Stelle die Vererbung integrieren.

```
13 private static void tricky(Point arg1, Point arg2) {
14    arg1.x = 100;
15    arg1.y = 100;
16    Point temp = arg1;
17    arg1 = arg2;
18    arg2 = temp;
19 }
20
21 public static void main(String[] args) {
22    Point pnt1 = new Point(0, 0);
23    Point pnt2 = new Point(0, 0);
24    System.out.println("X: " + pnt1.x + " Y: " + pnt1.y);
25    System.out.println("X: " + pnt2.x + " Y: " + pnt2.y);
26    System.out.println(" ");
27    tricky(pnt1, pnt2);
28    System.out.println("X: " + pnt1.x + " Y:" + pnt1.y);
29    System.out.println("X: " + pnt2.x + " Y: " + pnt2.y);
30 }
```

Listing 5: Pass-by reference or pass-by value?

https://github.com/dsenften/example-course.git

6 Objektsammlungen

6.1 Generische Klassen

Durch die Implementation verschiedener *Cache* Klassen handeln wir uns eine sehr starke Abhängigkeit ein und schreiben gleichzeitig redundanten Code.

```
package generics;

public class CacheString {
   private String text;

   public void addText(String text) {
      this.text = text;
   }

   public String getText() {
      return this.text;
   }
}
```

```
package generics;

public class CacheShirt {
   private Shirt shirt;

   public void addText(Shirt shirt) {
      this.shirt = shirt;
   }

   public Shirt getShirt() {
      return this.shirt;
   }
}
```

Listing 6: Einfacher *Cache* ohne *Generics*

Durch die Verwendung generischer Klassen können wir dieses Problem sehr elegant lösen.

```java
package generics;

public class Cache<T> {
    private T cache;

    public void add(T cache) {
        this.cache = cache;
    }

    public T get() {
        return this.cache;
    }
}
```

Listing 7: Definieren einer generischen Klasse

Die spezielle Schreibweise der Klasse Cache verwenden wir in einem Programm wie folgt:

```java
Cache<String> cache = new Cache<>();
```

Die hier eingeführte Notation mit den spitzen Klammern müssen wir allerdings noch etwas ausführlicher betrachten. Die Klasse selbst, die wir hier verwenden heisst Cache, aber sie erfordert die Angabe eines zweiten Typs als Parameter, wenn sie zur Deklaration von Attributen oder anderen Variablen verwendet wird.

Diese generischen Klassen definieren, im Gegensatz zu allen bisher betrachteten Klassen, nich einen einzelnen Typ, sondern beliebig viele Typen. Wir können beispielsweise weitere Objekte definieren:

```java
private Cache<Integer> cache = new Cache<>();
private Cache<Person> cache = new Cache<>();
private Cache<Circle> cache = new Cache<>();
```

6.2 Sammlungen | *Collections*

Bis zu diesem Punkt haben wir nur einfache Objekte betrachtet, die wir wahlweise auch mit einer anderen Bezeichnung (Attribut, Eigenschaft) benutzten. So trafen wir auf folgende Deklarationen:

```java
private int age;
private String firstName;
private Circle circle;
```

Neben diesen einfachen Strukturen sind auch statische Vektoren und mehrdimensionale Array mögliche. So sind wir auch folgender Deklarationen begegnet:

```java
String[] args; // Statischer (eindimensionaler) Array fester Grösse
```

Wollen wir allerdings eine dynamische Anzahl von Objekten in einer Datenstruktur speichern, dann ist dies nur mit entsprechenden Datentypen möglich. In Java sprechen wir vom sogenannten *Collection API*. Generell wird an dieser Stelle auch gerne der Begriff der *Abstrakten Datentypen* verwendet.

- Eine Sammlung ist eine Objekt, welches eine Reihe anderer Objekte verwaltet
  - Einzelne Objekte einer Sammlung nennen wir Elemente
  - Einfache/Primitive Datentype sind nicht erlaubt
- Es existieren viele vordefinierten Typen
  - *Stack*, *Queue*, *Hash*
- Das *Collection API* basiert stark auf generischen Klassen

In der Abblildung 4 (Seite 14) ist nur ein kleiner Ausschnitt des *Collection API's* dargestellt. Um sich einen kompletten Überblich zu machen ist sicher die Java Dokumentation ein guter Start. Auch die an dieser Stelle erwähnten Methoden sind nur auszugweise aufgeführt.

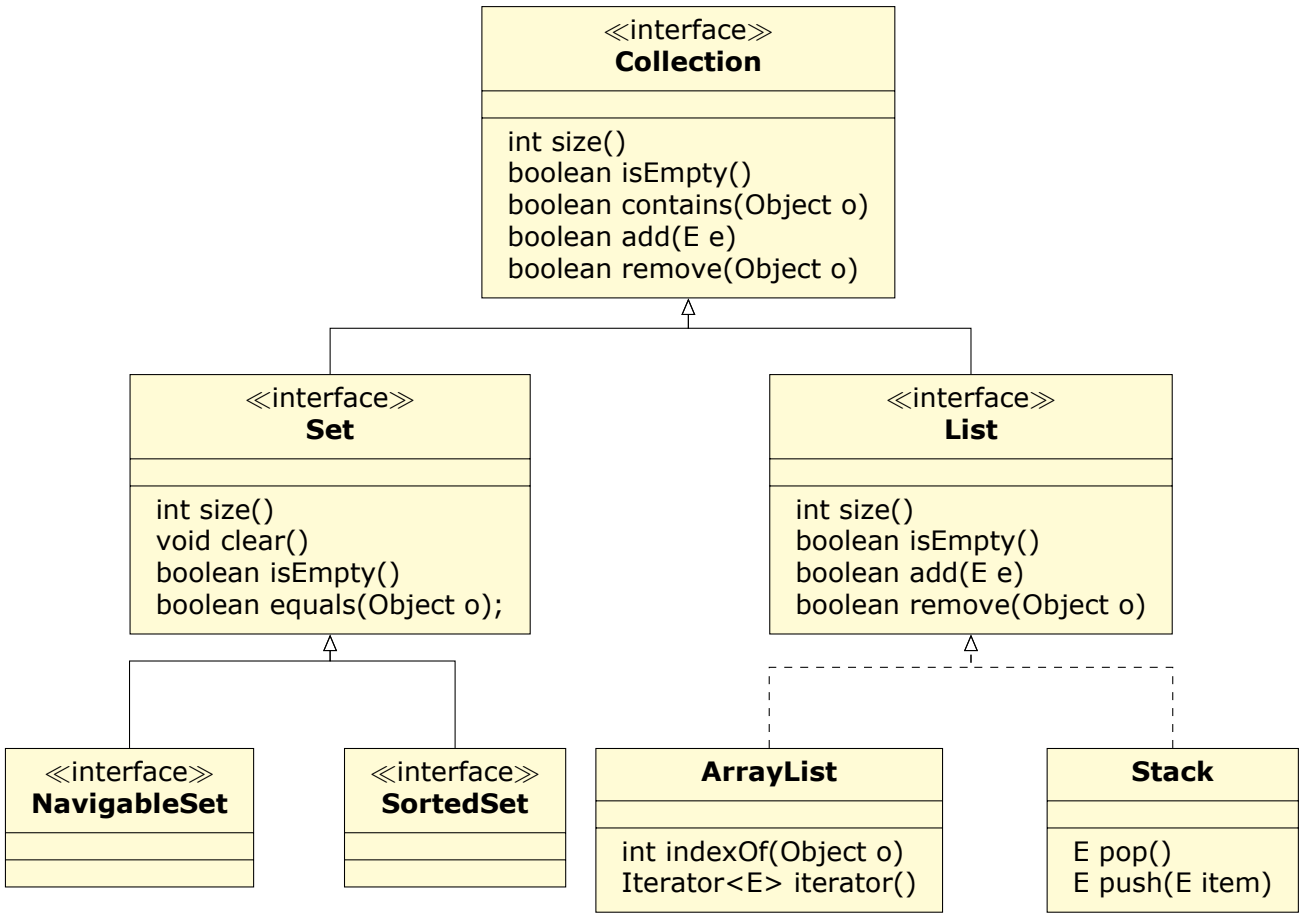

Abbildung 4: Collection API

Mit unserem Vorwissen in der Verwendung generischer Klassen konnen wir nun eine Liste für eine beliebige Anzahl von Obkten eines bestimmten Typs wie folgt deklarieren und gleichzeitig initialisieren:

```
ArrayList<Student> students = new ArrayList<>();
myPersonList.add(new Student("Senften, Daniel", "1234-123-45"));
```

7 Klassendesign durch Vererbung

In diesem Abschnitt kommen einige weitere objektorientierte Konzepte vor, mit welchen wir die Struktur unserer Anwendung(en) verbessern. Hierbei spielen Konzepte wie Vererbung und Polymorphie eine zentrale Rolle.

- **Vermeidung von Code-Duplizierung**

  Die Verwendung von Vererbung vermeidet, dass gleiche oder weitgehend ähnliche Quellabschnitte geschrieben werden.

- **Wiederverwendung von Quelltext**

  Bestehender Quelltext kann wiederverwendet werden. Wenn bereits eine Klasse besteht, die unseren Anforderungen sehr nahekommt, dann können wir sehr oft in einer Subklasse die bereits vorhandene Implementierung wiederverwenden.

- **Einfachere Wartung**

  Die Wartung der anwendung wird vereinfacht, denn die Beziehungen zwischen den Klassen sind explizit ausgedrückt. Eine Änderung an Objekten (Attributen oder Methoden) muss nur einmal vorgenommen werden. An dieser Stelle ist der Refactoring Mechanismuss unserer IDE eine weitere, grosse Hilfe.

- **Erweiterbarkeit**

  Durch Vererbung ist es wesentlich einfacher, eine bestehende Anwendung zu erweitern.

Es können viele Klassen von einer Superklasse erben und eine Subklasse kann wiederum selber Supperklasse weiterer Subklassen sein. So ergibt sich am Schluss eine Vererbungshierarchie. In Java besteht aus einem Klassenbaum, in welchem die Klasse Object dir Mutter aller Klassen representiert. Im Gegensatz zu anderen Programmiersprachen ist in Java nur eine Einfachvererbung (*single inheritance*) möglich.

Uns bestens bekannt ist vermutlich die Vererbungshierarchie in der Biologie. In Abbildung 5 (Seite 16) sehen wir einen kleinen Ausschnitt aus dieser Welt.

Bevor wir die Vererbung genauer ansehen, wollen wir kurz betrachten, wie Vererbung in Java ausgedrückt wird. In Listing 8 ist ein Ausschnitt der (Super-) Klasse (*parent class*) Employee. An dieser Klasse ist wie immer nichts aussergewöhliches festzustellen. Sie hat de Aufbau wie immer mit dem Namen der Klasse, Eigenschaften (Attributen) und Methoden.

Doch wie sieht es aus mit der spezialisierten (Sub-) Klasse? Wenn wir von dem*Keyword* extends absehen ist auch an der Klasse Manager nichts ausswergewöhnliches festzustellen. Die Klausel 'extends Employee' gibt an, dass diese Klasse eine Subklasse der Klasse Employee ist. Zweitens definiert die Klasse Manager nur Attribute, die spezifisch für Objekte der Klasse Manager sind. In diesem Fall sprechen wir vom Attribut department.

Sämtliche Eigenschaften (Attribute) der Supper-Klasse werden geerbt. Diese müssen wir nicht nicht noch einmal aufführen.

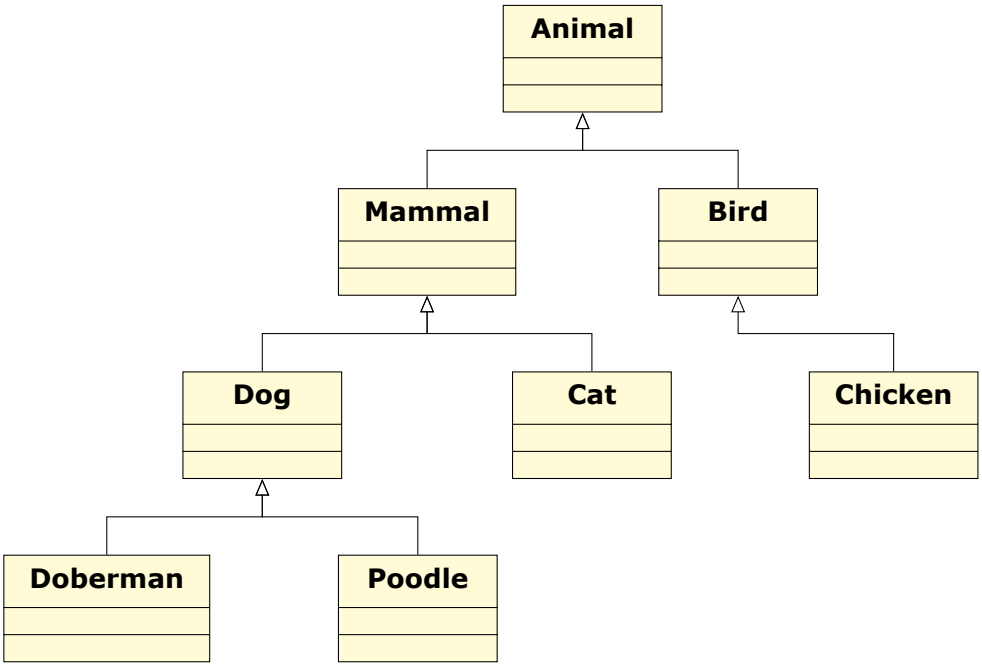

Abbildung 5: Klassen Hierarchie

```
8  public class Employee {
9      private final String firstName;
10     private final String lastName;
11
12     public Employee(String firstName, String lastName) {
13         this.firstName = firstName;
14         this.lastName = lastName;
15     }
16 }
```

Listing 8: Superklasse: *Employee*

7.1 Vererbung und Zugriffsrechte

Wir im vorangehenden Abschnitt beschrieben erbt eine Subklasse sämtliche Eigenschaften und Methoden der Superklasse. Dies heisst aber nicht automatisch, dass diese Objekte direkt sichtbar sind. Auch hier gib es eine Art Privatsphäre zwischen den beiden Klassen. Private Objekte bleiben auch in diesem Fall privat. Die in Tabelle 1 (Seite 7) vorgestellten Zugriffsrechte gelten natürlich auch hier. Daraus folgt, wenn wir in einer Subklasse auf private Objekte der Supperklasse zugreifen möchten, dann muss die Superklasse entsprechende Methoden zur Verfügung stellen.

7.2 Initialisierung von Subklassen

Wenn wir Objekte erzeugen, dann sorgt ein Konstruktor dafür, dass alle Attribute des Objektes in erstellt und gemäss den Regeln initialisiert werden. Wie sieht dies im Falle von

```
8  public class Manager extends Employee {
9      private final String department;
10
11     public Manager(String firstName, String lastName,
12                    String department) {
13         super(firstName, lastName);
14         this.department = department;
15     }
16 }
```

Listing 9: Subklasse: *Manager*

Subklassen aus?

Wenn wir ein Objekt vom Typ Employee erzeugen, dann übergeben wir mehrere Parameter an den Konstruktor dieser Klasse: den Vor- und den Nachnamen einer Person. Siehe hierzu auch unser Listing 8 (Seite 16).

Wenn wir nun ein Objekt vom Type Manager erzeugen, dann übergeben wir neben den Werten für Vor- und Nachname dieser Person auch noch die Abteilung (department), für welches dieser Manager verantwortlich ist. Da die Attribute für Vor- und Nachnamen Teil der Superklasse sind müssen wir nun den Konstruktor der Superklasse mit den entsprechenden Werten aufrufen mit:

```
super(firstName, LastName);
```

Wichtig: Der Aufruf des Superkonstruktors muss immer an erster Stelle stehen. Falls Wir diesen Aufruf nicht aufführen, dann wir an dieser Stelle in jedem Fall der *Default*-Konstruktor (ohne Argumente) aufgerufen.

Übung 5 Initialisierung von Subklassen

Versuche mit Hilfe des *Debuggers* die Initialisierung aller Objekte in der Vererbungshierarchie Employee und Manager nachzuvollziehen und zu verstehen.

Was passiert beispielsweise, wenn wir in der Klasse Employee folgende Zeile verwendet hätten:

```
private String firsName = "undefined";
```

Versuche auch eine weitere Spezialisierung (Engineer, Listing 10, Seite18) mit Hilfe des *Debuggers* zu verstehen. In welcher Reihenfolge werden welche Objekte initialisiert?

Übung 6 Vererbungshierarchie

Ordne die folgenden Begriffe in einer Vererbungshierarchie an: Apfel, Eiscreme, Brot, Frucht, Nahrungsmittel, Haferflocken, Orange, Dessert, Schockopudding, Baguette.

7.3 Weitere Techniken zur Abstraktion

```java
public class Engineer extends Employee {

   List<String> skills;

   {
      /*
       * The list returned when you call Arrays.asList is a thin
       * wrapper over the array, not a copy. The list returned is
       * fixed size: attempting to call add will throw an
       * UnsupportedOperationException exception. Therefore we
       * have to convert it to a 'real' ArrayList first :-)
       */
      skills = new ArrayList<>(
            Arrays.asList("Java", "C", "C++"));
   }

   public Engineer(String firstName, String lastName) {
      super(firstName, lastName);
   }

   public void addSkill(String skill) {
      skills.add(skill);
   }
}
```

Listing 10: Initialisierung von Subklassen

A Übungen und Programmbeispiele

Liste der Übungen

Übung 1 *Installation der Software* 5

Übung 2 *Installation der Entwicklungsumgebung* 5

Übung 3 *Ein einfacher Taschenrechner* 7

Übung 4 *Verwalten von Studenten und Kursbelegungen* 11

Übung 5 *Initialisierung von Subklassen* 17

Übung 6 *Vererbungshierarchie* 17

```java
public interface ElectronicDevice {
    void turnOn();
    void turnOff();
}
```

```java
public class Television implements ElectronicDevice {

    @Override
    public void turnOn() { }

    @Override
    public void turnOff() { }

    public void changeChannel(int channel) {}
    public void initializeScreen() {}
}
```

Listing 11: Deklaration und Implementation einer Schnittstelle

Programmbeispiele

1 Erstes Programm: *HelloWorld* 7
2 Syntaktischer Aufbau einer Java-Datei 7
3 Einfacher Additions-Rechner 9
4 Syntaktischer Aufbau einer Klasse 10
5 Pass-by reference or pass-by value? 12
6 Einfacher *Cache* ohne *Generics* 12
7 Definieren einer generischen Klasse 13
8 Superklasse: *Employee* 16
9 Subklasse: *Manager* 17
10 Initialisierung von Subklassen 18
11 Deklaration und Implementation einer Schnittstelle 19