Vorwort

»Alles, was einen Anfang hat, hat auch ein Ende, und meistens hat das, was ein Ende hat, auch eine Fortsetzung.«
– Sprichwort

Willkommen im zweiten Teil der Insel. Während der erste Band in die Grundlagen von Java einführte und sich dabei insbesondere auf die objektorientierten Konzepte und die Syntax der Sprache Java konzentrierte, kümmert sich der zweite Band um die Bibliotheken aus der Java SE.

Organisation der Kapitel

Den Einstieg ins Buch bildet **Kapitel 1**, »Neues in Java 9«, mit einer kompakten Darstellung der Neuerungen. Die Änderungen in den Bibliotheken fließen dabei in die jeweiligen Kapitel ein; so konzentriert sich das kurze Kapitel eher auf die neuen Spracheigenschaften. Die Inhalte sind Auszüge aus dem ersten Insel-Buch, »Java ist auch eine Insel«.

Die Java-Bibliothek besteht aus rund 6.000 Klassen, Schnittstellen, Aufzählungen, Ausnahmen und Annotationen. **Kapitel 2**, »Die Klassenbibliothek«, gibt eine Übersicht über die wichtigsten Pakete und greift einige Klassen aus der Bibliothek heraus, etwa zum Laden von Klassen. Hier sind auch Klassen zur Konfiguration von Anwendungen oder Möglichkeiten zum Ausführen externer Programme zu finden.

Neben den grundlegenden Basisklassen für Zeichenkettenverarbeitung gibt es spezielle Klassen, etwa für Mustererkennung, die **Kapitel 3**, »Fortgeschrittene String-Verarbeitung«, vorstellt.

Kapitel 4 befasst sich mit den »Datenstrukturen und Algorithmen«, die die Standardbibliothek anbietet. Die wichtigsten Klassen wie Listen, Mengen, Stapel, Bitmengen und Assoziativspeicher werden vorgestellt, und dann werden unterschiedliche Aufgaben mit den jeweils passenden Datenstrukturen gelöst. Als Algorithmen kommen beispielsweise vorgefertigte Sortierverfahren zum Einsatz. Das Kapitel diskutiert ebenfalls für Nebenläufigkeit optimierte Datenstrukturen.

Kapitel 5 beschäftigt sich mit »Threads und nebenläufiger Programmierung«. Das Kapitel umfasst auch die Koordination mehrerer kooperierender oder konkurrierender Threads.

Zeitzonen und unterschiedliche Ausgabeformate für Datumswerte sind Thema in **Kapitel 6**, »Raum und Zeit«. Darunter fallen auch Datumsberechnungen auf der Grundlage des gregorianischen Kalenders. Interessant sind Java-Möglichkeiten zur Internationalisierung, die

sich durchweg durch alle APIs zieht, die in irgendeiner Weise Nachrichten zum Benutzer geben.

Kapitel 7, »Dateien, Verzeichnisse und Dateizugriffe«, setzt den Fokus auf Dateiverarbeitung. Zuerst zeigen wir, wie sich Attribute von Dateien und Verzeichnissen auslesen lassen, und dann, wie sich wahlfreier Zugriff auf eine Datei realisieren lässt. Das in Java 7 eingeführte NIO.2-Paket nimmt dabei einen großen Raum ein, da es sehr leistungsfähig ist.

Nahtlos folgt in **Kapitel 8**, »Datenströme«, die Verarbeitung von Daten aus beliebigen Quellen und Senken. Während sich Kapitel 6 nur auf Dateien beschränkt, geht es in diesem Kapitel um allgemeinere Ein-/Ausgabe-Konzepte, die auch bei Datenströmen aus Netzwerken, Datenbanken oder Schnittstellen vorkommen. Die Datenströme können dabei durch Filter geschickt werden. Von Letzteren stellen wir einige vor, die zum Beispiel die Zeilennummer zählen, einen Datenstrom puffern oder ihn komprimieren. Eine elegante Möglichkeit ist das Serialisieren von Objekten. Dabei wird der Zustand eines Objekts ausgelesen und so in einen Datenstrom geschrieben, dass sich das Objekt später wiederherstellen lässt. Eine eigene Speicherroutine kann somit entfallen.

Heutzutage ist die Speicherung in XML populär, und auch Office-Dokumente werden in diesem Stil gespeichert. Doch keiner würde auf die Idee kommen, die Dokumente im XML-Format auszulesen und dort die Absätze und Zeichen herauszuziehen. Das Gleiche gilt für Grafikdateien: Es ist zwar eine Byte-Datei, doch die wird nicht Byte für Byte von uns eingelesen, sondern es gibt immer eine API, die sich darum kümmert, Zeichen oder Bytes in ein für uns komfortables Format zu bringen. Ein paar populäre Dateiformate stellt **Kapitel 9**, »Dateiformate«, vor. Zu den beschriebenen Formaten zählen nicht nur Dokumentenformate, wie für Microsoft Office, sondern auch Formate für die Kompression.

Tiefer in den XML-Dschungel geht **Kapitel 10**, »Die eXtensible Markup Language (XML) und JSON«. Java als plattformunabhängige Programmiersprache und XML als dokumentenunabhängige Beschreibungssprache sind ein ideales Paar, und die Kombination dieser beiden Technologien ist der Renner der letzten Jahre. Das Kapitel beginnt auf der höchsten Abstraktionsebene, beschreibt, wie XML-Daten auf Objekte übertragen werden, und geht dann weiter zur elementaren Verarbeitung der XML-Dokumente, wie sie zum Beispiel als Objektbaum aufgebaut und modifiziert werden.

Kapitel 11, »Netzwerkprogrammierung«, stellt vor, welche Informationen eine URL hat und wie mit dieser URL Daten von Webservern bezogen werden können. Bei Webservern werden wir CGI-Programme ansprechen, um an gewünschte Inhalte zu kommen. Mithilfe von Sockets wollen wir eine eigene Client-Server-Kommunikation aufbauen. Außer auf die gesicherte Verbindung TCP gehen wir auch auf ungesicherte UDP-Verbindungen ein.

Kapitel 12, »RESTful und SOAP-Web-Services«, stellt mit REST einen neuen Trend in der Kommunikation zwischen Browser und Webserver vor und mit SOAP einen Klassiker für den entfernten Methodenaufruf zwischen Systemen. SOAP ist Teil der Java SE, und für REST wird Jersey vorgestellt, die Referenzimplementierung der JAX-RS-API.

»Verteilte Programmierung mit RMI« in **Kapitel 13** zeigt auf, wie ein Java-Programm einfach Objekte und Methoden nutzen kann, die auf einem anderen Rechner gespeichert bzw. ausgeführt werden. Dabei wird der Aufruf einer Methode über das Netzwerk übertragen, und für das aufrufende Programm sieht es so aus, als ob es sich um einen normalen Methodenaufruf für ein lokales Objekt handelt.

Mit **Kapitel 14** widmen wir uns einer Java-typischen Technik: »Typen, Reflection und Annotationen«. Java-Klassen liegen selbst wieder als Meta-Objekte, als Exemplare der speziellen Klasse `Class`, vor. Diese `Class`-Objekte geben Auskunft über die verfügbaren und definierten Variablen, Methoden und Konstruktoren. So lassen sich beispielsweise dynamisch bestimmte Methoden aufrufen oder die Werte von dynamisch ausgewählten Objektvariablen abfragen. Annotationen ermöglichen es Entwicklern, Metadaten an Programmteilen festzumachen. Wie neue Annotationstypen erstellt und gesetzte Annotationen abgefragt werden, zeigt das Kapitel ebenfalls.

Wie Java-Programme zu Testzwecken überwacht werden können, zeigt **Kapitel 15**, »Logging und Monitoring«. Mit der JMX-API lassen sich MBeans an einem MBean-Server anmelden, und das Dienstprogramm *jconsole* ermöglicht den Zugriff und die Steuerung der Komponenten.

»Datenbankmanagement mit JDBC« ist das Thema von **Kapitel 16**. Als freie, quelloffene Beispieldatenbank wird HSQLDB vorgestellt, da sie sehr leicht zu installieren und zu betreiben ist und praktischerweise Beispieldaten mitbringt. Die Java-Beispiele bauen eine Verbindung zu HSQLDB auf, setzen SQL-Anweisungen ab, holen die Ergebnisse herein und visualisieren sie.

Kapitel 17, »Grafische Oberflächen mit Swing«, stellt die Swing-Komponenten zur Interaktion vor, wie zum Beispiel Schaltflächen. Es geht auf die Behandlung von Ereignissen ein, die aus Benutzeraktionen resultieren, und beschreibt Container, die andere Komponenten aufnehmen und layouten.

Das anschließende **Kapitel 18** deckt die zweite Aufgabe der grafischen Oberflächen ab, indem es auf die »Grafikprogrammierung« eingeht. Das AWT (Abstract Window Toolkit) ist die Java-Möglichkeit, grafische Oberflächen zu gestalten. Dabei gliedert es sich in zwei große Teile: zum einen in die direkte Ausgabe von Grafikprimitiven wie Linien und zum anderen in Komponenten für grafische Oberflächen. Das Kapitel behandelt die Themen Fenster, Zeichenketten und Zeichensätze, Farben und Bilder.

Ein alternatives GUI-Framework ist »JavaFX«, das **Kapitel 19** beleuchtet. JavaFX ist Teil des JDK, nicht der allgemeinen Java SE-Spezifikation. Das Kapitel fasst auf der einen Seite die fantastischen neuen Eigenschaften wie 2D/3D, die Transformationen und Animationen zusammen und auf der anderen Seite die neue Komponentenbibliothek von JavaFX.

Kapitel 20 zeigt kurz »Sicherheitskonzepte«, etwa das Sandkastenprinzip, und Sicherheitsmanager auf, zeigt aber auch, wie von Daten eine Prüfsumme gebildet werden kann und wie Daten mit DES oder RSA verschlüsselt werden.

Am Anfang war die JVM fest mit der Programmiersprache Java verbunden, doch das änderte sich im Laufe der Jahre. Heute gibt es unterschiedliche Programmiersprachen, die auf einer JVM laufen – von ganz entwickelten Skriptsprachen bis zu portierten. **Kapitel 21**, »Dynamische Übersetzung, Skriptsprachen, JShell«, stellt Skriptsprachen vor und auch die Möglichkeit, eigene Klassen mit der Compiler-API zu übersetzen.

Java kann nicht alles plattformunabhängig lösen. An einer bestimmen Stelle muss plattformabhängiger Programmcode eingebaut werden. Für den Zugriff von Java auf Nicht-Java-Code wie C(++) definiert Oracle das in **Kapitel 22** vorgestellte »Java Native Interface (JNI)«.

In **Kapitel 23**, »Dienstprogramme für die Java-Umgebung«, geht es um die zum JDK gehörigen Programme und um einige Extratools, die für unsere Arbeit nützlich sind. Im Mittelpunkt stehen Compiler, Interpreter und die Handhabung von JAR-Archiven. Dieses Archivformat ist vergleichbar mit den bekannten ZIP-Archiven und fasst mehrere Dateien zusammen. Mit den eingebetteten Dokumentationskommentaren in Java kann aus einer Quellcodedatei ganz einfach eine komplette HTML-Dokumentation der Klassen, Schnittstellen, Vererbungsbeziehungen und Eigenschaften inklusive Verlinkung erstellt werden. Unter den Programmen, die zu keiner Standardinstallation gehören, sind etwa Tools, die Java-Programme in C-Programme übersetzen, sie neu einrücken und formatieren und Bytecode wieder in lesbaren Java-Quellcode umwandeln.

*-Kapitel

Einsteiger in Java können noch nicht zwischen dem absolut notwendigen Wissen und einer interessanten Randnotiz unterscheiden. Die Insel gewichtet aus diesem Grund das Wissen auf zwei Arten. Zunächst gibt es vom Text abgesetzte Boxen, die zum Teil spezielle und fortgeschrittene Informationen bereitstellen. Des Weiteren enden einige Überschriften auf ein *, was bedeutet, dass dieser Abschnitt übersprungen werden kann, ohne dass dem Leser etwas Wesentliches für die späteren Kapitel fehlt.

Konventionen

In diesem Buch werden folgende Konventionen verwendet:

- Neu eingeführte Begriffe sind *kursiv* gesetzt, und der Index verweist genau auf diese Stelle. Des Weiteren sind *Dateinamen*, *HTTP-Adressen*, *Namen ausführbarer Programme*, *Programmoptionen* und *Dateiendungen* (*.txt*) kursiv. Einige Links führen nicht direkt zur Ressource, sondern werden über *http://tutego.de/go* zur tatsächlichen Quelle umgeleitet, was nachträgliche URL-Änderungen erleichtert.
- Begriffe der Benutzeroberfläche stehen in KAPITÄLCHEN.
- Listings, Methoden und sonstige Programmelemente sind in `nichtproportionaler` Schrift gesetzt. An einigen Stellen wurde hinter eine

Listingzeile ein abgeknickter Pfeil (↩) als Sonderzeichen gesetzt, der den Zeilenumbruch markiert. Der Code aus der nächsten Zeile gehört also noch zur vorigen.

- Bei Methodennamen und Konstruktoren folgt immer ein Klammerpaar, um Methoden/Konstruktoren von Attributen abgrenzen zu können. So ist bei der Schreibweise `System.out` und `System.gc()` klar, dass Ersteres ein Attribut ist und Letzteres eine Methode.
- Hat eine Methode/ein Konstruktor einen Parameter, so steht der Typ in Klammern. Beispiele: `Math.abs(double a)`, `Point(int x, int y)`. Oft wird auch der Parametername ausgelassen. Beispiel: »`System.exit(int)` beendet das Programm mit einem Rückgabewert.« Die Java-API-Dokumentation macht das genauso. Hat eine Methode/ein Konstruktor eine Parameterliste, ist diese aber im Text gerade nicht relevant, wird sie mit »…« abgekürzt. Beispiel: »Die Methode `System.out.print(…)` konvertiert die übergebenen Argumente in Strings und gibt sie aus.« Ein leeres Klammerpaar bedeutet demnach, dass eine Methode/ein Konstruktor wirklich keine Parameterliste hat. Um die Angabe kurz und dennoch präzise zu machen, gibt es im Text teilweise Angaben wie `getProperty(String key[, String def])`, was eine Abkürzung für `getProperty(String key)` und `getProperty(String key, String def)` ist.
- Um eine Gruppe von Methoden anzugeben, symbolisiert die Kennung **XXX** einen Platzhalter. So steht zum Beispiel `printXXX(…)` für die Methoden `println(…)`, `print(…)` und `printf(…)`. Aus dem Kontext geht hervor, welche Methoden gemeint sind.
- Lange Paketnamen werden teilweise abgekürzt, sodass `com.tutego.insel.game` etwa zu `c.t.i.g` wird.
- Um im Programmcode Compilerfehler oder Laufzeitfehler anzuzeigen, steht in der Zeile ein ☠. So ist auf den ersten Blick abzulesen, dass die Zeile nicht compiliert wird oder zur Laufzeit aufgrund eines Programmierfehlers eine Ausnahme auslöst. Beispiel:

```
int p = new java.awt.Point();     // ☠ Compilerfehler: Type mismatch
```

- Bei Compilerfehlern – wie im vorangehenden Punkt – kommen die Fehlermeldungen in der Regel von Eclipse. Sie sind dort anders benannt als in NetBeans bzw. dem Kommandozeilencompiler *javac*. Aber natürlich führen beide Compiler zu ähnlichen Fehlermeldungen.
- Raider heißt jetzt Twix, und Sun ging Anfang 2010 an Oracle. Auch wenn es für langjährige Entwickler hart ist: Der Name »Sun« verschwindet, und der geliebte Datenbankhersteller tritt an seine Stelle. Sun taucht immer nur dann auf, wenn es um eine Technologie geht, die von Sun initiiert wurde und in der Zeit auf den Markt kam, in der Sun sie verantwortete.

Programmlistings

Komplette Programmlistings sind wie folgt aufgebaut:

Listing 1.1 Listing Person.java

```
class Person {
}
```

Der abgebildete Quellcode befindet sich in der Datei *Person.java*. Befindet sich der Typ (Klasse, Aufzählung, Schnittstelle, Annotation) in einem Paket, steht die Pfadangabe beim Dateinamen:

Listing 1.2 Listing com/tutego/insel/Person.java

```
package com.tutego.insel.Person;
class Person { }
```

Um Platz zu sparen, stellt das Buch oftmals Quellcode-Ausschnitte dar. Der komplette Quellcode ist im Internet verfügbar (*http://www.rheinwerk-verlag.de/4469*). Wird ein Ausschnitt einer Datei *Person.java* abgebildet, steht »Ausschnitt« oder »Teil 1«, »Teil 2« ... dabei:

Listing 1.3 Listing Person.java, Ausschnitt

Listing 1.4 Listing Person.java, main(), Teil 1

Gibt es Beispielprogramme für bestimmte Klassen, so enden die Klassennamen dieser Programme im Allgemeinen auf -Demo. Für die Java-Klasse `DateFormat` heißt somit ein Beispielprogramm, das die Funktionalität der Klasse `DateFormat` vorführt, `DateFormatDemo`.

API-Dokumentation im Buch

Attribute, Konstruktoren und Methoden finden sich in einer speziellen Auflistung, die es ermöglicht, sie leicht im Buch zu finden und die Insel als Referenzwerk zu nutzen.

```
abstract class java.text.DateFormat
extends Format
implements Cloneable, Serializable
```

- `Date parse(String source) throws ParseException`
  Parst einen Datum- oder einen Zeit-String.

Im Rechteck steht der vollqualifizierte Klassen- oder Schnittstellenname (etwa die Klasse `DateFormat` im Paket `java.text`) bzw. der Name vom Annotations- oder Aufzählungstyp. In den nachfolgenden Zeilen sind die Oberklasse (`DateFormat` erbt von `Format`) und die implementierten Schnittstellen (`DateFormat` implementiert `Cloneable` und `Serializable`) aufgeführt. Da jede Klasse, die keine explizite Oberklasse hat, automatisch von `Object` erbt, ist diese nicht extra angegeben. Die Sichtbarkeit ist, wenn nicht anders angegeben, `public`, da dies für Bibliotheksmethoden üblich ist. Wird eine Schnittstelle beschrieben, sind die Methoden automatisch abstrakt und öffentlich, und die Schlüsselwörter `abstract` und `public` werden nicht zusätzlich angegeben. In der anschließenden Aufzählung folgen Konstruktoren, Methoden und Attribute. Wenn nicht anders angegeben, ist die Sichtbarkeit `public`. Sind mit `throws` Fehler angegeben, dann handelt es sich nicht um `RuntimeException`s, son-

dern nur um geprüfte Ausnahmen. Veraltete (deprecated) Methoden sind nicht aufgeführt, lediglich, wenn es überhaupt keine Alternative gibt.

Ausführbare Programme

Ausführbare Programme auf der Kommandozeile sind durch ein allgemeines Dollarzeichen am Anfang zu erkennen (auch wenn andere Betriebssysteme und Kommandozeilen ein anderes Prompt anzeigen). Die vom Anwender einzugebenden Zeichen sind fett gesetzt, die Ausgabe nicht:

```
$ java FirstLuck
Hart arbeiten hat noch nie jemanden getötet. Aber warum das Risiko auf sich nehmen?
```

Über die richtige Programmierer-»Sprache«

Die Programmiersprache in diesem Buch ist Englisch, um ein Vorbild für »echte« Programme zu sein. Bezeichner wie Klassennamen, Methodennamen und auch eigene API-Dokumentationen sind auf Englisch, um eine Homogenität mit der englischsprachigen Java-Bibliothek zu schaffen. Zeichenketten und Konsolenausgaben sowie die Zeichenketten in Ausnahmen (Exceptions) sind in der Regel auf Deutsch, da es in realistischen Programmen kaum hart einkodierte Meldungen gibt – spezielle Dateien halten unterschiedliche Landessprachen vor. Zeilenkommentare sind als interne Dokumentation ebenfalls auf Deutsch vorhanden.

Vorwort zur 3. Auflage

Java 9 bietet – wenn wir von der Modularisierung einmal absehen – keine großen Änderungen an der Sprache, daher sind die Anpassungen im Buch gering. Auch die Änderungen in den Bibliotheken sind gering und fordern vom Autor nur wenig Nacharbeit. Auf den ersten Blick klingt es so, als ob Unternehmen deswegen sofort auf Java 9 umstellen können, doch das täuscht. Die Modularisierung ändert so viel am zentralen Distributionskonzept der Java-Archive, dass der Einzug von Java 9 vermutlich viel länger dauert als der von Java 8, wo die neuen Sprachänderungen euphorisch aufgenommen wurden.

Da bei dieser Aufgabe des Buches Java 8 als gesetzt gelten kann, sind alle Beispiele auf die Möglichkeiten von Java 8 gebracht. So gibt es vermehrt bei den Beispielen Lambda-Ausdrücke.

Da die Java-Bibliothek gerne einmal mehrere Möglichkeiten zur Realisierung einer Aufgabe mitbringt, sind die Prioritäten nun verschoben auf den »modernen Weg«. Das heißt, `Path` und `Files` kommen vor `java.io.File` und Date-Time-API vor Date. Außerdem gibt es ein paar weitere kleine Umorganisationen der Kapitel.

Eine größere Änderung liegt in der Einführung von Maven; die Java-Insel-Eclipse-Projekte wurden in Maven-Projekte konvertiert. Das heißt auch, dass Java-Archive nicht Teil der Projekte sind, sondern Eclipse sie beim Start erst vom zentralen Maven-Server beziehen muss.

Vorwort zur 2. Auflage

Das String-Kapitel aus Band 1, »Java ist auch eine Insel«, wurde aufgeteilt, und so finden sich nun fortgeschrittene String-Techniken – insbesondere zu regulären Ausdrücken – in einem neuen Kapitel hier im Buch wieder. Fundamentale Fehler gibt es keine zu berichtigen, ein paar Dinge sind sprachlich genauer; wo es früher etwa »Hash-Tabelle« hieß (eine technische Realisierung), heißt es nun allgemeiner »Assoziativspeicher«.

Natürlich enthält das Buch alle Updates zu Java 8, angefangen bei den Lambda-Ausdrücken, die im ersten Kapitel eine zentrale Rolle einnehmen. Auf der Bibliotheksseite sind besonders hervorzuheben die Stream-API in der Collection-API und die Date-Time-API. Da in Java 8 auch nicht nur Dinge hinzukamen, sondern auch verschwanden, tritt der seltene Fall ein, dass ein Abschnitt entfällt, und zwar über die JDBC-ODBC Bridge. Zudem taucht mit JavaFX eine neue API zur GUI-Entwicklung auf, die die Java Foundation Classes (AWT, Swing, Java 2D ...) auf lange Sicht verdrängen werden, daher gibt es ein neues JavaFX-Kapitel, während die anderen GUI-Kapitel gekürzt wurden. Das Applet-Kapitel wurde komplett entfernt, die Zeit der Java-Applets ist (vorerst) vorbei.

Mit der aktuellen Java-Version 8 sind auch alle Beispiele auf die vorhergehende Version 7 umgestellt. Die Syntax von Java 7 bietet zwar nur kleine Verbesserungen, doch nutzen zum Beispiel alle Demos mit Ressourcen das Sprachkonstrukt `try` mit Ressourcen. Da nun die `File`-Klasse durch neue NIO.2-Typen uninteressant wurde, verschwand `File` aus den Beispielen fast komplett. Und während das IO-Kapitel bisher die Abschnitte zu NIO.2 hinter `File` setzte, ist es nun andersherum: NIO.2 führt, und `File` folgt.

Christian Ullenboom
Sonsbeck, im Oktober 2017