Java - Spezifikationen

- stark typ- und sicherheitsorientiert
- Compiler erzeugt aus Java-Anweisungen Objekt- bzw. Bytecode, der auf einer „virtuellen Maschine", einem in ANSI C programmiertem Bytecode-Interpreter, unabhängig vom Betriebssystem ausgeführt wird
- char-Eingaben im 16bit UTF-Zeichensatz
- viele frei erhältliche Zusatzpakete, Frameworks und IDEs; feste Größe im Open-Source-Bereich

- Produktionszyklus:
  editieren(*.java-Datei) → kompilieren (*.class-Datei) → interpretieren

Einsatzgebiete und Editionen

- **Einsatzgebiete**
  - Applikationen: kommandozeilenorientierte Applikationen, Fensterapplikationen, Rich-Client-Plattformen

  - Applets: werden in HTML eingebunden und laufen im Browser; erfordern bestimmte Klassenstruktur; gilt als veraltete Technologie, da es vom Internet Explorer nie in aktuellen Versionen unterstützt wurde und soll vom neueren JavaFX abgelöst werden

  - serverseitiges Java: Java-Anweisungen für dynamische Webseiten (JSP, JNLP) benötigen eine entsprechende Server-Laufzeitumgebung (z.B. Tomcat, JBoss)

  - mobiles Java: Applikationen für mobile Endgeräte (Handy, PDA, Organizer, Handhelds, Smartphones)

- **Editionen**
  - Java SE (Standard Edition): Grundlage der Java-Applikations-Programmierung

  - Java ME (Micro Edition): angepasst an mobile Endgeräte hinsichtlich Peripherie und Benutzerschnittstellen; eigenständiges SDK

  - Java EE (Enterprise Edition): umfassende Edition für Forschungs-Einrichtungen und Firmen mit Zusatztechnologien (z.B. EJB, Server); eigenständiges SDK auf der Basis des Java SE mit Applikations-Servern usw.

Sprachelemente

- **grundlegende Sprachelemente**
  - Operanden (Literale, elementare und komplexe Datentypen, Felder, Klassen, Interfaces, Objekte, Methoden)
  - Operatoren
  - Kontrollstrukturen
  - Programmblöcke

- Kommentare

- **elementare Datentypen**
  - numerische Datentypen
    - Ganzzahlen: **byte** (-128 bis 128), **short**(-2^{15} bis 2^{15}), **int** (-2^{31} bis 2^{31}), **long** (-2^{63} bis 2^{63})
    - Fließkommazahlen: **float** (32bit), **double** (64bit)
  - char
  - boolean

- **komplexe Datentypen**
  - Arrays
  - Klassen und Objekte
  - Interfaces

- **Operatoren**
  - arithmetische Operatoren (z.B. **++**, **%**)
  - relationale Operatoren bzw. Vergleichoperatoren (z.B. **!=**, **<=**)
  - logische Operatoren (z.B. **&&**, **|**)
  - bitweise bzw. Shift-Operatoren (z.B. **~**, **>>**)
  - einfache Zuweisungsoperatoren und akkumulierende Zuweisungsoperatoren (z.B. **+=**, **|=**)

  - Einteilung nach Anzahl der Operanden
    unäre Operatoren (z.B. **-**, **~**)
    binäre Operatoren (z.B. a*****b)
    ternärer Operator bzw. Fragezeichenoperator: Bed **?** Wert1 **:** Wert2

  - zusätzliche Operatoren
    Seperatoren (z.B. Punktoperator **.** Und **[** bzw. **]**)
    Objekt-Operatoren (z.B. **new**, **this**)

- **Kontrollstrukturen**
  - Konditionale Strukturen (conditionals):
    Verzweigungen **if-else**, **elseif**
    Fallunterscheidungen **switch-case**

  - Schleifen (loops):
    for und erweiterte **for**-Schleife (foreach)
    while
    do-while

  - Ausnahmebehandlung:
    throws
    try-catch[-**finally**]

  - Zusicherung:
    assert

- **Programmblock**

  - Ausdruck:
    gültige Kombination aus lexikalischen Elementen (Token) und Operatoren, die mit einem Semikolon abgeschlossen wird

- Anweisungsblock
  - von geschweiften Klammern umschlossen
  - lokale Variablen existieren nur in ihrem Block und überschreiben dort ggf. vorhandenen globale Felder oder Variablen gleichen Namens

- Kommentare
  - Implementationskommentare:
    - mehrzeiliger Block-Kommentar (**/*** und ***/**)
    - einzeiliger End-Of-Line-Kommentar (**//**)
  - Dokumentationskommentar:
    - Javadoc-Kommentar (**/**** und ***/**)

Syntax und Aufbau

- **Besonderheit**
  - **JAVA PROGRAMME BESTEHEN VOLLSTÄNDIG AUS KLASSEN UND OBJEKTEN, DIE IN PAKETEN ORGANISIERT SIND**
  - Methoden in Klassen und Objekten gekapselt
  - die „main" - Funktion startet den Programmablauf (Ausnahme: Applet)
  - je Datei darf es eine öffentlich zugängliche Klasse geben
  - Konstruktorname = Klassenname = Dateiname
  - Paket ohne selbst vergebenem Name bildet ein anonymes Standardpaket, Paketname = Dateiordner

- **Klassen**
  - Muster: [Sichtbarkeit] [Modus] **class** Klassenname [**extends** OberKlasse]
  - abstrakte Klassen funktionieren als generalisierte Oberklassen und können keine Objekte erzeugen

- **Methodensignaturen**
  - Muster: [Sichtbarkeit] [Modus] Rückgabedatentyp methodenName ([Parameterliste]) [Zusatz]
  - Rückgabetyp **void** liefert keinen Wert zurück
  - Zusatz **abstract** kennzeichnet eine Methode als abstrakte Methode, d.h. sie darf keinen Methodenrumpf besitzen (z.B. bei Interfaces)
  - Klassen mit mindestens einer abstrakten Methode müssen selbst abstrakt sein

- **Felder**
  - Muster: [Sichtbarkeit] [Modus] Rückgabedatentyp eigenschaft
  - Felder können mit einem passenden Wert initalisiert werden
  - Konstanten bekommen den Zusatz „static final" und müssen initalisiert werden
  - Feldeigenschaften können elementare oder komplexe Datentypen sein, wobei elementare als Wert (call by value) und komplexe als Referenz(call by reference, Referenzobjekt) realisiert werden
  - der Wert einer Konstanten darf im weiteren Programmablauf nicht verändert werden

- **Objekte**
  - Erzeuge Instanz(=Objekt) vom Datentyp Klasse:
    KlasseName objektName = **new** Klassenkonstruktor[(Parameterliste)]
  - Zugriff:
    Instanzfeld: objektName.eigenschaft

Instanzmethode: objektName.methodenName([Parameterliste])
Klassenfeld: KlasseName.eigenschaft
Klassenmethode: KlasseName.methodenName([Parameterliste])
konstantes Klassenfeld: KlasseName.EIGENSCHAFT

- Klassenkonstanten und -methoden müssen immer **static** sein

- **Deklarationsalternativen für Array und String**
  - Array:
    Datentyp**[]** arrayName = **new** Datentyp**[**anzahl**]**
    Datentyp**[]** arrayName = **{**a,b,c**}** (mit literaler Initalisierung)
  - String-Objekt:
    String name = **new String(**„Ein String"**)** (↔ **String** name)

- **Schlüsselwörter für Sichtbarkeit und Modi (Auswahl)**
  - **public** öffentlich zugänglich (Schnittstelle)
  - **private** nur innerhalb der Klasse verwendbar
  - **protected** innerhalb der definierenden Klasse und in davon abgeleiteten Klassen verwendbar
  - **static** benötigt als Methode kein Objekt für den Zugriff und kann als Klasse keine Objekte erzeugen(z.B. **System**, **Math**, **GraphicsEnvironment**)
  - **final** darf nicht modifiziert werden
  - **static** final bezeichnet eine Konstante
  - **this** bezieht sich auf das aktive Objekt
  - **super** startet Konstruktor der nächsthöheren Oberklasse

- **Klassentypen in Java**
  - Abgeleitete Klasse: erhält im Klassenkopf den Zusatz **extends** Oberklasse und erbt alle Eigenschaften und Methoden der Oberklasse – auch jene, die diese Oberklasse selbst von einer höheren Oberklasse geerbt hat
  - Abstrakte Klasse: Klasse mit mindestens einer abstrakten Methode, d.h. einer Methode ohne Rumpf, die als Oberklasse für abgeleitete Unterklassen fungiert
  - Adapterklasse: implementiert Interface, so dass deren Methoden nicht implementiert, sondern nur bei Bedarf überschrieben werden müssen
  - Anonyme Klasse: namenlose Erweiterung der umgebenden Klasse (z.B. zur Implementierung von Interface-Methoden)
  - Innere bzw. Lokale Klasse: wird innerhalb einer Klasse definiert und ist nur über diese Klasse erreichbar; wird auch als „Klassenklasse" verwendet, d.h. als statische innere Klasse, z.B. **System.in**)
  - Hüllklasse(„Wrapper"): umschließen elementare Datentypen und versehen sie dadurch mit Objekteigenschaften
  - Oberklasse: Klasse, von der mindestens eine Unterklasse abgeleitet wurde

- **Interface**
  - Klassenschnittstelle, die der implementierenden Klasse zusätzliche Funktionalität bereitstellt (meist im Zusammenhang mit Listeners in der GUI-Programmierung und Threads)
  - ein Ersatz für Mehrfachvererbung: in einer Klassen können mehrere Interfaces verwendet werden
  - Klasse benötigt für angegeben Methodenschnittstellen wenigstens vollständige Deklaration im Rumpf, die bei Bedarf mit entsprechenden Implementationen zu versehen sind
  - alternativ kann man eine innere oder anonyme Klasse von einer der Interface-Adapter-Klassen ableiten, die das entsprechende Interface bereits implementiert, und die benötigte Methode überschreiben(=neu

definieren)
- Interfaces können wie Klassen erben und vererben, allerdings sind Mehrfachvererbungen von mehreren Ober-Interfaces möglich
- Namenskollisionen bei Mehrfachvererbungen werden aufgelöst, indem der Name des Oberinterfaces dem jeweiligen Feld oder der Methode vorangestellt wird

GUI-Programmierung mit JFC

- **JFC (Java Foundation Classes)**
  Hauptkomponenten: AWT, Swing und Java2D
  weitere Komponenten: Accessibility, Drag&Drop, Java3D

  - AWT (Abstract Window Toolkit)
    - Bestandteil der Standardbibliothek seit 1996 und Grundlage für alle grafischen Komponenten
    - „Heavyweight": benötigt viele Systemressourcen
    - Klassen für einfache Layouts mit nativen GUI-Elemente (Peers), die sich am jeweiligen OS orientieren; werden jedoch selten verwendet, sondern i.d.R. mit Swing-Klassen umgesetzt
    - stellt Interfaces für Event-Behandlung bereits (Listener für Mausereignisse und Tastatureingaben)

  - Swing
    - Erweiterung von AWT um zahlreiche visuelle Komponenten, insbesondere Formularelemente, Tabellen und Widgets
    - Swing-Klassen beginnen zur besseren Unterscheidung von ihren AWT-Vorläufer immer mit J (z.B. **JFrame**, **JMenu**, **Jlabel**, usw.)
    - „Lightweight": benötigt weniger Systemressourcen, setzt allerdings schwergewichtige AWT-Komponenten voraus
    - stellt GUI-Komponenten zur Verfügung, die auf allen Systemen möglichst einheitlich ausfallen sollen
    - Alternative zu Swing im eclipse-Universum: SWT(Standard Widget Toolkit)

  - Java2D
    - Sammelsurium von Klassen für geometrische Objekte, Pixelbild-Darstellungen und Texte, wobei Texte als Grafiken gezeichnet und nicht als Strings ausgegeben werden
    - basiert hauptsächlich auf AWT-Klassen zur Darstellung verschiedener grafischer Effekte

- **Grundlagen des GUI-Layouts**
  - ein GUI verbindet grafische Komponenten und fast diese auf oberster Ebene in einem Toplevel-Container zusammen
  - ein Fenster besteht grundsätzlich aus einer Titelleiste (mit Platz für den Titel der Applikation und optionalen Schaltflächen) und der eigentlichen Oberfläche für die Einbettung grafischer Komponenten

  - GUI-Elemente
    - Container: Behälter, die andere Container und Komponenten aufnehmen können (meist JFrame); Top-Level-Container (z.B. **JFrame**) sind mehrschichtige Container, wobei die in ihnen enthaltenen Container die Inhaltsschicht bilden

  - Panels: Zeichenflächen für die grafische Ausgabe(**JPanel**)
  - Komponenten: Schaltflächen (**JButton**), Anzeigentext(**JLabel**), Verzeichnisstrukturen(**JTree**), Menükomponenten(z.B. **JMenuItem**), usw.
  - Ereignisse: werden von Komponenten abgefangen und können von beliebigen Komponenten verarbeitet werden(z.B. Auswahl eines Listenelementes mit der Maus – Element wird farbig unterlegt)
  - Listener(„Beobachter"): legen fest, wie eine Komponente auf ein konkretes Ereignis reagiert;ein Listener kann mehrere Komponenten steuern und eine Komponente von mehreren Listener beobachtet werden
  - Layout-Manager: ordnet die sichtbaren Bestandteile innerhalb der Oberfläche eines Containers an
    - Border-Layout: unterteilt einen Container in 5 Bereiche(**North**, **East**, **South**, **West**, **Center**)und ordnet jedem Bereich eine Komponente zu
    - Flow-Layout: stellt die vorhandenen Komponenten nacheinander von links oben nach rechts unten dar
    - Grid-Layout: teilt einen Container in eine tabellarische Struktur mit x Zeilen und y Spalten mit identischen Zellen auf
    - Gridbag-Layout: basiert auf einer komplexen Tabellenstruktur, ermöglicht jedoch im Gegensatz zum Grid-Layout auch Zellen mit flexibler Größe

- da Container andere Container aufnehmen können, entsteht u.U. eine komplexe Container-Hierarchie(„Wurzelbaum", „Vater-Komponente", „Kind-Komponente")
- bei exakten Pixelangaben bildet die linke obere Ecke der jeweiligen übergeordneten Komponente den Bezugspunkt(0,0)

Konventionen

- **Deklarationen**
  - Bezeichner(=Namen) dürfen nur mit einem Buchstaben beginnen
  - der Name einer Methode soll eine Tätigkeit widerspiegeln,
  - wenn Namen aus mehreren Wörter bestehen, sollen man sie zusammen und den ersten Buchstaben des zweiten Wortes groß schreiben: meineErsteVariable
  - Eigenschaften, Methoden und Objektnamen beginnen mit kleinen Buchstaben:
    zahl, addiereZahl(), addition
  - Klassen- und Interfacenamen dagegen mit einem Großbuchstaben, wobei Interfaces ein großes I vorangestellt werden kann:
    TextFenster, ITextEvents
  - Bezeichner für Klassenkonstanten stehen in Großbuchstaben, wobei mehrere Wörter mit Unterstrich verbunden werden:
    LINKE_SEITE
  - obwohl für Bezeichner alle alphanumerischen UTF-16-Zeichen (plus **$** und **_**) möglich sind, sollte man sich bei der Deklaration auf Standard-ASCII beschränken, um Probleme mit PC zu vermeiden, die einen anderen Zeichensatz verwenden (z.B. Französisch, Russisch)

- **Programmaufbau**
  - eine Anweisung je Programmzeile
  - main-Methode so schlank wie möglich halten
  - Programm so modular wie möglich gestalten

- veränderbare Klassenfelder privat halten und Zugriff stattdessen über setter- und getter-Methoden realisieren
- Variablen deklarieren und nach Möglichkeit immer initialisieren

- **Projektarbeit**
  - mehrere Java-Dateien (*.java, *.class) und zusätzliche Ressourcen (z.B. Grafiken, Sounds) in packages organisieren
  - der Name eines Paketes ergibt sich aus der Umkehrung des Domain-Namens des Paket-Anbieters, wobei Topleveldomain, Domain und eventuelle Subdomains einem Verzeichnis entsprechen
  - beim Anlegen eigener Pakete sollen Namensbestandteile des Paketes eine entsprechende Verzeichnisstruktur abbilden:
    „com.sun.java.Klassenname" ↔ „com/sun/java/Klassenname"
    Verzeichnisnamen werden mit kleinen Buchstaben bezeichnet
  - kleine Projekte können natürlich flachere Strukturen benutzen

  - gehört eine Datei zu einem Paket, muss diese Information an erster Stelle in der java-Datei erscheinen
  - der Zugriff auf Klassen eines fremden Paketes erfolgt nach der Syntax **package** paketordner.KlassenName von einem übergeordneten Wurzel-Verzeichnis

  - Paket-Klassen müssen am Dateinanfang mit **import** paketordner.KlassenName eingebunden werden
  - dabei muss man Achtgeben, dass tatsächlich ein Ordner mit der entsprechenden class-, oder java-Datei existiert

Ressourcen

Internet

Java API Download. http://java.sun.com/docs/

Java Coding Convention. http://java.sun.com/docs/codeconv/

Java-Forum. http://www.java-forum.org/

SWT mit Eclipse. http://www.eclipse.org/swt/

Literatur

Erlenkötter, Hartmut (2001) Java. rororo.

Krüger, Giudo / Stark, Thomas (2008) Handbuch der Java Programmierung. Standardedition Version 6. 5.Auflage. Addison-Wesley.

Ullenboom, Christian (2008) Java ist auch eine Insel. 8.Aufl. Galileo OpenBooks.

Seeboerger-Weichselbaum, Michael (2006) Einsteigerseminar Java mit Eclipse. BHV.