Einführung in die Softwareentwicklung

Übersicht über die Vorlesungsinhalte

Kapitel 1	**Rahmenbedingungen**
Voraussetzungen:	keine
Inhalte:	Es werden die Rahmenbedingungen für die Softwareentwicklung abgesteckt: • Was sind Programme, wie führt der Computer sie aus (Notwendigkeit von Programmiersprachen, Compiler, Interpreter) • Softwareentwicklungsumgebung Eclipse kennenlernen
Vorläufige Vereinfachungen:	Programme müssen vorerst abgetippt werden. Bevor auf das Verständnis der Befehle und die Entwicklung eigener Programme mithilfe der Befehle eingegangen wird, sollen die Rahmenbedingungen, unter denen die Softwareentwicklung stattfinden wird, bekannt sein.
Erworbene praktische Fähigkeiten:	Man ist in der Lage, in Eclipse ein Projekt anzulegen und in dem Projekt einen „Rohbau“ für ein Programm anzulegen. Innerhalb des Rohbaus ist man in der Lage, Befehle zu platzieren (ohne diese zu verstehen). Man kann Programme aus Eclipse heraus starten.

Kapitel 2	**Struktur eines einfachen Programmes**
Voraussetzungen:	Kapitel 1
Inhalte:	Ein einfaches (fertiges) Programm wird analysiert, um ein Gefühl für die Granularität von Programmanweisungen zu bekommen. Es wird dabei aufgezeigt, dass... • es die Möglichkeit gibt, Informationen in Variablen abzulegen, wobei die Informationsart vorher festgelegt werden muss, z. B. `double` • es Befehle gibt, um Informationen zu verarbeiten (hier Rechenoperationen) • es Strukturierungselemente wie Klammern gibt, um Zusammengehörigkeiten zu definieren • es das Semikolon gibt, um Anweisungen abzuschließen • es den Ausgabebefehl `System.out.println( ... )` gibt.
Vorläufige Vereinfachungen:	Die Begriffe „Klasse“ und „Methode“ werden vorerst nicht erläutert. Man beschränkt sich darauf, dass ein Programm einen gewissen „Rohbau“ hat (den Eclipse generieren kann). Nur innerhalb dieses Rohbaus werden vorerst Anweisungen platziert. Es wird vermittelt, dass `System.out.println( ... )` bzw. `Math.sqrt( ... )` Befehle sind, die zur Verfügung stehen, was nur eine sehr vereinfachte Darstellung dessen ist, was sich hinter den Anweisungen technisch verbirgt.

Erworbene praktische Fähigkeiten:	Man ist in der Lage, einfache lineare Programmabläufe zu verstehen wie z. B. das Ausrechnen mathematischer Formeln.

Kapitel 3	**Primitive Datentypen, Operatoren, Fallunterscheidungen**
Voraussetzungen:	Kapitel 1, 2
Inhalte:	Es wird gezeigt, mit welchen Informationsarten Java von Haus aus umgehen kann, z. B. `int`, `long`, `double`, `boolean`. Es wird gezeigt, wie entsprechende Variablen definiert und verwendet werden. Weiterhin werden grundlegende Aktionen, die mit diesen Informationen durchgeführt werden können, beschrieben: • Rechenoperationen wie z. B. Addieren und Multiplizieren • Vergleichsoperationen wie z. B. „gleich“, „kleiner“ • Logische Operationen wie z. B. „und“, „oder“ Mit Fallunterscheidungen (sogenannten `if`-Bedingungen) wird es möglich, bestimmte Programmbestandteile nur unter bestimmten Voraussetzungen auszuführen.
Vorläufige Vereinfachungen:	keine
Erworbene praktische Fähigkeiten:	Man ist in der Lage, einfache lineare Programmabläufe selbst zu entwickeln wie z. B. das Ausrechnen mathematischer Formeln. Man kann mit Rechenoperationen, Vergleichsoperationen und Logischen Operationen umgehen. Man ist auch in der Lage, Programme zu schreiben, die je nach Zustand einiger Werte unterschiedliche Abläufe ausführen, z. B. die Berechnung von pq-Formeln, wobei entweder keine, eine oder zwei Lösungen berechnet werden müssen.

Kapitel 4	**Programmschleifen, Statische Methoden**
Voraussetzungen:	Kapitel 1, 2, 3
Inhalte:	Mit Programmschleifen (`for` / `while` / `do-while`) ist es möglich, Anweisungen wiederholt direkt hintereinander ausführen zu lassen. Es werden die grds. Unterschiede zwischen kopf- und fußgesteuerten Schleifen aufgezeigt. Mit dem Schreiben statischer Methoden wird es möglich, häufig wiederkehrende Aufgaben einmalig zu programmieren und aus anderen Stellen des Programms auf diese Programmierung zurückzugreifen. Diese Technik wird z. B. auch bei der Klasse `Math` angewendet (siehe Vorläufige Vereinfachungen von Kapitel 2). Außerdem wird deutlich, dass die `main`-Anweisung auch nur die Definition einer statischen Methode ist.
Vorläufige Vereinfachungen:	Die Methoden verwenden vorerst alle das Schlüsselwort `static`. Der genaue Unterschied einer Methode mit bzw. ohne diesem Schlüsselwort kann erst erfolgen, wenn der Klassen-/Objektbegriff eingeführt wurde.

Erworbene praktische Fähigkeiten:	Mit den Kenntnissen aus Kapitel 1-4 ist man in der Lage, komplexe Programmabläufe mit bedingten Anweisungen (if-Bedinungen), Schleifen und in Methoden ausgelagertem Programmcode zu schreiben. Mit den Eingabebefehlen aus uebungstools.jar kann man nun kleine Programme wie BMI-Rechner, Zinskalkulationen, mathematische Simulationen (z. B. das aus der Stochastik bekannte Ziegenproblem) oder sogar kleine Spiele (z. B. Stäbchen-Nimm oder Mastermind) schreiben.

Kapitel 5	**Konvertieren primitiver Datentypen, Arrays**
Voraussetzungen:	Kapitel 1, 2, 3
Inhalte:	Es wird gezeigt, wie Informationen, die in einem bestimmten Typ vorliegen (z. B. Kommazahl) in einen anderen Typ umgewandelt werden können (z. B. Ganzzahl). Arrays sind sogenannte List-Datentypen. Mit ihnen wird es möglich, hinter einem einzigen Variablennamen eine ganze Reihe von Werten abzulegen, auf die dann mit einem Index zugegriffen werden kann (ähnlich wie in der Mathematik Vektoren oder Matritzen). Da es sich um eine komplexe Datenstruktur handelt, wird auch erklärt, wie Java solche Daten im Speicher ablegt – nur so können gewisse Eigenheiten beim Arbeiten mit Arrays verstanden werden. Es wird auch die sog. For-each-Schleife erläutert, die in Kapitel 4 vorläufig weggelassen wurde.
Vorläufige Vereinfachungen:	keine
Erworbene praktische Fähigkeiten:	Durch List-Datentypen können nun Programme geschrieben werden, die das Handling von beliebig großen Datenstrukturen benötigen. So ist es nun z. B. auch möglich, Spielfelder wie z. B. für Schach umzusetzen (sei es nun, um ein Spiel zu schreiben oder nur, um Lösungen für das Springer- oder Damenproblem errechnen zu lassen).

Kapitel 6	**Einführung in objektorientierte Programmierung**
Voraussetzungen:	Kapitel 1, 2, 3, 4
Inhalte:	Das objektorientierte Denken hilft zuerst einmal bei der Analyse von Problemstellungen. Da Java eine objektorientierte Programmiersprache ist, können die Analyseergebnisse einfacher in das Design und die Programmierung einer späteren Software einfließen. Dies ist insbesondere für große Softwareprojekte ein ernstzunehmender Vorteil. Es wird der Begriff der „Klasse“ als Bauplan für sog. „Objekte“ eingeführt. Es wird gezeigt, wie Klassen vom Entwickler definiert/programmiert werden. Weiterhin wird gezeigt, wie in einem Programm dann Objekte (gemäß des durch die Klassen definierten „Bauplans“) erzeugt und verwendet werden können. Im übrigen: In Kapitel 1 wurde der Begriff der Klasse noch nicht erläutert – bestimmte Anweisungen der ersten einfachen Javaprogramme wurden als „notwendiger Rohbau“ abgetan. Nun wird darüber aufgeklärt, dass ein Programm aus sehr vielen Klassen bestehen kann und es ergibt sich daraus,

	dass Anweisungen notwendig sind, die Klassen definieren können.
Vorläufige Vereinfachungen:	keine
Erworbene praktische Fähigkeiten:	Man sollte in der Lage sein, sich z. B. mit der zur Verfügung gestellten alten Hausarbeit auseinanderzusetzen und alle wesentlichen Objekte zu identifizieren, mit denen das Programm hantieren muss. Man sollte daraus schon grob ein Klassenmodell erstellen können, dass die wichtigsten Klassen des zukünftigen Programms mit den wichtigsten Attributen und Methoden benennt. (Programmatisch lösbar wäre die Aufgabenstellung jedoch noch nicht, z. B. weil noch nicht bekannt ist, wie man Informationen aus Dateien ausliest.)

Kapitel 7	**private, final, static**
Voraussetzungen:	Kapitel 1, 2, 5, 6
Inhalte:	Der Zugriffsmodifizierer `private` wird erklärt. Er dient dazu, bestimmte Dinge einer Klasse nach außen hin zu verbergen (z. B. um zu verhindern, dass andere Programmteile willkürlich die Attribute eines Objektes ändern können) Der Modifizierer `final` wird erklärt. Er dient bei Attributen dazu, Konstanten zu definieren, d.h. ein als „final“ markiertes Attribut kann nur genau 1x einen Wert zugewiesen bekommen. Der Modifizierer `static` wird erklärt. Er wird zwar schon seit Kapitel 4 verwendet, jedoch kann erst jetzt (wo bekannt ist, was Klassen/Objekte sind) dessen Auswirkung und Einsatzzweck erklärt werden.
Vorläufige Vereinfachungen:	keine
Erworbene praktische Fähigkeiten:	Die Schlüsselwörter `static` und `final` dienen insbesondere dazu, die Eigenschaften von Klassen präziser den tatsächlichen Gegebenheiten nachzubilden, was die Softwarequalität erhöht. Das Schlüsselwort `private` dient der Steuerung, welche Eigenschaften eine Klasse anderen zugänglich machen will und welche nicht – ein wichtiges Instrument, um zu steuern, wie die eigenen Klassen von fremden Klassen verwendet werden sollen.

Kapitel 8	**Klasse „String“, Pakete und Importe, Dialogboxen**
Voraussetzungen:	Kapitel 1, 2, 3, 5, 6
Inhalte:	Objekte der Klasse `String` bilden Zeichenketten, also Texte ab. Zwar wurden Strings bisher schon indirekt verwendet, jedoch wurden sie bisher noch nicht als Objekte betrachtet und genutzt. Dies wird zum Anlass genommen, die verschiedenen Möglichkeiten zur Stringverarbeitung (Strings teilen, Zeichen in Strings suchen, Strings in Großbuchstaben umwandeln usw.) kennenzulernen. Mit sog. Paketen können größere Javaprogramme strukturiert und gegliedert werden. Der Zugriff einer Klasse auf eine andere Klasse eines anderen Paketes ist dann nur noch möglich, indem der `import`-Befehl verwendet wird. Bisher standen Befehle zum Abfragen von Informationen vom Anwender nur über bereitgestellte Hilfskomponenten (uebungstools.jar) zur Verfügung. Es

	wird nun eine Java-eigene Möglichkeit hierfür aufgezeigt.
Vorläufige Vereinfachungen:	keine
Erworbene praktische Fähigkeiten:	Es ist nun möglich, Programme zu schreiben, die Texte (ggf. auch vom Anwender eingegebene) analysieren. Es sind damit z. B. Palindromtester möglich. Weiterhin können nun alle Programme sehr gut mit Hilfe von Klassen und Paketen strukturiert werden, was die Entwicklung größerer Softwareprojekte möglich macht.

Kapitel 9	**Javadoc, Jars**
Voraussetzungen:	Kapitel 1, 2, 3, 5, 6, 7
Inhalte:	Bei Javadoc handelt es sich um Kommentare, die in einer speziellen Syntax gehalten sind. Dadurch sind sie maschinell auswertbar. Diese Kommentare werden vor Klassen, Attributen, Konstruktoren oder Methoden platziert, um eben diese zu beschreiben. Programme wie Eclipse werten die Kommentare aus und können sie z. B. als Hilfestellung während der Programmierung einblenden. Bei jars handelt es sich im Prinzip um zip-Dateien, die alle compilierten Klassen eines Programms beinhalten. Wenn Javaprogramme ausgeliefert werden, dann geschieht dies in der Regel als jar-Datei. Diese können von einer JVM ausgeführt werden. Eine solche ist auf fast jedem Betriebssystem vorhanden und mit jar-Dateien verknüpft (in Windows genügt also z. B. ein Doppelklick auf eine jar-Datei und das Programm wird ausgeführt).
Vorläufige Vereinfachungen:	keine
Erworbene praktische Fähigkeiten:	Durch Javadoc wertet sich der Programmcode qualitativ auf und wird leichter wartbar. Jars sind das Mittel der Wahl, um Programme an Endanwender auszuliefern.

Kapitel 10	**Vererbung und Polymorphie**
Voraussetzungen:	Kapitel 1, 2, 3, 5, 6
Inhalte:	Hierbei handelt es sich um ein Konzept, mit dem Gemeinsamkeiten verschiedener Klassen in eine gemeinsame Klasse ausgelagert werden können. Durch diese Zusammenfassung der Gemeinsamkeiten können Objekte verschiedener Klassen gleichbehandelt werden, solange nur die gemeinsamen Eigenschaften der Objekte relevant sind.
Vorläufige Vereinfachungen:	keine
Erworbene praktische Fähigkeiten:	Es können bestimmte Programmabläufe deutlich vereinfacht werden. Beispiel: In einem Programm müssen die Eigenschaften von Lagerstätten/Arbeitsstellen/Tankstellen verwaltet werden. Allen gemeinsam ist, dass sie einen Namen, eine x-Koordinate und eine y-Koordinate haben. Wer

	Vererbung und Polymorphie verstanden hat, wird diese Gemeinsamkeiten z. B. in eine Klasse „Ort“ auslagern. Denn: Es ist nun möglich, z. B. eine Methode `berechneEntfernung(Ort a, Ort b)` zu schreiben, der es völlig egal ist, ob sie eine Lagerstätte & Arbeitsstelle oder zwei Arbeitsstellen oder eine Arbeitsstelle & Tankstelle usw. übergeben bekommt.

Kapitel 11	**Fehlerbehandlung, ArrayList**
Voraussetzungen:	Kapitel 1, 2, 3, 5, 6, 10
Inhalte:	Es wird gezeigt, was exakt passiert, wenn während eines Programmablaufs ein Fehler auftritt. Außerdem werden Möglichkeiten aufgezeigt, noch im Programm auf Fehlerzustände zu reagieren und den Zustand selbst zu beheben. Am Beispiel der `ArrayList` wird eine Alternative für normale Arrays aufgezeigt. Diese Alternative bietet z. B. die Möglichkeit, die Größe nicht vorher festlegen zu müssen.
Vorläufige Vereinfachungen:	keine
Erworbene praktische Fähigkeiten:	Durch Fehlerbehandlung ist es z. B. möglich, auf Fehleingaben des Anwenders zu reagieren – bisher sind Programme oft abgebrochen. Durch ArrayListen gibt es bequemere Möglichkeiten, Mengen von gleichartigen Informationen/Objekten im Speicher abzulegen.

Kapitel 12	**Ein-/Ausgabe**
Voraussetzungen:	Kapitel 1, 2, 3, 6, 10, 11
Inhalte:	Es wird das Ein-/Ausgabekonzept von Java vorgestellt. Es findet bei jedem Datenfluss von einer externen Datenquelle oder zu einem Datenziel Anwendung, z. B. Netzwerkkommunikation oder Abspielen von Audiodateien. Der Fokus liegt jedoch auf Dateiein-/-ausgabe, d.h. dem Lesen aus Dateien oder dem Schreiben in Dateien.
Vorläufige Vereinfachungen:	keine
Erworbene praktische Fähigkeiten:	Es ist nun möglich, Informationen über das Ende des Programmablaufs zu erhalten (indem sie in Dateien geschrieben und beim nächsten Start wieder gelesen werden). Außerdem besteht nun die Möglichkeit, Informationen aus einer anderen Quelle als durch eine manuelle Eingabe vom Anwender in das Programm zu bekommen.

Hinweis: Die Voraussetzungen sind Referenzen auf vorhergehende Kapitel, die behandelt worden sein müssen, bevor das aktuelle Kapitel behandelt wird. Es werden ausschließlich thematische Abhängigkeiten dokumentiert, keine praktischen. (D.h. der theoretische Stoff kann verstanden werden, ggf. gibt es jedoch im Foliensatz Übungen oder Beispiele, für die auch andere vorhergehende Kapitel bekannt sein müssen.)