Zusammenfassung PRG1

Objektorientierte Programmierung

Sie können den Unterschied zwischen einer Klasse und einem Objekt erklären.

Objekte repräsentieren ‚Dinge' der realen Welt oder eines Problembereiches. Eine Klasse ist der Bauplan für ein Objekt.

Sie kennen die Grundkonzepte von Abstraktion und Modularisierung.

Die Abstraktion bezeichnet einen Vorgang, bei dem die Sicht von der realen Welt oder einem Modell in ein (vereinfachtes) Modell gefasst wird, welches die wesentlichen Bestandteile hervorhebt. Kurz: Man konzentrert sich nur aufs Wesentliche.
Bei der Modularisierung geht es darum, eine Gesamtaufgabe in Teilaufgaben mit erforderlichen Schnittstellen zu zerlegen. Somit kann man unabhängig voneinander daran arbeiten.

Sie können Beziehungen zwischen Klassen erkennen und gemäss Lehrbuch/BlueJ darstellen.

Klassen- / Objektdiagramm

Sie kennen das Konzept des Information Hiding und wissen, wie es durch Zugriffsmodifizierer unterstützt wird.

Die Inhalte einer Klasse sind gegenüber anderen Klassen nicht sichtbar. Mit den Zugriffsmodifizierer kann geregelt werden, worauf wer zugreiffen darf. Nur ein möglichst eines Interface ist sichtbar (Was). Die Implementation (Wie) ist verborgen.

Sie kennen die Begriffe Kohäsion und Kopplung.

Kohäsion steht für den inneren Zusammenhalt einer Klasse (eine Klasse = eine Einheit, wenig Codedublikationen).
Die Kopplung für den Zusammenhang zwischen Klassen (möglichst wenig Beziehungen und möglichst unabhängig voneinander).

Sie wissen, wie sich Kopplung im Source Code manifestiert.

Möglichst wenig Beziehungen zwischen Klassen und möglichst unabhängig voneinander.

Sie kennen 3 Folgen von gutem und von schlechtem Klassendesign.

Gut:

- Bessere Lesbarkeit
- Bessere Testbarkeit
- Bessere Wiederverwendung

Schlecht

- Langsame Entwicklung
- Schwierige Entwicklung
- Fehlerhafte Entwicklung

Sie können das Konzept der Vererbung anwenden und kennen Vor- und Nachteile.

Vorteil:

- Verhindert Code-Duplikatoinen
- Unterstützt Code-Wiederverwendung
- Vereinfacht den Unterhalt
- Unterstütz einfaches erweitern der bestehenden Klassen

Nachteil:

- Erhöht die Kopplung

Sie können das Konzept des Subtyping, insbesondere das entsprechende Handling von Referenzvariablen, nutzen.

Analog zur Klassenhierarchie bilden Klassentypen auch eine Hierarchie.

- erlaubte Zuweisungen (implizites Casting): vom Sub-Klassentyp zum Basis-Klassentyp
- umgekehrt braucht es explizites Casting: nur vom Basis-Klassentyp → Sub-Klassentyp
- Aber Achtung: Zur Laufzeit können trotz Casting Fehler auftreten!

```
Fahrzeug fahrzeug1, fahrzeug2;  Auto auto;  Velo velo;

auto = new Auto();
fahrzeug1 = auto;          ← Korrekt (implizites Casting)
auto = (Auto)fahrzeug1;    ← Korrekt (explizites Casting)

velo = (Velo)auto;         ← Compiler-Fehler! (Subklasse → Subklasse)

fahrzeug2 = auto;
velo = (Velo)fahrzeug2;    ← Laufzeit-Fehler!
```

Sie verstehen das Konzept der Polymorphie von Klassentyp Variablen.

Variablen eines Klassentypen in Java sind polymorph. Sie können sowohl die deklarierten Objekte sowie auch alle die davon abgeleiteten Objekte aufnehmen.

Klassen in Java

Sie kennen den prinzipiellen Aufbau einer Klasse.

Der Rahmen (z.B. public class Testklasse), gefolgt von den Instanzvariablen, dem Konstruktor (z.B. public Testklasse(int x)) und den Methoden.

Sie wissen, wozu Instanzvariablen, Methoden und Konstruktoren dienen.

Instanzvariablen definieren die Eigenschaften des Objektes, die Methoden, wie sich das Objekt verhalten soll und der Konstruktor um die Initialisierung vorzunehmen.

Sie kennen mindestens je 3 Eigenheiten von Instanzvariablen, Klassenvariablen, lokalen Variablen und von formalen Parametern.

Instanzvariablen
- Existieren solange wie das Objekt
- Innerhalb der ganzen Klasse sichtbar
- Zugriff typisch via Methode
- Werden automatisch initialisiert

Klassenvariablen
- Existieren solange wie die Klasse
- Innerhalb der ganzen Klasse sichtbar
- Zugriff je nach Zugriffsmodifizierer

Lokalen Variablen
- Existieren innerhalb eines Blockes
- Innerhalb des Blockes sichtbar
- Besitzen keine Zugriffsmodifizierer
- Müssen explizit initialisiert werden

Formaler Parameter

- „lokale Variablen“, weche mit dem Methodenkopf deklariert werden
- Beim Methdenaufruf werden sie mit dem aktuellen Parameter initialisiert
- Existieren bis die Methode abgearbeitet ist

Sie können die Rollen von Methodenkopf und Methodenrumpf erklären.

Methodenkopf: Zugriffsmodifizierer, Rückgabewert, Signatur (Name & Parameter)
Methodenrumpf: alles, zwischen den geschweiften Klammern

Sie können einen Konstruktor-Kopf ohne/mit Parameter(n) formulieren.

Mit: public int Summe(int zahl1, int zahl2)
Ohne: public void printKosten()

Sie können einen Methoden-Kopf ohne/mit Parameter bzw. ohne/mit Rückgabewert formulieren.

Mit/Mit: public int Summe(int zahl1, int zahl2)
Mit/Ohne: public void setGewinn(int zahl1)
Ohne/Mit: public int getGewinn()
Ohne/Ohne: public void printKosten()

Sie können überladene Konstruktoren und Methoden erstellen.

Name der Methode / Konstruktor bleibt gleich, jeodhc muss die Anzahl der Parameter verschieden sein oder die jeweiligen Dateittypen anders sein.

Sie können Accessor Methoden formulieren.

Public void getZins()

Sie können Mutator Methoden formulieren.

Public void setZins(int Wert)

Sie können den Unterschied zwischen einem formalen und einem aktuellen Parameter erklären.

Formel Parameter: public void setZins(**int Wert**)
Aktueller Parameter: setZins(**5**)

Sie können Instanzvariablen und lokale Variablen deklarieren.

Instanz: private int x = 1;
Lok. Variablen: int x = 1;

Sie wissen, wie man mit Objekten arbeitet.

Einflussnahme auf Objekte

Sie wissen, wieso und wie man Klassen importiert.

Oft vorkommende Codesegmente / Codelösungen kann man so einfach importieren und muss es nicht selber in den Code implementieren (Klassen-Bibliothek)
Der Import erfolg über Syntax: import java.util.ArrayList
Automatisch Importiert: import java.lang (z.B. für String oder Math)

Sie können Klassenmethoden definieren.

Public static long round(double x)

Sie können Klassenvariablen definieren.

private static int x = 5;

Sie können Konstanten und Klassenkonstanten definieren.

Private final int x =5;
Private static final int x=5; (normerweise wenn dann, dann Klassenkonstante)

Sie können eine main()-Methode implementieren.

```
Public static void main(String[] args)
{
        Game game = new Game();
        game.start();
}
```

Sie verwenden die Interfacebeschreibung einer Klasse (vgl. API-Dokumentation), um sie in Ihren Programmen zu nutzen.

```
 **/
* Beschreibung der Methode. Der zweite Satz kommt nicht mehr in der Kurzvorschau
* @param Beschreibung Parameter 1
* @param Beschreibung Parameter 2
* @Return Beschreibung des Rückgabewerts
*/
```

Sie können einfache Klassenhierarchien in Java implementieren.

Dies kann mir vererbung erreicht werden.

Sie verstehen die Bedeutung der Klasse Object und kennen überblicksmässig deren Basisfunktionalität.

Die Klasse Object ist die oberste Klasse in Java und wird von jedem Java Programm automatisch eingebunden. Die wichtigsten Funktionalitäten:

- Boolean Equals(Object obj)
- String toString()
- Int hashCode()

Sie kennen die vier Zugriffsmodifizierer und wissen, wie man sie anwendet.

Public:	Sichtbar innerhalb Klasse, für User und für abgeleitete Klassen (Methoden)
Private:	Sichtbarkeit nur innerhalb der Klasse. Verdeckt für User und abgeleitet Klassen.
Protected:	Sichtbar innerhalb Klasse und deren Ableitungen + für Klassen desselben Packets
- (keiner):	Sichtbar innerhalb Klasse + für Klassen desselben Packets

Datentypen und Operatoren in Java

Sie können die acht primitiven Datentypen benennen und einordnen

Byte	Länge: 1 Byte	Boolean	Länge: 1 Byte
Short	Länge: 2 Byte	Char	Länge: 2 Byte
Int	Länge: 4 Byte	Float	Länge: 4 Byte
Long	Länge: 8 Byte	Double	Länge: 8 Byte

Sie können vergleichende Bedingungen formulieren und logische Operatoren anwenden.

==	gleich
< / >	Kleiner / Grösser als
&&	and
||	or
^	xor

Sie können arithmetische und logische Ausdrücke auswerten.

True ^ (false || true) = false
7 / (2 + 1.0) = 2,33 (double)

Sie können das implizite und das explizite Typecasting anwenden.

Float x; x=5+5.0f; x=10.0f
Char -> double (zahlenwert in Unicode)

Sie können Arrays deklarieren, initialisieren sowie auf Arrays zugreifen.

```
Int[] a;
a = new int[5];
oder: int[] a = new int[5];
oder: int[] a = {2,4,5,6,5}

Zugriff aufs Array:
For(int n=0;n<a.length; n++) {
        System.out.println(a[n]);
}
```

Sie können einfache Methoden mit Array-Parametern implementieren und diese aufrufen.

```
public int[] arrayInhaltVerdoppeln(int[] a)
{
        for(int n=0; n<a.length;n++) {
                a[n] = a[n]*2;
        }
        Return a;
}
```

Selektion und Iteration in Java

Sie können die Selektion mit if anwenden.

```
If (n < 0) {
        isNegative();
 } else if (number = = 0) {
        isZero();
} else {
        isPositiv();
}
```

Sie können eine einfache Selektion mit switch formulieren.

```
switch(tageszeit) {
        case 1: tagesTeil = „morgen“;
                break;
        case 2: tagesTeil = „mittag“;
                break;
        default: tagesTeil = „sonst was“;
                break;
        }
```

Sie kennen die Eigenheiten der verschiedenen Schleifen-Anweisungen (while, do, for, foreach).

While: Anzahl Durchläufe unbekannt, evtl. keinen Durchlauf
Do while: Anzahl Durchläufe unbekannt, mindestens einen Durchlauf
For: Anzahl Druchläufe berechnenbar, evtl. keinen Durchlauf
Foreach: Zugriff auf Collection / Array möglich

```
        For(int counter : arrayX) {
                System.out.println(counter);
        }
```

- **Sie können für eine gegebene Aufgabenstellung die geeignete Schleifen-Anweisung bestimmen.**

Je nach bekanntheit der Durchläufe bzw. ob min. einmal durchlaufen muss.

Sie können einfache Methoden mit Schleifen-Anweisungen implementieren.

Siehe oben.

Klassen und Methoden der Java Klassenbibliothek

Sie können Klassen der Klassenbibliothek importieren und wissen, wieso man Klassen importiert.

An erster Stelle im Code mit:
Import java.util.ArrayList
Import java.util.Iterator

Sie wissen, dass die Klasse String existiert, und können Variablen des Typs String deklarieren.

String muss nicht explizit importiert werden, das es im Package ‚java.lang' vorhanden ist und somit standardmässig bei jeder Java-Applikation dabei ist.
Deklarartion: private String x = „hallo welt";

Sie können die Stringkonkatenation mit + ausführen.

System.out.println(wort1 + „ „ + wort2);

Sie können Strings vergleichen.

If(wort1.equals(wort2))

Sie verwenden die Methode System.out.println().

System.out.println(„Hallo „ + name);

Sie kennen überblicksmässig das Konzept der Wrapper-Klassen (z.B. Integer, Long, Float).

Zu jedem primitiven Datentyp in Java gibt es eine korrespondierende Wrapper-Klasse. Diese kapselt die primitive Variable in einer objektorientierten Hülle und stellt eine Reihe von Methoden zum Zugriff auf die Variable zur Verfügung. Zwar wird man bei der Programmierung meist die primitiven Typen verwenden, doch gibt es einige Situationen, in denen die Anwendung einer Wrapper-Klasse sinnvoll sein kann.

Sie können Wrapper Klassen anwenden und verstehen den auto-boxing / unboxing Mechanismus.

Boxing: Einpacken des primitiven Datentypes in ein (z.B. float) Arraylist
Unboxing: Entpacken
Wenn der Compiler genung informatoinen hat, kann er dass automatisch machen.

Sie verstehen "Generics" und können sie anwenden.

Typensichere Collections. Ohne Generics können diverse Typen in eine Collection aufgenommen werden. Aber es empfielt sich, nur auf Generics zu setzen.

Mit Generics:

```
ArrayList<Balloon> balloons;
balloons = new ArrayList<Balloon>();
…
balloons.add(new Balloon(„red"));
…
for(Balloon b : balloons) {
        b.blowup();
}
```

Sie kennen die Container-Klassen überblicksmässig.

Java Collections Framework (JCF) stellt wichtige Container zur Verfügung:

- Set (keine 2 gleichen Objekte)
- List (Arraylist)

- Queue
- Map (stets 2 Objekte)

Sie kennen das Konzept der Iteratoren und können dieses anwenden.

Zwei Möglichkeiten: Iterator und ListIterator. ListIterator hat aber ein paar Funktionen mehr.

• Sie kennen verschiedene Möglichkeiten, um alle Elemente einer Collection (z.B. ArrayList) zu traversieren bzw. bearbeiten zu können (for, foreach, Iterator).

Import java.util.*

Deklaration und Initialisierung:
ArrayList<String> words = new ArrayList<String>();

Iterator:
```
for (Iterator<String> i = words.iterator; i.hasNext();  ) {
        System.out.println(i.next());
```

Foreach-Schlaufe:
```
For (String s : words) {
        System.out.println(s);
}
```

For-Schlaufe:
```
for(int i=0, n = words.size(); i<n; i++) {
        System.out.println(words.get(i);
}
```

Sie können die wichtigsten Methoden der Klassen List, ArrayList, Set, HashSet, Map und HashMap anwenden.

size(), isEmpty(), get(), set(), iterator() und listIterator(), add()

Algorithmen

Sie kennen die Merkmale eines Algorithmus.

Verfahren, mit dem eine Problemkategorie gelöst werden kann

- Schrittweises Verfahren
- Nächster Schritt eindeutig
- Ausführbare Schritte
- Endet nach denlich vielen Schritten

Sie können den Begriff der Komplexität eines Algorithmus definieren.

Komplexität wirkt sich auf die Zeitdauer aus:

proc0(n) = O(n^2)

- eine Verdoppelung von n
  → $2^2 = 4$ fache Verlängerung der Ausführungszeit

$$\frac{(2n)^2}{(n)^2} = 4$$

- Verdreifachung von n
  → $3^2 = 9$ fache Verlängerung der Ausführungszeit

$$\frac{(3n)^2}{(n)^2} = 9$$

- Verzehnfachung von n
  → $10^2 = 100$ fache Verlängerung der Ausführungszeit

$$\frac{(10n)^2}{(n)^2} = 100$$

Sie verstehen den Begriff ‚Ordnung eines Algorithmus' und können die Ordnung einer gegebenen Funktion bestimmen.

Abhängigkeit des Aufwandes von den Eingabedaten.

Geordnet nach Wachstumsgrösse (kleines Wachstum zuerst)

1. Konstant O(1)
2. Logarithmisch O(ln(n))
3. Linear O(n)
4. n log n O(n *ln(n))
5. Expnential $O(d^n)$
6. Fakultät O(n!)

Sie kennen das Prinzip der Rekursion.

Die Rekursion ruft sich selber auf. Sie besteht immer aus einer Rekursionsbasis und einer Rekursionsvorschrift.

Sie können die Rekursionsbasis und Rekursionsvorschrift bestimmen.

Die Rekursionsbasis ist die Grundlage um das Problem zu lösen.
Die Rekursionsvorschrift gibt an, was passiert, um das Problem zu vereinfachern.

Sie wissen, wie eine rekursive Methode abgearbeitet wird.

Die Rekursion ruft sich so oft auf, bis die Rekursionsbasis erreicht wird und arbeitet sich danach „rückwärts" wieder ab.

Sie können das Konzept „Rekursion" an einem Beispiel erklären, z.B. mit den Türmen von Hanoi.

Fakultät berechnen:

```
Public long fak(int n)
{
        If(n==1 || n==0) {        //Rekursionsbasis
                return 1;
        } else
        return n * fak(n-1);      //Rekursionsvorschrift
        }
}
```

Sie können einfache rekursive Methoden implementieren.

Siehe vorheriger Punkt.

Sie können das Konzept „Backtracking" erklären und anwenden.

Systematische Suche in einem vorgegebenen Lösungsraum. Läuft man in eine Sackgasse, wird jeweils der letzte Schritt rückgängig gemacht.

Sie können an einem Beispiel erläutern, was die Eigenschaft „Stabilität“ eines Sortieralgorithmus bedeutet.

Zwei gleich grosse Zahlen bleiben in der Anfangsreihenfolge nebeneinander.

Sie kennen die Ordnungen der behandelten direkten/einfachen und höheren Sortieralgorithmen.

Direktes Einfügen	$O(n^2)$	stabil
Direktes Auswählen	$O(n^2)$	stabil
Direktes Austauschen (Bubblsort)	$O(n^2)$	stabil
Shellsort	$\sim O(n^2)$	instabil
Quicksort	O(n*logn)	instabil
Mergesort	O(n*logn)	stabil

Sie können die Arbeitsweise der behandelten direkten/einfachen und höheren Sortieralgorithmen erklären und an einfachen Beispielen aufzeigen.

Die höheren Algorithmen vertauschen zu erst ganz grob und werden danach immer detailierter. Dies geht in der Regel optimaler als die einfachen Algorithmen. Den die einfachen durchkämmen bei deiner Suche das ganze Feld „detailiert“ vom Anfang bis zum Schluss.

Sie kennen die Vor- und Nachteile der behandelten Sortieralgorithmen.

• Sie verstehen die im Unterricht betrachteten konkreten Implementierungen von Sortieralgorithmen, z.B. von Quicksort.

Sie haben einen Überblick über Bibliotheksklassen von Java zum Sortieren.

Java.util.Arrays
Java.util.Collections

• Sie können einen einfachen endlichen Automaten verstehen, aufzeichnen und in Java umsetzen.

Sie können die Ränder aller Teilworte eines Wortes bestimmen.

Wort: abbabaabaa
Rand: a

• Sie können einen „Musterautomaten“ zu einem einfachen Muster aufzeichnen.

Sie kennen den Aufwand und die Merkmale des KMP Suchalgorithmus, von Quicksearch und von Optimal Mismatch.

KMP	O(m+n)
Quicksearch	sehr schnell
Optimal Mismatch	noch bisschen schneller

Software Engineering

Sie können die fünf Hauptdisziplinen des Software Engineering benennen.

1. Kundenspezifikation 15%
2. Systemspezifikatoin 20%
3. Realisierung 40%
4. Test 20%
5. Verteilung und Einsatz 5%

Sie sehen die Notwendigkeit des Testens ein und können Gründe dafür benennen.

Qualität der Software (z.B. Performance, Fehler, Sicherheit, Benutzbarkeit,..)

Sie kennen die Haupteigenschaften von Blackbox-Test, Whitebox-Test, Walkthrough und Debugger-Einsatz.

Blackbox-Test

- Man kennt den Code nicht
- Betrachtung gilt dem Ein- Ausgängen
- Testspezifikation wird anhand der Programmspezifikation erstellt
- Man schaut, ob die erwarten Outputs vgl. mit den Inputs übereinstimmen

Whitebox-Test

- Code ersichtlich
- Alle verwendeten Variabeln werden initialisiert
- Alle Programmpfade müssen mindestens einmal durchgearbeitet werden

Walkthrough

- Fremde Person studiert den Programmcode
- Geleitet durch den Programmierer

Debugger

- Hilfsmittel der Entwicklungsumgebung
- Ermöglicht Hilfe beim Testen z.B. Step-by-Step oder Breakpoints

Sie können auf Grund einer einfachen Programmspezifikation eine Testspezifikation erstellen.

Unterteilen in: Normalfälle, zulässige Spezialfälle und unzulässige Spezialfälle

Sie können Ihren Code gemäss den Richtlinien von javadoc unter Verwendung von javadoc Tags kommentieren.

```
**/
* Dies ist eine Klassendokumentation
* @autor T. Stuber
* @version 0.1
*/

**/
* Beschreibung der Methode. Der zweite Satz kommt nicht mehr in der Kurzvorschau
* @param Beschreibung Parameter 1
* @param Beschreibung Parameter 2
* @Return Beschreibung des Rückgabewerts
*/
```

Sie können die drei Vorgehensschritte beim Refactoring beschreiben.

1. Testspezifikation schreiben und Code testen
2. Code redesignen und ohne funktionale Erweiterungen erneut testen
3. Code mit zusätzlichen funktionalen Erweiterungen versehen und erneut testen