

FOP Reference Sheet

Jonas Milkovits

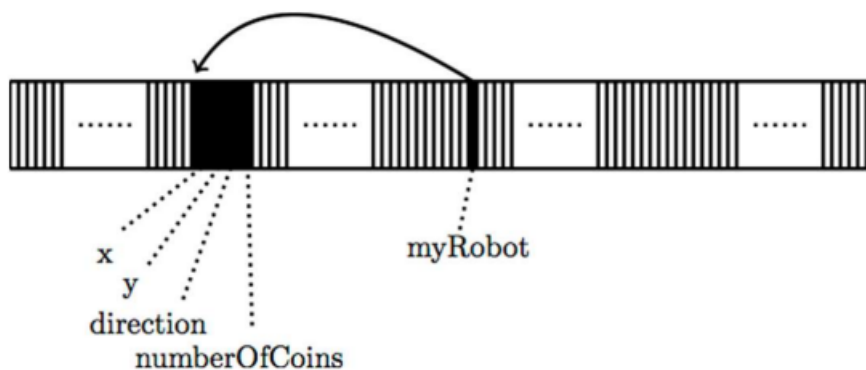
Last Edited: 7. April 2020

Inhaltsverzeichnis

1	Computerspeicher	1
2	Datenstrukturen	1
3	Datentypen	2
4	Fehler	2
5	Graphics (java.awt.Graphics;)	3
6	Interfaces	3
7	Klassen	4
8	Konversionen	4
9	Methoden	4
10	Packages und Zugriffsrechte	5
11	Programme und Prozesse	5
12	Schleifen und if	5
13	String (java.lang.String)	6
14	Syntax	6
15	Vererbung	7

1 Computerspeicher

Unsere Vorstellung	▷ groSSes Feld aus Maschinenwörtern mit eindeutiger Adresse
Erzeugung eines neuen Objekts	▷ Reservierung von ungenutztem Speicher in ausreichender GröSSe
Referenz	▷ Name der Variable, die die Anfangsadresse des Objekts speichert ▷ Kann auch an komplett anderer Stelle als das Objekt gespeichert sein
Speicherort primitiver Datentypen	▷ Name verweist tatsächlich auf Speicherstelle, an der Wert abgespeichert wird
Prozessablauf	▷ Program Counter enthält Adresse der nächsten Anweisung ◊ Zählt nach jeder Anwendung hoch und verweist auf nächsten Speicher ▷ CPU verarbeitet parallel die momentane Anweisung aus Program Counter
Methodenausführung	▷ Einrichtung einer Variable StackPointer bei Programmstart ▷ StackPointer enthält die Adresse des Call-Stacks ▷ Bei Methodenaufruf wird im Speicher Platz reserviert, genannt Frame ▷ Frame wird dann auf dem Call-Stack abgelegt ▷ Der StackPointer wird dann mit der Adresse des neuen Frames überschrieben ▷ Methodenaufruf vorbei: Frame wird wieder vom Call-Stack genommen ▷ StackPointer wird auf Adresse des vorherigen Frames gesetzt
Methodentabelle	▷ Enthält bei Objekt die Anfangsadressen der verfügbaren Methoden



2 Datenstrukturen

Array	▷ Verwendet zum Speichern von mehreren Variablen des selben Typs ▷ Erzeugung: <code>int[] test = new int[n];</code> ▷ <code>n</code> gibt in diesem Fall die feste Anzahl der speicherbaren Variablen an ▷ Natürlich auch Arrays von Objekten möglich ▷ Zugriff auf Variablen: <code>test[0]</code> für ersten Wert (Index) ▷ Zugriff auf Länge: <code>test.length</code>
-------	--

3 Datentypen

Konstanten	<ul style="list-style-type: none"> ▷ Variable/Referenz wird dadurch unveränderbar ▷ z.B.: <code>final myClass ABC = new myClass();</code> <ul style="list-style-type: none"> ◊ Referenz zwar nicht veränderbar, Objekt aber schon ▷ <code>Integer.MAX_VALUE</code> / <code>Integer.MIN_VALUE</code> ▷ Unendlich: <code>Double.POSITIVE_INFINITY</code> / <code>Double.NEGATIVE_INFINITY</code>
Primitive Datentypen	<ul style="list-style-type: none"> ▷ Ganze Zahlen: <code>byte</code> → <code>short</code> → <code>int</code> → <code>long</code> ▷ Gebrochene Zahlen: <code>float</code> → <code>double</code> ▷ Logik: <code>boolean</code> ▷ Zeichen: <code>char</code>
Literale	<ul style="list-style-type: none"> ▷ wörtlich hingeschriebene Werte eines Datentyps ▷ Zahlen standardmäßig <code>int</code>, falls <code>long</code> gewünscht: <code>123L</code> oder <code>123l</code> ▷ Bei gebrochenen <code>double</code>, falls <code>float</code> gewünscht: <code>12.3F</code> oder <code>12.3f</code> ▷ <code>null</code>: Nutzung für Referenzen → verweist auf nichts
Boolean	<ul style="list-style-type: none"> ▷ nur <code>true</code> und <code>false</code> ▷ Negation <code>!a</code> ▷ Logisches Und: <code>a && b</code> ▷ Logisches Oder: <code>a b</code> (inklusive) ▷ Gleichheit: <code>a == b</code>
Zeichentyp <code>char</code>	<ul style="list-style-type: none"> ▷ z.B.: <code>char c = 'a';</code> ▷ Interne Kodierung als Unicode ▷ <code>\t</code> Horizontaler Tab ▷ <code>\b</code> Backspace ▷ <code>\n</code> Neue Zeile ▷ Auch Darstellung im Hexacode (<code>\u0041</code>)
Enumeration	<ul style="list-style-type: none"> ▷ Zusammenfassung mehrerer Konstanten (feste Anzahl) ▷ Erzeugung meist in eigener <code>.java</code> Datei ▷ <code>enum MyDirection {DOWN, RIGHT}</code> ▷ Keine Objekterzeugung von Enumeration möglich ▷ Abspeichern in Variable des Enum-Typs ist jedoch möglich ▷ <code>MyDirection dir = MyDirection.DOWN;</code>
Referenztypen	<ul style="list-style-type: none"> ▷ Alle Typen, die keine primitiven Datentypen sind ▷ Unterscheidung zwischen Referenz und eigentlichem Objekt ▷ Gleichheitsoperator <code>==</code> vergleicht nur die Referenz (Objektidentität) <ul style="list-style-type: none"> ◊ Verweis auf dasselbe Objekt ▷ Wertgleichheit bezieht sich auf das Objekt an sich ▷ Deep Copy ⇒ An allen parallelen Stellen Wertgleichheit ▷ Shallow Copy ⇒ Nur Kopie der Adressen

4 Fehler

Kompilierzeitfehler (compile-time errors)	<ul style="list-style-type: none"> ▷ Falsche Klammersetzung, falsche Schlüsselwörter,... ▷ Programm wird nicht übersetzt ⇒ Fehlermeldung vom Compiler
Laufzeitfehler (run-time errors)	<ul style="list-style-type: none"> ▷ Tritt während der Ausführung auf ▷ Führt zum Abbruch des Programms ⇒ Ausgabe der Fehlermeldung ▷ <code>IndexOutOfBoundsException</code>, <code>NullPointerException</code>,...

5 Graphics (java.awt.Graphics;)

Applet	<ul style="list-style-type: none">▷ leichtgewichtige Variante an Graphikprogrammen▷ <code>import java.awt.Applet;</code>▷ 1. Erstellen eigener Applet-Klasse (<code>extends Applet</code>)▷ 2. Überschreiben der Methode <code>paint</code><pre>public void paint (Graphics graphics) {...}</pre>Klasse <code>Graphics</code> verknüpft Programm mit Zeichenfläche▷ 2.1 <code>GeomShape2D</code>-Array<pre>GeomShape2D pic = new GeomShape2D[3];</pre>Füllen des erstellten Arrays mit Formen (z.B.: <code>new Circle(0,0,0);</code>)▷ 2.2 Erstellen jeder Form mithilfe Randfarbe, Füllfarbe und Zeichnen<pre>pic[0].setBoundaryColor(Color.RED); // Randfarbe pic[0].setFillColor(Color.RED); // Füllfarbe pic[0].paint(graphics); // Eigentliches Zeichnen</pre>
GeomShape2D	<ul style="list-style-type: none">▷ Abstrakte Klasse (Methode <code>paint</code> ist abstrakt)▷ Attribute:<pre>int positionX; int positionY; int rotationAngle; int transparencyValue; Color boundaryColor; Color fillColor;</pre>▷ Subklassen: <code>Rectangle</code>, <code>Circle</code>, <code>StraightLine</code>

6 Interfaces

Erzeugung	<ul style="list-style-type: none">▷ Meist in eigener Datei▷ <code>public interface MyInterface {...}</code>▷ Alle Methodes und das Interface müssen <code>public</code> sein
Methoden	<ul style="list-style-type: none">▷ Werden hier nicht implementiert, sondern nur definiert▷ <code>public</code> kann weggelassen werden, da ohnehin notwendig▷ Implementierte Methoden müssen dann auch <code>public</code> sein▷ Falls eine der Methoden nicht implementiert wird ⇒ Klasse abstrakt
Verwendung	<ul style="list-style-type: none">▷ <code>implements MyInterface</code> nach Klassenname▷ Beliebig viele Interfaces möglich (seperiert durch ,)▷ Ein Interface kann mehrere andere Interfaces erweitern (<code>extends</code>

7 Klassen

Erzeugung	<ul style="list-style-type: none"> ▷ meist in separater .java Datei ▷ <code>public class MyClass {}</code> ▷ <code>new MyClass();</code> <ul style="list-style-type: none"> ◊ Reserviert ausreichend Speicherplatz für das Objekt ▷ <code>MyClass x = new MyClass();</code> <ul style="list-style-type: none"> ◊ Speichern der Adresse des neuen Objekts in der Referenz x
Attribute	<ul style="list-style-type: none"> ▷ Eigenschaften der Objekte/Klassen ▷ z.B.: <code>private int x;</code> (Objektattribut) ▷ z.B.: <code>private static int x;</code> (Klassenattribut)
Konstruktor	<ul style="list-style-type: none"> ▷ Wird zur Erzeugung von neuen Objekten einer Klasse verwendet ▷ Methode mit selben Namen wie Klasse und ohne Rückgabotyp ▷ z.B.: <code>public MyClass (int x, int y) {this.x = x; this.y = y;}</code> ▷ Erzeugung eines neuen Objekts: <code>MyClass test = new MyClass(2,4);</code> ▷ Falls kein Konstruktor angegeben wird → Standardkonstruktor
Abstraktion	<ul style="list-style-type: none"> ▷ <code>abstract public class MyClass {...}</code> ▷ Notwendig, sobald Klasse eine abstrakte Methode beinhaltet ▷ Keine Objekterzeugung möglich ▷ Meist als Klasse mit Rahmenbedingungen für Subklassen verwendet
Klasse aller Klassen	<ul style="list-style-type: none"> ▷ <code>java.lang.Object</code> ▷ Jede Klasse ist direkt oder indirekt von <code>Object</code> abgeleitet ▷ Methoden: <ul style="list-style-type: none"> ◊ <code>boolean equals (Object obj) {...}</code> // Test auf Wertgleichheit ◊ <code>String toString() {...}</code> // Zustand des Objekts als String ◊ Werden oft an jeweilige Klasse angepasst
Verborgene Informationen▷	

8 Konversionen

Implizit	<ul style="list-style-type: none"> ▷ Immer möglich, wenn kein Informationsverlust entstehen kann ▷ z.B.: kleinerer Datentyp in gröSSeren
Explizit	<ul style="list-style-type: none"> ▷ Meist Informationsverlust ▷ Durchführung durch Angabe des Datentyps in Klammern davor ▷ z.B.: <code>int i = (int)testDouble;</code>

9 Methoden

Methodenkopf	<ul style="list-style-type: none"> ▷ Modifier Rückgabewert Methodenname (Parameter) {Anweisung} ▷ z.B.: <code>public void setX (int x) {this.x = x;}</code> (Objektmethode) ▷ z.B.: <code>public static void setY (int y) {this.y = y;}</code> (Klassenmethode) ▷ <code>this.x</code> steht hier für das Objektattribut und nicht den Parameter
Ausführung	<ul style="list-style-type: none"> ▷ Objektmethoden: <code>myObject.setX(2);</code> ▷ Klassenmethoden: <code>MyClass.setY(2);</code>
return	<ul style="list-style-type: none"> ▷ Wird für Rückgabe bei Methoden mit Rückgabewert benötigt
Abstraktion	<ul style="list-style-type: none"> ▷ <code>abstract</code> vor Modifier (z.B.: <code>public</code>) ▷ Abstrakte Methoden haben keinen Methodenrumpf
Parameter	<ul style="list-style-type: none"> ▷ Parameterliste in Definition: Formale Parameter ▷ Parameterliste bei Methodenaufruf: Aktuelle Parameter <ul style="list-style-type: none"> ◊ Kommt von actual ⇒ tatsächlich, vorliegend ▷ Verhalten bei Referenzen: <ul style="list-style-type: none"> ◊ Kopie der Adresse des Objekts bei Initialisierung des formalen durch aktuellen Parameter

10 Packages und Zugriffsrechte

Package	<ul style="list-style-type: none">▷ Zusammenfassung von mehreren Dateien▷ Wird zur Gruppierung von ähnlichen Funktionalitäten verwendet▷ Ermöglicht selbe Dateinamen in unterschiedlichen Packages▷ Bestehen nur aus Kleinbuchstaben▷ Am Anfang der Quelldatei: <code>package mypackage;</code><ul style="list-style-type: none">◊ Datei gehört damit zum Package <code>mypackage</code>◊ <code>mypackage</code> wird automatisch importiert
Import	<ul style="list-style-type: none">▷ <code>import package.*;</code>▷ <code>*</code> steht für alle Definitionen aus <code>package</code>▷ <code>*</code> importiert aber nicht die Inhalte von Subpackages▷ Import-Anweisungen müssen immer am Anfang des Quelltextes stehen▷ Durch Importanweisungen sind Teile danach nur noch mit Namen ansprechbar▷ Wichtigstes Package: <code>java.lang.*</code> (automatisch importiert)▷ Konstanten: <code>import static java.lang.Math.PI;</code><ul style="list-style-type: none">◊ Ermöglicht Schreiben von <code>PI</code> statt <code>Math.PI</code>
Zugriffsrechte	<ul style="list-style-type: none">▷ Klassen/Enum: nur <code>public</code> oder nichts<ul style="list-style-type: none">◊ Nur eine Klasse darf <code>public</code> sein (Damit auch Dateiname)▷ <code>private</code>: Zugriff innerhalb der Klasse▷ Keine Angabe: <code>private</code> + im Package▷ <code>protected</code>: Keine Angabe + in allen Subklassen▷ <code>public</code>: <code>protected</code> + an jeder Import-Stelle

11 Programme und Prozesse

Quelltest	▷ z.B. selbst geschriebener Java-Code
Java-Bytecode	▷ Wird durch Übersetzung des Java-Quelltextes erzeugt
Programm	▷ Sequenz von Informationen
Aufruf eines Programms	▷ Starten eines Prozesses, der die Anweisungen des Programmes abarbeitet
Prozesse	<ul style="list-style-type: none">▷ CPU besteht aus mehreren Prozessorkernen▷ Mehrere Prozesse laufen dementsprechend parallel▷ Allerdings bearbeitet jeder Kern nur einen Prozess gleichzeitig (sehr kurz)<ul style="list-style-type: none">◊ Illusion von Multitasking

12 Schleifen und if

while-Schleife	<ul style="list-style-type: none">▷ <code>while (Bedingung) {Anweisung;}</code>▷ Schleife wird ausgeführt, solange die Bedingung wahr ist▷ <code>{}</code> kann bei einzelner Anweisung auch weggelassen werden
do-while-Schleife	<ul style="list-style-type: none">▷ <code>do {Anweisung;} while (Bedingung);</code>▷ Anweisungsblock wird immer mindestens einmal ausgeführt
for-Schleife	<ul style="list-style-type: none">▷ <code>for (Anweisung davor; Bedingung; Anweisung danach) {Anweisung}</code>▷ z.B.: <code>for (int i = 0; i < 10; i++) {...}</code><ul style="list-style-type: none">◊ Zehnmalige Ausführung der Anweisung▷ Kurzform: <code>for (Position p : positions) {}</code><ul style="list-style-type: none">◊ (Komponententyp Identifier : ArrayName)
if-Anweisung	<ul style="list-style-type: none">▷ <code>if (Bedingung) {...}</code><ul style="list-style-type: none">◊ Führt den Code in der Anweisung nur aus, falls die Bedingung erfüllt ist▷ <code>if (Bedingung) {} else {}</code><ul style="list-style-type: none">◊ Code, der ausgeführt wird, falls Bedingung nicht erfüllt ist

13 String (java.lang.String)

Eigenschaften	<ul style="list-style-type: none">▷ Sonderrolle, da Klasse, aber trotzdem Literale in Java▷ Zeichenketten, die aus allen möglichen chars bestehen
Methoden:	<ul style="list-style-type: none">▷ <code>String str = "Hello World";</code><ul style="list-style-type: none">◊ <code>str.length;</code> // 11◊ <code>str.charAt(2);</code> // e◊ <code>str.indexOf('e');</code> // 2◊ <code>str.matches("He.+rld");</code> // true<ul style="list-style-type: none">.+ ⇒ . als Platzhalter für beliebiges Zeichen, + erlaubt Wiederholung⇒ Regular Expression◊ <code>String str 2 = str.concat("b");</code> // Anhängen◊ <code>String str 2 = str1 + "b";</code> // Kurzform

14 Syntax

Keywords	<ul style="list-style-type: none">▷ Können nur an bestimmten Stellen im Code stehen▷ z.B. <code>class</code>, <code>import</code>, <code>public</code>, <code>while</code>,...
Identifizier	<ul style="list-style-type: none">▷ Namen für Klassen, Variablen, Methoden,..▷ Erstes Zeichen darf keine Ziffer sein▷ Keine Keywords als Identifizier ▷ Identifizier sind case-sensitive
Konventionen	<ul style="list-style-type: none">▷ Variablen / Methoden beginnen mit Kleinbuchstaben (<code>testInt</code>)▷ Klassen beginnen mit Großbuchstaben (<code>testClass</code>)▷ Wortanfänge im Inneren mit Großbuchstaben▷ Konstanten bestehen aus <code>_</code> und Großbuchstaben (<code>CENTS_PER_EURO</code>)▷ Packagenamen nur aus Kleinbuchstaben und <code>_</code> bei unzulässigen Zeichen
Kommentare	<ul style="list-style-type: none">▷ <code>//</code> Einzelne Zeile▷ <code>/*...*/</code> Mehrere Zeilen▷ <code>/**...*/</code> Erzeugung von Javadoc

15 Vererbung

Zweck	▷ Weitergabe von allen Methoden und Attributen
Verwendung	▷ <code>public class MySubClass extends MyClass {}</code>
Konstruktor	▷ Aufruf des Konstruktors der Superklasse mithilfe von <code>super(Parameter);</code> ▷ Dieser Aufruf erfolgt im Konstruktor der Subklasse ▷ z.B.: <code>public MySubClass (int x) { super(x);<v></code>
Overwrite	▷ Methoden in Subklassen können auch neu geschrieben werden ◊ Die Implementation der Superklasse wird sozusagen überschrieben ▷ Selber Name und Parameterliste notwendig
Overload	▷ Methoden mit selbem Bezeichner, aber unterschiedlicher Parameterliste ▷ Die Methode wird überladen ▷ Konstruktoren kann man auch überladen ◊ Für manche Werte werden dann Standardwerte gesetzt ◊ Anderer Konstruktor auch in Konstruktor aufrufbar (<code>this(1);</code>)
Subtypen	▷ Abgeleitete Klassen / Interfaces (extends) ▷ Überall wo ein Referenztyp (Supertyp) erwartet wird: ◊ Verwendung eines Objekts eines Subtyps möglich in Zuweisung an Variable als Parameterwert als Rückgabewert
Statischer Typ	▷ Der Typ, mit dem Referenz definiert wird ▷ Statischer Typ unveränderlich mit Referenz verknüpft ⇒ statisch ▷ z.B.: <code>X a = new Y();</code> ⇒ X hier statischer Typ ▷ Entscheidet , auf welche Attribute/Methoden zugegriffen werden darf ◊ Müssen im statischen Typ vorhanden sein (definiert oder ererbt)
Dynamischer Typ	▷ Der Typ des Objekts einer Referenz, auf das diese Referenz ▷ Muss gleich dem statischen Typ oder ein Subtyp des statischen Typs sein ▷ Kann sich beliebig häufig ändern ⇒ dynamisch ▷ z.B.: <code>X a = new Y();</code> ⇒ Y hier dynamischer Typ ▷ Entscheidet , welche Implementation der Methode aufgerufen wird
Downcast	▷ <code>if (y instanceof X) {...}</code> ◊ Gibt <code>true</code> zurück, falls y (Variable von Referenztyp) gleich dem Typen von X oder ein Subtyp von X ist ▷ Downcast ◊ Vorherige Überprüfung mit <code>isinstanceof</code> ◊ Ermöglicht z.B.: <code>X z;</code> <code>z = (X) y;</code> ◊ Warum? Zugriff auf Funktionen, die nicht im statischen Typ existieren
Garbage Collector	▷ Teil des Laufzeitsystems ▷ Wird selbstständig aufgerufen, um Objekte ohne Referenz zu löschen ▷ Kann zwecks Laufzeitoptimierung konfiguriert werden