FOP Reference Sheet

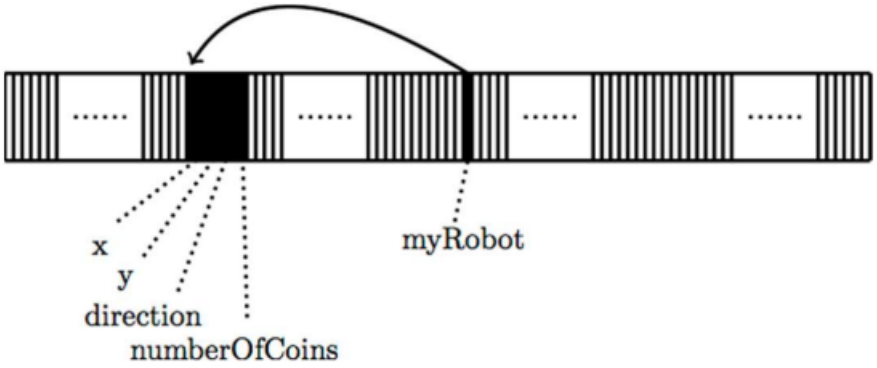
Jonas Milkovits

Last Edited: 2. April 2020

Inhaltsverzeichnis

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 abgespeichet 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: `int[] test = new int[n];` ▷ `n` gibt in diesem Fall die feste Anzahl der speicherbaren Variablen an ▷ Natürlich auch Arrays von Objekten möglich ▷ Zugriff auf Variablen: `test[0]` für ersten Wert (Index) ▷ Zugriff auf Länge: `test.length`

3 Datentypen

Konstanten	▷ Variable/Referenz wird dadurch unveränderbar ▷ z.B.: `final myClass ABC = new myClass();` ⇒ Referenz zwar nicht veränderbar, Objekt aber schon ▷ `Integer.MAX_VALUE / Integer.MIN_VALUE` ▷ Unendlich: `Double.POSITIVE_INFINITY / Double.NEGATIVE_INFINITY`
Primitive Dateitypen	▷ Ganze Zahlen: byte → short → int → long ▷ Gebrochene Zahlen: float → double ▷ Logik: boolean ▷ Zeichen: char
Literale	▷ wörtlich hingeschriebene Werte eines Datentyps ▷ Zahlen standardmäßig int, falls `long` gewünscht: `123L oder 123l` ▷ Bei gebrochenen double, falls `float` gewünscht: `12.3F oder 12.3f` ▷ null: Nutzung für Referenzen → verweist auf nichts
Boolean	▷ nur `true` und `false` ▷ Negation `!a` ▷ Logisches Und: `a && b` ▷ Logisches Oder: `a || b` (inklusiv) ▷ Gleichheit: `a == b`
Zeichentyp char	▷ z.B.: `char c = ´a´;` ▷ Interne Kodierung als Unicode ▷ ´\t´ Horizontaler Tab ▷ ´\b´ Backspace ▷ ´\n´ Neue Zeile ▷ Auch Darstellung im Hexacode (´\u039A´)
Enumeration	▷ Zusammenfassung mehrerer Konstanten (feste Anzahl) ▷ Erzeugung meist in eigener .java Datei ▷ `enum MyDirection {DOWN, RIGHT}` ▷ Keine Objekterzeugung von Enumeration möglich ▷ Abspeichern in Variable des Enum-Types ist jedoch möglich ▷ `MyDirection dir = MyDirection.DOWN;`
Referenztypen	▷ Alle Typen, die keine primitiven Datentypen sind ▷ Unterscheidung zwischen Referez und eigentlichem Objekt ▷ Gleichheitsoperator `==` vergleicht nur die Referenz (Objektidentität) ⇒ Verweis auf dasselbe Objekt ▷ Wertgleichheit bezieht sich auf das Objekt an sich ▷ Deep Copy ⇒ An allen parallelen Stellen Wertgleichheit ▷ Shallow Copy ⇒ Nur Kopie

4 Fehler

Kompilierzeitfehler (compile-time errors)	▷ Falsche Klammersetzung, falsche Schlüsselwörter,.. ▷ Programm wird nicht übersetzt ⇒ Fehlermeldung vom Compiler
Laufzeitfehler (run-time errors)	▷ Tritt während der Ausführung auf ▷ Führt zum Abbruch des Programms ⇒ Ausgabe der Fehlermeldung ▷ IndexOutOfBounds, NullPointerException,..
	▷
	▷
	▷
	▷
	▷
	▷

5 Graphics (java.awt.Graphics;)

Applet	▷ leichtgewichtige Variante an Graphikprogrammen
	▷ `import java.awt.Applet;`
	▷ 1. Erstellen eigener Applet-Klasse (`extends Applet`)
	▷ 2. Überschreiben der Methode `paint`
	`public void paint (Graphics graphics) {...}`
	Klasse `Graphics` verknüpft Programm mit Zeichenfläche
	▷ 2.1 `GeomShape2D`-Array
	`GeomShape2D pic = new GeomShape2D[3];`
	Füllen des erstellten Arrays mit Formen (z.B.: `new Circle(0,0,0);`)
	▷ 2.2 Erstellen jeder Form mithilfe Randfarbe, Füllfarbe und Zeichnen
	`pic[0].setBoundaryColor(Color.RED); // Randfarbe`
	`pic[0].setFillColor(Color.RED); // Füllfarbe`
	`pic[0].paint(graphics); // Eigentliches Zeichnen`

GeomShape2D	▷ Abstrake Klasse (Methode `paint` ist abstrakt)
	▷ Attribute:
	`int positionX; int positionY; int rotationAngle;`
	`int transparencyValue; Color boundaryColor; Color fillColor;`
	▷ Subklassen: `Rectangle, Circle, StraightLine`

6 Interfaces

Erzeugung	▷ Meist in eigener Datei ▷ `public interface MyInterface {...}` ▷ Alle Methodes und das Interface **müssen** `public` sein
Methoden	▷ Werden hier nicht implementiert, sondern nur definiert ▷ `public` kann weggelassen werden, da ohnehin notwending ▷ Implementierte Methoden müssen dann auch `public` sein ▷ Falls eine der Methoden nicht implementiert wird ⇒ Klasse abstrakt
Verwendung	▷ `implements MyInterface` nach Klassenname

7 Klassen

Erzeugung	▷ meist in seperater .java Datei ▷ `public class MyClass {}` ▷ `new MyClass();`: Reserviert ausreichend Speicherplatz für das Objekt ▷ `MyClass x = new MyClass();`: Speichern der Adresse des neuen Objekts in der Referenz x
Attribute	▷ Eigenschaften der Objekte/Klassen ▷ z.B.: `private int x;` (Objekteigenschaft) ▷ z.B.: `private static int x;` (Klasseneigenschaft)
Konstruktor	▷ Wird zur Erzeugung von neuen Objekten einer Klasse verwendet ▷ Methode mit selben Namen wie Klasse und ohne Rückgabetyp ▷ z.B.: `public MyClass (int x, int y) {this.x = x; this.y = y;}` ▷ Erzeugung eines neuen Objekts: `MyClass test = new MyClass(2,4);` ▷ Falls kein Konstruktur angegeben wird → Standardkonstruktor
Abstraktion	▷ `abstract public class MyClass {...}` ▷ Notwendig, sobald Klasse eine abstrakte Methode beinhaltet ▷ Keine Objekterzeugung möglich ▷ Meist als Klasse mit Rahmenbedingungen für Subklassen verwendet

8 Konversionen

Implizit	▷ Immer möglich, wenn kein Informationsverlust entstehen kann ▷ z.B.: kleinerer Datentyp in größeren
Explizit	▷ Meist Informationsverlust ▷ Durchführung durch Angabe des Datentyps in Klammern davor ▷ z.B.: `int i = (int)testDouble;`

9 Methoden

Methodenkopf	▷ Modifier Rückgabewert Methodenname (Parameter) {Anweisung} ▷ z.B.: `public void setX (int x) {this.x = x;}` (Objektmethode) ▷ z.B.: `public static void setY (int y) {this.y = y;}` (Klassenmethode) ▷ `this.x` steht hier für das Objektattribut und nicht den Parameter
Ausführung	▷ Objektmethoden: `myObject.setX(2);` ▷ Klassenmethoden: `MyClass.setY(2);`
return	▷ Wird für Rückgabe bei Methoden mit Rückgabewert benötigt
Abstraktion	▷ `abstract` vor Modifier (`z.B.: public`) ▷ Abstrakte Methoden haben keinen Methodenrumpf

10 Packages und Zugriffsrechte

Package	▷ 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: `package mypackage;` ⇒ Datei gehört damit zum Package `mypackage` ⇒ `mypackage` wird automatisch importiert
Import	▷ `import package.*;` ▷ `*` steht für alle Definitionen aus `package` ▷ `*` 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: `java.lang.*` (automatisch importiert)
Zugriffsrechte	▷ Klassen/Enum: nur `public` oder nichts ⇒ Nur eine Klasse darf `public` sein (Damit auch Dateiname) ▷ `private:` Zugriff innerhalb der Klasse ▷ `Keine Angabe:` `private` + im Package ▷ `protected:` `Keine Angabe` + in allen Subklassen ▷ `public:` `protected` + 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	▷ CPU besteht aus mehreren Prozessorkernen ▷ Mehrere Prozesse laufen dementsprechend parallel ▷ Allerdings bearbeitet jeder Kern nur einen Prozess gleichzeitig (sehr kurz) ⇒ Illusion von Multitasking
	▷

12 Schleifen und if

while-Schleife	▷ `while (Bedingung) {Anweisung;}` ▷ Schleife wird ausgeführt, solange die Bedingung wahr ist ▷ {} kann bei einzelner Anweisung auch weggelassen werden
do-while-Schleife	▷ `do {Anweisung;} while (Bedingung);` ▷ Anweisungsblock wird immer mindestens einmal ausgeführt
for-Schleife	▷ `for (Anweisung davor; Bedingung; Anweisung danach) {Anweisung}` ▷ z.B.: `for (int i = 0; i < 10; i++) {...}` ⇒ Zehnmalige Ausführung der Anweisung ▷ Kurzform: `for (Position p : positions) {}`
if-Anweisung	▷ `if (Bedingung) {...}` ▷ Führt den Code in der Anweisung nur aus, falls die Bedingung erfüllt ist ▷ `if (Bedingung) {} else {}` ▷ Code, der ausgeführt wird, falls Bedingung nicht erfüllt ist

13 String (java.lang.String)

Eigenschaften	▷ Sonderrolle, da Klasse, aber trotzdem Literale in Java ▷ Zeichenketten, die aus allen möglichen chars bestehen

Methoden	▷ `String str = "Hello World";`
	`str.length; // 11`
	`str.charAt(2); // e`
	`str.indexOf('e'); // 2`
	`str.matches("He.+rld"); // true`
	`.+` ⇒ `.` als Platzhalter für beliebiges Zeichen, `+` erlaubt Wiederholung
	⇒ Regular Expression
	`String str 2 = str.concat("b"); // Anhängen`
	`String str 2 = str1 + "b"; // Kurzform`

	▷
	▷
	▷
	▷
	▷
	▷

14 Syntax

Keywords	▷ Können nur an bestimmten Stellen im Code stehen ▷ z.B. `class, import, public, while,..`
Identifier	▷ Namen für Klassen, Variablen, Methoden,.. ▷ Erstes Zeichen darf keine Ziffer sein ▷ Keine Keywords als Identifier ▷ Identifier sind case-sensitive
Konventionen	▷ Variablen / Methoden beginnen mit Kleinbuchstaben (`testInt`) ▷ Klassen beginnen mit Großbuchstaben (`testClass`) ▷ Wortanfänge im Inneren mit Großbuchstaben ▷ Konstanten bestehen aus `_` und Großbuchstaben (`CENTS_PER_EURO`) ▷ Packagenamen nur aus Kleinbuchstaben und `_` bei unzulässigen Zeichen
Kommentare	▷ `//` Einzelne Zeile ▷ `/*...*/` Mehrere Zeilen ▷ `/**...*/` Erzeugung von Javadoc

15 Vererbung

Zweck	▷ Weitergabe von allen Methoden und Attributen
Verwendung	▷ `public class MySubClass extends MyClass {}`
Konstruktor	▷ Aufruf des Konstruktors der Superklasse mithilfe von `super(Parameter);` ▷ Dieser Aufruf erfolgt im Konstruktor der Subklasse ▷ z.B.: `public MySubClass (int x) { super(x);<v}`
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 (`this(1);`)