



AvHG Informatik Sek II

Vera Lüthje, Hartmut Meyer
h.meyer6@schule.bremen.de

Version vom 10. Januar 2018

Inhaltsverzeichnis

| | |
|---|-----------|
| 1 Eclipse als Java-Entwicklungsumgebung | 1 |
| 1.1 Vorbereitung | 1 |
| Installation des Java-SDKs | 1 |
| Anlegen eines Git-Repositories für deine Java-Dateien | 1 |
| 1.2 Eclipse herunterladen und installieren | 5 |
| 1.3 Eclipse konfigurieren | 5 |
| Zeichensatz festlegen | 5 |
| Änderungen durch andere Programme im Workspace überwachen | 5 |
| Git-Funktionalität in Eclipse konfigurieren und dein eigenes Repository einbinden | 7 |
| Git: Commit and Push | 9 |
| Git: Pull | 10 |
| Das Kurs-Repository einbinden | 11 |
| Zeilenummern anzeigen lassen | 11 |
| Automatische Updates | 11 |
| Package „hilfe“ importieren | 12 |
| Erzeugung eines Templates (Vorlage) für HJFrame | 12 |
| Wahl eines externen PDF-Betrachters | 12 |
| Installation des WindowsBuilder Plugins | 12 |
| Installation des MySQL-Java-Connectors | 13 |
| Plugin „SQL-Explorer“ installieren und konfigurieren | 13 |
| 1.4 Eclipse kennen lernen | 14 |
| 2 Hardware und Software | 17 |
| 2.1 Hardware | 17 |
| Grundstruktur eines Computers | 17 |
| Speicherarten | 18 |
| 2.2 Software | 19 |
| BIOS | 19 |
| Betriebssystem | 19 |
| Treiber | 20 |
| Anwendungssoftware | 20 |
| 2.3 Hardware und Software – Übungen | 21 |
| Aufgabe 1: Speicherarten | 21 |
| Aufgabe 2: Software | 21 |
| Aufgabe 3: Stärken und Schwächen von MS Windows | 21 |
| 3 Informatik – Worum geht's? | 23 |
| 3.1 Informatik | 23 |
| Gegenstand der Informatik | 23 |
| Teilgebiete der Informatik | 23 |
| 3.2 Was Computer können | 24 |
| 3.3 Computational Thinking | 24 |
| 3.4 Erste Übung – erster Teil | 25 |
| 3.5 Variablen | 26 |
| 3.6 Kontrollstrukturen | 26 |
| Bedingte Anweisung und Verzweigung | 26 |
| Wiederholungsanweisungen (Schleifen) | 27 |
| 3.7 Erste Übung – zweiter Teil | 28 |
| 3.8 Eine mögliche Lösung: Brute Force | 29 |
| 3.9 Eine bessere mögliche Lösung: Bisektion | 30 |
| 3.10 Implementierung in echten Programmiersprachen | 33 |
| Implementierung in Python | 34 |
| Implementierung in Java | 35 |
| 3.11 Wer es genauer wissen will | 36 |

| | |
|---|-----------|
| 4 Java Grundlagen | 37 |
| 4.1 Wichtige Regeln | 37 |
| 4.2 Variablen | 37 |
| Deklaration und Initialisierung | 37 |
| Lebensdauer | 37 |
| Konstanten | 38 |
| 4.3 Kommentare | 38 |
| 4.4 Namenskonventionen | 38 |
| 4.5 Datentypen | 38 |
| 4.6 Typkonvertierungen | 39 |
| 4.7 Literale | 39 |
| 4.8 Operatoren | 40 |
| Arithmetische Operatoren | 40 |
| Vergleichs-Operatoren | 40 |
| Logische Operatoren | 41 |
| Zuweisungsoperatoren | 41 |
| 4.9 Kontrollstrukturen | 41 |
| Einfache Verzweigung | 41 |
| Mehrfach-Verzweigung | 41 |
| Schleifen | 42 |
| 4.10 Methoden | 42 |
| 5 Erste Übungen mit Java: Turtle-Grafik | 45 |
| 5.1 Turtle-Grafik | 45 |
| Kommandos der Turtle | 45 |
| 5.2 Programmieraufgaben zur Turtle-Grafik | 45 |
| Aufgabe 1: Gleichseitiges Dreieck | 45 |
| Aufgabe 2: Quadrat | 46 |
| Aufgabe 3: Sechseck | 46 |
| Aufgabe 4: Achteck | 46 |
| Aufgabe 5: Treppe | 46 |
| Aufgabe 6: Spirale | 46 |
| Aufgabe 7: Verschachtelte Quadrate | 47 |
| Aufgabe 8: Achtzehn Sechsecke | 47 |
| Aufgabe 9: Sechseck aus Dreiecken | 47 |
| Aufgabe 10: Figur aus vier Dreiecken | 47 |
| Aufgabe 11: Stern aus Dreiecken | 48 |
| 6 Rekursion | 49 |
| 6.1 Rekursions-Begriff | 49 |
| Blick in den Spiegel | 49 |
| Rekursiv Programmieren | 50 |
| 6.2 Rekursion: Programmieraufgaben | 51 |
| Aufgabe 1: Berechnung der Fakultät | 51 |
| Aufgabe 2: Zweier-Potenzen | 51 |
| Aufgabe 3: Papierformate | 51 |
| 6.3 Rekursion mit Turtle-Grafik | 52 |
| Aufgabe 1: Binärer Baum | 52 |
| Aufgabe 2: Farnwedel | 53 |
| 6.4 Rekursion mit Turtle-Grafik (Fortsetzung) | 54 |
| Aufgabe 1: Kochsche Kurve I | 54 |
| Aufgabe 2: Schneeflocke | 54 |
| Aufgabe 3: Variation zur Kochschen Kurve | 55 |
| Aufgabe 4: Kochsche Kurve II | 55 |
| Aufgabe 5: Kochsche Kurve III | 55 |
| Aufgabe 6: Sierpinski-Dreieck | 55 |
| Aufgabe 7: Baum des Pythagoras | 56 |

| | |
|--|-----------|
| 7 Java Dialoge | 57 |
| 7.1 Dialoge verwenden | 57 |
| Nachricht ausgeben | 57 |
| Eingabe-Dialog | 57 |
| 7.2 Nebenläufigkeit von Events in Swing | 58 |
| 7.3 Umwandlung String ↔ Zahl | 59 |
| Umwandlung von Strings in Zahlen | 59 |
| Umwandlung von Zahlen in Strings | 59 |
| 7.4 Dialoge – Übungen | 60 |
| Aufgabe 1: Zahl zwischen 1 und 100 | 60 |
| Aufgabe 2: Zahl kleiner als 10 oder größer als 20 | 60 |
| Aufgabe 3: Gerade Zahl | 60 |
| Aufgabe 4: Umwandlung von Notenpunkten in Noten | 60 |
| Aufgabe 5: Mathe-Trainer | 60 |
| Aufgabe 6: Lösung quadratischer Gleichungen | 61 |
| Aufgabe 7: Body Mass Index (BMI) | 61 |
| Aufgabe 8: Niedersachsenticket | 61 |
| 8 Eigene Java Programme | 63 |
| 8.1 Eigene Java-Programme basierend auf HJFrame | 63 |
| Beispielprogramm | 63 |
| 8.2 Malen mit der Klasse Graphics | 64 |
| Methoden von Graphics (Auswahl) | 64 |
| 8.3 Eigene Programme mit HJFrame – Übungen | 65 |
| Aufgabe 1: Abstrakte Kunst | 65 |
| Aufgabe 2: Gespenster | 65 |
| Aufgabe 3: Aufrufe von myPaint() zählen | 65 |
| Aufgabe 4: Quadrate zählen die Aufrufe von myPaint() | 65 |
| Aufgabe 5: Gespenster-Dialog | 66 |
| Aufgabe 6: Ineinander geschachtelte Kreise | 66 |
| Aufgabe 7: Pyramide | 67 |
| 9 Farben und Zufallszahlen | 69 |
| 9.1 Farben | 69 |
| Standardfarben | 69 |
| Farben Mischen | 69 |
| 9.2 Zufallszahlen | 69 |
| 9.3 Farben und Zufallszahlen – Übungen | 71 |
| Aufgabe 1: Wald | 71 |
| Aufgabe 2: Wald mit Zufallspositionen | 72 |
| Aufgabe 3: Diagonale | 72 |
| Aufgabe 4: Farbsäule | 72 |
| 10 Klassen und Objekte | 73 |
| 10.1 Darstellung einer Klasse mit der Modellierungssprache UML | 73 |
| 10.2 Beispiel: Klasse Hund | 73 |
| Eine erste Version der Klasse Hund | 73 |
| Klasse Hund mit selbstdefinierten Konstruktoren | 75 |
| 10.3 Objekte erzeugen | 77 |
| 10.4 Variablen einer Klasse | 77 |
| Zugriff auf Variablen | 78 |
| 10.5 Methoden | 78 |
| 10.6 Konstruktor | 78 |
| 10.7 Animationen erzeugen | 79 |
| Beispiel für eine Animation | 79 |
| 10.8 Klassen aus dem Package hilfe | 80 |
| Zeichnen von Dreiecken | 80 |

| | |
|---|------------|
| 10.9 Klassen und Objekte – Übungen | 81 |
| Aufgabe 1: Hunde | 81 |
| Aufgabe 2: Raupe | 81 |
| Aufgabe 3: Strichmännchen | 82 |
| Aufgabe 4: Blumen | 82 |
| Aufgabe 5: Verhältnis von Klassen und Objekten | 83 |
| Aufgabe 6: Fachbegriffe | 83 |
| Aufgabe 7: Variablendeclaration | 83 |
| Aufgabe 8: Entwurf einer Methode (1) | 83 |
| Aufgabe 9: Entwurf einer Methode (2) | 83 |
| Aufgabe 10: Programmierung von Klassen und Objekten | 84 |
| Aufgabe 11: Verweis auf sich selbst | 84 |
| Aufgabe 12: Konstruktor | 84 |
| Aufgabe 13: Leseübung | 84 |
| Aufgabe 14: Autorennen | 85 |
| Aufgabe 15: Sternen-Himmel | 86 |
| Aufgabe 16: Bälle (alte Klausuraufgabe) | 87 |
| 11 UML-Zustandsdiagramme | 89 |
| 11.1 Notation | 89 |
| Darstellung der Zustände | 89 |
| Darstellung von Zustandsübergängen | 89 |
| Zustand oder Zustandsübergang? | 90 |
| 11.2 Beispiele | 90 |
| Zustandsdiagramme für das Strichmännchen | 90 |
| Komplexes Beispiel | 91 |
| 11.3 UML-Zustandsdiagramme – Übungen | 92 |
| Aufgabe 1: Windows | 92 |
| Aufgabe 2: Fähre | 92 |
| Aufgabe 3: Seehund | 92 |
| Aufgabe 4: Getränke-Automat | 92 |
| Aufgabe 5: xkcd: My Problem With Phone Alarms | 93 |
| 12 Datenkapselung | 95 |
| 12.1 Problembeschreibung | 95 |
| 12.2 Lösung: Datenkapselung | 96 |
| 12.3 Datenkapselung – Übungen | 97 |
| Aufgabe 1: Auf ein Haus zufahren | 97 |
| Aufgabe 2: Lampe | 97 |
| Aufgabe 3: Ampel | 98 |
| Aufgabe 4: Lichterkette | 98 |
| Aufgabe 5: Digitaluhr | 98 |
| 13 Grafik-Dateien | 101 |
| 13.1 Grafik-Dateien in eigenen Java-Programmen nutzen | 101 |
| Unterstützte Bildformate | 101 |
| Erzeugung eines Image-Objektes | 101 |
| Malen eines Image-Objektes | 102 |
| Beispiel-Programm | 102 |
| 13.2 Grafik-Dateien – Übungen | 104 |
| Aufgabe 1: Himmel mit Vögeln und Wolken | 104 |
| Aufgabe 2: Schiffe auf dem Meer | 104 |
| 14 Sound | 105 |
| 14.1 Wiedergabe von *.wav Dateien | 105 |
| 14.2 Wiedergabe von *.mp3 Dateien | 106 |

| | |
|---|------------|
| 15 UML-Klassendiagramme | 109 |
| 15.1 Allgemeine Beziehungen: Assoziation | 109 |
| Multiplizität | 109 |
| 15.2 „Ist-Teil-von“-Beziehung: Aggregation | 110 |
| 15.3 „Ist-ein“-Beziehung: Vererbung | 110 |
| Klassenhierarchie | 110 |
| Abstrakte Klassen | 111 |
| 15.4 UML-Klassendiagramme – Übungen | 112 |
| Aufgabe 1: Kartenverkauf im Theater | 112 |
| Aufgabe 2: Assoziationen | 112 |
| Aufgabe 3: Bestandteile eines Computers | 112 |
| Aufgabe 4: Möbelfirma | 113 |
| Aufgabe 5: Bank | 113 |
| Aufgabe 6: Werkstatt | 114 |
| 16 Vererbung | 115 |
| 16.1 Wie man eine Klasse ableitet | 115 |
| 16.2 Überschreiben von Methoden | 115 |
| Noch ein paar Besonderheiten | 115 |
| Konstruktor | 115 |
| 16.3 Datenkapselung: vollständiger Überblick | 116 |
| 16.4 Vererbung – Übungen | 117 |
| Aufgabe 1: Übung zur Datenkapselung | 117 |
| Aufgabe 2: Leseübung | 117 |
| Aufgabe 3: Zwei spezielle Lampen | 118 |
| Aufgabe 4: Drachen | 119 |
| Aufgabe 5: Schiffe | 120 |
| Aufgabe 6: Billardkugeln | 121 |
| 17 Wiederholung | 125 |
| 17.1 Programmierung | 125 |
| Aufgabe 1: Kette | 125 |
| Aufgabe 2: Kreisende Bälle | 126 |
| 17.2 Theorie | 127 |
| Aufgabe 1: Schlüsselworte | 127 |
| Aufgabe 2: Datenkapselung | 127 |
| Aufgabe 3: Vererbung | 127 |
| 17.3 UML | 127 |
| Aufgabe 1: UML-Klassendiagramme | 127 |
| Aufgabe 2: Model-Agentur | 127 |
| Aufgabe 3: Weihnachtsmann-Roboter | 128 |
| Aufgabe 4: Weihnachts-Lieferservice | 128 |
| 18 Arrays | 129 |
| 18.1 Arrays von primitiven Datentypen(boolean, int, double, char) | 129 |
| Eindimensionale Arrays | 129 |
| Mehrdimensionale Arrays | 130 |
| 18.2 Arrays von Objekten | 130 |
| 18.3 Arrays – Übungen | 132 |
| Aufgabe 1: Zahlen | 132 |
| Aufgabe 2: Summe und Extremwerte von Zahlen | 132 |
| Aufgabe 3: Strichmännchen | 133 |
| Aufgabe 4: Billardkugeln | 133 |
| Aufgabe 5: Meer mit Fischen | 133 |
| Aufgabe 6: Game of Life | 134 |
| Aufgabe 7: Mandelbrot-Menge | 136 |
| Aufgabe 8: Clifford-Attraktor | 140 |

| | |
|---|------------|
| 19 Sortierverfahren | 143 |
| 19.1 Gruppe 1: Sortieren durch Einfügen (Insertion Sort) | 144 |
| 19.2 Gruppe 2: Sortieren durch Auswählen (Selection Sort) | 145 |
| 19.3 Gruppe 3: Bubble Sort | 146 |
| 19.4 Sortierverfahren – Übungen | 147 |
| Aufgabe 1: Sortieren der Zahlenreihe „8 3 2 1“ | 147 |
| Aufgabe 2: Namen sortieren | 147 |
| Aufgabe 3: Welches Verfahren sortiert am schnellsten? | 147 |
| Aufgabe 4: Programmierübung | 148 |
| 20 Abstrakte Klassen und Interfaces | 149 |
| 20.1 Abstrakte Klassen | 149 |
| 20.2 Interfaces | 149 |
| 20.3 Abstrakte Klassen und Interfaces – Übungen | 151 |
| Aufgabe 1: Ableitung von Abstrakte Klassen | 151 |
| Aufgabe 2: Implementierung eines Interfaces | 151 |
| 21 Tastaturereignisse | 153 |
| 21.1 Ereignisbehandlung | 153 |
| 21.2 Datentyp char | 153 |
| 21.3 Tastaturereignisse – Übungen | 155 |
| Aufgabe 1: Leseübung | 155 |
| Aufgabe 2: Game of Life | 156 |
| Aufgabe 3: Stoppuhr | 156 |
| Aufgabe 4: Dartspiel | 156 |
| Aufgabe 5: Autorennen | 157 |
| Aufgabe 6: Streifen | 158 |
| 22 Mausereignisse | 159 |
| 22.1 Normale Mausereignisse | 159 |
| 22.2 Mausbewegungen | 159 |
| 22.3 Details über Mausereignisse erfahren | 159 |
| 22.4 Adapterklassen | 159 |
| 22.5 Mausereignisse – Übungen | 161 |
| Aufgabe 1: Markierungen setzen | 161 |
| Aufgabe 2: Zeichenbrett | 161 |
| Aufgabe 3: Ameisen fangen | 162 |
| Aufgabe 4: Das Lampenproblem | 162 |
| Aufgabe 5: Game of Life erweitern | 163 |
| Aufgabe 6: Snake | 163 |
| 23 Fenster | 165 |
| 23.1 Klassen-Hierarchie in AWT und Swing | 165 |
| 23.2 Erstellung eines eigenständigen Java-Programms | 166 |
| 23.3 Erzeugung eines Programmfensters | 166 |
| 23.4 Methoden von JFrame (Auswahl) | 166 |
| 23.5 Fenster-Ereignisse | 166 |
| Methoden des Interfaces WindowListener | 167 |
| Wichtige Methode von WindowEvent | 167 |
| 23.6 Beispiel | 167 |
| Erläuterungen zum Beispiel | 168 |
| 23.7 WindowListener verwenden | 168 |
| 23.8 Auf das Objekt des Anwendungsfensters zugreifen | 169 |
| 23.9 Programmierung von komplexen Dialogen | 169 |
| 23.10 Fenster – Übungen | 170 |
| Aufgabe 1: windowClosing() | 170 |
| Aufgabe 2: Beleidigtes Fenster | 171 |
| Aufgabe 3: JFrame -Template von Eclipse benutzen | 171 |

| | |
|---|------------|
| 24 GUI-Komponenten | 173 |
| 24.1 Struktur eines Swing UI | 173 |
| Hauptfenster | 173 |
| Container | 173 |
| Komponenten | 173 |
| Layout-Manager | 173 |
| Arbeiten mit einem GUI-Designer | 173 |
| 24.2 Einfache Dialogelemente | 174 |
| Gemeinsamkeiten aller Komponenten (geerbt von JComponent) | 174 |
| JButton | 175 |
| JLabel | 176 |
| JCheckBox | 176 |
| JTextField | 177 |
| JTextArea | 178 |
| 24.3 Einfache Dialogelemente – Übungen | 179 |
| Aufgabe 1: Layout-Manager austesten | 179 |
| Aufgabe 2: Knopfdrücke zählen | 179 |
| Aufgabe 3: Fahne im Wind | 179 |
| Aufgabe 4: Anrede | 179 |
| Aufgabe 5: RGB-Farben mischen | 179 |
| 24.4 Allgemeines zur Implementierung von Event-Listenern (Ereignisbehandlung) | 181 |
| 24.5 Typkonvertierung | 182 |
| Beispiel | 182 |
| Regeln für die Typkonvertierungen | 183 |
| 24.6 Typkonvertierungen – Übungen | 185 |
| Aufgabe 1: Typkonvertierungen | 185 |
| Aufgabe 2: Tic Tac Toe | 185 |
| Aufgabe 3: Taschenrechner | 186 |
| 24.7 Arbeiten mit Zeichenketten (Strings) | 188 |
| Methoden der Klasse String (Auswahl) | 188 |
| 24.8 Methoden der Klasse Character | 189 |
| 24.9 Strings – Übungen | 190 |
| Aufgabe 1: Vergleich von Strings | 190 |
| Aufgabe 2: String-Operationen | 190 |
| Aufgabe 3: Noch mehr String-Operationen | 190 |
| Aufgabe 4: Initialen | 191 |
| Aufgabe 5: Rechnen mit versteckten Zahlen | 191 |
| Aufgabe 6: Geheimbotschaft | 191 |
| 24.10 Komplexere Dialogelemente | 193 |
| Allgemein | 193 |
| JList | 193 |
| JScrollPane | 194 |
| JComboBox | 195 |
| 24.11 Komplexe Dialogelemente – Übungen | 197 |
| Aufgabe 1: Einkaufsliste | 197 |
| 25 Fehlerbehandlung | 199 |
| 25.1 Fehler-Klassen | 199 |
| Wichtige Methoden der Super-Klasse Throwable | 199 |
| 25.2 Beispiel | 200 |
| 25.3 Weiterleiten von Exceptions | 201 |
| 25.4 Behandlung verschiedenartiger Exceptions | 202 |
| 25.5 Fehlerbehandlung – Übungen | 203 |
| Aufgabe 1: Währungsrechner | 203 |
| Aufgabe 2: Benzinkosten berechnen | 203 |
| Aufgabe 3: Andere Programme starten | 204 |

| | |
|--|------------|
| 26 Nebenläufige Programmierung: Threads | 207 |
| 26.1 Wofür benötigt man Threads? | 207 |
| 26.2 Beispiel | 207 |
| 26.3 Animation steuern mit einem eigenen Timer-Thread | 209 |
| Beispiel | 209 |
| 26.4 Threads – Übungen | 211 |
| Aufgabe 1: Rennendes Pferd | 211 |
| Aufgabe 2: Oldtimer | 211 |
| Aufgabe 3: Jumping Potatoes | 212 |
| Aufgabe 4: Störrischer Esel | 212 |
| 26.5 Thread-Synchronisation | 214 |
| Beispiel | 214 |
| Lösung des Problems | 214 |
| 26.6 Thread-Synchronisation – Übungen | 216 |
| Aufgabe 1: Drucker-Simulation | 216 |
| Aufgabe 2: Dining Philosophers | 216 |
| 27 Dateizugriffe | 219 |
| 27.1 Dateien lesen und schreiben | 219 |
| 27.2 <code>FileInputStream</code> und <code>FileOutputStream</code> | |
| <code>InputStreamReader</code> und <code>OutputStreamWriter</code> | 219 |
| Datei öffnen | 219 |
| Einlesen von Daten | 220 |
| Ausgabe von Daten | 220 |
| Datei schließen | 220 |
| 27.3 Dateinamen und -pfade | 220 |
| 27.4 Dateien aus JAR Archiven zum Lesen öffnen | 221 |
| 27.5 Beispiel: Schreiben in eine Datei | 221 |
| 27.6 Beispiel: Lesen aus einer Datei | 222 |
| 27.7 Dateizugriffe – Übungen | 223 |
| Aufgabe 1: Zeichen zählen | 223 |
| Aufgabe 2: Einfache Verschlüsselung mit dem Caesar-Verfahren | 223 |
| Aufgabe 3: Prüfsumme | 224 |
| Aufgabe 4: Notizbuch | 225 |
| 27.8 Konfigurations-Dateien (<code>java.util.Properties</code>) | 227 |
| 28 UML Wiederholung | 229 |
| 28.1 UML Zustandsdiagramme | 229 |
| Aufgabe 1: Seehund | 229 |
| Aufgabe 2: Sprechende Puppe | 229 |
| 28.2 UML Klassendiagramme | 229 |
| Aufgabe 1: Zelten | 229 |
| Aufgabe 2: Gitarren | 229 |
| Aufgabe 3: Landleben | 229 |
| 29 Softwaretechnik | 231 |
| 29.1 Die Entwicklung eines Video-Überwachungssystems oder warum die Firma LogoSoft Pleite ging | 231 |
| Was vorher geschah | 231 |
| Der Vertrag wird abgeschlossen | 231 |
| Das Projekt beginnt | 232 |
| Ablauf des ersten halben Jahres | 232 |
| Das Projekt muss verlängert werden | 232 |
| Der Geldhahn wird zugedreht | 233 |
| Technische Mängel | 233 |
| Die Test-Phase | 233 |
| Das Ende | 233 |
| 29.2 Kontrollfragen zum Text | 234 |

| | |
|---|------------|
| 29.3 Cartoon: How Projects Really Work | 237 |
| 30 Javadoc | 239 |
| 30.1 Javadoc als Sonderfall von Kommentaren | 239 |
| 30.2 Javadoc zum Dokumentieren von Klassen, globalen Variablen und Methoden | 239 |
| 30.3 Javadoc-Tags | 240 |
| 30.4 HTML-Formatierungen | 240 |
| 30.5 Beispiel | 240 |
| 31 Projektarbeit | 243 |
| 31.1 Programm-Planung | 243 |
| 31.2 Programmierung und Test | 243 |
| 31.3 Benotung | 244 |
| 32 SQL – Einführung | 245 |
| 32.1 Relationale Datenbanken – Überblick | 245 |
| Relationale Datenbanken | 245 |
| Datenbanksysteme | 245 |
| Organisation eines Datenbankmanagementsystems (DBMS) | 245 |
| Terminologie | 245 |
| Front End und Back End | 246 |
| SQL – Structured Query Language | 246 |
| 32.2 Installationsanleitung für MySQL | 246 |
| Installation | 246 |
| 32.3 SQL – Anlegen von Datenbanken | 247 |
| Groß- und Kleinschreibung | 247 |
| Datenbank erstellen | 247 |
| Datenbank auswählen | 247 |
| Den verwendeten Zeichensatz festlegen | 247 |
| Tabelle erstellen | 247 |
| Datenbanken und Tabellen löschen | 248 |
| Tabellenstruktur ändern | 248 |
| 32.4 SQL – Daten eingeben | 249 |
| Zeichenketten und Datum- / Zeit-Angaben | 249 |
| Daten einfügen | 249 |
| Daten der Tabelle ansehen | 249 |
| Daten löschen | 249 |
| Daten aktualisieren | 249 |
| Vergleichsoperatoren | 249 |
| SQL-Anweisungen aus einer Datei einfügen | 250 |
| 32.5 SQL – Daten abfragen | 250 |
| SELECT-Anweisung | 250 |
| 32.6 SQL – Übung 1: Eine einfache Haustierdatenbank | 252 |
| Aufgabe 1: Datenbank erzeugen | 252 |
| Aufgabe 2: Daten eingeben | 252 |
| Aufgabe 3: Daten abfragen | 253 |
| 32.7 SQL – Übung 2: Die Haustierdatenbank wird erweitert | 254 |
| Aufgabe 1: Datenstruktur erweitern | 254 |
| 32.8 Tabellen verknüpfen: Primär- und Fremdschlüssel | 255 |
| Primärschlüssel | 255 |
| Fremdschlüssel | 255 |
| Referentielle Integrität | 256 |
| Verknüpfung von Tabellen | 256 |
| 32.9 SQL – Übung 3: Erweiterte Haustierdatenbank | 257 |
| Aufgabe 1: Daten eingeben | 257 |
| Aufgabe 2: Daten ändern | 257 |
| Aufgabe 3: Daten abfragen | 257 |

| | |
|--|------------|
| 32.10 SQL – Daten abfragen (Fortsetzung) | 258 |
| Kartesisches Produkt | 258 |
| LEFT JOIN und RIGHT JOIN | 258 |
| Verknüpfung einer Tabelle mit sich selbst | 259 |
| Unterabfragen | 259 |
| 32.11 SQL – Übung 4: Die Haustierdatenbank (Fortsetzung) | 260 |
| Aufgabe 1: Datenbank erweitern | 260 |
| Aufgabe 2: Daten abfragen | 260 |
| Aufgabe 3: Daten verändern | 260 |
| Aufgabe 4: Daten abfragen 2 | 260 |
| 32.12 SQL – Weiterführendes | 261 |
| Ähnlichkeitssuche mit LIKE | 261 |
| MAX(), MIN(), AVG() und SUM() | 261 |
| Mathematische Funktionen | 261 |
| Datentyp BLOB | 261 |
| Unterabfragen | 261 |
| Stored Procedures | 262 |
| 33 SQL – Komplexes Beispiel | 263 |
| 33.1 Firmen-Datenbank | 263 |
| Aufgabe 1: Datenbank Firma untersuchen | 263 |
| Aufgabe 2: Datenbankabfragen | 263 |
| 34 Entity-Relationship-Modell | 265 |
| 34.1 Begriffe im ER-Modell | 265 |
| Entität und Entitätsotyp | 265 |
| Attribute | 265 |
| Beziehungen | 265 |
| 34.2 Einfaches Beispiel eines ER-Diagramms | 266 |
| 34.3 ER-Modell – Übung | 267 |
| Aufgabe 1: Interpretation eines ER-Diagramms | 267 |
| Aufgabe 2: Kardinalitäten | 267 |
| Aufgabe 3: Datenbankentwurf | 267 |
| Aufgabe 4: Vergleich mit UML | 268 |
| 35 Normalisierung | 269 |
| 35.1 Erster Entwurf einer Tabelle | 269 |
| 35.2 Erste Normalform | 269 |
| 35.3 Zweite Normalform | 270 |
| 35.4 Dritte Normalform | 271 |
| 35.5 Höhere Normalformen | 271 |
| 35.6 Lesetipp | 271 |
| 35.7 Normalisierung – Übung | 272 |
| Aufgabe 1 | 272 |
| Aufgabe 2 | 272 |
| 36 SQL – Wiederholung | 273 |
| 36.1 Aufgaben zur Wiederholung | 273 |
| Aufgabe 1: Datenbankentwurf | 273 |
| Aufgabe 2: Normalformen | 273 |
| Aufgabe 3: Urlaubs-Datenbank | 273 |
| 37 Datenbankzugriffe mit Java | 275 |
| 37.1 SQL-Package | 275 |
| 37.2 Verbindung zur Datenbank aufbauen | 275 |
| 37.3 Datenbankabfragen | 276 |
| 37.4 Datenbankänderungen | 276 |
| 37.5 Beispiel | 276 |

| | | |
|-----------|---|------------|
| 37.6 | Programmierübung zur Haustierdatenbank | 278 |
| | Aufgabe 1: Ausgabe mit einer JList-Komponente | 278 |
| | Aufgabe 2: Datensätze durchblättern | 278 |
| | Aufgabe 3: Tiere löschen | 279 |
| | Aufgabe 4: Neue Tiere hinzufügen | 279 |
| 37.7 | Programmierung eines Terminplaners | 279 |
| | Aufgabe 1: Erzeugen einer geeigneten Datenbank | 279 |
| | Aufgabe 2: Programmierung des Terminplaners | 280 |
| | Aufgabe 3: Ein Programm zum Anzeigen der aktuellen Termine | 281 |
| 37.8 | SQL-Injection | 282 |
| 38 | Java mit SQL – Wiederholung | 283 |
| | Aufgabe 1: Umwandlung von Strings in Zahlen | 283 |
| | Aufgabe 2: String-Vergleich | 283 |
| | Aufgabe 3: Programmierübung zur Haustier-Datenbank | 283 |
| | Aufgabe 4: KFZ-Werkstatt | 284 |
| | Aufgabe 5: Programmierung einer Software für ein Reiseunternehmen | 284 |
| | Aufgabe 6: Programmierung einer Software für einen Getränkehandel | 286 |
| 39 | Wiederholung | 289 |
| 39.1 | Java | 289 |
| | Aufgabe 1: String-Manipulationen | 289 |
| | Aufgabe 2: Count Down (Thread-Übung) | 289 |
| | Brötchenbestellung | 290 |
| | Aufgabe 4: Lauftext (Thread-Übung) | 290 |
| | Aufgabe 5: Steinen ausweichen | 290 |
| 39.2 | UML-Zustandsdiagramme | 292 |
| | Aufgabe 1: Hausbau | 292 |
| | Aufgabe 2: Igelspiel | 292 |
| 40 | Datensicherheit | 293 |
| 41 | Kryptologie | 295 |
| 41.1 | Monoalphabetische Verfahren | 295 |
| | Das Caesar-Verfahren | 295 |
| | Substitutionsverfahren | 296 |
| 41.2 | Polyalphabetische Verschlüsselungsverfahren | 296 |
| | Vigenère-Verfahren | 296 |
| | One-Time-Pad | 297 |
| | DES | 297 |
| | IDEA | 297 |
| 41.3 | Asymmetrische Verschlüsselung | 297 |
| | RSA-Algorithmus | 298 |
| 41.4 | Anwendungsgebiete der Kryptographie | 299 |
| 41.5 | Kryptoanalyse | 299 |
| 41.6 | Kryptologie – Übungen | 301 |
| | Aufgabe 1: Caesar-Verfahren I | 301 |
| | Aufgabe 2: Caesar-Verfahren II | 301 |
| | Aufgabe 3: Caesar-Verfahren III | 301 |
| | Aufgabe 4: Substitutionsverfahren I | 301 |
| | Aufgabe 5: Substitutionsverfahren II | 301 |
| | Aufgabe 6: Substitutionsverfahren III | 301 |
| | Aufgabe 7: Substitutionsverfahren IV | 301 |
| | Aufgabe 8: Vigenère-Verfahren I | 302 |
| | Aufgabe 9: Vigenère-Verfahren II | 302 |
| | Aufgabe 10: Programmierübung | 302 |
| | Aufgabe 11: Asymmetrische Verfahren | 302 |
| | Aufgabe 22: Digitale Signatur | 302 |

| | |
|--|------------|
| Aufgabe 13: RSA | 302 |
| Aufgabe 14: Vergleich von symmetrischer und asymmetrischer Verschlüsselung | 303 |
| Aufgabe 15: Web of Trust | 303 |
| 42 Netzwerke | 305 |
| 42.1 Netzwerk-Kommunikation | 305 |
| Peer to Peer (P2P) | 305 |
| Client-Server | 305 |
| 42.2 Protokoll | 305 |
| 42.3 OSI-Schichtenmodell | 306 |
| 42.4 TCP/IP im vereinfachten Schichtenmodell | 306 |
| 42.5 IP-Adressen | 308 |
| 42.6 Host- und Domain-Namen | 308 |
| Loopback-Adresse | 309 |
| 42.7 IP-Adressen in Java | 309 |
| 42.8 Port-Nummern | 310 |
| 42.9 Übungsaufgaben: Protokolle | 311 |
| Aufgabe 1: Vier Gewinnt | 311 |
| Aufgabe 2: Schiffe Versenken | 311 |
| Aufgabe 3: Veranschaulichung des Schichtenmodells | 311 |
| Aufgabe 4: TCP und UDP | 312 |
| 42.10 Übungsaufgaben: IP-Adressen und Ports | 312 |
| Aufgabe 5: IP-Adresse des eigenen Rechners | 312 |
| Aufgabe 6: Ports | 312 |
| Aufgabe 7: Programmierübung | 313 |
| 43 Client/Server | 315 |
| 43.1 Client | 315 |
| Sockets | 315 |
| Streams | 315 |
| Einlesen von Daten | 316 |
| Beispiel: Einlesen von Daten aus einem Socket (Code-Auszug) | 316 |
| Ausgabe von Daten | 316 |
| Verbindung beenden | 316 |
| 43.2 Server | 317 |
| 43.3 Allgemeiner Aufbau von Client- und Server-Programmen | 317 |
| Tipps | 318 |
| Server-Gerüst | 318 |
| Client-Gerüst | 321 |
| 43.4 Übungsaufgaben: Client/Server | 326 |
| Aufgabe 1: Einfacher Client | 326 |
| Aufgabe 2: Echo-Client | 326 |
| Aufgabe 3: HTTP-Client | 327 |
| Aufgabe 4: POP3-Protokoll | 329 |
| Aufgabe 5: Einfacher Server | 329 |
| Aufgabe 6: Echo-Server | 330 |
| Zusatzaufgabe: Chat-Client und -Server | 330 |
| 44 Abi-Training | 333 |
| 44.1 Allgemeines | 333 |
| Aufgabe 1: Programmietechnik | 333 |
| Aufgabe 2: Netzwerktechnik | 333 |
| Aufgabe 3: Kryptologie | 333 |
| 44.2 Einfache Programmieraufgaben | 334 |
| Aufgabe 1: Satz in Wörter zerlegen | 334 |
| Aufgabe 2: Zeichen extrahieren | 334 |
| 44.3 Q2.2: Client/Server | 335 |
| Aufgabe 1: Dateiinhalte Senden | 335 |

| | |
|--|------------|
| Aufgabe 2: Rechentrainer | 335 |
| 44.4 Q2.1: Datenbanken | 338 |
| Aufgabe 1: Osterhasen GmbH | 338 |
| Aufgabe 2: Schuldatenbank I | 338 |
| Aufgabe 3: Schuldatenbank II | 339 |
| Aufgabe 4: Kartenverkauf | 340 |
| Aufgabe 5: Fahrzeug-Datenbank | 340 |
| Aufgabe 6: Fahrzeuge hinzufügen | 341 |
| Aufgabe 7: Fahrzeuge abmelden (mit JList) | 341 |
| Aufgabe 8: Fahrzeuge abmelden (mit JComboBox) | 342 |
| 44.5 Gemischte Aufgaben (Datenbank plus Client/Server) | 342 |
| Aufgabe 1: Buchhandel | 342 |
| 45 Typische Fehler ... und wie man sie vermeidet | 347 |
| 45.1 Debugging: Fehler finden | 347 |
| Debugging mit einfachen Mitteln: Textausgabe auf der Konsole | 347 |
| 45.2 Fehlende geschweifte Klammern | 348 |
| 45.3 Das tödliche Semikolon | 348 |
| ... und wie man es vermeiden kann | 349 |

1 Eclipse als Java-Entwicklungsumgebung

1.1 Vorbereitung

Installation des Java-SDKs

Installiere das aktuelle JDK auf deinem Computer:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

→ *JDK* → *Download* → *Accept License Agreement*

Aus der Liste wählst du die für dein Betriebssystem und Architektur (32bit oder 64bit) passende Datei aus.

Anlegen eines Git-Repositories für deine Java-Dateien

Spätestens wenn mehrere Personen (oft dazu noch an unterschiedlichen Orten) gemeinsam an einem Software-Projekt arbeiten, macht es keinen Sinn mehr seine Programm-Quelltexte auf der lokalen Festplatte zu speichern. Vielmehr werden dann Versionsverwaltungssysteme – sogenannte Repositories – benutzt, die die Daten online an einer zentralen Stelle bereit halten. Siehe

<http://de.wikipedia.org/wiki/Repository>

Git

Git ist ein modernes System zur verteilten Versionsverwaltung. Es wurde 2005 von Linus Torvalds, dem Erfinder von Linux, als Open Source Alternative zum bis dahin für die Linux Kernel Entwicklung eingesetzten BitKeeper System entwickelt.

Die Versionsverwaltung mit Git erfolgt zunächst einmal in einem Verzeichnis auf dem lokalen Rechner. Üblicherweise wird man irgendwo im Internet einen Git-Server benutzen, der als zentrales Repository dient, über den der Austausch zwischen beliebig vielen lokalen Repositories (für uns: typischerweise eines zu Hause und eines in der Schule) läuft.

Es gibt mehrere Anbieter von kostenlosen Git-Servern. Der bekannteste ist wohl Github. Bei einem anderen Provider, nämlich GitLab, kann man nicht nur öffentliche, sondern auch private (also für andere nicht einsehbare) kostenlos Repositories anlegen:

https://gitlab.com/users/sign_in

Dazu musst du dich mit einer gültigen e-Mail Adresse registrieren (siehe Abbildung 1.1).

Du kannst Eclipse auch ohne solch ein Git-Repository nutzen, oder ein kostenloses Repositories eines anderen Providers nutzen.

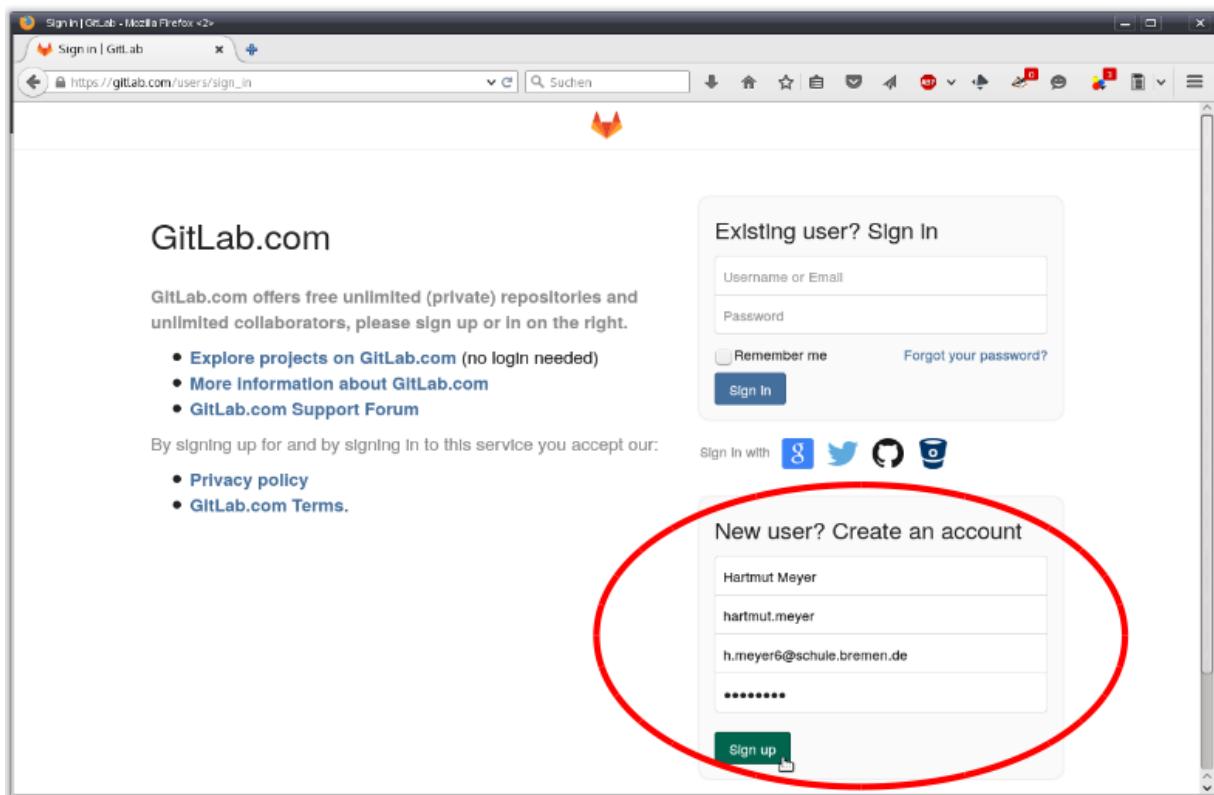


Abbildung 1.1: Anlegen eines Accounts bei gitlab.com

Am Beispiel von GitLab wird jetzt gezeigt, wie du dir ein eigenes Projekt anlegst. Bei anderen Providern würde es ganz ähnlich funktionieren.

Nach der erfolgreichen Registrierung bei GitLab hast du dort ein Benutzerkonto, aber noch keine Projekte. Nach der Anmeldung findest du direkt auf der Startseite einen Button, um ein neues Projekt anzulegen (Abbildung 1.2).

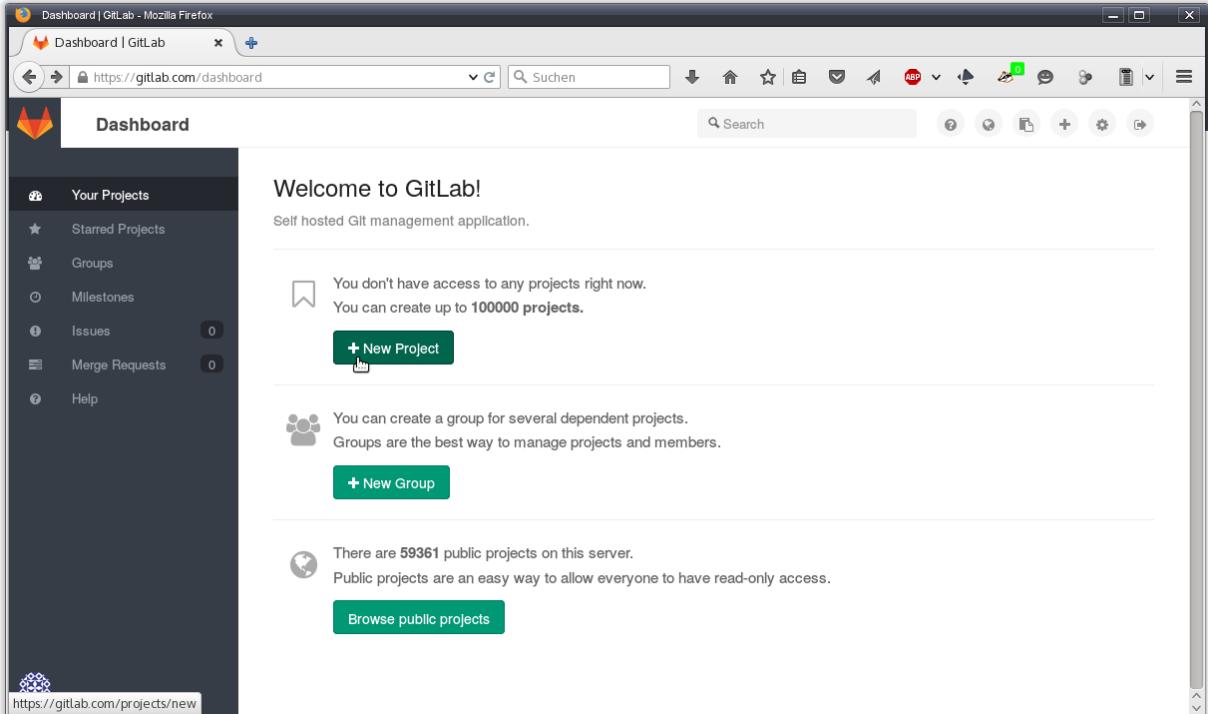


Abbildung 1.2: Anlegen eines eigenen Projects bei GitLab (1)

Auf der folgenden Seite musst du einen Namen für dein Projekt wählen (Abbildung 1.3).

Im letzten Schritt solltest du dir die Adresse (URL) deines Repositories notieren. Dazu musst du zunächst auf den Button *HTTPS* klicken. Außerdem solltest du an dieser Stelle die angebotene Möglichkeit nutzen, eine README Datei anzulegen. Erst mit dieser Readme Datei kann man das Repository anschließend direkt nutzen! Siehe Abbildung 1.4.

Der Inhalt der README Datei ist völlig nebensächlich. Ein Punkt, oder irgendein Nonsense. Aber eben mindestens ein Zeichen. Damit die neue Datei im Repository angelegt werden kann, musst du auch eine sogenannte *Commit Message* schreiben. Erst danach ist der Button *Commit Changes* aktiv.

Der Inhalt der Commit Message ist hier übrigens genauso egal wie der Inhalt der README Datei.

Nach dem Commit ist dein eigenes Projekt angelegt, und du kannst es nun als Git-Repository nutzen.

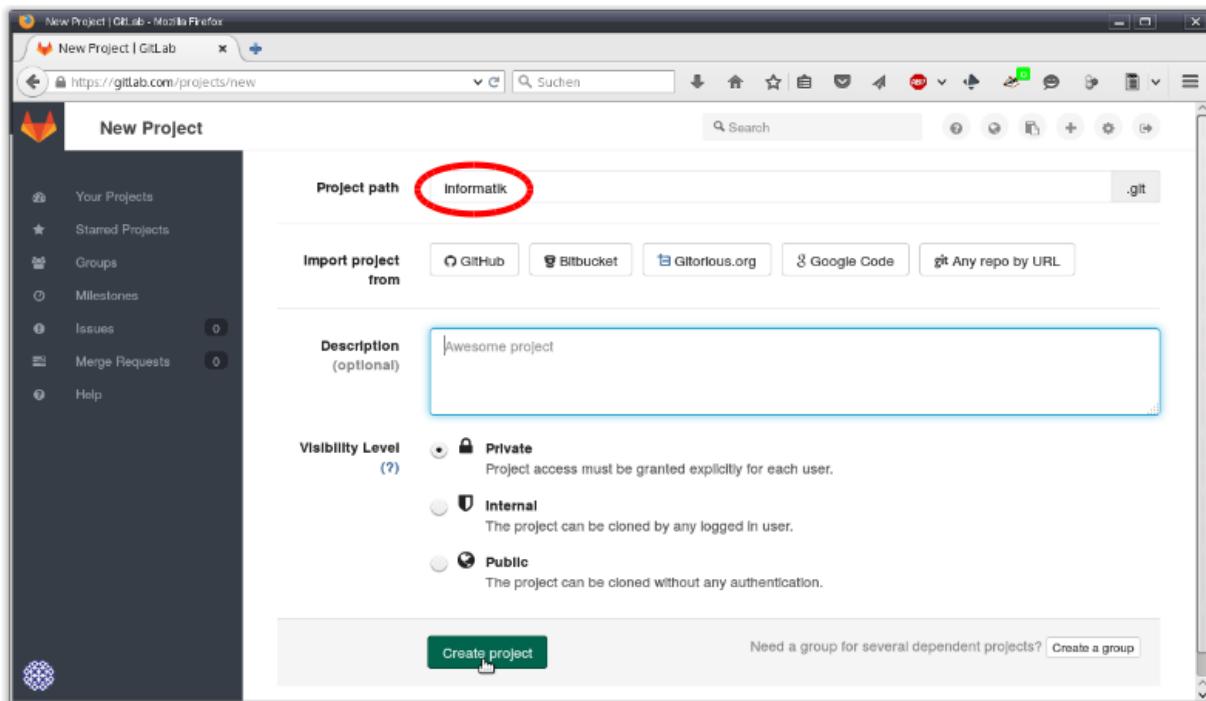


Abbildung 1.3: Anlegen eines eigenen Projects bei GitLab (2)

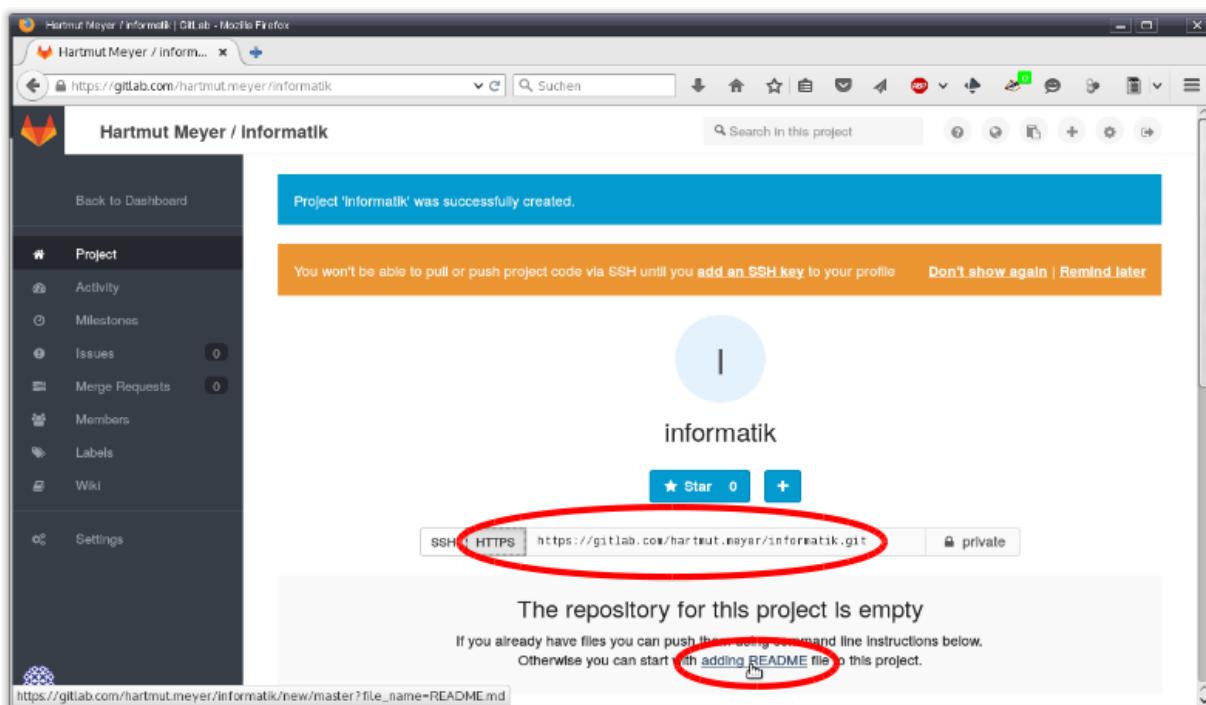


Abbildung 1.4: Anlegen eines eigenen Projects bei GitLab (3)

1.2 Eclipse herunterladen und installieren

Die jeweils aktuellste stabile Version von Eclipse findest du hier:

<https://www.eclipse.org/downloads/>

Dort wählst du den 64-Bit Download von Eclipse, passend für dein Betriebssystem. Heruntergeladen wird so der Eclipse-Installer, den du anschließend startest.

Von den dort angebotenen Paketen solltest du „Eclipse IDE for Java Developers“ auswählen. Wenn du mit Windows arbeitest, wird dir zum Abschluss der Installation angeboten, eine Verknüpfung auf dem Desktop anzulegen und einen Eintrag im Startmenü zu erzeugen.

1.3 Eclipse konfigurieren

Zeichensatz festlegen

Damit ihr untereinander und mit mir problemlos Dateien austauschen könnt, solltet ihr als erstes den Zeichensatz festlegen, mit dem Eclipse eure Textdateien (also auch die Java-Quelltext Dateien) interpretiert. Dies ist im Besonderen bedeutsam für die Kodierung von Umlauten und sonstigen Sonderzeichen.

In Eclipse: *Window → Preferences → General → Workspace → Text file encoding → Other → UTF-8*

(Abbildung 1.5)

Änderungen durch andere Programme im Workspace überwachen

Eclipse geht normalerweise davon aus, dass die Dateien im Eclipse-eigenen Arbeitsverzeichnis (Workspace) nur aus Eclipse selbst heraus angelegt, gelöscht oder verändert werden.

Das hat zur Folge, dass Eclipse durcheinander kommen kann, wenn du beispielsweise mit einem externen Programm eine Datei im Workspace veränderst oder Dateien dort hinein kopierst, löscht oder umbenennst. Es ist aber leicht, Eclipse so einzustellen, dass externe Änderungen an Dateien im Workspace erkannt werden:

Window → Preferences → General → Workspace → Häkchen setzen vor Refresh using native hooks or polling.

(Abbildung 1.5)

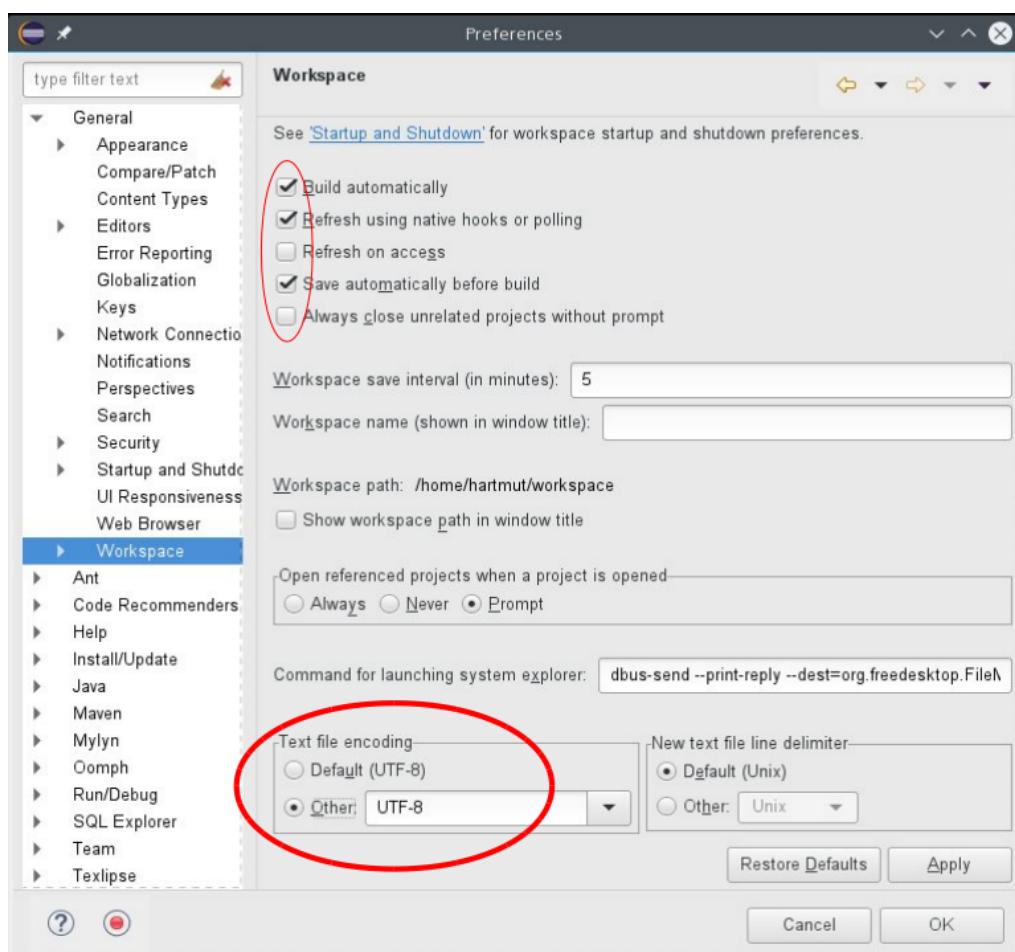


Abbildung 1.5: Preferences-Dialog für die Workspace-Einstellungen

Git-Funktionalität in Eclipse konfigurieren und dein eigenes Repository einbinden

Um nun dein eigenes Repository in Eclipse nutzen zu können, musst du einige wenige Konfigurationsschritte durchführen:

Fall 1: Erstmaliges Einbinden eines zuvor beim Git-Hoster angelegten Repositories (machst du typischerweise in der Schule)

Wenn du bisher noch kein eigenes Java-Projekt angelegt hast, dann bist du hier richtig. Ansonsten folgst du bitte den Anweisungen im nächsten Abschnitt (Fall 2).

File → Import... → Git → Projects from Git → Next → Clone URI → Next Per Copy & Paste die URI (<https://gitlab.com/...>) einfügen sowie dein Benutzername und Passwort für GitLab(!) (Abbildung 1.6)

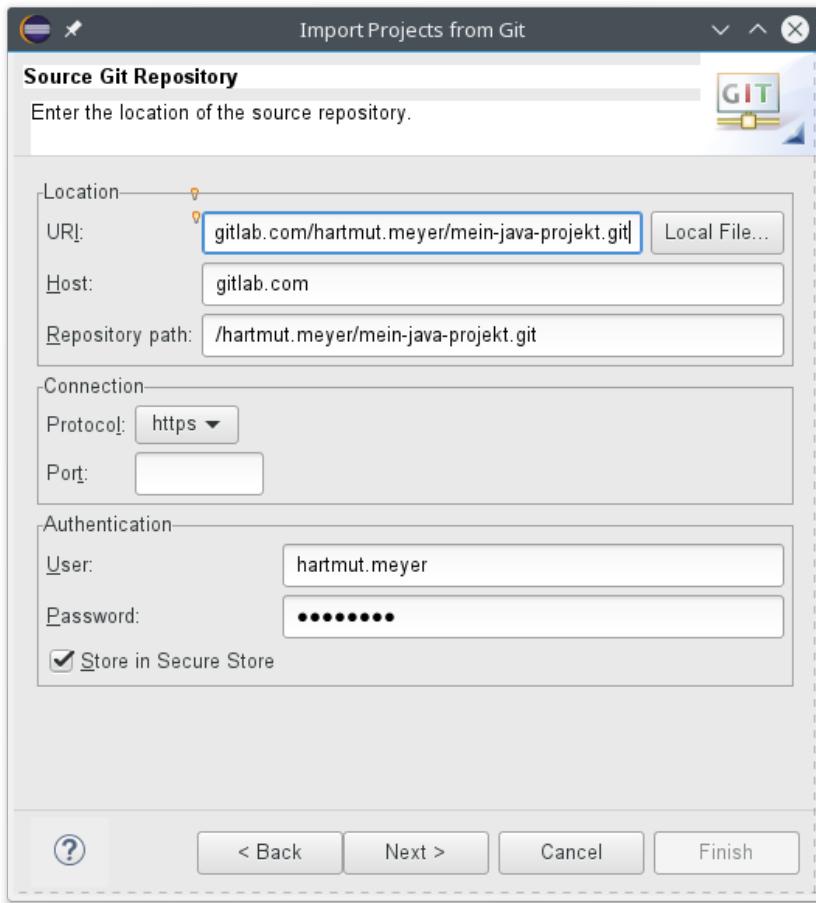


Abbildung 1.6: Ein eigenes Java-Projekt in Eclipse anlegen (erster Schritt)

Von dort aus weiter: → *Next* → *Next* → *Next*

(Abbildung 1.7)

→ *Finish* → *Java* → *Java Project*

(Abbildung 1.8)

→ *Next* → Bei *Project Name*: einen Namen für dein Projekt wählen (kann, muss aber nicht identisch sein, mit dem Namen, den du bei GitLab für dein Projekt gewählt hast) *Next* → *Finish*

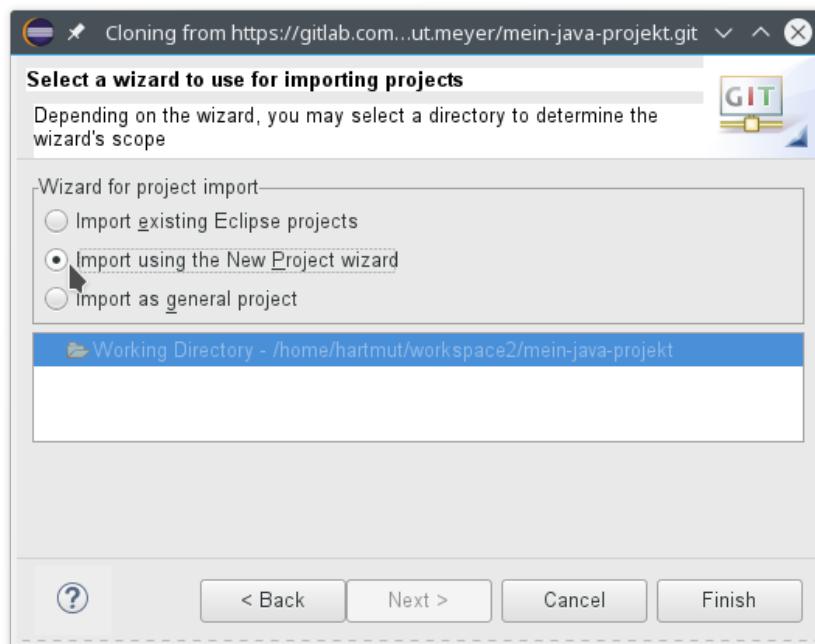


Abbildung 1.7: Ein eigenes Java-Projekt in Eclipse anlegen (using Project Wizard)

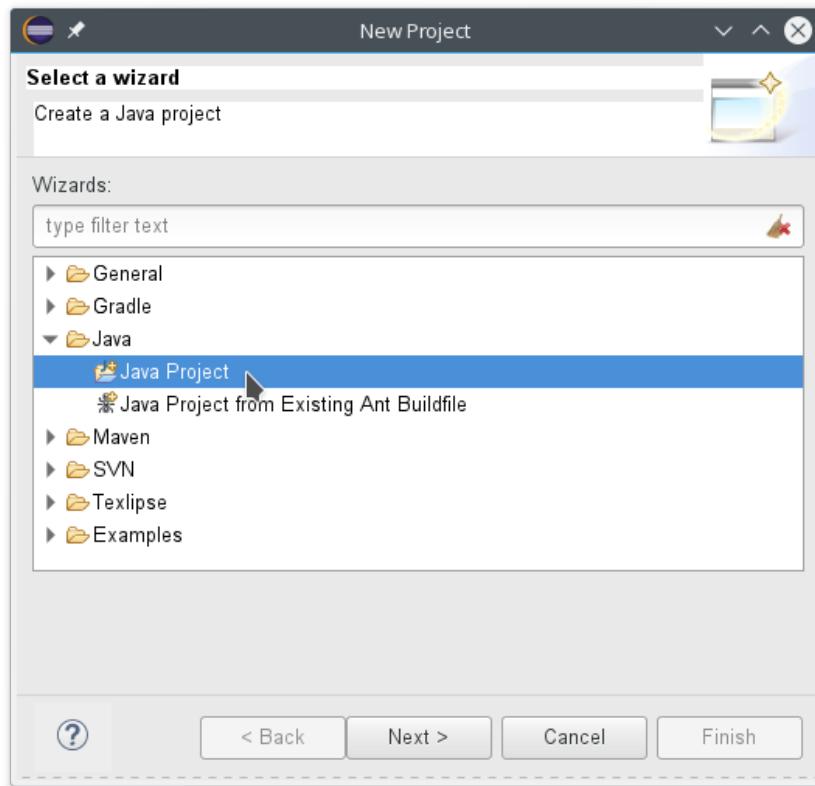


Abbildung 1.8: Ein eigenes Java-Projekt in Eclipse anlegen (Project Wizard: Java Project)

Mit einem Rechts-Klick auf das neue Projekt (im Package-Explorer nun sichtbar) wählst du *New → Package*. Paket-Namen sollten immer mit einem kleinen Buchstaben beginnen. Indem du das Projekt „auffaltest“ (Projekt → src) siehst du die vorhandenen Pakete des Projektes. Ein Rechts-Klick auf ein Paket und ein *New → Class* erzeugt einen Dialog, in dem man den Namen der neuen Java-Klasse wählen kann. Mit einem Rechts-Klick auf ein Projekt im Package-Explorer kann man das Projekt auch in das zuvor angelegte Git-Repository schieben: *Team → Commit →* Dort muss ein Kommentar eingegeben werden, der den Commit beschreibt; Haken setzen um alle neuen/geänderten Dateien im Projekt für den Commit zu markieren (Abbildung 1.10) → *Commit and Push*

Dieses Projekt ist nun im Repository angelegt und man kann ab sofort Dateien des Projekts in dieses Repository sichern (*Team → Commit ...*) oder auch auf die Dateien aus dem Repository auf die lokale Festplatte bringen (*Team → Pull*) – etwa weil du zu Hause gearbeitet hast und nun in der Schule die Dateien auf dem aktuellen Stand haben möchtest.

Fall 2: Du hast dein eigenes Git-Repository bereits einmal in Eclipse eingebunden und dort als Java-Projekt angelegt

Wenn du dein eigenes Java-Projekt bereits einmal in dein damit verknüpftes Git-Repository „geschoben“ (*Commit and Push*) hast, aber es auf dem Computer an dem du gerade arbeitest (zum Beispiel zu Hause) noch nicht in Eclipse eingebunden ist, dann kannst du es leicht aus deinem Git-Repository importieren. Wichtig ist dabei, dass du dabei nicht noch einmal den Projekt-Wizard von Eclipse benutzt, sondern einfach das bereits bestehende Eclipse-Projekt aus dem Git-Repository importierst.

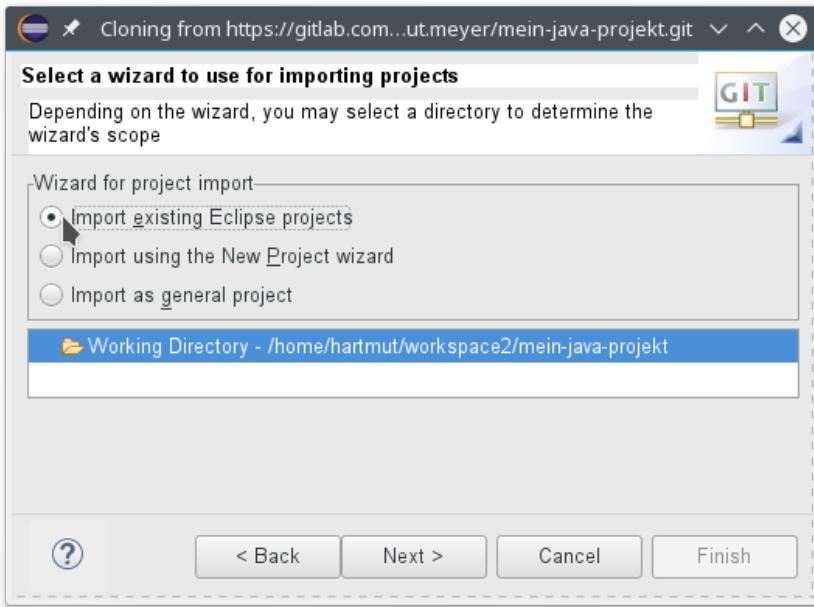


Abbildung 1.9: Ein eigenes Java-Projekt in Eclipse anlegen (Import existing Eclipse projects)

Abgesehen davon folgst du der Anleitung aus dem letzten Abschnitt (Fall 1).

Jetzt können lokale Änderungen in das Online-Repository geschrieben ("Commit and Push") werden und umgekehrt das lokale Repository auf den Stand des Online-Repositories gebracht werden ("Pull"):

Git: Commit and Push

Im Project Explorer: Rechtsklick auf eigenes Projekt → *Team → Commit...* → Dort muss ein Kommentar eingegeben werden, der den Commit beschreibt; Haken setzen um alle neuen/geänderten Dateien im Projekt für den Commit zu markieren (Abbildung 1.10) → *Commit and Push*

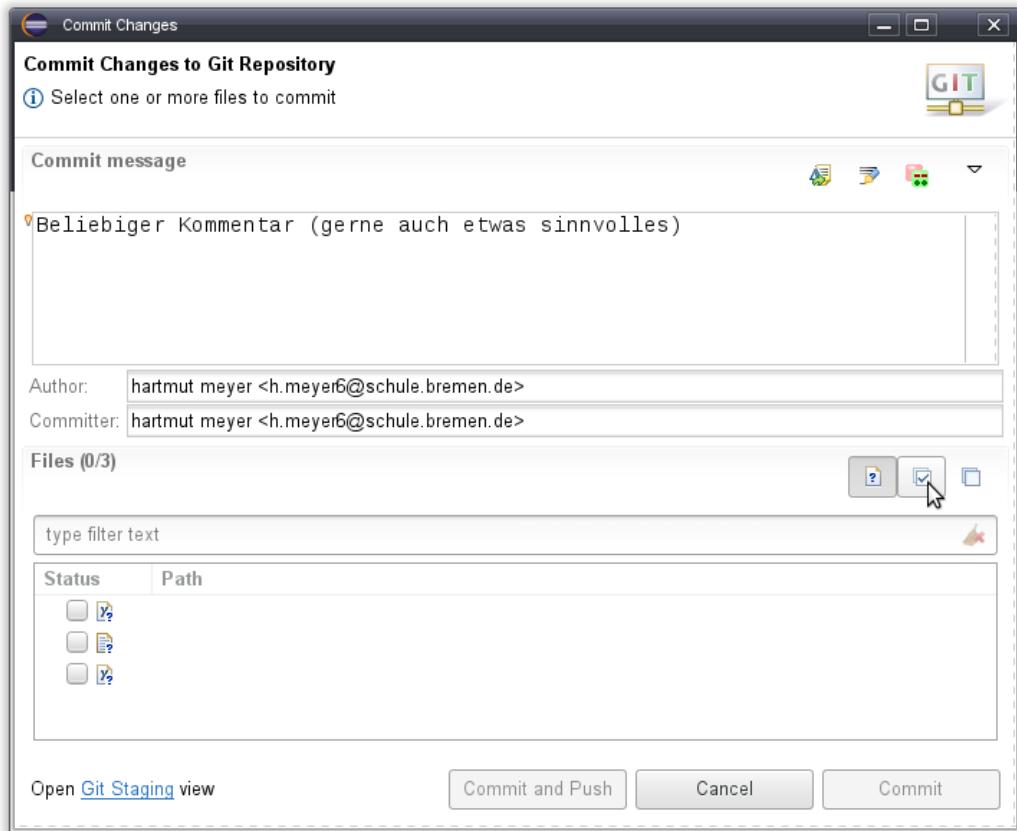


Abbildung 1.10: Commit: Geänderte/neue Dateien in das Online-Repository schreiben



Git: Pull

Im Project Explorer: Rechtsklick auf eigenes Projekt → Team → Pull

Das Kurs-Repository einbinden

Alle Arbeitsblätter und sonstige Dateien werden euch über ein Kurs-Repository zur Verfügung gestellt. Um auf dieses Kurs-Repository zugreifen zu können, kannst du im Git Repositories View ein existierendes Git-Repository zu *clonen* (diese Möglichkeit wird über ein recht unscheinbares kleines Icon am oberen rechten Rand des Git Repositories View angeboten: siehe Abbildung 1.11).

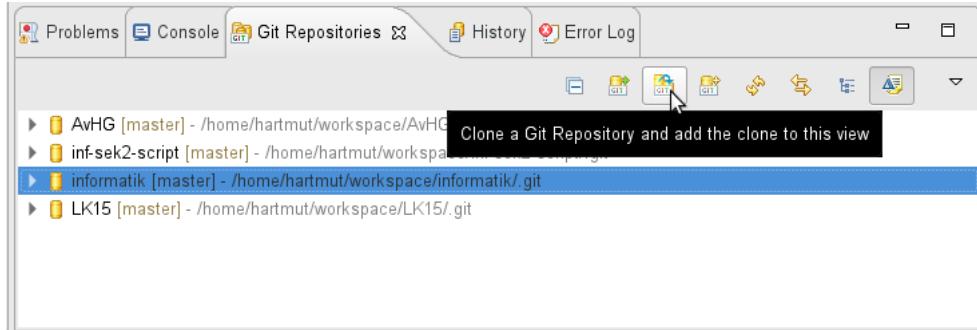


Abbildung 1.11: Commit: Ein existierendes Git-Repository clonen

Dort gibst du im Feld *URI*, die URL für euer Kurs-Repository ein. Etwa <https://github.com/hartmutmeyer/LK15.git> an (der genaue Name eures Kursverzeichnisses wird dir von mir mitgeteilt). Im dem folgenden Dialog *Clone Git Repository: Local Destination*, wählst du als Ordner deinen Eclipse workspace aus. Dieser wird dann automatisch um den Namen des Repositories ergänzt.

Wenn du das Häkchen bei *Import all existing Eclipse projects after clone finishes* gesetzt lässt, wird nach Abschluss des Dialogs (und ggf. einer mehr oder weniger langen Wartezeit, während derer die Dateien aus dem Online-Repository auf die lokale Festplatte kopiert werden), im Projekt Explorer neben deinem eigenen Repository auch das Kurs-Repository erscheinen.

Wichtig: Im Kurs-Repository hast du nur Lese-, aber keine Schreibrechte. Folglich kannst du lokale Änderungen auch nicht mit einem Commit in das Repository sichern. Wenn du Dateien aus dem Kurs-Repository ändern willst musst du diese deshalb immer zuerst in dein eigenes Repository kopieren. Dort kannst du sie dann bearbeiten und auch verändert mit einem Commit in dein eigenes Repository sichern. Falls du versehentlich doch mal etwas im Kurs-Repository verändert haben solltest (erkennbar am schwarzen Stern vor dem Ordner), dann kannst du diese Änderungen ganz einfach mit Rechtsklick auf den Ordner → *Team* → *Replace With* → *Branch, Tag, or Reference...* → Doppelklick auf *Remote Tracking* → *origin/master* → *Replace* rückgängig machen.

Zeilennummern anzeigen lassen

Beim Programmieren ist es hilfreich, sich im Editor die Zeilennummern anzeigen zu lassen. Dies stellst du wie folgt ein:

Window → *Preferences* → *General* → *Editors* → *Text Editors* → *Show line numbers*

Automatische Updates

Eclipse kann automatisch nach Updates suchen:

Window → *Preferences* → *Install/Update* → *Automatic Updates*

In diesem Dialog ein Häkchen Setzen vor *Automatically find new updates and notify me*. Außerdem im Abschnitt *Update schedule* des selben Dialogs auf *Look for updates on the following schedule*: → *Every Saturday* (oder ein beliebiger anderer Wochentag). Im Abschnitt *Download options* wählt du *Download new updates automatically and notify me when ready to install them*. Schließlich den Dialog mit *OK* verlassen.

Package „hilfe“ importieren

Zunächst werdet ihr bei der Java-Programmierung noch häufiger auf Hilfs-Klasse `HJFrame` zurück greifen. Um diese nutzen zu können, müsst ihr in eurem Projekt ein Package `hilfe` anlegen (Rechtsklick auf das Projekt → *new* → *package*) und in dieses anschließend die Dateien `HJFrame.java`, `HZeichnen.java` und `EclipseJFrameHilfe.txt` importieren (Rechtsklick auf das Package → *Import ...* → *General* → *File System* → *Next* → *From directory:* (*Browse* – dort das Verzeichnis wählen, in dem die gewünschten Java-Dateien liegen) → *Next* → im folgenden Dialog alle gewünschten Dateien auswählen → *Finish*).

Die Java-Dateien sollten jetzt im package `hilfe` sichtbar sein.

Erzeugung eines Templates (Vorlage) für `HJFrame`

Window → *Preferences* → *Java* → *Editor* → *Templates* → *New*

Im folgenden Dialog als Namen `HJFrame`, als Beschreibung (kann auch weg gelassen werden) Template für die Ableitung eigener Klassen von der `HJFrame`-Klasse und in dem großen Textfeld *Pattern* den Text (Copy & Paste) aus der Datei `EclipseJFrameHilfe.txt` einfügen. Anschließend zwei mal mit *OK* bestätigen.

Um diese Vorlage zu nutzen genügt es im Editor die Zeichenfolge `HJFrame` einzutippen (Groß- und Kleinschreibung ist dabei egal) und durch ein <Strg>-<Leertaste> zu bestätigen.

Wahl eines externen PDF-Betrachters

Unter Windows werden PDF-Dateien zumindest in älteren Eclipse-Versionen direkt geöffnet. Störend dabei ist vor allem, dass Eclipse jedes Mal meint, die PDF-Datei habe sich verändert und es deshalb beim Schließen der PDF-Ansicht immer die Rückfrage gibt, ob die Änderungen in der Datei gespeichert werden sollen. Das ist unsinnig und irritierend und kann im schlimmsten Fall sogar zu Inkonsistenzen im Repository führen.

Dieses Problem umgehst du, indem du Eclipse anweist, PDF-Dateien nicht selber anzuzeigen, sondern für diesen Zweck ein externes Programm zu benutzen:

Window → *Preferences* → *General* → *Editors* → *File Associations*

Der folgende Dialog ist in einen oberen und einen unteren Bereich aufgeteilt. Zunächst klickst du auf *Add ...* im oberen Bereich und gibst dann `*.pdf` ein. Anschließend wählst du im unteren Bereich ebenfalls *Add ...* und klickst dann im folgenden Dialog auf den Radio-Button für *external Programs*. Aus der folgenden Liste wählst du einen geeigneten PDF-Betrachter aus. Beispielsweise den Acrobat Reader. Nach Bestätigen mit *OK* ist dieser Konfigurationsschritt abgeschlossen.

Auf die gleiche Art und Weise kannst du auch für andere Dateitypen festlegen, mit welchen internen oder externen Betrachtern bzw. Editoren sie geöffnet werden sollen.

Installation des WindowsBuilder Plugins

Im aktuellen Eclipse Release (Oxygen) ist das Plugin WindowsBuilder leider noch nicht vorinstalliert. Ab Q1 werdet ihr dieses Plugin jedoch brauchen.

Help → *Eclipse Marketplace...*

Im Suchfeld gibst du `Windowbuilder` ein und lässt den Marktplatz durchsuchen.

Anschließend kannst du das WindowBuilder Plugin installieren lassen:

→ *Confirm* → Häkchen setzen vor *Accept Licence Agreement* → *Finish* → *Restart Now*

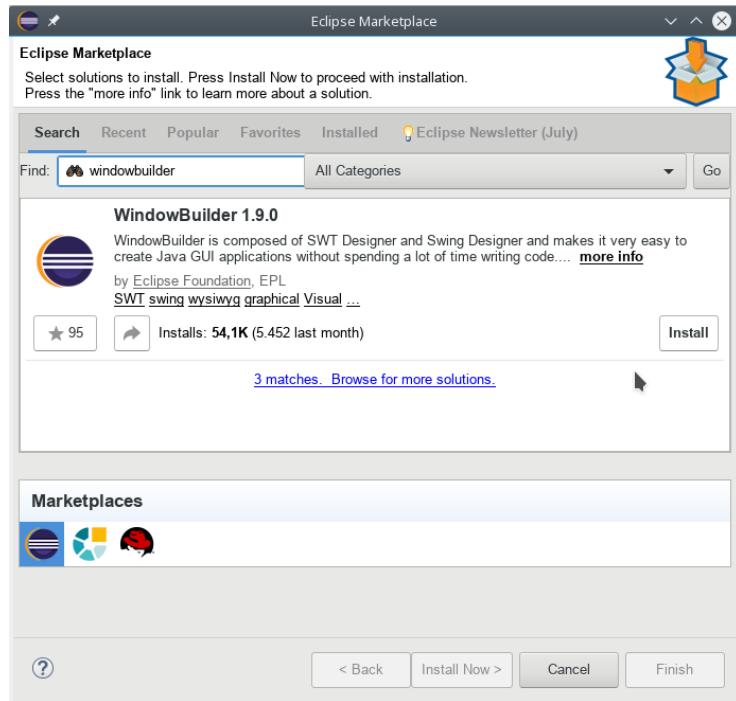


Abbildung 1.12: WindowsBuilder Plugin installieren (falls nötig)

Installation des MySQL-Java-Connectors

Zunächst musst du den Connector herunter laden:

<http://dl.dropbox.com/u/31241540/mysql-connector-java-5.1.43-bin.jar>

(Sobald wir im Unterricht bei SQL angekommen sind, wirst du diese Datei auch in unserem Kurs-Repository finden. Und bis dahin brauchst du dich um die Einbindung des Connectors auch noch nicht zu kümmern.)

Rechtsklick auf dein Projekt → *Import ...* → *General* → *File System* → *Next* → *From directory*: (*Browse* – dort das Verzeichnis wählen, in dem die JAR-Datei des Connectors liegt) → *Next* → im folgenden Dialog die gewünschte Datei auswählen → *Finish*.

Rechtsklick auf deinen Projekt-Ordner im Package-Explorer → *Build Path* → *Configure Build Path ...* → Wähle den Reiter *Libraries* → *Add JARs ...* → wähle mysql-connector-java-5.1.43-bin.jar in deinem Projekt → *OK* → *OK*.

Plugin „SQL-Explorer“ installieren und konfigurieren

Help → *Install New Software ...* → *Add* → bei Name: SQL-Explorer und bei Location:

<http://eclipsesql.sourceforge.net/>

eintragen. Mit *OK* bestätigen. Wenn man den nun sichtbaren Eintrag für *SQL Explorer* auffaltet, sieht man möglicherweise mehrere Versionen des Plugins. Davon die aktuellste auwählen. Mit *Next* bestätigen. Nochmals *Next*. Dann *I accept the terms of the license agreement* → *Finish*. Die im Installationsverlauf erscheinende Warnung bezüglich der Installation nicht-signierter Inhalte mit *OK* quittieren. Ebenso den Hinweis auf den nötigen Neustart von Eclipse.

Anschließend muss noch die Verbindung zum lokalen MySQL-Server konfiguriert werden:

Windows → *Show View* → *Other ...* → *SQL Explorer* → *Connections* → *OK* Im nun verfügbaren Connections-Tab (unten) → Rechtsklick → *New Connection Profile ...* → Name: MySQL localhost → *Add/Edit Drivers* → *SQL Explorer* auffalten → *JDBC Drivers* → Doppelklick auf *MySQL Driver* → *Extra Class Path* → *Add*

JARs . . . → Den Pfad zu mysql-connector-java-5.1.43-bin.jar (oder neuere Version) auf der lokalen Platte (vermutlich in einem Unter-Ordner des workspace-Ordners) auswählen.

Siehe Abbildung 1.13.

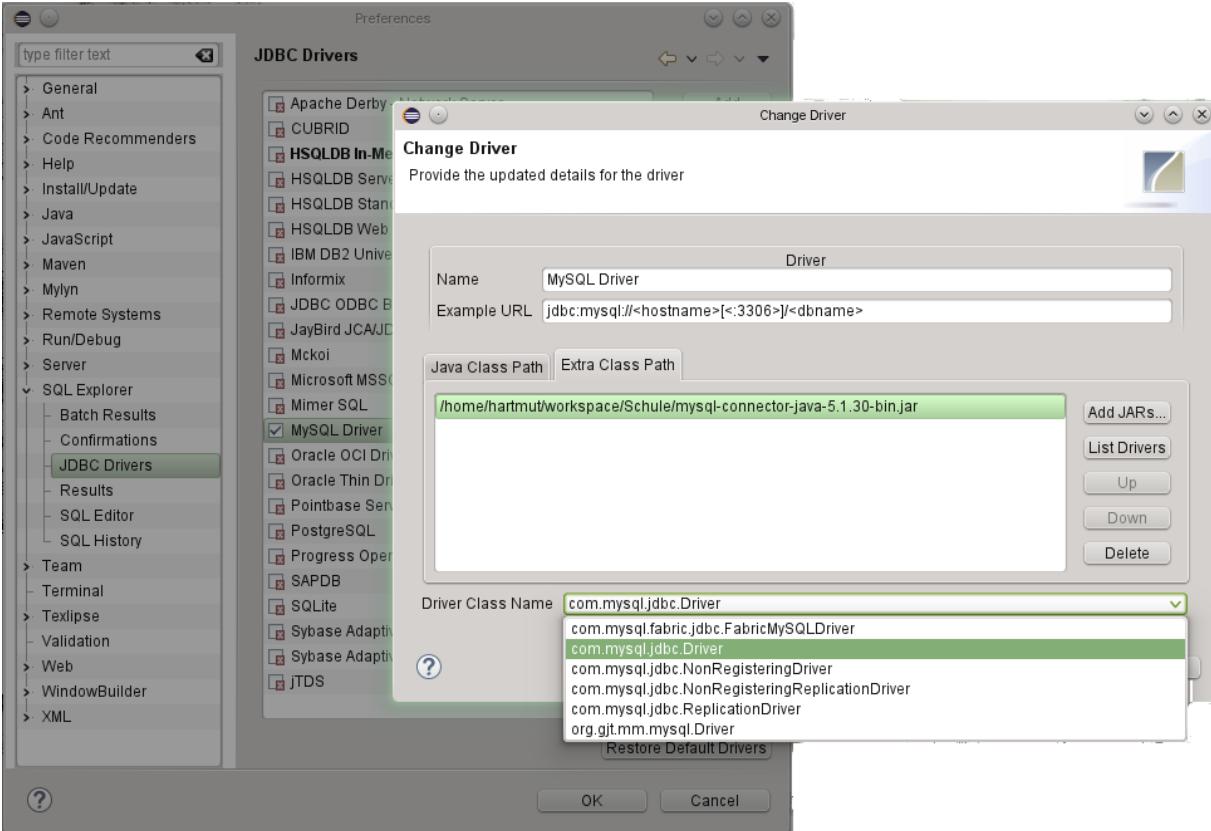


Abbildung 1.13: Auswahl des MySQL-Connectors

→ *List Drivers* → aus der Dropdown-Liste für *Driver Class Name* den Eintrag `com.mysql.jdbc.Driver` auswählen → *OK* → *OK* → *Driver: MySQL Driver* → Häkchen Setzen bei *Auto Logon* und *Auto Commit* → Anpassen der URL auf `jdbc:mysql://localhost:3306/` → *User:* „root“ → *Password:* „root“ → *OK*.

Siehe Abbildung 1.14.

Ab sofort werden SQL-Skripte in Eclipse mit dem SQL-Explorer geöffnet. In diesem kann (einen lokal laufenden MySQL-Server vorausgesetzt) dann einfach eine Verbindung zum lokalen MySQL-Server hergestellt werden. Im Drop-Down-Menü, links von *Limit Rows* in der Menüzeile des SQL-Explorer Fensters kann dazu einfach der Eintrag *MySQL localhost/root* ausgewählt werden.

1.4 Eclipse kennen lernen

Für die ersten Schritte in Eclipse empfele ich dir

http://dl.dropbox.com/u/31241540/JavaBuch_Eclipse.html

Dabei handelt es sich um ein Kapitel aus dem „Handbuch der Java-Programmierung“ von Guido Krüger und Heiko Hansen. Dieses ist als HTML-Version kostenlos unter <http://www.javabuch.de/> zu finden oder als richtiges Buch (Addison-Wesley Verlag) auch zu kaufen.

Außerdem gibt es auf YouTube jede Menge Video-Tutorials zu Java mit Eclipse. Beispielsweise auch eine ganze Serie (von brauchbarer Qualität):

<http://www.youtube.com/playlist?list=PL71C6DFDDF73835C2>

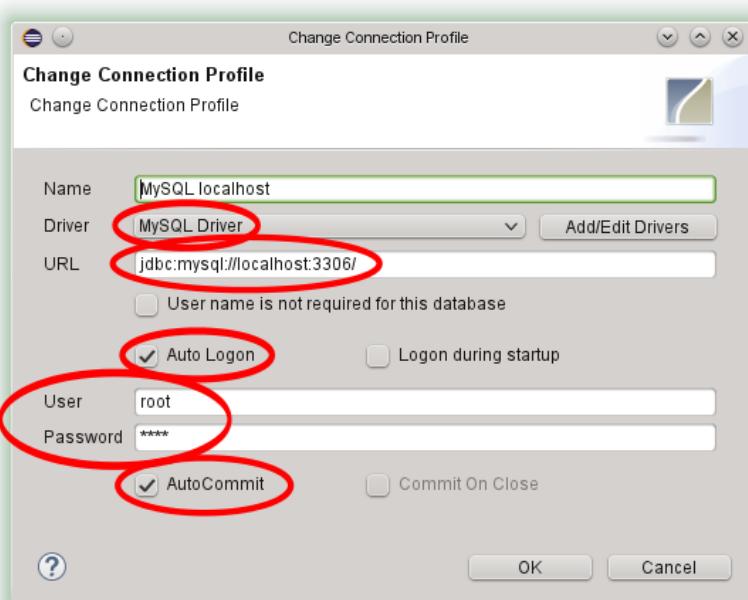


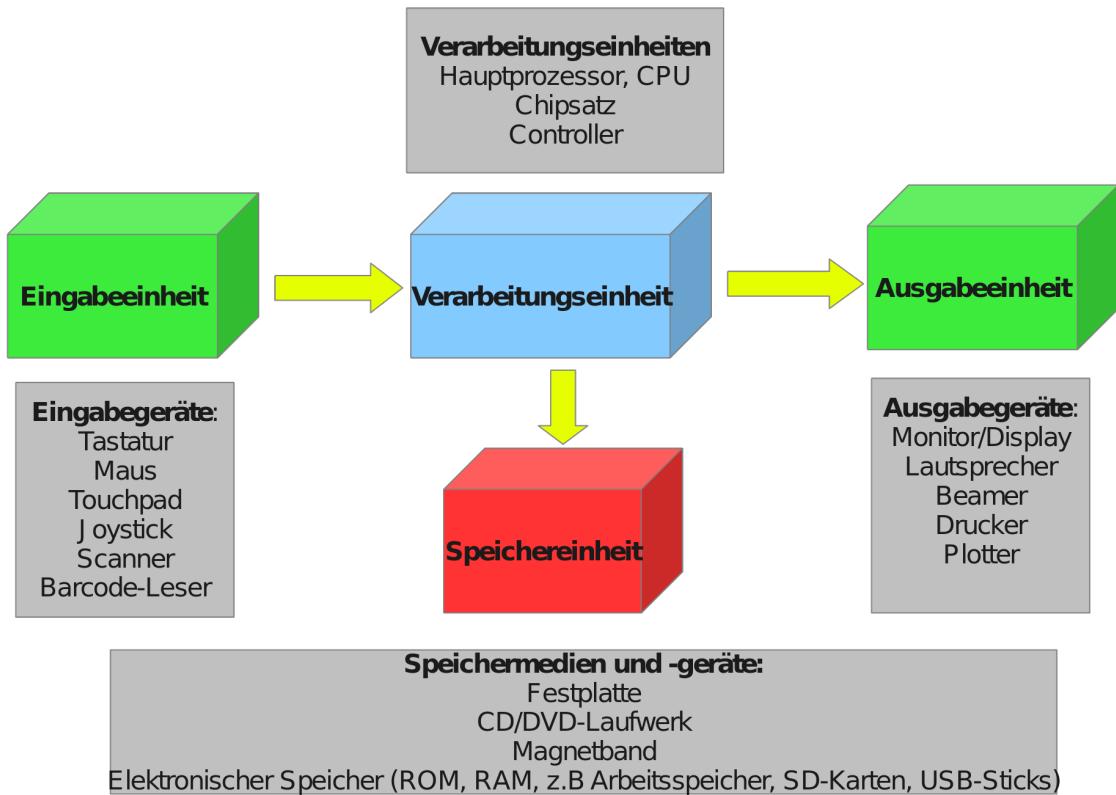
Abbildung 1.14: Einstellungsdialog für die Verbindung zum lokalen MySQL-Server

2 Hardware und Software

2.1 Hardware

Grundstruktur eines Computers

Um von den elektrischen Vorgängen in einem Rechner abstrahieren zu können, ist es zweckmäßig, den Rechner als Datenverarbeiter anzusehen. Die Verarbeitung von Daten funktioniert nach dem sogenannten EVA-Prinzip:



Der Computer besteht vereinfacht aus drei Einheiten:

- Ein-/Ausgabegeräte
- Hauptspeicher (engl. RAM = Random Access Memory)
- Prozessor (engl. CPU = Central Processing Unit)

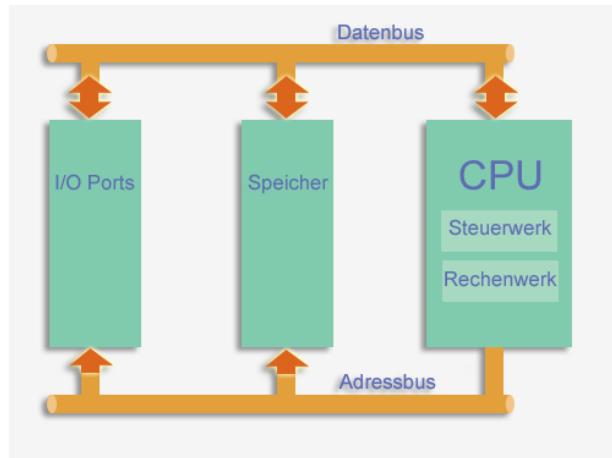
Die Einheiten werden durch Datenleitungen miteinander verbunden, die man als Bus bezeichnet.

Eingabe und Ausgabegeräte

Die Eingabe- und Ausgabegeräte haben die Aufgabe, die zu verarbeitenden Daten von der Außenwelt in den Hauptspeicher zu übertragen, bzw. die Ergebnisse der Verarbeitung wieder in die Außenwelt zu bringen. Solche Geräte werden auch Peripheriegeräte genannt. (Peripherie = Umgebung; Rand)

Hauptspeicher (RAM)

Die Aufgabe des Hauptspeichers ist es, alle aktuell benötigten Daten zu speichern. Dazu gehört auch der Programmcode der momentan laufenden Programme.



Um Daten in den Hauptspeicher zu schreiben bzw. von dort zu lesen, sind die Speicherzellen des Hauptspeichers durchnummieriert. Jede Speicherzelle hat eine eindeutige Nummer, die sogenannte Adresse, mit der man sie ansprechen kann. Bei einem Zugriff auf den Hauptspeicher transportiert der Adressbus die gewünschte Adresse und der Datenbus die Daten.

Prozessor (CPU)

Die CPU ist das „Gehirn“ des Rechners. Sie hat die Aufgabe, die Daten, die im Hauptspeicher liegen, zu verarbeiten, d.h. sie zu lesen, mit ihrer Hilfe Berechnungen durchzuführen und die Berechnungsergebnisse in den Hauptspeicher zu schreiben.

Die CPU lädt die Kommandos eines Programms Schritt für Schritt aus dem Hauptspeicher und führt sie nacheinander aus. Der Hauptspeicher unterscheidet nicht zwischen Daten und Programmen. Erst durch die CPU wird eine Datenfolge im Hauptspeicher zu einem Programm.

Die Leistungsfähigkeit eines Computers ist unter anderem davon abhängig, wie schnell die Befehle vom Prozessor abgearbeitet werden. Der Prozessor ist mit einer Uhr verbunden, die ihn in gewissen Zeitabständen anweist, den nächsten Befehl aus dem Hauptspeicher zu holen und auszuführen. Je schneller dieser Uhrtakt schlägt, desto schneller wird auch der Computer arbeiten. Ein Gigahertz entspricht 1.000.000.000 Stromimpulsen pro Sekunde.

Adress- und Datenbus

Der Datenbus überträgt die eigentlichen Daten zwischen den Geräten, wie z.B. Bilder, Texte, Programmcode usw. Über den Adressbus übermittelt die CPU die Speicheradressen, an denen sich die zu transportierenden Daten befinden. Auf diese Weise steuert die CPU, welche Daten übertragen werden sollen.

Speicherarten

Aufwändige Aufgaben, wie z.B. die Wettervorhersage, beruhen auf der Verarbeitung von Millionen von Messdaten. Durch die immens gesteigerte Verarbeitungsgeschwindigkeit der Computer während der letzten Jahre, können immer komplexere Probleme durch Computer berechnet werden. Die ersten Rechner benötigten einige Sekunden, um eine Addition von zwei ganzen Zahlen durchzuführen, heutige CPUs führen einige hundert Millionen Rechenoperationen pro Sekunde durch.

Der Speicher im Computer muss in der Lage sein, die CPU mit der entsprechenden Geschwindigkeit mit Daten zu versorgen und er muss möglichst viele Daten speichern können. Die Speicherbausteine, die mit der Geschwindigkeit der CPU mithalten können, haben allerdings zwei Nachteile: Sie sind teuer und sie sind flüchtig, d.h. sie können die Daten nur speichern, solange sie mit Strom versorgt werden.

Deshalb gibt es in einem Computer verschiedene Arten von Speichern:

- In der CPU gibt es spezielle Speicherzellen, sogenannte *Register*, die dazu dienen, die Operanden der gerade ausgeführten Anweisung zu halten. Die Register sind Teil des CPU-Kernes, d.h. ebenso schnell wie die CPU, und ändern ihren Inhalt teilweise nach jeder Anweisung. Ein Register speichert ein Computer-„Wort“ (heutzutage je nach CPU 32 bis 128 Bit).
- Der schnelle *Hauptspeicher* (englisch *RAM = Random Access Memory*) speichert die Daten, die für die laufenden Programme benötigt werden. Der Hauptspeicher ist über den Adress- und Datenbus direkt mit der CPU verbunden. Da Hauptspeicher teuer ist, kann dieser Speicher nicht beliebig groß sein.
- Alle anderen Daten (und Programme) werden in sogenannten *Hintergrundspeichern* gehalten (Festplatte, CD-ROM-Laufwerk, USB-Stick, usw.). Hintergrundspeicher sind langsamer als Hauptspeicher, billiger, meist deutlich größer und nicht flüchtig.
- Moderne CPUs sind vom Hauptspeicher durch sogenannte *Caches* entkoppelt. Caches sind schneller (und teurer) als Hauptspeicher und haben die Aufgabe häufig benötigte Bereiche des Hauptspeichers zwischen zu lagern, um so die CPU möglichst unabhängig von der Geschwindigkeit des Hauptspeichers zu machen.

2.2 Software

Im Folgenden wird die Software von PCs betrachtet.

BIOS

Startet man den Rechner („hochfahren“, booten), wird, bevor das Betriebssystem in Aktion tritt, das sogenannte *BIOS* (Basic Input Output System) gestartet. Das BIOS ist ein Ein- und Ausgabesystem, welches unabhängig vom nachfolgend installierten Betriebssystem vom Hersteller mitgeliefert wird.

Das BIOS steht schreibgeschützt in einem Speicherbaustein auf dem Mainboard (ROM-Baustein, engl. für Read Only Memory). Es testet bei jedem Computerstart alle Hardwarekomponenten, startet den Rechner und lädt das Betriebssystem von der Festplatte. Während des laufenden Betriebs kann das BIOS elementare Aufgaben zur Ansteuerung von Hardwarekomponenten wie Tastatur, Maus, Festplatte und Grafikkarte übernehmen und arbeitet damit dem Betriebssystem zu.¹

Betriebssystem

Das Betriebssystem ist eine spezielle Software zur Steuerung der Grundfunktionen eines Computers und zur Verwaltung der Hardware. Es besteht aus einer Reihe von Systemprogrammen (Systemsoftware), die den Umgang mit CPU, Hardwarekomponenten und Anwendungsprogrammen ermöglichen, und ist damit die Schnittstelle zwischen Mensch und Computer sowie zwischen Anwendungsprogrammen und Hardware.

Aufgaben eines Betriebssystems

Das Betriebssystem steuert alle zum Betrieb des Rechners notwendigen Verwaltungsaufgaben. Ein Betriebssystem gehört in der Regel zur Grundausstattung des Computers. Es ist jedoch prinzipiell austauschbar. Die wichtigsten Aufgaben des Betriebssystems sind:

Speicherverwaltung Verwaltung des Hauptspeichers

Dateiverwaltung Organisation der Daten auf Festplatten und anderen Datenträgern. Dazu gehört z.B. die Vorbereitung der Datenträger (Partitionieren von Festplatten, Formatieren und Benennen von Datenträgern), die Erstellung von Verzeichnissen, das Kopieren, Löschen, Verschieben und Umbenennen von Dateien.

¹Moderne Betriebssysteme nutzen diese Funktionalität allerdings nur noch während der ersten Phase des Bootens. Denn dort gibt es ein klassisches Henne-Ei-Problem: Zum Booten muss das Betriebssystem Daten von der Festplatte laden und gegebenenfalls auch auf Tastatureingaben des Benutzers reagieren können. Ebenso sollen Ausgaben auf dem Bildschirm erscheinen. Aber das Betriebssystem läuft ja noch gar nicht ... Anschließend werden diese BIOS-Funktionen dann nicht mehr benutzt.

Programmverwaltung (Fachwort: *Prozessverwaltung*) Ausführung von Anwendungsprogrammen und Aufteilung der Rechenzeit der CPU zwischen allen momentan laufenden Programmen.

Geräteverwaltung Steuerung der Arbeit aller angeschlossenen internen und externen Geräte.

Bestandteile eines Betriebssystems

Betriebssysteme vereinen sogenannte residente und transiente Bestandteile:

Residente Bestandteile werden nach dem Start sofort in den Arbeitsspeicher (RAM) geladen und verbleiben dort während der gesamten Betriebszeit des Computers. Diese Bestandteile ermöglichen die Kommunikation zwischen Nutzer und Rechner sowie das Laden und Starten von Anwendungsprogrammen.

Transiente Bestandteile befinden sich in eigenständigen Dateien, die auf der Festplatte in einem gesonderten Verzeichnis gespeichert sind und nur in den Arbeitsspeicher geladen werden, wenn es notwendig ist.

Grafische Benutzeroberfläche

Die Benutzeroberfläche ist der Teil eines Programms, den der Benutzer auf dem Bildschirm sehen kann. Sie bildet die Schnittstelle zwischen dem Programm und seinem Benutzer, welcher über die Tastatur oder die Maus Dienste anfordern kann.

Die einfachste Version einer Benutzeroberfläche zeigt eine Kommandozeile. Der Benutzer hat die Möglichkeit zeilenweise Text einzugeben und erhält als Antwort vom Programm wieder eine Textausgabe. Moderne Programme und Betriebssysteme haben eine sogenannte grafische Benutzeroberfläche, die dem Benutzer ein Fenstersystem mit vielen bunten Bildern präsentiert.

Die Benutzeroberfläche des Betriebssystems eines PCs ist der sogenannte *Desktop* (deutsche Übersetzung: „Schreibtisch“).

Treiber

Treiber sind Programme zur Ansteuerung einer Software- oder Hardware-Komponente. Ein Treiber ermöglicht einem Anwendungsprogramm die Benutzung einer Komponente, ohne deren detaillierten Aufbau zu kennen.

Zum Beispiel kann ein Textverarbeitungsprogramm nicht im Voraus alle verschiedenen Drucker kennen, die es vielleicht einmal bedienen soll. Wenn der Benutzer aus dem Datei-Menü den Befehl „Drucken“ auswählt, soll der Druck funktionieren, egal welcher Drucker angeschlossen ist. Die Druckbefehle des Textverarbeitungsprogramms (z.B. „Drucke ein kursives ‘a’“, „Neue Zeile anfangen“ oder „Seitenumbruch“) werden deshalb vom Betriebssystem an ein sogenanntes Treiberprogramm (kurz Treiber) weitergeleitet, das vom Hersteller des ausgewählten Druckers geschrieben wurde. Das Treiberprogramm weiß genau, wie der spezielle Drucker angesteuert werden soll und übersetzt die Befehle der Textverarbeitung in geeignete Drucker-Befehle.

Treiber für gängige Geräte sind meist schon im Betriebssystem vorhanden. Wenn aber ein neues Gerät, das dem Betriebssystem noch nicht bekannt ist, an einen Rechner angeschlossen wird, so muss auch ein zugehöriges Treiberprogramm installiert werden.

Anwendungssoftware

Die Art der Aufgaben, die ein Computer erledigen kann, ist nicht von vornherein festgelegt. Für den Computer können beliebig viele Anwendungsprogramme generiert werden. Auf einem PC können mehrere Anwendungsprogramme parallel laufen. Ein ausführbares Programm besteht aus einem für den Rechner zugeschnittenen Maschinencode und kann daher immer nur auf einem bestimmten Betriebssystem laufen. Unter Windows werden ausführbare Programme mit der Endung *.exe gekennzeichnet. „exe“ steht dabei für das englische Wort *executable* (auf deutsch: ausführbar).

Typische Aufgaben für Anwendungsprogramme sind:

Textverarbeitung, Tabellenkalkulation, Buchhaltung, Spiele, Internet Browser, usw.

2.3 Hardware und Software – Übungen

Aufgabe 1: Speicherarten

- a) Was versteht man unter „flüchtigem“ Speicher?
- b) Was ist der Unterschied zwischen dem Hauptspeicher und dem RAM?
- c) Erkläre die unterschiedlichen Aufgaben von Hauptspeicher und Festplatte.
- d) Wozu dient der Cache?
- e) Fülle die folgende Tabelle aus:

| | Register | Cache | RAM | Festplatte | USB-Stick |
|--------------------------------------|----------|-------|-----|------------|-----------|
| flüchtiger Speicher (ja/nein) | | | | | |
| Zugriffsgeschwindigkeit | | | | | |
| übliche SpeichergröÙe | | | | | |

Aufgabe 2: Software

- a) Welcher Teil der Software liegt nicht auf der Festplatte, sondern befindet sich auf einem ROM-Baustein (engl. *Read Only Memory*, schreibgeschützter Speicher) auf der Hauptplatine (Mainboard)?
- b) Wie wird ein Betriebssystem hergestellt?
- c) Nenne die Namen einiger dir bekannter Betriebssysteme.
- d) Nenne die Namen einiger dir bekannter Anwendungsprogramme.
- e) Welche Aufgaben hat ein Betriebssystem?
- f) Ein Benutzer führt mit dem Textverarbeitungsprogramm Word die nachfolgenden Aktionen durch. Gib an, bei welchen der Aktionen neben dem Textverarbeitungsprogramm auch das Betriebssystem oder ein Gerätetreiber in Aktion treten muss.
 1. Starten des Programms Word bzw. LibreOffice Writer.
 2. Öffnen der bereits vorhandenen Datei MeinText.doc.
 3. Veränderung der Schriftart des Textes von „Arial“ nach „Times New Roman“.
 4. Abspeichern des geänderten Textes.
 5. Ausgabe des Textes auf den Drucker.
- g) Begründe, wieso das Betriebssystem in residente und transiente Bestandteile unterteilt wird.
- h) Wie kann man auf einem PC die Einstellungen des BIOS verändern?

Aufgabe 3: Stärken und Schwächen von MS Windows

Die unten stehende Anekdote zeigt die Schwachstellen der heutigen Betriebssysteme auf, die inzwischen meist völlig unübersichtlich und voller Fehler sind.

1997 soll Bill Gates, der damalige Kopf des Microsoft-Konzerns, auf einer großen Computer-Messe (COMDEX) die Computer-Industrie mit der Auto-Industrie verglichen und das folgende Statement abgegeben haben:

If GM had kept up with technology like the computer industry has, we would all be driving \$25 cars that got 1.000 MPG.²

²Das Zitat stammt so nicht tatsächlich von Bill Gates. Vielmer wurde eine echte Aussage von ihm im nachhinein zugespitzt. Das echte Zitat ist: „The PC industry is different than any other industry. The volume, the openness, the innovation, it's really unequaled. In fact, comparisons are often done between this industry and others, and it's just stunning when you look at it. The price of a mid-sized auto, it's about double what it used to be. Cereal, I admit I don't buy that much cereal, but research shows

Als Antwort darauf veröffentlichte General Motors eine Presse-Erklärung mit folgendem Inhalt:

If GM had developed technology like Microsoft, we would all be driving cars with the following characteristics:

1. For no reason at all, your car would crash twice a day.
2. Every time they repainted the lines on the road, you would have to buy a new car.
3. Occasionally, executing a manoeuvre such as a left-turn would cause your car to shut down and refuse to restart, and you would have to reinstall the engine.
4. When your car died on the freeway for no reason, you would just accept this, restart and drive on.
5. Only one person at a time could use the car, unless you bought 'Car95' or 'CarNT', and then added more seats.
6. Apple would make a car powered by the sun, reliable, five times as fast, and twice as easy to drive, but would run on only five per cent of the roads.
7. Oil, water temperature and alternator warning lights would be replaced by a single 'general car default' warning light.
8. New seats would force every-one to have the same size butt.
9. The airbag would say 'Are you sure?' before going off.
10. Occasionally, for no reason, your car would lock you out and refuse to let you in until you simultaneously lifted the door handle, turned the key, and grabbed the radio antenna.
11. GM would require all car buyers to also purchase a deluxe set of road maps from Rand-McNally (a subsidiary of GM), even though they neither need them nor want them. Trying to delete this option would immediately cause the car's performance to diminish by 50 per cent or more. Moreover, GM would become a target for investigation by the Justice Department.
12. Every time GM introduced a new model, car buyers would have to learn how to drive all over again because none of the controls would operate in the same manner as the old car.
13. You would press the 'start' button to shut off the engine.

Aufgabe:

1. Kommentiere die Aussage von Bill Gates.
2. Auf welche Schwachstellen der gängigen Computersysteme spielt General Motors in seiner Presseerklärung an? Was wird mit den 13 Aussagen kritisiert? Welche Kritikpunkte sind berechtigt?

that, too, has doubled in price. And if you take that and say, what would those prices be if it were like the PC industry, the car would cost about \$27, and the cereal would cost about one cent. So, I think there's a lot to be learned by watching how this industry has done what it's done.“

3 Informatik – Worum geht's?

3.1 Informatik

Gegenstand der Informatik

Die meisten Menschen denken bei Informatik sofort an Computer. Nicht ganz zu Unrecht, denn Computer sind als Werkzeug für die Informatik von zentraler Bedeutung.

Dem gegenüber steht ein bekanntes Zitat des niederländischen Informatikers Edsger Wybe Dijkstra:

In der Informatik geht es genauso wenig um Computer wie in der Astronomie um Teleskope.

Auch hier wird auf den Werkzeugcharakter des Computers hingewiesen.

Die Informatik beschäftigt sich als Wissenschaft mit der automatisierten Informationsverarbeitung. Um Informationen automatisiert verarbeiten zu können braucht es zum einen entsprechende Maschinen (Computer) und zum anderen genaue und eindeutige Handlungsanweisungen, mit deren Hilfe die Daten der jeweiligen Aufgabenstellung entsprechend verarbeitet werden können.

Eine solche eindeutige Handlungsanweisung zur Lösung eines Problems wird *Algorithmus* genannt. Dies ist ein zentraler Begriff in der Informatik.

Teilgebiete der Informatik

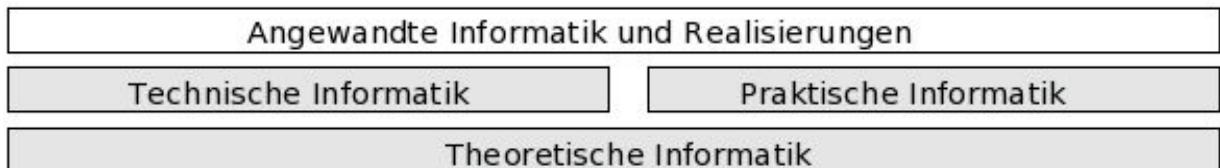


Abbildung 3.1: Teilgebiete der Informatik

Im Informatikunterricht werden wir uns schwerpunktmäßig mit Themen der *Praktischen Informatik* beschäftigen (mit anderen Worten: Ihr werdet lernen passende Algorithmen für gegebene Probleme zu finden und diese zu implementieren/programmieren).

Aber auch Themen aus den anderen Teilgebieten der Informatik werden wir behandeln:

Theoretische Informatik:

- Aussagenlogik
- Kryptologie

Technische Informatik:

- Rechnerarchitektur
- Aufgaben und Funktionsweise von Betriebssystemen
- Netzwerkprotokolle

Angewandte Informatik:

- Datenschutz und Urheberrecht
- Informatik und Gesellschaft
- Anwendungsgebiete in Wirtschaft und Wissenschaft

3.2 Was Computer können

Computer können nur zwei Dinge tun:

1. Berechnungen ausführen.
2. Sich die Ergebnisse solcher Berechnungen merken.

Das ist tatsächlich nicht viel. Diese zwei Dinge können Computer allerdings *sehr gut!* Ein typischer PC führt pro Sekunde eine Milliarde Rechenoperationen aus. Selbst in der unvorstellbar kurzen Zeit, die Licht braucht um von der Schreibtischlampe auf eine nur 60 Zentimeter entfernte Schreibtischoberfläche zu gelangen, hat der PC bereits zwei Berechnungen durchgeführt. Und dabei ist Licht wirklich verdammt schnell: Pro Sekunde legt Licht eine Entfernung von dreihunderttausend Kilometern zurück. Mit dieser Geschwindigkeit kann man die Erde in einer Sekunde acht Mal umkreisen!

Während des überwiegenden Teils der Menschheitsgeschichte waren die Möglichkeiten, Berechnungen anzustellen und deren Ergebnisse festzuhalten, durch die Leistungsfähigkeit des menschlichen Gehirns beschränkt. Das bedeutete, dass nur die kleinsten und einfachsten durch Berechnung zu lösenden Probleme angegangen werden konnten. Selbst mit der Geschwindigkeit moderner Computer bleiben noch unlösbare Probleme. Zum Beispiel die langfristigen Wetter- und Klimavorhersagen. Aber mit der zunehmenden Leistungsfähigkeit der Computer werden mehr und mehr Problemstellungen einer Lösung mit Hilfe von Computern zugänglich.

Zwar kann der Computer nur mit Zahlen rechnen (genau genommen nur mit Null und Eins), aber auch Buchstaben, Ton, Bild, Wetterdaten – ganz Allgemein: *Informationen* – lassen sich als Zahlen darstellen. Und somit auch berechnen.

3.3 Computational Thinking

Computational Thinking ist ein relativ neuer Begriff in der Informatik. Beschrieben wird damit die Fähigkeit, auch alltägliche Problemstellungen daraufhin zu untersuchen, ob und wie man sie mit Hilfe von Computern lösen könnte.

Populär wurde der Begriff durch Jeannette Wing, Leiterin des Computer Science Departments an der Carnegie Mellon University in Pittsburgh (USA).

Zitat aus einem Artikel in der österreichischen Tageszeitung *Der Standard* vom 28. September 2010:

Wer analytisch und präzise wie ein Computerwissenschaftler denken lernt, kann komplexe Probleme effizienter lösen.

Mit dem Denkmodell könnten selbst alltägliche Aufgaben wie das Kaffeeholen in der Cafeteria effizienter gestaltet werden. Den Unterschied zu gewissermaßen „herkömmlichem“ logischem Denken erklärt Wing mit dem herausragenden Stellenwert von Effizienz in der Computerwissenschaft: „Computational Thinking befähigt dazu, Strukturen und Muster zu erkennen, wo man vorher keine gesehen hat, oder effizienter in der täglichen Routine zu sein.“

Besonderes Augenmerk legt die Wissenschaftlerin auf die Implementierung ihres Konzeptes in das Bildungswesen. „Zusätzlich zu den Fertigkeiten wie Schreiben, Rechnen und Lesen sollte in Zukunft jedes Kind als weitere analytische Fähigkeit die grundlegenden Konzepte der Computerwissenschaften verinnerlichen.“

„Wenn wir lernen, schriftlich zu dividieren oder zwei große Zahlen zu multiplizieren, lernen wir eigentlich einen Algorithmus. Nur sagt uns das niemand“, so Wing.

Quelle: <http://derstandard.at/1285199497847/>

3.4 Erste Übung – erster Teil

Für die folgende Übung braucht ihr nur Blatt und Stift. Wer will darf zusätzlich auch einen Taschenrechner benutzen.

Gegeben ist das folgende Problem: Die Quadratwurzel einer beliebigen Zahl soll möglichst genau bestimmt werden *ohne(!)* die Wurzelfunktion des Taschenrechners zu benutzen.

Setzt euch in Zweier- oder Dreiergruppen zusammen und überlegt, wie sich das Problem lösen lässt. Gesucht ist eine Schritt-für-Schritt Anleitung (mit anderen Worten: ein *Algorithmus*), wie man die Quadratwurzel einer beliebigen Zahl finden kann. Dabei spielt es zunächst keine Rolle, wie kurz bzw. effizient der Lösungsweg ist. Für diese Aufgabe habt ihr 15 Minuten Zeit.

Schreibt eure Lösung auf, so dass ihr sie anschließend den anderen vorstellen könnt.

3.5 Variablen

Das Computer Berechnungen wie

```
2 + 5
12 * 7
```

ausführen können, wird euch nicht überraschen. Aber wie sollen die Ergebnisse abgespeichert werden. Oder wie könnte man solche Rechenvorschriften verallgemeinern (beispielsweise nicht $2 + 5$, sondern allgemein die Summe von zwei Zahlen)?

Dazu werden sogenannte *Variablen* benutzt. Variablen sind Speicherorte im „Gedächtnis“ des Computers, die wir als Programmierer über Namen ansprechen können.

Damit können wir etwa schreiben

```
a = 2
b = 5
summe = a + b
PRINT summe
```

3.6 Kontrollstrukturen

Wenn Probleme so beschrieben werden sollen, dass sie als Handlungsanweisung Schritt für Schritt zu einem vom Computer zu berechnenden Ergebnis führen, dann braucht man neben der einfachen Anweisung (etwa: Verdopple eine gegebene Zahl und merke dir das Ergebnis) typischerweise auch sogenannte *Kontrollstrukturen*.

Vermutlich habt ihr solche Kontrollstrukturen bereits bei der Lösung der ersten Aufgabe benutzt.

Etwa um in Abhängigkeit vom vorangegangen Ergebnis zu entscheiden, wie es weiter gehen soll, oder um einen (oder auch mehrere) Schritte der Berechnung wiederholt ausführen zu lassen.

Obwohl es viele verschiedene Programmiersprachen gibt, teilen sich doch die meisten von Ihnen die gleichen Kontrollstrukturen. Was ihr hier in der Schule lernt wird euch in der Zukunft deshalb auch dann von direktem Nutzen sein, wenn ihr dann möglicherweise mit ganzen anderen Programmiersprachen arbeiten müsst oder wollt.

Bedingte Anweisung und Verzweigung

Bedingte Anweisung werden im Unterschied zu normalen Anweisungen nur dann ausgeführt, wenn eine vorgenannte Bedingung erfüllt ist:

```
FALLS (Ampelfarbe ist grün):
  Die Kreuzung überqueren.
```

Nur wenn die Bedingung erfüllt ist, wird der folgende Anweisungsblock (eine oder auch mehrere Anweisungen, deren Ausführung durch die Bedingung kontrolliert wird) ausgeführt.

Die Bedingte Anweisung wird zur *Verzweigung*, wenn es für den Fall, dass die Bedingung nicht erfüllt ist, einen alternativen Anweisungsblock gibt:

```
FALLS (Ampelfarbe ist grün):
  Die Kreuzung überqueren.
SONST:
  Anforderungsknopf drücken.
  An der Ampel warten.
```

In fast allen Programmiersprachen wird dies mit dem Schlüsselwort *if* gekennzeichnet.

Wiederholungsanweisungen (Schleifen)

Als weitere Kontrollstruktur gibt es die *Wiederholungsanweisung*, die oft auch als *Schleife* bezeichnet wird. Auch hier kontrolliert eine Bedingung die Ausführung eines Anweisungsblockes. Im Unterschied zur Bedingten Anweisung, wird der folgende Anweisungsblock jedoch nicht maximal einmal ausgeführt, sondern beliebig oft.

Wir können mit Hilfe einer Schleife die Handlungsanweisung für das Überqueren einer Fußgängerampel vervollständigen:

FALLS (Ampelfarbe ist grün):

 Die Kreuzung überqueren.

SONST:

 Anforderungsknopf drücken.

 WIEDERHOLE SOLANGE (Ampel nicht grün):

 An der Ampel warten.

 Die Kreuzung überqueren.

3.7 Erste Übung – zweiter Teil

Noch einmal zu unserem Problem aus der ersten Übung: Die Quadratwurzel einer beliebigen Zahl soll möglichst genau bestimmt werden *ohne(!)* die Wurzelfunktion des Taschenrechners zu benutzen.

Setzt euch nochmal in Zweier- oder Dreiergruppen zusammen.

Versucht diesmal euren Algorithmus mit Hilfe von Variablen und Kontrollstrukturen zu beschreiben. Dabei dürft ihr den Algorithmus im Vergleich zum ersten Übungsteil auch noch verändern.

Ihr habt wieder 15 Minuten Zeit.

3.8 Eine mögliche Lösung: Brute Force

Es gibt viele mögliche Algorithmen, um die Quadratwurzel einer beliebigen positiven Zahl zu bestimmen.

Ein ganz einfacher Ansatz basiert auf dem Wissen, dass die Lösung irgendwo im Bereich von 0,0 und Unendlich liegen muss. Folglich könnte man bei Null beginnend und mit einer hinreichend kleinen Schrittweite alle möglichen Lösungen durchprobieren (je kleiner wir die Schrittweite wählen, desto genauer wird unser Ergebnis sein).

Das Programm als sogenannter *Pseudocode*:

```

radikand = benutzereingabe()
gerateneLösung = 0.0
quadrat = gerateneLösung * gerateneLösung
schrittweite = 0.0001

WIEDERHOLE SOLANGE (quadrat < radikand):
    gerateneLösung = gerateneLösung + schrittweite
    quadrat = gerateneLösung * gerateneLösung

PRINT gerateneLösung

```

Der gleiche Algorithmus als sogenanntes *Flussdiagramm* (englischer Fachausdruck *Flowchart*):

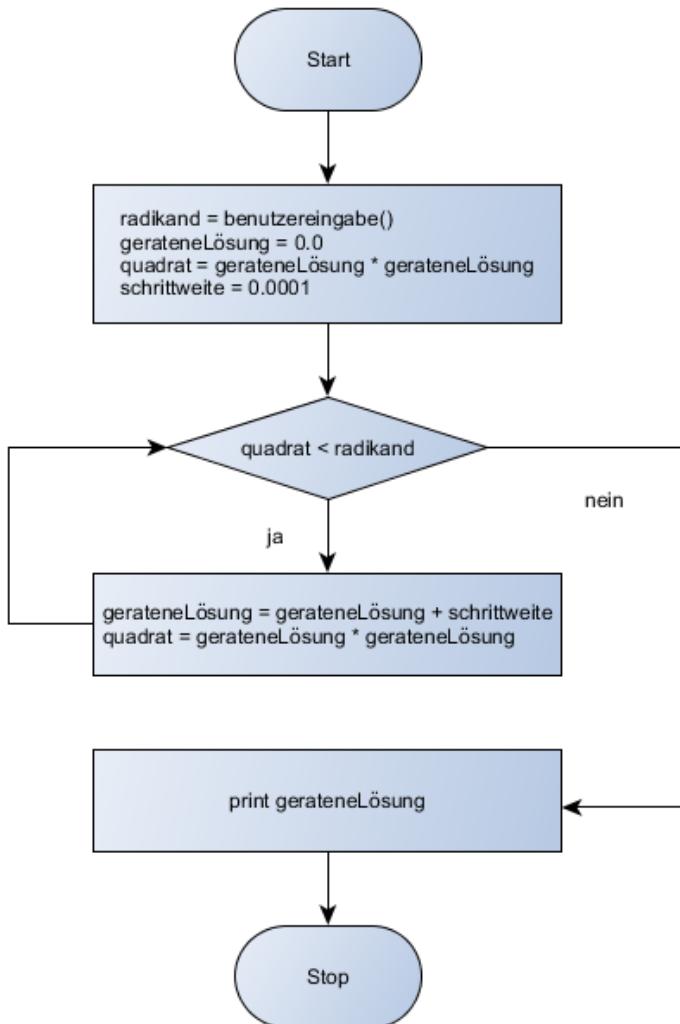


Abbildung 3.2: Flussdiagramm: Ermittlung der Quadratwurzel mit dem Brute-Force-Verfahren

Dieser Algorithmus ist gut in dem Sinne, dass der Lösungsweg direkt einleuchtend ist. Außerdem ist er sehr kurz.

Alles andere als kurz (und gut) ist allerdings der nötige Aufwand, der so zu betreiben ist: Selbst um die Quadratwurzel von 100 zu bestimmen, sind bei der oben gewählten Schrittweite bereits 100000 (einhunderttausend) Wiederholungen der Schleife nötig, um das Ergebnis zu finden.

3.9 Eine bessere mögliche Lösung: Bisektion

Statt – wie im ersten Lösungsansatz – stumpf alle Möglichkeiten durchzuprobieren (solch eine Lösungsstrategie wird oft als *Exhaustive Enumeration* oder auch *Brute Force* bezeichnet), gibt es oft intelligenteren Ansätze, durch die der Rechenaufwand erheblich reduziert werden kann.

Bisektion ist solch ein Ansatz. Er wird oft verwendet, wenn eine Lösung in einer großen, aber geordneten Lösungsmenge gesucht wird. Ein Zahlenstrahl – etwa die Menge der Reellen Zahlen – ist solch eine geordnete Lösungsmenge. Geordnet in dem Sinne, dass alle Zahlen links von einer gegebenen Zahl kleiner sind als diese und alle Zahlen rechts von ihr größer.

Wenn wir zu Beginn eine Ober- und eine Untergrenze für die mögliche Lösung kennen, dann wissen wir in welchem Bereich der Lösungsmenge wir nach unserer richtigen Lösung suchen müssen. Suchen wir etwa die Quadratwurzel einer positiven Zahl, die größer als 1 ist, so wissen wir, dass die Lösung irgendwo im Bereich zwischen 0 und der gewählten positiven Zahl (dem Radikand) liegen muss.

Das Bisektions-Verfahren teilt nun einfach immer wieder die Lösungsmenge in zwei gleich große Teile und testet, ob sich die Lösung in der oberen oder unteren Hälfte befindet. Und wiederholt diese Teilung (mit den jeweils neu gefundenen Ober- und Untergrenzen) solange, bis die Lösung hinreichend genau bestimmt wurde.

In unserem Beispiel aus dem Brute-Force-Ansatz (oben), wurde die Quadratwurzel von 100 gesucht.

Mit dem Bisektionsansatz würden wir zu Beginn als untere Grenze unserer Lösungsmenge 0 und als obere Grenze 100 setzen. Diese Lösungsmenge teilen wir nun in zwei gleich große Hälften und müssen entscheiden, in welcher der beiden Hälften die richtige Lösung liegt. Um dies zu entscheiden prüfen wir, ob

$$\left(\frac{\text{untere Grenze} + \text{obere Grenze}}{2} \right)^2 = \left(\frac{0 + 100}{2} \right)^2 = 50^2 = 2500$$

größer oder kleiner als unser Radikand (100) ist. Der Ausdruck ist größer. Also wissen, dass wir in der unteren Hälfte der ursprünglichen Lösungsmenge nach der richtigen Lösung suchen müssen und die obere Hälfte komplett ignorieren können. Wir haben also in einem Schritt die Lösungsmenge halbiert!

Die neue obere Grenze ist nun 50. Die untere Grenze ist unverändert geblieben.

Wieder teilen wir unsere Lösungsmenge in zwei gleich große Teile. Und wieder überprüfen wir, ob

$$\left(\frac{\text{untere Grenze} + \text{obere Grenze}}{2} \right)^2 = \left(\frac{0 + 50}{2} \right)^2 = 25^2 = 625$$

größer oder kleiner als der gegebene Radikand ist.

Und so weiter, und so weiter. Bereits nach 22 Schritten, haben wir die Lösung auf mindestens vier Nachkommastellen genau bestimmt! Noch mal zum Vergleich: Mit der Brute-Force-Methode waren es einhunderttausend Schritte!

Hier das passende Programm als Pseudocode:

```
radikand = benutzereingabe()
akzeptierterFehler = 0.0001
untereGrenze = 0.0
obereGrenze = MAXIMUM(1, radikand)
gerateneLösung = (untereGrenze + obereGrenze) / 2
quadrat = gerateneLösung * gerateneLösung
abweichung = BETRAG(quadrat - radikand)

WIEDERHOLE SOLANGE (abweichung > akzeptierterFehler):
    FALLS (quadrat > radikand):
        obereGrenze = gerateneLösung
    SONST:
        untereGrenze = gerateneLösung
        gerateneLösung = (untereGrenze + obereGrenze) / 2
        quadrat = gerateneLösung * gerateneLösung
        abweichung = BETRAG(quadrat - radikand)

PRINT gerateneLösung
```

Wer genau hinschaut sieht, dass dieses Programm noch den Fall berücksichtigt, dass der Radikand kleiner als 1 ist. Dann liegt die gesuchte Lösung nicht im Bereich zwischen 0 und dem gegebenen Radikand, sondern zwischen 0 und 1!

Abbildung 3.3 (nächste Seite) zeigt das passende Flussdiagramm.

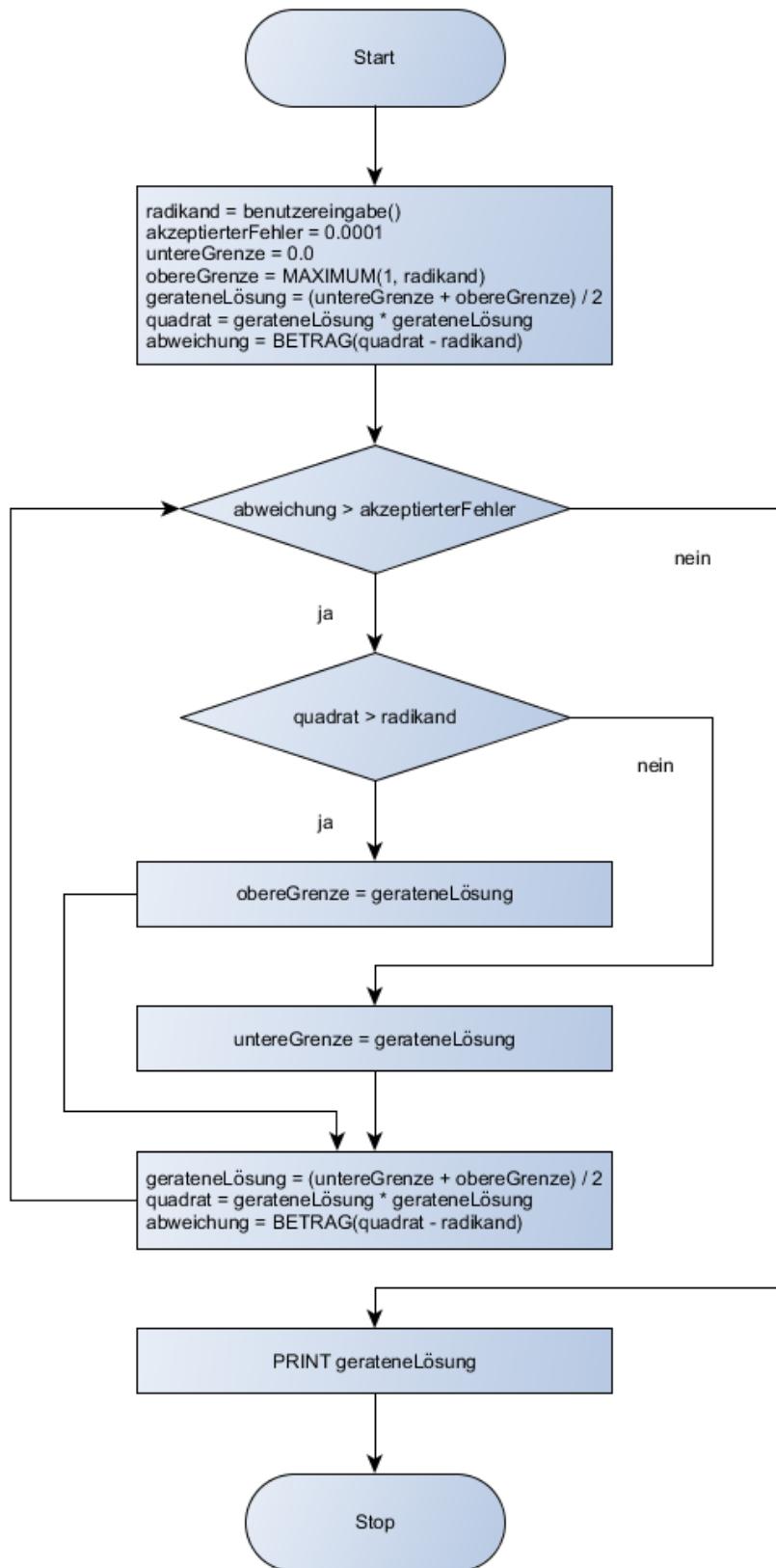


Abbildung 3.3: Flussdiagramm: Ermittlung der Quadratwurzel mit dem Bisektionsverfahren

3.10 Implementierung in echten Programmiersprachen

Auf der nächsten Doppelseite seht ihr, wie der bisher in Pseudocode und als Flussdiagramm beschriebene Algorithmus tatsächlich – also mit einer „echten“ Programmiersprache – implementiert werden kann.

Es gibt viele verschiedene Programmiersprachen. Und natürlich gibt es Unterschiede zwischen ihnen. Die Kontrollstrukturen, wie bedingte Anweisungen, Verzweigungen und Schleifen, haben aber die allermeisten Programmiersprachen gemein.

Beispielhaft wurde hier die Implementierung einmal in Python und einmal in Java gewählt. Bei beiden handelt es sich um imperative, objektorientierte Programmiersprachen. Im Unterschied zu Java muss man in Python aber den Datentyp von Variablen nicht vorab bestimmen (Java gehört zu den *stark typisierenden* Sprachen, Python nicht).

Außerdem unterscheiden sich verschiedene Programmiersprachen durch die vorgefertigten Funktionen, etwa zur Ein- und Ausgabe von Text, oder auch für mathematische Funktionen wie zur Bestimmung des Maximums mehrerer gegebener Werte oder auch des Betrages einer Zahl.

Manche Programmiersprachen sind für spezielle Aufgabenzwecke konzipiert (etwa die Programmiersprache R, die oft für statistische Auswertungen benutzt wird), andere sind sogenannte *General Purpose* Programmiersprachen. Sowohl Java als auch Python gehören zu diesem allgemeinen Typ.

Welche Programmiersprache man bevorzugt bzw. einsetzt hängt zum einem von persönlichen Vorlieben ab, zum anderen von möglichen Vorgaben. So wie etwa bei uns im Informatik-Unterricht: Wir werden im weiteren Verlauf mit Java arbeiten.

Implementierung in Python

```
radikand = float(raw_input('Gib eine positive Zahl ein: '))
akzeptierterFehler = 0.0001
anzahlSchritte = 0
untereGrenze = 0.0
obereGrenze = max(1.0, radikand)
gerateneLoesung = (untereGrenze + obereGrenze) / 2.0
quadrat = gerateneLoesung * gerateneLoesung
abweichung = abs(quadrat - radikand)

while abweichung > akzeptierterFehler:
    anzahlSchritte += 1
    if quadrat < radikand:
        untereGrenze = gerateneLoesung
    else:
        obereGrenze = gerateneLoesung
    gerateneLoesung = (obereGrenze + untereGrenze) / 2.0
    quadrat = gerateneLoesung * gerateneLoesung
    abweichung = abs(quadrat - radikand)

print 'Anzahl Schritte =', anzahlSchritte
print gerateneLoesung, 'ist die Quadratwurzel von', radikand
```

Implementierung in Java

```
import java.util.Scanner;

public class Qudratwurzel {

    public static void main(String[] args) {
        double radikand;
        double akzeptierterFehler = 0.0001;
        int anzahlSchritte = 0;
        double untereGrenze = 0.0;
        double obereGrenze;
        double gerateneLösung;
        double quadrat;
        double abweichung;

        Scanner scanner = new Scanner(System.in);
        System.out.print("Gib eine positive Zahl ein: ");
        radikand = scanner.nextDouble();           // Eingabe mit Komma statt Punkt!
        obereGrenze = Math.max(1.0, radikand);
        gerateneLösung = (untereGrenze + obereGrenze) / 2.0;
        quadrat = gerateneLösung * gerateneLösung;
        abweichung = Math.abs(quadrat - radikand);

        while (abweichung > akzeptierterFehler) {
            anzahlSchritte++;
            if (quadrat < radikand) {           // Die geratene Wurzel ist zu klein
                untereGrenze = gerateneLösung;
            } else {                           // Die geratene Wurzel ist zu groß
                obereGrenze = gerateneLösung;
            }
            gerateneLösung = (obereGrenze + untereGrenze) / 2.0;
            quadrat = gerateneLösung * gerateneLösung;
            abweichung = Math.abs(quadrat - radikand);
        }

        System.out.println("Anzahl Schritte = " + anzahlSchritte);
        System.out.println(gerateneLösung + " ist die Quadratwurzel von " + radikand);
    }
}
```

3.11 Wer es genauer wissen will

<http://de.wikipedia.org/wiki/Pseudocode>

[http://de.wikipedia.org/wiki/Programmablaufplan \(Flussdiagramme\)](http://de.wikipedia.org/wiki/Programmablaufplan_(Flussdiagramme))

<http://de.wikipedia.org/wiki/Brute-Force-Methode>

<http://de.wikipedia.org/wiki/Bisektion>

<http://de.wikipedia.org/wiki/Programmiersprache>

4 Java Grundlagen

4.1 Wichtige Regeln

- Es wird zwischen Groß- und Kleinschreibung unterschieden!
- Alle Anweisungen müssen mit einem Semikolon beendet werden!
- Mehrere Anweisungen können zu einem Anweisungsblock zusammengefasst werden. Der Beginn eines Anweisungsblocks wird durch { und das Ende durch } gekennzeichnet.
- Prozeduren (tun etwas, liefern aber kein Ergebnis) und Funktionen (tun etwas und geben ein Ergebnis als Rückgabewert zurück) werden in Java als *Methoden* bezeichnet. Methoden erkennt man immer an den runden Klammern nach dem Namen. In diesen Klammern steht entweder nichts oder die Parameter, die von der Methode verarbeitet werden.
Beispiele: `close()`, `compare(a, b)`
- In Java ist es üblich, die Namen von Variablen und Methoden mit einem kleinen Anfangsbuchstaben zu beginnen (auch wenn der Deutschlehrer das nicht gut findet, der sieht euren Code nicht!).
Beispiele: `auto`, `zeichneAuto()`
- In Java ist jedes Programm eine *Klasse*. Was das genau bedeutet, werden wir uns später ansehen. Alle Variablen und Methoden müssen innerhalb der geschweiften Klammern der Klasse stehen. Der Name der Datei muss immer genau so heißen wie der Name der Klasse. Namen von Klassen beginnen üblicherweise mit einem Großbuchstaben, z.B. `Auto`, `MeinProgramm`.

4.2 Variablen

Deklaration und Initialisierung

Die *Deklaration* (Vereinbarung) einer Variablen hat die Form:

`Typname Variablenname;`

Beispiel:

```
int i, j;
double zahl;
```

Bei der Deklaration darf die Variable auch gleichzeitig *initialisiert* werden. Das bedeutet, dass sie einen Anfangswert erhält. Beispiel:

```
int i = 45, j = 3;
double zahl = 0.0;
```

Lebensdauer

Die Lebensdauer einer lokalen Variablen, die innerhalb einer Methode deklariert wird, erstreckt sich von der Stelle ihrer Deklaration bis zum Ende der Methode. Falls eine Variable innerhalb eines geklammerten Blocks deklariert wird, lebt die Variable nur bis zum Ende des Blocks.

Globale Variablen, die außerhalb einer Methode stehen, gehören immer zu einer Klasse und müssen innerhalb der Klasse stehen.

Konstanten

Einer Variable, deren Wert nicht verändert werden soll, wird das Schlüsselwort **final** vorangestellt. Damit wird die Variable zu einer *Konstanten*. Die Namen von Konstanten werden üblicherweise komplett groß geschrieben. Beispiel:

```
final float PI = 3.14;
final String NAME = "Otto Mustermann";
```

4.3 Kommentare

```
int i, j; // Kommentar bis zum Zeilenende

/* Kommentar über
 * mehrere Zeilen
 */
```

4.4 Namenskonventionen

Zur besseren Lesbarkeit des Codes gibt es folgende Konventionen:

| Art | Namensgebung | Schreibweise | Beispiel |
|------------------|----------------------|--|---------------------|
| Klasse | Substantiv (Einzahl) | CamelCase mit großem Anfangsbuchstaben | String, Auto |
| Variable, Objekt | | camelCase mit kleinem Anfangsbuchstaben | meinAuto |
| Methode | Verb | camelCase mit kleinem Anfangsbuchstaben | zeichneAuto() |
| Konstante | | komplett mit Großbuchstaben verbunden mit Unterstrichen | WIDTH MIN_HEIGHT |

Alle Namen müssen immer mit einem Buchstaben beginnen und dürfen keine Leerzeichen enthalten.

Außerdem gilt als Grundregel: Bezeichner sollten möglichst kurz und „sprechend“ (selbsterklärend) sein. Ausgenommen von dieser Regel sind Variablen, die nur kurzzeitig (etwa als Zähler innerhalb einer Schleife) benutzt werden. Solche „Wegwerfvariablen“ werden oft wie folgt benannt:

i, j, k, m, und n für Ganzahl-, x, y, und z für Fließkomma- und schließlich c, d, und e für Character-Variablen.

Siehe auch <http://www.oracle.com/technetwork/java/codeconventions-135099.html>

4.5 Datentypen

Folgende Datentypen stehen zur Verfügung:

| Datentyp | Bedeutung | Anzahl Bytes | Wertebereich | Standardwert |
|----------|-------------------------|------------------------------|---|---------------|
| boolean | Wahrheitswerte | 1 | true, false | false |
| char | Buchstaben | 2 (Fußnote ¹) | Alle Unicode-Zeichen | \u0000 |
| byte | ein Byte große Zahlen | 1 | -128 ... 127 | 0 |
| short | kleine ganze Zahlen | 2 | $-2^7 \dots 2^7 - 1$ (-32.768...32.767) | 0 |
| int | ganze Zahlen | 4 | $-2^{31} \dots 2^{31} - 1$ | 0 |
| long | große ganze Zahlen | 8 | $-2^{63} \dots 2^{63} - 1$ | 0 |
| float | kleine Fließkommazahlen | 4 | $\pm 3.40282347 \cdot 10^{38}$ (Fußnote ²) | 0.0 |
| double | große Fließkommazahlen | 8 | $\pm 1.79769313486231570 \cdot 10^{308}$ | 0.0 |
| String | Zeichenketten | | | leerer String |

¹Einige seltene Unicode-Zeichen werden als eine Folge von zwei char-Objekten verpackt.

Siehe <http://docs.oracle.com/javase/tutorial/i18n/text/unicode.html>

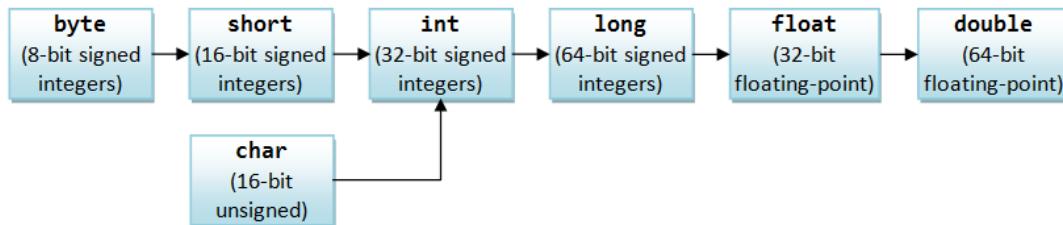
²Tatsächlich ist der Wertebereich von float und double im negativen sogar noch größer als im positiven.

Siehe http://de.wikibooks.org/wiki/Java_Standard:_Primitive_Datentypen

Neben den Standard-Datentypen kann man sich eigene Datentypen definieren, in dem man eine eigene Klasse schreibt. Der Typ **String** ist eine vordefinierte Klasse aus dem Package `java.lang`.

4.6 Typkonvertierungen

Zwischen den primitiven Datentypen besteht die folgende Beziehung:



Die Konvertierung eines „kleineren“ in einen „größeren“ Datentyp (also von links nach rechts im Schaubild) führt das System automatisch durch. Eine Konvertierung in umgekehrter Richtung kann einen Datenverlust bedeuten und wird deshalb nur auf explizite Anweisung des Programmierers vorgenommen. Beispiel:

```

int i = 56;
double d = 24.5;
double d2 = i;           // geht automatisch
int i2 = (int) d;        // explizite Typkonvertierung, vom Programmierer erzwungen
char c = (char) i;       // explizite Typkonvertierung
  
```

Das Schema einer expliziten Typkonvertierung lautet:

```
Variable_vom_Typ1 = (Typ1) Variable_vom_Typ2;
```

Andere Konvertierungen (z.B. zwischen `int` und `String`) können nur mit Hilfe spezieller Methoden durchgeführt werden. Zahlenvariablen werden jedoch automatisch in einen `String` umgewandelt, wenn man sie mit einem '+' an einen `String` anhängt. Beispiel:

```
String text = "Der Inhalt von i ist: " + i;
```

4.7 Literale

Ganzzahlige Zahlenwerte (z.B. 304) sind grundsätzlich vom Typ `int`. Nur wenn der Suffix `L` hinten angehängt wird (z.B. 304L) sind sie vom Typ `long`.

Zahlenwerte von Fließkommazahlen enthalten im Gegensatz zu ganzen Zahlen einen Punkt zur Trennung der Nachkommastelle (z.B. ist 0 eine ganze Zahl aber 0.0 ist eine Fließkommazahl). Standardmäßig sind Fließkommazahlen vom Typ `double`. Wenn man eine Fließkommazahl vom Typ `float` haben möchte, muss man das Suffix `f` oder `F` an die Zahl anhängen (z.B. 0.0F).

Für Fließkommazahlen sind einige symbolische Zahlenwerte verfügbar:

| Name | Bedeutung |
|-------------------|--|
| MAX_VALUE | Größter darstellbarer positiver Wert |
| MIN_VALUE | Kleinster darstellbarer positiver Wert |
| NaN | Not-A-Number |
| NEGATIVE_INFINITY | Negativ unendlich |
| POSITIVE_INFINITY | Positiv unendlich |

Buchstaben (char) werden grundsätzlich in einfache Hochkommata gesetzt.

Zeichenketten (String) werden mit doppelten Hochkommata geschrieben. Zur Darstellung von Sonderzeichen gibt es eine Reihe von Standard-Escape-Sequenzen:

| Zeichen | Bedeutung |
|---------|--|
| \b | Rückschritt (Backspace) |
| \t | Horizontaler Tabulator |
| \n | Zeilenschaltung (Newline) |
| \f | Seitenumbruch (Formfeed) |
| \r | Wagenrücklauf (Carriage return) |
| \" | Doppeltes Anführungszeichen |
| \' | Einfaches Anführungszeichen |
| \\\ | Backslash |
| \nnn | Oktalzahl nnn (kann auch kürzer als 3 Zeichen sein, darf nicht größer als oktal 377 sein) |
| \uxxxx | Unicode-Escape-Sequenz. xxxx steht für eine Folge von 4 hexadezimalen Ziffern. Z.B. ist \u0020 die Hexadezimaldarstellung des Leerzeichens. |

Beispiele:

```
char c = 'a', b = '\t';
String text = "Guten Tag.\nWie geht es dir?";
```

4.8 Operatoren

Arithmetische Operatoren

| Operator | Bezeichnung | Bedeutung |
|----------|-------------------|---|
| + | Addition | a + b ergibt die Summe von a und b |
| - | Subtraktion | a - b ergibt die Differenz von a und b |
| * | Multiplikation | a * b ergibt das Produkt von a und b |
| / | Division | a / b ergibt den Quotienten von a und b |
| % | Modulo (Restwert) | a % b ergibt den Rest der ganzzahligen Division von a und b Lässt sich in Java auch auf Fließkommazahlen anwenden. |
| ++ | Präinkrement | x = ++a; erhöht a um 1 und schreibt das Ergebnis (a+1) in x |
| ++ | Postinkrement | x = a++; schreibt a in x und erhöht a anschließend um 1 |
| -- | Prädekrement | x = --a; verringert a um 1 und schreibt das Ergebnis (a-1) in x |
| -- | Postdekrement | x = a--; schreibt a in x und verringert a anschließend um 1 |

Vergleichs-Operatoren

| Operator | Bezeichnung | Bedeutung |
|----------|----------------|--|
| == | gleich | a == b ergibt true, wenn a gleich b ist. Sind a und b Referenztypen, so ist der Rückgabewert true, wenn beide Werte auf dasselbe Objekt zeigen. |
| != | ungleich | a != b ergibt true, wenn a ungleich b ist. Sind a und b Objekte, so ist der Rückgabewert true, wenn beide Werte auf unterschiedliche Objekte zeigen. |
| < | kleiner | a < b ergibt true, wenn a kleiner b ist. |
| <= | kleiner gleich | a <= b ergibt true, wenn a kleiner oder gleich b ist. |
| > | größer | a > b ergibt true, wenn a größer b ist. |
| >= | größer gleich | a >= b ergibt true, wenn a größer oder gleich b ist. |

Logische Operatoren

| Operator | Bezeichnung | Bedeutung |
|-------------------------|------------------------------------|---|
| ! | logisches NICHT | $!a$ ergibt <code>false</code> , wenn a wahr ist, und <code>true</code> , wenn a falsch ist. |
| <code>&&</code> | UND mit Short-Circuit-Evaluation | $a \&\& b$ ergibt <code>true</code> , wenn sowohl a als auch b wahr sind. Ist a bereits falsch, so wird <code>false</code> zurückgegeben und b nicht mehr ausgewertet. |
| <code> </code> | ODER mit Short-Circuit-Evaluation | $a b$ ergibt <code>true</code> , wenn mindestens einer der beiden Ausdrücke a oder b wahr ist. Ist bereits a wahr, so wird <code>true</code> zurückgegeben und b nicht mehr ausgewertet. |
| <code>&</code> | UND ohne Short-Circuit-Evaluation | $a \& b$ ergibt <code>true</code> , wenn sowohl a als auch b wahr sind. Beide Teilausdrücke werden ausgewertet. |
| <code> </code> | ODER ohne Short-Circuit-Evaluation | $a b$ ergibt <code>true</code> , wenn mindestens einer der beiden Ausdrücke a oder b wahr ist. Beide Teilausdrücke werden ausgewertet. |
| <code>^</code> | Exklusiv-ODER | $a ^ b$ ergibt <code>true</code> , wenn beide Ausdrücke einen unterschiedlichen Wahrheitswert haben. |

Zuweisungsoperatoren

| Operator | Bezeichnung | Bedeutung |
|-----------------|--------------------------|---|
| <code>=</code> | einfache Zuweisung | $a = b$; weist a den Wert von b zu und liefert b als Rückgabewert. |
| <code>+=</code> | Additionszuweisung | $a += b$; ist eine Abkürzung für $a = a + b$; |
| <code>-=</code> | Subtraktionszuweisung | $a -= b$; ist eine Abkürzung für $a = a - b$; |
| <code>*=</code> | Multiplikationszuweisung | $a *= b$; ist eine Abkürzung für $a = a * b$; |
| <code>/=</code> | Divisionszuweisung | $a /= b$; ist eine Abkürzung für $a = a / b$; |
| <code>%=</code> | Modulozuweisung | $a \%= b$; ist eine Abkürzung für $a = a \% b$; |

4.9 Kontrollstrukturen

Einfache Verzweigung

```
if (Bedingung) {
    Anweisungen;
}
else {
    Anweisungen;
}
```

Der `else`-Teil kann auch entfallen. Die Klammern dürfen weggelassen werden, wenn es nur eine einzige Anweisungszeile gibt.

Mehrfach-Verzweigung

```
switch (Ausdruck) {
case Wert1:
    Anweisungen1;
    break;           // Achtung: ohne break; wird auch Anweisungen2 ausgeführt!
case Wert2:
    Anweisungen2;
    break;
...
default:          // falls kein anderer Wert passt (optional)
    Anweisungen3;
}
```

Beispiel:

```
switch (ampelfarbe) {
    case "rot":
    case "gelb":
        warten();
        break;
    case "grün":
        gehen();
        break;
    default:
        System.out.println("Das ist aber eine komische Ampel!");
}
```

Schleifen

a) `while (Bedingung) {
 Anweisungen;
}`

Beispiel:

```
int zaehler = 0;
while (zaehler<10) {
    System.out.println("Der Zähler hat den Wert: " + zaehler);
    zaehler++; // Kurzform für: zaehler = zaehler + 1;
}
```

Die Anweisungen in den geschweiften Klammern werden ausgeführt, solange die Bedingung `true` ergibt.

b) `do {
 Anweisungen;
} while (Bedingung);`

Die Anweisungen in den geschweiften Klammern werden ausgeführt, solange die Bedingung `true` ergibt (mindestens aber ein mal).

c) `for (Vorab; Bedingung; Wert-Verändern) {
 Anweisungen;
}`

Der Teil `Vorab` wird nur einmal ausgeführt, bevor die Schleife betreten wird. Hier fügt man den Code zur Initialisierung der Variable ein. Die Schleife wird ausgeführt, solang die `Bedingung` wahr ist. Nach jedem Schleifendurchgang wird der Code `Wert-Verändern` einmal ausgeführt. Das obige Beispiel für die `while`-Schleife kann man mit der `for`-Schleife folgendermaßen schreiben:

```
for (int zaehler = 0; zaehler < 10; zaehler++) {
    System.out.println ("Der Zähler hat den Wert: " + zaehler);
}
```

4.10 Methoden

Aufgaben die mehrere Anweisungen umfassen und die immer wieder, auch an verschiedenen Stellen des Programms ausgeführt werden sollen, kann man in *Methoden* „verpacken“:

Methoden müssen immer innerhalb einer Klasse stehen. Eine Methode hat folgendes Schema:

```
Sichtbarkeit Rückgabewert Name(Typ1 Parameter1, Typ2 Parameter2, ...){  
    Anweisungen;  
    return Wert; // Rückgabe eines berechneten Wertes. Kann entfallen.  
}
```

Die *Sichtbarkeit* regelt, wer das Recht hat, die Methode aufzurufen. Der Wert `public` gibt z.B. an, dass die Methode öffentlich ist und jeder darauf zugreifen kann.

Der *Rückgabewert* gibt an, welchen Typ die Werte haben, die die Methode zurück liefert. Wenn die Methode keinen Wert zurück liefert, schreibt man `void`. Wenn die Methode einen Wert zurück gibt, gibt man an dieser Stelle den Typ des Werts an.

Standard-Datentypen (`int`, `boolean`, usw.) können in Java nur als Werte-Parameter übergeben werden, d.h. es wird eine Kopie einer Variablen übergeben. Objekt-Variablen sind dagegen immer Referenz-Parameter, d.h. die Speicheradresse der Objekte wird übergeben.

Beispiele:

```
public void schreibeText() {  
    System.out.println ("Hallo");           // schreibt Text auf die Konsole  
}  
  
public int mult(int zahl1, int zahl2) {  
    int ergebnis = zahl1 * zahl2;  
    return ergebnis;  
}
```

Der Aufruf der obigen Methoden von einer anderen Methode innerhalb derselben Klasse kann z.B. folgendermaßen aussehen:

```
schreibeText();                      // die Klammern dürfen nicht fehlen!  
int zahl = mult(5, 4);
```


5 Erste Übungen mit Java: Turtle-Grafik

5.1 Turtle-Grafik

Zum Einstieg in die Programmierung mit der Sprache Java werden wir mit einer sogenannten Turtle-Grafik geometrische Figuren zeichnen. Bei der Turtle-Grafik stellt man sich vor, dass ein Roboter (die Schildkröte, engl. „turtle“) einen Stift auf einer Zeichenfläche hin und her bewegt.

Kopiere dir die Dateien `Turtle.java` und `TurtleGrafik.java` aus dem Kurs-Repository in dein eigenes Repository.

Erzeuge dazu zunächst ein neues *Package* unterhalb des `src` Ordners in deinem eigenen Projekt (Rechtsklick auf `src` → `src` → *Package*). Als Name für das Package bietet sich `turtle` an. Package-Namen werden üblicherweise komplett aus Kleinbuchstaben gebildet.

Die Datei `TurtleGrafik.java` enthält bereits ein halb fertiges Java-Programm mit einer Reihe von Schaltflächen (engl. Buttons). Du musst noch den Code einfügen, der ausgeführt wird, wenn man auf die Buttons drückt. Nur der Button *Löschen* ist schon fertig programmiert.

Kommandos der Turtle

Vorwärts oder rückwärts gehen

```
t.vor(pixel);
```

Bewegt die Turtle um die angegebene Anzahl von Pixeln vorwärts. Wenn ein negativer Wert übergeben wird, geht die Turtle rückwärts.

Drehen

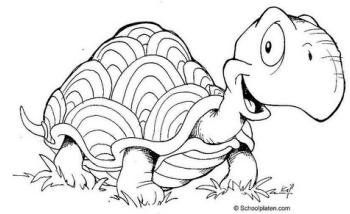
```
t.drehen(winkel);
```

Dreht die Turtle um den angegebenen Winkel gegen den Uhrzeigersinn. Wenn ein negativer Wert übergeben wird, dreht sich die Turtle im Uhrzeigersinn.

Beispiel

```
t.vor(50);      // 50 Pixel vorwärts gehen
t.drehen(-90); // um 90 Grad im Uhrzeigersinn drehen
t.vor(100);
t.drehen(-90);
t.vor(50);
```

Was für eine Figur wird gezeichnet?

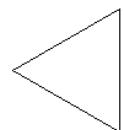


5.2 Programmieraufgaben zur Turtle-Grafik

Aufgabe 1: Gleichseitiges Dreieck

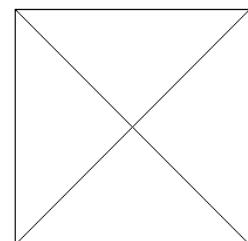
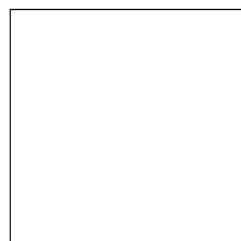
Zeichne ein gleichseitiges Dreieck.

Überlege dir dazu zunächst, wie groß die Innenwinkel in einem gleichseitigen Dreieck sein müssen. Die Turtle muss sich allerdings um den Außenwinkel drehen. Wie groß ist der?



Aufgabe 2: Quadrat

- a) Zeichne ein Quadrat mit einer Seitenlänge von 200 Pixeln. Programmiere dazu eine kleine while-Schleife, in der du viermal hintereinander um 200 Pixel gehst und die Turtle anschließend um 90° Grad gegen den Uhrzeigersinn drehst.
- b) Erweitere das Quadrat so, dass du zusätzlich noch die Diagonalen einzeichnest. Nach Teil (a) blickt die Turtle wieder nach oben. Wenn du sie jetzt 45° Grad gegen den Uhrzeigersinn drehst, hat sie die richtige Blickrichtung für die erste Diagonale. Die Länge der Diagonale kannst du mit dem Satz des Pythagoras berechnen.



Aufgabe 3: Sechseck

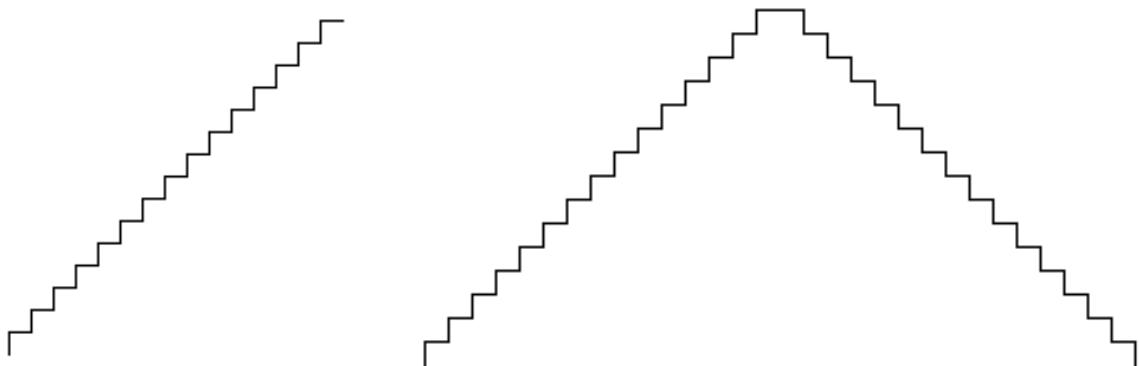
Bewege die Turtle mit einer while-Schleife 6-mal hintereinander um 100 Pixel vorwärts und drehe sie dann um 60° gegen den Uhrzeigersinn. Es entsteht ein Sechseck.

Aufgabe 4: Achteck

Bewege die Turtle mit einer for-Schleife 8-mal hintereinander um 100 Pixel vorwärts und drehe sie dann um 45° gegen den Uhrzeigersinn. Es entsteht ein Achteck.

Aufgabe 5: Treppe

- a) Zeichne in einer Schleife die links abgebildete Figur, die aus 15 Treppenstufen besteht. Die Höhe und Breite jeder einzelnen Stufe beträgt 10 Pixel.
- b) Hänge an die Figur eine zweite Treppe an, die spiegelverkehrt nach unten führt.



Aufgabe 6: Spirale

Zeichne eine Spirale. Legt dazu eine lokale Variable laenge an und setzt sie am Anfang auf 15. Führe nun in einer Schleife 64 mal hintereinander die folgenden Schritte aus:

- Bewege die Turtle um die laenge nach vorne.
- Drehe die Turtle um 90° Grad gegen den Uhrzeigersinn.
- Verlängere die Variable laenge um $1/15$ ihrer aktuellen Länge.

Aufgabe 7: Verschachtelte Quadrate

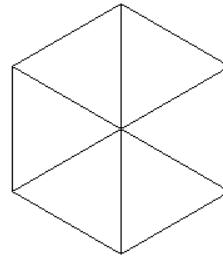
- Programmiere eine Methode `quadrat()`, die ein Quadrat einer beliebigen Länge zeichnen kann. Die Methode erhält die Seitenlänge als Parameter übergeben. Sie zeichnet in einer Schleife 4-mal hintereinander eine Linie der angegebenen Länge und dreht sich anschließen um 90° im Uhrzeigersinn.
- Zeichne in einer Schleife 22 ineinander liegende Quadrate, in dem du jeweils die Methode `quadrat()` aufrufst. Die Seitenlänge der Quadrate soll immer erhöht werden. Das erste Quadrat erhält die Seitenlänge 4. Bei jedem nachfolgenden Quadrat wird die Seitenlänge um $1/4$ ihrer aktuellen Länge erhöht. Es entsteht eine hübsche Figur.

Aufgabe 8: Achtzehn Sechsecke

- Programmiere eine parameterlose Methode `sechseck()`, die ein Sechseck mit einer Seitenlänge von 100 Pixeln malt. Du kannst dazu deine Lösung von Aufgabe 3 wieder verwenden.
- Rufe in einer Schleife 18-mal hintereinander die Methode `sechseck()` auf und drehe dich anschließend jeweils um 20° gegen den Uhrzeigersinn.

Aufgabe 9: Sechseck aus Dreiecken

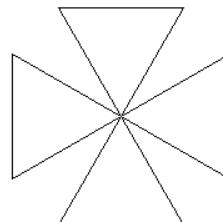
- Programmiere eine parameterlose Methode `dreieck()`, die ein Dreieck mit einer Seitenlänge von 100 Pixeln malt. Du kannst dazu deine Lösung von Aufgabe 1 wieder verwenden.
- Rufe in einer Schleife 6-mal hintereinander die Methode `dreieck()` auf und drehe dich anschließend auf geeignete Weise, so dass die abgebildete Figur entsteht.



Aufgabe 10: Figur aus vier Dreiecken

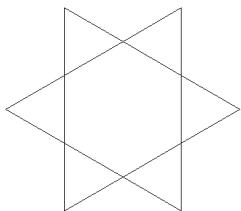
Rufe in einer Schleife 4-mal hintereinander die Methode `dreieck()` auf und drehe dich anschließend auf geeignete Weise, so dass die abgebildete Figur entsteht.

Beachte, dass du die Turtle vor dem Beginn der Schleife in eine geeignete Ausgangsposition drehen musst.



Aufgabe 11: Stern aus Dreiecken

Zeichne den abgebildeten Stern, in dem du in einer Schleife die Methode dreieck() 6-mal hintereinander aufrufst und dich anschließend geeignet drehst.



6 Rekursion

6.1 Rekursions-Begriff

Rekursion bedeutet, dass auf etwas Bekanntes zurückgegriffen wird. In der Programmierung tritt eine Rekursion auf, wenn eine Funktion in ihrem Code auf sich selber zurückgreift.

Blick in den Spiegel

Wenn man den Bildschirminhalt filmt (Screen-Capturing) und das Ergebnis am Bildschirm betrachtet, dann erhält man einen rekursiven Effekt:

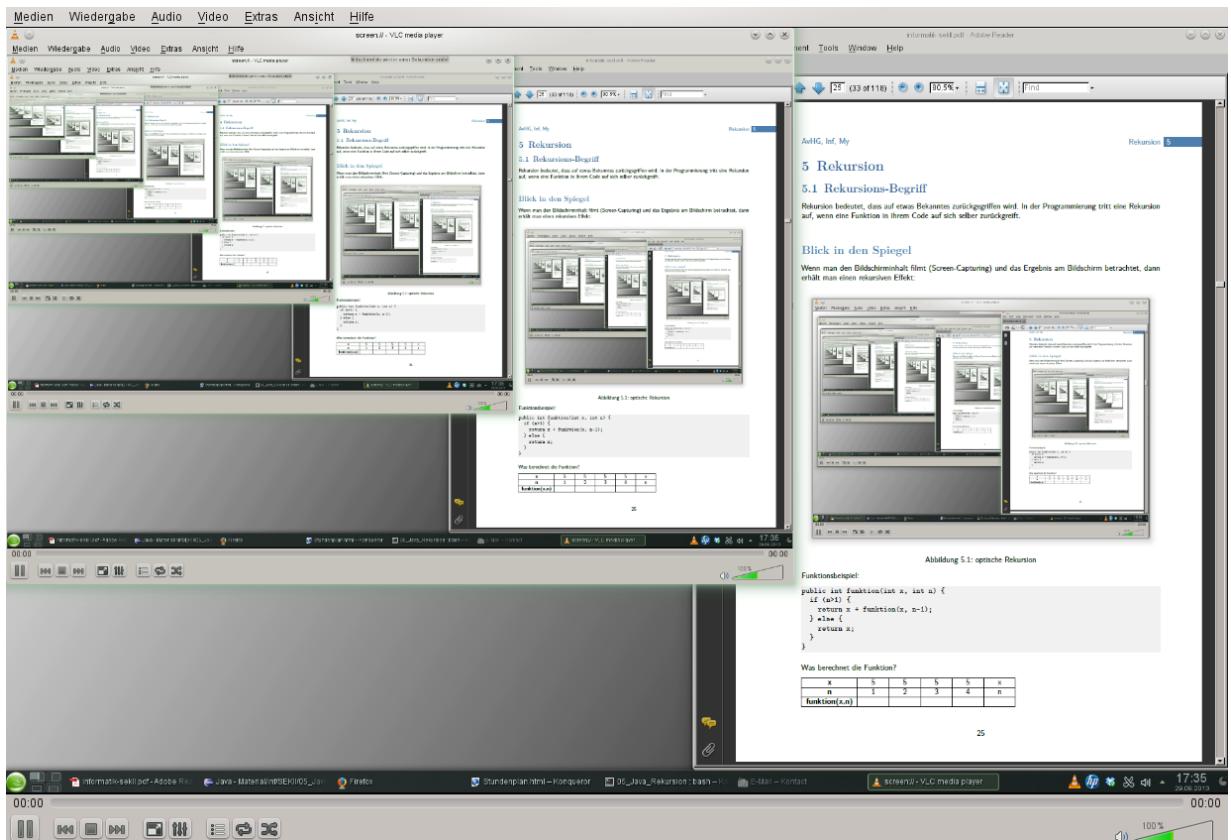


Abbildung 6.1: optische Rekursion

Funktionsbeispiel:

```
public int funktion(int x, int n) {
    if (n>1) {
        return x + funktion(x, n-1);
    } else {
        return x;
    }
}
```

Was berechnet die Funktion?

| | | | | | |
|---------------|---|---|---|---|---|
| x | 5 | 5 | 5 | 5 | x |
| n | 1 | 2 | 3 | 4 | n |
| funktion(x,n) | | | | | |

Rekursiv Programmieren

Nicht alle Probleme lassen sich rekursiv lösen. Aber es gibt eine Reihe von Problemen, die sich entweder nur rekursiv lösen lassen, oder die sich zumindest mit Hilfe eines rekursiven Ansatzes sehr viel eleganter lösen lassen als ohne.

Wenn du den rekursiven Ansatz wählst, musst du folgende beiden Fragen beantworten:

1. Für welchen Parameter-Wert ist die Lösung bekannt? Am Beispiel der Fakultät: Die Fakultät von 0 ist definiert als 1.
2. Wie lässt sich das Ergebnis für alle anderen Parameter-Werte berechnen, wenn man annimmt, dass das Ergebnis für den um 1 verringerten Parameter-Wert bereits bekannt ist?

Um die Fakultät einer gegebenen positiven ganzen Zahl zu berechnen ruft die rekursiv programmierte Methode `fakultaet()` sich immer wieder mit einem jeweils um 1 verringerten Parameter-Wert auf. Dabei warten die zuerst aufgerufenen Methoden `fakultaet()` solange auf die Ergebnisse ihrer „Kinder“ bis der Parameter-Wert auf diese Weise den Wert 0 erreicht hat. Sobald das passiert ist ein konkretes Ergebnis bekannt. Zunächst nur das Ergebnis des „jüngsten Kindes“, dann des zweitjüngsten usw. Bis schließlich die zuerst aufgerufene `fakultaet()`-Methode von seinem Kind das Ergebnis liefert bekommt und damit in die Lage versetzt wird, das endgültige Ergebnis zu berechnen.

Schau dir dazu noch mal das Programm-Beispiel auf der vorherigen Seite an.

6.2 Rekursion: Programmieraufgaben

Erzeuge in deinem eigenen Projekt ein neues Java-Package mit dem Namen `rekursion`. Kopiere anschließend die Datei `Rekursion1.java` aus dem Kurs-Repository in das gerade angelegte Package in deinem Projekt und bearbeite die Aufgaben dort.

Aufgabe 1: Berechnung der Fakultät

In der Wahrscheinlichkeitsrechnung benutzt man sehr häufig die Fakultätsfunktion, die rekursiv definiert ist:

$$0! = 1 \quad (0! \text{ wird „0 Fakultät“ gesprochen})$$

$$n! = n \cdot (n - 1)!$$

Programmiere eine Java-Methode `fakultaet()`, die `n` als Parameter erhält und die Fakultät von `n` zurück gibt (beides sollen ganze Zahlen sein). Kommentiere anschließend in der Datei den Aufruf der Methode unterhalb des Kommentars „// Aufgabe 1“ aus und teste die Methode.

Aufgabe 2: Zweier-Potenzen

Die Zahlenfolge 1, 2, 4, 8, 16, ... lässt sich folgendermaßen rekursiv definieren (bitte selber überlegen und eintragen):

$$\text{zahlenfolge}(0) =$$

$$\text{zahlenfolge}(n) =$$

| | | | | | |
|-----------------------|---|---|---|---|----|
| n | 0 | 1 | 2 | 3 | 4 |
| zahlenfolge(n) | 1 | 2 | 4 | 8 | 16 |

Programmiere eine Java-Methode `zahlenfolge()`, die `n` als Parameter erhält und das Ergebnis der Folge für `n` zurück gibt (beides sollen ganze Zahlen sein). Kommentiere anschließend in der Datei den Aufruf der Methode unterhalb des Kommentars „// Aufgabe 2“ aus und teste die Methode.

Aufgabe 3: Papierformate

Hast du dich schon einmal gefragt, warum wir eigentlich meist auf A4-Blättern schreiben? Papier könnte ja irgendein beliebiges Format haben. Es könnte quadratisch sein, es könnte 20 x 25 cm groß oder 10 x 30 cm groß sein. Aber nein, unser Papier (auch das, welches du gerade liest), ist eben meist DIN A4, also 21,0 x 29,7 cm groß.

Früher war das nicht so. Jede Person, die Papier herstellte, schnitt es irgendwie zu: gerade so, wie es ihr passte. Diese Tatsache bereitete vielen Leuten Mühe, etwa den Sekretärinnen, deren Briefpapiere nie in irgendwelchen Couverts Platz fanden.

Einige praktische Köpfe nahmen sich deshalb vor, ein für allemal aufzuräumen mit diesem Papier-Durcheinander. Man erfand die DIN-Norm 472 (DIN = Deutsche Industrie Norm). Darin wurde festgelegt, in welchen Größen die Papierbögen herzustellen und zu gebrauchen sind. Zuerst wurde das größte Papierformat festgelegt. Man nannte es A0 und definierte folgendes:

Ein A0 Papierbogen ist 1 m^2 groß und sein Seitenverhältnis beträgt $1 : \sqrt{2}$. Dies führt zu einer Länge von 118,9 cm und einer Breite von 84,1 cm.

Die weiteren Formate wurden wie folgt festgelegt:

Das Papierformat A1 erhält man, indem man ein A0 Blatt so faltet, dass seine Breitseiten aufeinander zu liegen kommen.

Auf dieselbe Weise – durch wiederholtes Falten – erhält man die weiteren Papierformate der DIN-A-Reihe.

Für alle Formate außer A0 gilt also: Die Höhe ist genau so groß wie die Breite des nächstgrößeren Formats und die Breite entspricht der halben Höhe des nächstgrößeren Formats.

Programmiere eine Java-Methode `breite()`, die die Breite eines Papierformats als Fließkommazahl zurück gibt. Programmiere eine Java-Methode `hoehe()`, die die Höhe eines Papierformats als Fließkommazahl zurück gibt.

Beide Methoden sollen die Nummer des Papierformats (0 bis 6) als Parameter erhalten.

Kommentiere anschließend in der Datei die Aufrufe der Methoden unterhalb des Kommentars „// Aufgabe 3“ aus. Teste die Methoden und fülle die Tabelle aus:

| Format | A0 | A1 | A2 | A3 | A4 | A5 | A6 |
|--------------|-------|----|----|----|------|----|----|
| Breite in cm | 84,1 | | | | 21,0 | | |
| Höhe in cm | 118,9 | | | | 29,7 | | |

6.3 Rekursion mit Turtle-Grafik

Figuren, bei denen im Kleinen die gleiche Struktur wie im Großen auftritt, nennt man *Fraktale*.

Für die folgenden Aufgaben kopierst du dir die Datei Rekursion2.java aus dem Kurs-Repository in dein eigenes Projekt (und dort in das bereits existierende Package `rekursion`).

Aufgabe 1: Binärer Baum

Programmiere eine Methode

```
public void baum(double laenge, int stufen)
```

die einen Binären Baum mit der angegebenen Anzahl Stufen zeichnet. Der Aufruf der Methode mit der im Textfeld eingegebenen Stufenzahl braucht im Code nur noch auskommentiert werden. Der Parameter `laenge` gibt die Länge des untersten Astes an. Von jedem Ast gehen zwei kleinere Äste im Winkel von $+45^\circ$ und -45° ab. Die Länge der Ästen beträgt $6/10$ der aktuellen Astlänge. In den nachfolgenden Abbildungen siehst du wie ein Baum rekursiv entsteht:

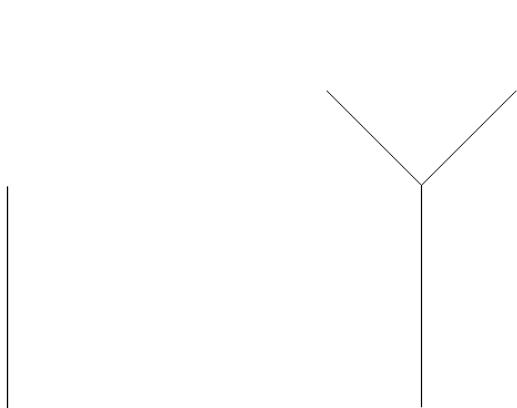


Abbildung 6.2: Stufe 1

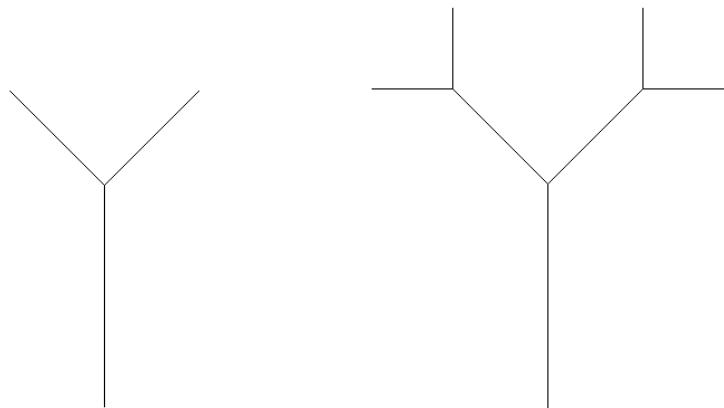


Abbildung 6.3: Stufe 2

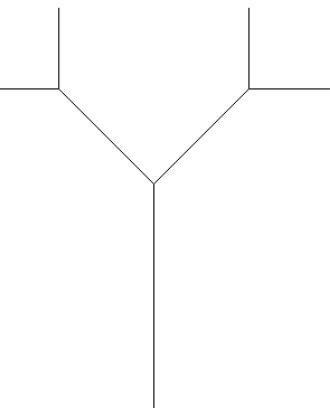


Abbildung 6.4: Stufe 3

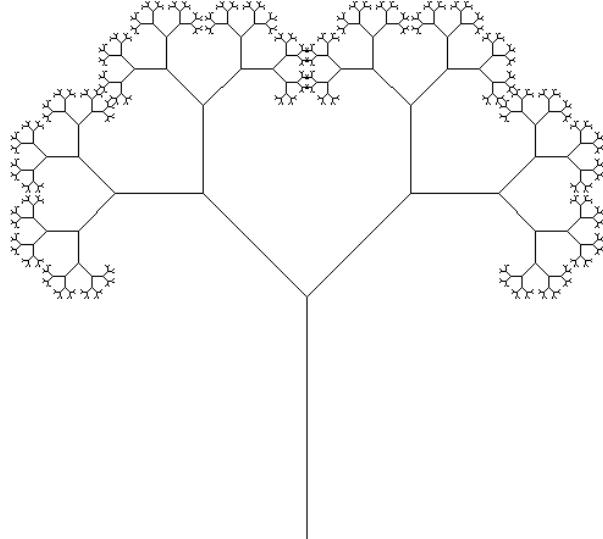


Abbildung 6.5: Ein Binärer Baum der Stufe 10

Lösungsansatz:

Die Turtle soll einen Baum mit einer beliebigen Stufenanzahl zeichnen. Falls die Stufenanzahl größer als 0 ist, geht sie dazu folgendermaßen vor:

Sie geht um die angegebene Länge vorwärts, dreht sich nach links und zeichnet rekursiv einen Baum, der eine Stufe kleiner ist. Dann dreht sie sich nach rechts und zeichnet auch dort rekursiv einen Baum, der eine Stufe kleiner ist. Sie dreht sich wieder so, dass sie nach oben blickt, und läuft dann mit `t.vor(-laenge)` rückwärts zu ihrem Ausgangspunkt zurück.

Aufgabe 2: Farnwedel

Programmiere eine Methode

```
public void farnwedel(double laenge, int stufen)
```

die auf ähnliche Weise arbeitet wie die Methode der vorhergehenden Aufgabe. Der „Baum“ ist jetzt jedoch nicht mehr binär, sondern hat drei Äste, die in unterschiedlichen Winkeln abzweigen. Der linke und der rechte Ast besitzen $\frac{1}{2}$ der Länge des alten Astes. Der mittlere Ast dagegen $\frac{4}{5}$ der alten Astlänge.



Abbildung 6.6: Ein Farnwedel der Stufe 2

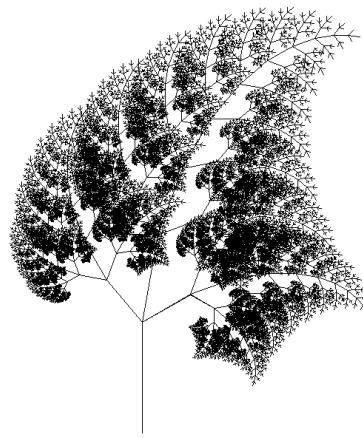


Abbildung 6.7: Ein Farnwedel der Stufe 11

6.4 Rekursion mit Turtle-Grafik (Fortsetzung)

Für die folgenden Aufgaben kopierst du dir die Datei Rekursion3.java aus dem Kurs-Repository in dein eigenes Projekt (und dort in das bereits existierende Package rekursion).

Aufgabe 1: Kochsche Kurve I

Die Turtle steht mit Blick nach rechts. Eine Kochsche Kurve der Stufe 1 ist eine einfache Linie.

Eine Kochsche Kurve der Stufe 2 sieht wie rechts abgebildet aus. Die Linie wird gedrittelt. Über dem Mittelstück wird ein gleichseitiges Dreieck angesetzt. Die Seiten des Dreiecks sind alle ein Drittel der ursprünglichen Linie lang.

Überlege dir, wie groß der Innenwinkel des Dreiecks sein muss!

Abbildung 6.8: Kochsche Kurve Stufe 1



Abbildung 6.9: Kochsche Kurve Stufe 2

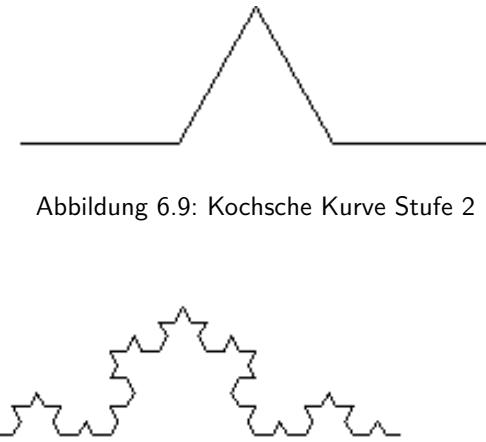


Abbildung 6.10: Kochsche Kurven der Stufe 3 und 4



Programmiere zur Lösung der Aufgabe die Methode:

```
public void koch(double laenge, int stufe)
```

Tipp: Nur bei Stufe 1 wird tatsächlich eine Linie gemalt. Kochsche Kurven einer höheren Stufe, rufen zum „Zeichnen“ vier Mal rekursiv die Methode `koch()` mit einem Drittel von `laenge` auf und drehen sich zwischen den Aufrufen auf geeignete Weise.

Aufgabe 2: Schneeflocke

Die Turtle blickt zu Beginn nach oben. Drehe die Turtle um 30 Grad nach rechts und rufe die Methode `koch()` aus Aufgabe 1 mit der gewünschten Anzahl Stufen auf. Drehe dich dann um 120 Grad nach rechts und rufe die Methode `koch()` erneut auf. Drehe dich ein zweites Mal um 120 Grad nach rechts und rufe die Methode `koch()` zum dritten Mal auf. Dadurch werden drei Kochsche Kurven in Dreieck-Form hintereinander gehängt und man erhält eine Art Schneeflocke.

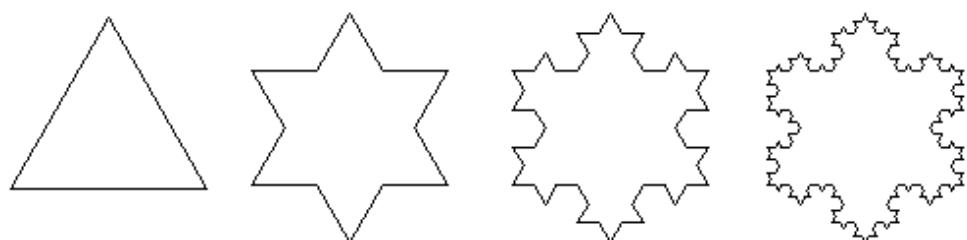


Abbildung 6.11: Kochsche Kurven werden zu Schneeflocken

Aufgabe 3: Variation zur Kochschen Kurve

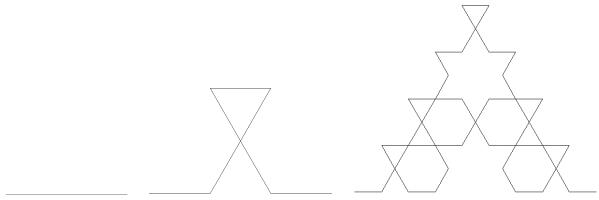
Der Ausgangspunkt ist derselbe wie in Aufgabe 2. Du sollst erneut drei Kochsche Kurven in der Form eines Dreiecks anordnen, aber diesmal sollst du anders herum gehen. Dadurch wird die Kochsche Kurve nicht nach außen sondern nach innen entwickelt. Was für eine Figur entsteht?

Aufgabe 4: Kochsche Kurve II

Ordne vier Kochsche Kurven in Form eines Quadrats an. Zeichne das Quadrat so herum, dass die Kochschen Kurven stufenweise nach innen entwickelt werden. Was für eine Figur entsteht?

Aufgabe 5: Kochsche Kurve III

Die Turtle blickt zu Beginn nach rechts. Zeichne eine erweiterte Kochsche Kurve, bei der auf das gleichseitige Dreieck noch ein zweites, umgedrehtes gleichseitiges Dreieck drauf gesetzt wird. Es entsteht ein hübsches Dreieck mit Schneeflocken-Muster.



Aufgabe 6: Sierpinski-Dreieck

Ein Sierpinski-Dreieck der ersten Stufe besteht aus drei gleich langen Strichen, die im 120° Winkel vom Mittelpunkt abgehen. In der 2. Stufe wird am Ende jedes Striches noch einmal die gleiche Figur angefügt, wobei die Seitenlängen um die Hälfte verkürzt werden, usw.

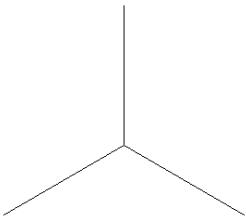


Abbildung 6.12: Stufe 1

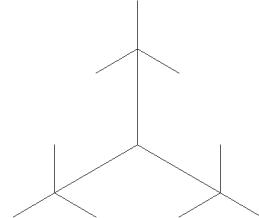


Abbildung 6.13: Stufe 2

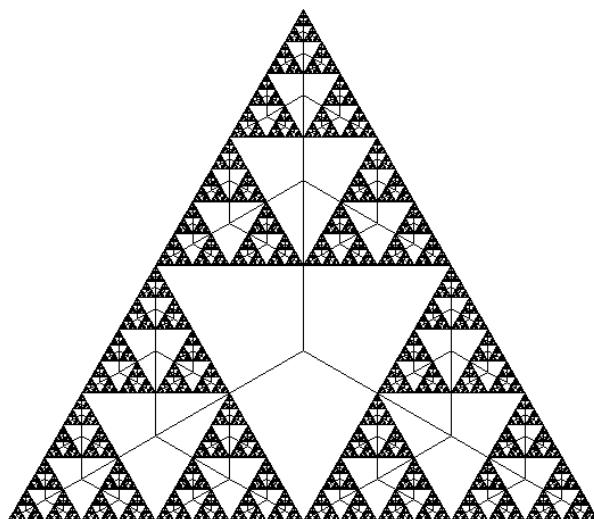


Abbildung 6.14: Sierpinski Dreieck Stufe 10

Aufgabe 7: Baum des Pythagoras

Die Grundstruktur des „Baums des Pythagoras“ besteht aus einem Quadrat, über dem ein rechtwinkliges Dreieck liegt. Auf jede Kathete des Dreiecks werden wieder ein Quadrat und ein rechtwinkliges Dreieck gezeichnet, und so weiter.

Programmiere eine Methode

```
public void pythagoras(double seite, int alpha, int stufe)
```

Durch den Aufruf

```
pythagoras(120, 35, 10);
```

soll ein Baum der Stufe 10 (beim Weg von der Wurzel bis zu einer Baumspitze erscheinen 10 Quadrate) gezeichnet werden, dessen größtes Quadrat die Seitenlänge 120 hat. Der Winkel Alpha im Dreieck sei 35 Grad groß.

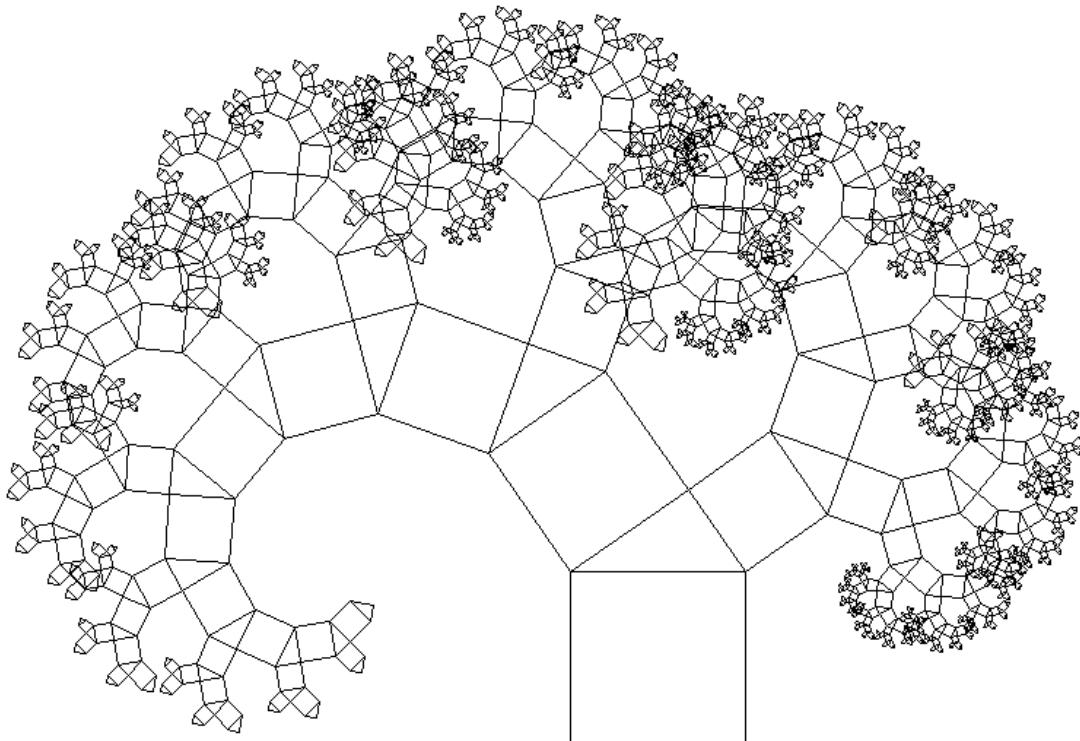


Abbildung 6.15: Baum des Pythagoras (10. Stufe)

7 Java Dialoge

7.1 Dialoge verwenden

Einfache Standard-Dialoge kann man mit der Klasse `JOptionPane` aus dem Package `javax.swing` erzeugen. Um auf die Klasse zugreifen zu können, muss man folgende import-Anweisung oben in der Datei einfügen:

```
import javax.swing.*;
```

Das Sternchen bedeutet, dass alle Klassen aus dem Package `javax.swing` verwendet werden können. Alternativ kann man auch explizit die Klasse angeben, auf die man zugreifen möchte, also:

```
import javax.swing.JOptionPane;
```

In Java gibt es zwei große Bibliotheken (d.h. Code-Sammlungen), die fertigen Code bereit stellen: AWT und Swing. Jede Bibliothek enthält mehrere Unterkomponenten, die sogenannten Packages. Die Klasse `JOptionPane` gehört – wie man am Package-Namen erkennen kann – zur Swing-Bibliothek.

Nachricht ausgeben

Mit der folgenden Methode kann man einen einfachen Dialog mit einem *OK*-Button erzeugen:

```
public static void showMessageDialog(Component parentComponent, Object message)
```

Als ersten Parameter muss man das Anwendungsfenster übergeben.

Als zweiten Parameter wird die Nachricht angegeben (normalerweise als String).

Beispiel:

```
import javax.swing.*; // Oben in der Datei
.
.
.
JOptionPane.showMessageDialog(this, "Falsche Eingabe!"); // Irgendwo in einer
// Methode
```



`this` bezeichnet immer das aktuelle Objekt (so als wenn wir Menschen „ich“ sagen). Wenn man im Anwendungsfenster ist, bezeichnet `this` das Anwendungsfenster.

Methoden, die mit dem Schlüsselwort `static` gekennzeichnet sind, kann man benutzen, indem man einfach den Klassennamen davor setzt.

Eingabe-Dialog

Mit folgender Methode kann man einen Dialog zum Einlesen eines Strings erzeugen:

```
public static String showInputDialog(Object message)
```

Beispiel:

```
String text = JOptionPane.showInputDialog("Gib eine Zahl ein:");
```



Wenn der Benutzer den *OK*-Button drückt, gibt die Methode `showInputDialog()` den eingegebenen Text als String zurück. Wenn der *Abbrechen*-Button gedrückt wird, ist der Rückgabewert `null` (Bedeutung: kein Objekt vorhanden).

7.2 Nebenläufigkeit von Events in Swing

In Programmen werden oft mehrere „Handlungsstränge“ (sogenannte *Threads*) parallel ausgeführt. Ohne diese Nebenläufigkeit wäre es beispielsweise nicht möglich die Benutzerschnittstelle (User-Interface oder auch kurz UI) für Benutzerinteraktionen frei zu halten, während das Programm gleichzeitig eine komplexe Berechnung ausführt oder auf etwas wartet.

Swing ist nicht „Thread-Safe“

Im Unterschied zu AWT ist Swing nicht Thread-Safe. Das bedeutet, dass der Programmierer selbst dafür sorgen muss, dass nicht mehrere Threads gleichzeitig versuchen auf ein Objekt zuzugreifen. Bis auf Weiteres reicht es wenn ihr euch merkt, dass es im Zweifelsfall sinnvoll ist, Swing-Methoden nicht direkt auszuführen, sondern diese an den sogenannten *Event Dispatch Thread (EDT)* zu übergeben:

```
EventQueue.invokeLater(new Runnable() {
    public void run() {
        JOptionPane.showMessageDialog(null, "Game Over");
    }
});
```

Beachte dabei, dass als erstes Argument von `showMessageDialog()` nicht mehr `this` sondern `null` benutzt werden muss!

Statt des in diesem Beispiel verwendeten `showMessageDialog()` könnten es auch beliebige andere Swing Methoden sein. Auch mehrere.

Zu beachten ist dabei, dass der Zeitpunkt der tatsächlichen Ausführung dieses Befehls bzw. dieser Befehlsfolge nicht mehr unter der Kontrolle des Programmierers liegt. Das folgendes funktioniert nicht:

```
name = "Rumpelstilzchen";
EventQueue.invokeLater(new Runnable() {
    public void run() {
        name = JOptionPane.showInputDialog("Gib deinen Namen ein:");
    }
});
System.out.println("Hallo " + name + "!");
```

Richtig wäre es so:

```
name = "Rumpelstilzchen";
EventQueue.invokeLater(new Runnable() {
    public void run() {
        name = JOptionPane.showInputDialog("Gib deinen Namen ein:");
        System.out.println("Hallo " + name + "!");
    }
});
```

Du findest dazu im Kursordner die Dateien `EDTfalsch.java` und `EDTrichtig.java`.

Die Formulierung „im Zweifelsfall“ (oben) bedeutet, dass es durchaus auch ohne den Umweg über den EDT funktionieren kann (und oft genug auch tut). Spätestens dann, wenn Swing Elemente (Fenster und Dialoge) oder Teile davon nicht korrekt dargestellt werden solltest du dich an den Hinweis auf den EDT besinnen und diesen benutzen.

7.3 Umwandlung String ↔ Zahl

Bei der Verwendung von Dialogen gibt es oft die Notwendigkeit zwischen Strings und Zahlenwerten zu konvertieren.

Umwandlung von Strings in Zahlen

String → int

Die Klasse `Integer` besitzt folgende Methode zur Umwandlung eines Strings in eine ganze Zahl:

```
public static int parseInt(String s)
```

Beispiel:

```
String text = "10";
int zahl = Integer.parseInt(text);
```

String → double

Die Klasse `Double` besitzt folgende Methode zur Umwandlung eines Strings in eine Fließkommazahl:

```
public static double parseDouble(String s)
```

Beispiel:

```
String text = "5.25";
double zahl = Double.parseDouble(text);
```

Umwandlung von Zahlen in Strings

Am einfachsten wandelt man eine Zahl in einen String um, in dem man sie mit + an einen Leerstring anhängt.

Beispiel:

```
int zahl = 10;
String text = "" + zahl;
```

Dabei ist es unerheblich, ob es sich um einen Ganzzahl- oder einen Fließkommazahlwert handelt.

7.4 Dialoge – Übungen

Bearbeite die Aufgaben in der Datei Dialoge.java, welche du im Kurs-Repository findest.

Aufgabe 1: Zahl zwischen 1 und 100

- a) Schreibe eine Methode, die überprüft ob eine als Parameter übergebene ganze Zahl zwischen 1 und 100 liegt. Das Ergebnis wird als boolescher Wert zurück gegeben.
- b) Der Benutzer wird in einem Eingabe-Dialog nach einer Zahl gefragt, die an die Methode aus Teil (a) übergeben wird. Das Ergebnis wird in einem Ausgabe-Dialog angezeigt.

Aufgabe 2: Zahl kleiner als 10 oder größer als 20

- a) Schreibe eine Methode, die überprüft ob eine als Parameter übergebene Fließkommazahl kleiner als 10 oder größer als 20 ist. Das Ergebnis wird als boolescher Wert zurück gegeben.
- b) Der Benutzer wird in einem Eingabe-Dialog nach einer Zahl gefragt, die an die Methode aus Teil (a) übergeben wird. Das Ergebnis wird in einem Ausgabe-Dialog angezeigt.

Aufgabe 3: Gerade Zahl

- a) Schreibe eine Methode, die überprüft ob eine als Parameter übergebene ganze Zahl gerade (das heißt durch 2 teilbar) ist. Das Ergebnis wird als boolescher Wert zurück gegeben.
- b) Der Benutzer wird in einem Eingabe-Dialog nach einer Zahl gefragt, die an die Methode aus Teil (a) übergeben wird. Das Ergebnis wird in einem Ausgabe-Dialog angezeigt.

Aufgabe 4: Umwandlung von Notenpunkten in Noten

- a) Schreibe eine Methode, die eine Zensur in Punktangabe in die übliche Notenbezeichnung aus der Mittelstufe umrechnet:

| | | |
|--------------------|---|---------------------|
| 0 | → | 6 |
| 1 bis 3 | → | 5 |
| 4 bis 6 | → | 4 |
| 7 bis 9 | → | 3 |
| 10 bis 12 | → | 2 |
| 13 bis 15 | → | 1 |
| Alle anderen Werte | → | -1 (als Fehlerwert) |
- b) Der Benutzer wird in einem Eingabe-Dialog nach einer Zahl gefragt, die an die Methode aus Teil (a) übergeben wird. Das Ergebnis wird in einem Ausgabe-Dialog angezeigt.
- c) Der Benutzer soll nacheinander mehrere Zahlen überprüfen können ohne erneut auf den Button zu drücken. Rufe den Code aus Teilaufgabe (b) in einer Schleife wiederholt auf, bis der Benutzer den *Abbrechen*-Button drückt.

Aufgabe 5: Mathe-Trainer

- a) In einem Eingabe-Dialog werden nacheinander die folgenden Rechenaufgaben angezeigt, die der Benutzer lösen muss:

5*7 6*7 7*7
 5*8 6*8 7*8
 5*9 6*9 7*9

Verwende dazu zwei ineinander geschachtelte for-Schleifen. Die eine Schleife zählt die erste Zahl von 5 bis 7 hoch. Die andere Schleife zählt die zweite Zahl von 7 bis 9 hoch. Nachdem der Benutzer seine Lösung eingegeben hat, wird die Korrektheit der Lösung überprüft. In einem Ausgabedialog wird dann entweder „Korrekt.“ ausgegeben oder es wird ausgegeben „Falsch. Die richtige Lösung ist *Lösung*.“

- b) Erweitere das Programm so, dass der Benutzer nach Eingabe einer falschen Lösung so lange wiederholt dieselbe Aufgabe erhält, bis er die Lösung korrekt angegeben hat.

Aufgabe 6: Lösung quadratischer Gleichungen

Gleichungen der Form $x^2 + p \cdot x + q = 0$ können mit der *p-q*-Formel gelöst werden.

Die *p-q*-Formel liefert zwei Lösungen:

$$x_1 = -\frac{p}{2} + \sqrt{\left(\frac{p}{2}\right)^2 - q} \quad \text{und} \quad x_2 = -\frac{p}{2} - \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

Frage den Benutzer in zwei Eingabe-Dialogen nach einem Wert für *p* und *q*. Wandle die Benutzereingaben in Fließkommazahlen um und berechne zunächst den Wert unter der Wurzel. Falls dieser Wert negativ ist, wird in einem Ausgabe-Dialog „keine Lösung“ ausgegeben, da man keinen Wert aus einer negativen Zahl ziehen kann. Falls der Wert 0 ist, gibt es nur eine Lösung, die in einem Ausgabe-Dialog angezeigt wird, nämlich $x = -\frac{p}{2}$. Falls der Wert positiv ist, gibt es zwei Lösungen. Berechne beide Lösungen und gib sie in einer zusammenhängenden Zeichenkette mit einem einzigen Ausgabe-Dialog aus.

Teste dein Programm mit den Testdaten in der folgenden Tabelle.

| Aufgabe | Lösung |
|-------------------------------|--|
| $x^2 - 4 \cdot x - 5 = 0$ | $x_1 = 5, x_2 = -1$ |
| $x^2 + 7,9 \cdot x + 2,5 = 0$ | $x_1 = -0,330\dots, x_2 = -7,569\dots$ |
| $x^2 + 4,6 \cdot x - 1 = 0$ | $x_1 = 0,207\dots, x_2 = -4,807\dots$ |
| $x^2 + 4 \cdot x + 4 = 0$ | $x = -2$ |
| $x^2 - 5 \cdot x + 17 = 0$ | keine Lösung |
| $x^2 + 0 \cdot x + 6 = 0$ | keine Lösung |

Mathematische Funktionen

```
double wurzel = Math.sqrt(3); // √3
double hoch = Math.pow(4,2); // 42
```

Aufgabe 7: Body Mass Index (BMI)

Der Body Mass Index wird von Ernährungswissenschaftlern verwendet, um festzustellen ob eine Person Übergewicht oder Untergewicht hat. Der Body Mass Index berechnet sich nach folgender Formel:

$$\text{BMI} = \frac{\text{Gewicht in kg}}{(\text{Größe in m})^2}$$

Erwachsene gelten als normalgewichtig, wenn ihr BMI zwischen 18,5 und 25 liegt. Wenn ihr BMI über 25 liegt, haben sie Übergewicht. Wenn er unter 18,5 liegt, haben sie Untergewicht. Frage den Benutzern in zwei Eingabedialogen zuerst nach seinem Gewicht und dann nach seiner Größe (Achtung: Die Größe muss in Metern eingeben werden, also als Kommawert!). Wandle beide Werte in Fließkommazahlen um, berechne den BMI und gib in einem Ausgabedialog aus, ob der Benutzer Übergewicht, Untergewicht oder Normalgewicht hat.

Aufgabe 8: Niedersachsenticket

Ein Niedersachsenticket ist einen Tag gültig und kostet 28€. Maximal 5 Personen können auf dem Ticket an Wochentagen in Regionalzügen durch Bremen, Hamburg und Niedersachsen fahren. Wie viel eine Person z.B. für eine Tagesfahrt von Bremen nach Hamburg zahlt, hängt von der Anzahl der Mitfahrer ab. Wenn beispielsweise 2 Personen nach Hamburg fahren, zahlt jeder 28€ : 2 = 14€. Wenn statt dessen 6 Personen mitfahren, müssen zwei Ticket gekauft werden. Jeder zahlt dann 56€ : 6 = 9,33€.

- a) Schreibe eine Methode, die als Parameter die Anzahl der Personen erhält und als Rückgabewert den Fahrpreis für jede einzelne Person ermittelt. Welchen Datentyp sollte man für den Parameter und den Rückgabewert wählen?
- b) Der Benutzer wird in einem Eingabe-Dialog nach der Anzahl der Personen gefragt, die an die Methode aus Teil a) übergeben wird. Der berechnete Fahrpreis wird in einem Ausgabe-Dialog angezeigt.
- c) Der Benutzer soll nacheinander mehrere Zahlen eingeben können ohne erneut auf den Button zu drücken. Rufe den Code aus Teilaufgabe b) in einer Schleife wiederholt auf, bis der Benutzer den Abbrechen-Button drückt.

8 Eigene Java Programme

8.1 Eigene Java-Programme basierend auf HJFrame

Bisher hast du in Java Aufgaben gelöst, indem du vorgegebene Java-Dateien mit eigenen Java-Befehlen ergänzt hast. Ab sofort sollst du aber die Programme komplett selber schreiben. Vereinfacht wird dies zunächst dadurch, dass du deine Programme von der Klasse `HJFrame` ableitest und dadurch alle Eigenschaften und Möglichkeiten von `HJFrame` erbst. (Was man unter „ableiten“ und „erben“ im Zusammenhang mit Objektorientierter Programmierung genau versteht, wirst du später noch lernen.)

Wenn du der Anleitung im ersten Kapitel richtig gefolgt bist, hast du bereits ein Package `hilfe` in deinem eigenen Java-Projekt in Eclipse und hast auch eine Vorlage für die einfache Erstellung eines Programm-Gerüsts, welches eine neue Java-Klasse basierend auf `HJFrame` erstellt.

Um diese Vorlage zu nutzen genügt es im Editor die Zeichenfolge `HJFrame` einzutippen (Groß- und Kleinschreibung ist dabei egal) und durch ein <Strg>-<Leertaste> zu bestätigen.

Beispielprogramm

```

1 import java.awt.Color;
2 import java.awt.EventQueue;
3 import java.awt.Graphics;
4 import hilfe.*;
5
6 public class Beispiel extends HJFrame {
7     final static int WIDTH = 500;
8     final static int HEIGHT = 500;
9     final static Color BACKGROUND = Color.WHITE;
10    final static Color FOREGROUND = Color.BLACK;
11    String text = "Guten Tag.";
12
13    public Beispiel(final String title) {
14        super(WIDTH, HEIGHT, BACKGROUND, FOREGROUND, title);
15        // eigene Initialisierung
16        text = text + " Wie geht es dir?";
17    }
18
19    @Override
20    public void myPaint(Graphics g) {
21        // wird aufgerufen, wenn das Fenster neu gezeichnet wird
22        g.drawString(text, 10, 80);
23        g.setColor(Color.RED);
24        g.fillOval(100, 100, 200, 200);
25    }
26
27    public static void main(final String[] args) {
28        EventQueue.invokeLater(new Runnable() {
29            @Override
30            public void run() {
31                try {
32                    Beispiel anwendung = new Beispiel("Beispiel");
33                } catch (Exception e) {
34                    e.printStackTrace();
35                }
36            }
37        });
38    }
39 }
```

Anmerkungen zum Beispiel-Programm:

- In den Zeilen 1 bis 4 werden Klassen aus anderen Packages, die in der Datei verwendet werden, aufgeführt. Ein Package enthält eine Gruppe fertiger Java-Dateien.
Der * bedeutet, dass alle Klassen des Paketes ausgewählt werden.
- In Zeile sechs wird der Name der Klasse festgelegt. Er muss identisch mit dem Dateinamen (ohne die *.java Dateiendung) sein. Auch die Groß- und Kleinschreibung muss dabei beachtet werden!
- In den Zeilen 13 bis 17 sieht man den Konstruktor. Er hat denselben Namen wie die Klasse. Der Konstruktor enthält Code, der einmal beim Start des Programms aufgerufen wird.
- Zeile 20 bis 25: Die Methode `myPaint()` wird vom System jedes Mal aufgerufen, wenn das Fenster neu gezeichnet werden muss.
- Die Methode `main()` (ab Zeile 27) wird als erstes aufgerufen, wenn das Programm gestartet wird.

8.2 Malen mit der Klasse Graphics

Der Code zum Malen des Fensterinhalts wird in die Methode `myPaint()` eingefügt. Das System ruft diese Methode auf, wenn das Fenster neu gezeichnet werden muss, z.B. weil es vorübergehend durch ein anderes Fenster verdeckt wurde.

Vor dem Aufruf von `myPaint()` wird der Inhalt des Fensters automatisch gelöscht.

Damit das Malen für den Programmierer einfacher wird, gibt es die Klasse **Graphics**, die einfache Malbefehle beherrscht, wie z.B. das Zeichnen von Linien oder Rechtecken. Der Programmierer erhält als Parameter der `myPaint()`-Methode ein Objekt der Klasse **Graphics** übergeben.

Die nachfolgende Tabelle listet die wichtigsten Methoden der Klasse **Graphics** auf. In den Methoden gibt man die Koordinaten von gewünschten Punkten (Pixel) auf der Zeichenfläche an. Die linke obere Ecke der Zeichenfläche hat die Position (0,0) des gedachten Koordinatensystems.

Methoden von Graphics (Auswahl)

| | |
|--|--|
| <code>void setFont(Font font)</code> | Festlegen der Schriftart |
| <code>void setColor(Color c)</code> | Festlegen der Vordergrundfarbe |
| <code>void drawString(String str, int x, int y)</code> | schreibt einen Text (x,y = linke untere Ecke) |
| <code>void drawLine(int x1, int y1, int x2, int y2)</code> | malt eine Linie von (x1, y1) nach (x2, y2) |
| <code>void drawRect(int x, int y, int width, int height)</code> <code>void fillRect(int x, int y, int width, int height)</code> | malt ein Rechteck (x,y = linke obere Ecke) <code>fillRect()</code> : ausgefüllt, <code>drawRect()</code> : hohl |
| <code>void drawOval(int x, int y, int width, int height)</code> <code>void fillOval(int x, int y, int width, int height)</code> | malt im Bereich des angegebenen Rechtecks einen Kreis oder eine Ellipse (x,y = linke obere Ecke) <code>fillOval()</code> : ausgefüllt, <code>drawOval()</code> : hohl |
| <code>void clearRect(int x, int y, int width, int height)</code> | löscht den angegebenen Bereich durch Übermalen mit der Hintergrundfarbe (x,y = linke obere Ecke) |

8.3 Eigene Programme mit HJFrame – Übungen

Aufgabe 1: Abstrakte Kunst

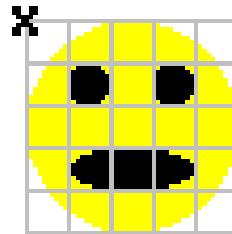
Zeichne ein abstraktes Kunstwerk oder eine Figur. Teste dabei alle vorgestellten Methoden der Klasse **Graphics** einmal aus. Verändere außerdem die Schriftart sowie die Vorder- und Hintergrundfarbe des Programms. Ändere auch die Größe des Anwendungsfensters.

Aufgabe 2: Gespenster

- Erzeuge ein neues Anwendungsfenster mit einem schwarzen Hintergrund.
- Programmiere eine Methode, die ein Gespenst an einer angegebenen Position im Fenster zeichnet:

```
public void gespenst(Graphics g, int x, int y)
```

Die Methode erhält das **Graphics**-Objekt („den Zeichenstift“) als Parameter und die linke obere Ecke des Gespenstes, die in der Zeichnung mit einem Kreuz markiert ist. Zeichne das Gespenst als ausgefüllten gelben Kreis. Auf den Kreis werden die Augen und der Mund als ausgefüllte schwarze Ovale aufgesetzt. Ein Kästchen in der Zeichnung entspricht einer Einheit von 10 Pixeln.



- Rufe die Methode **gespenst()** in der **myPaint()**-Methode mehrfach auf, so dass du Gespenster an verschiedenen Positionen erhältst.

Aufgabe 3: Aufrufe von myPaint() zählen

- Schreibe ein Programm, das zählt wie oft die **myPaint()**-Methode aufgerufen wird. Der Zähler ist eine globale Integer-Variable, die zu Anfang auf 0 gesetzt wird. Bei jedem Aufruf von **myPaint()** wird der Zähler um eins erhöht. Anschließend wird der Zähler mit der **Graphics**-Methode **drawString()** im Anwendungsfenster ausgegeben. Die Integer-Variable wird dazu mit + an einen String angehängt.
- Teste aus, in welchen Situationen der Zähler erhöht wird. Verdecke dazu vorübergehend das Programmfenster durch ein anderes Fenster oder verkleinere das Fenster zu einem Icon.

Aufgabe 4: Quadrate zählen die Aufrufe von myPaint()

Füge in das Programm aus Aufgabe 3 eine Zeile mit ausgefüllten grünen Quadraten ein. Beim ersten Aufruf der **myPaint()**-Methode wird nur ein Quadrat gezeichnet, beim zweiten Aufruf werden zwei Quadrate gezeichnet, beim dritten Aufruf drei Quadrate, usw.

Damit das Aussehen und die Position der Quadrate später leicht verändert werden können, werden folgende Werte als Konstanten deklariert:

- die Breite der Quadrate (Beispiel: `final static int BREITE = 30;`)
- der Abstand zwischen den Quadranten
- die y-Position, an der die Quadrate beginnen
- die x-Position, an der das erste Quadrat beginnt.

Programmiere die `myPaint()`-Methode entsprechend. Verändere anschließend das Aussehen der Quadrate, indem du die vier Konstanten veränderst. Wenn dein Code sauber programmiert ist, muss nach der Änderung der Konstanten ohne weitere Änderungen am Code noch alles funktionieren.

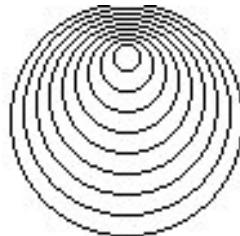
Aufgabe 5: Gespenster-Dialog

Im Programm soll eine Zeile mit lauter Gespenstern gezeichnet werden (benutze dazu die Methode aus Aufgabe 2), zwischen denen jeweils 10 Pixel Abstand liegen. Der Benutzer soll die Anzahl der Gespenster in einem Dialog eingeben. Erzeuge den Dialog im Konstruktor des Programms und merke dir die vom Benutzer eingegebene Zahl in einer globalen Variablen, die du in der `myPaint()`-Methode abfragst.

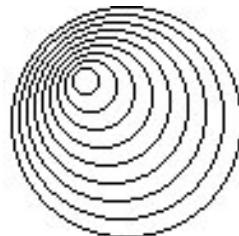
Was passiert, wenn man den Dialog in der `myPaint()`-Methode erzeugt?

Aufgabe 6: Ineinander geschachtelte Kreise

Programmiere eine Methode, die die hier abgebildete Figur zeichnet:



- Die Methode erhält als Parameter das `Graphics`-Objekt, die Zeichenfarbe und die linke obere Ecke der Figur (x- und y-Position). Alle anderen Werte wie zum Beispiel die Breite dürfen in der Methode mit festen Zahlen eingestellt werden.
- Stelle mit Hilfe der in a) programmierten Methode ein Bild aus mehreren Figuren zusammen, die sich gegenseitig überlagern. Jede Figur hat eine eigene Farbe.
- Programmiere eine Methode, die die hier abgebildete Figur zeichnet:



Die Methode erhält dieselben Parameter wie die Methode aus Aufgabe a). Erstelle anschließend wieder ein Bild aus mehreren Figuren mit unterschiedlichen Farben, die sich gegenseitig überlagern.

- Programmiere eine Methode die, die die unten abgebildete Figur zeichnet. Die Kreise werden bei dieser Figur nicht mehr hohl gezeichnet sondern mit zwei sich abwechselnden Farben ausgefüllt. Beide Farben werden der Methode als Parameter übergeben (zusätzlich zum `Graphics`-Objekt und der x- und y- Position). Beachte, dass die Vordergrundfarbe vor jedem Malen eines Kreises mit der Methode `setColor()` gewechselt werden muss.

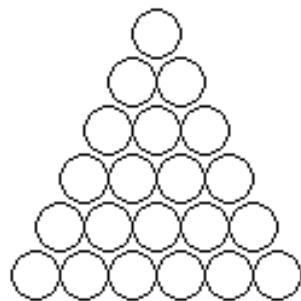


Erstelle ein Bild aus mehreren Figuren mit unterschiedlichen Farben.

Aufgabe 7: Pyramide

Programmiere ein Anwendungsfenster, das eine Pyramide aus Quadraten oder Kreisen (Kreise erhalten exakt dieselben Parameter wie Quadrate) darstellt. Die Breite der Quadrate soll im Code als Konstante vereinbart werden. Dadurch kann man später leicht die Größe der Pyramide verändern, indem man einfach die Konstante umsetzt. Außerdem werden der x- und der y-Wert der linken oberen Ecke der Pyramide als Konstanten vereinbart und die Anzahl der Zeilen, die die Pyramide haben soll. Wenn du die Lösung sauber programmiert hast, muss man später ohne weitere Änderungen des Codes das Aussehen der Pyramide über die Konstanten verändern können.

Tipp für die Programmierung: Zeichne die Pyramide zunächst auf Karo-Papier und vergegenwärtige dir die Position der linken oberen Ecke der einzelnen Kästchen.



Erweiterung: Der Benutzer kann die Anzahl der Zeilen, die die Pyramide haben soll, in einem Eingabe-Dialog eingeben. Die Pyramide wird je nach Eingabe des Benutzer mit unterschiedlicher Größe gezeichnet.

9 Farben und Zufallszahlen

9.1 Farben

Standardfarben

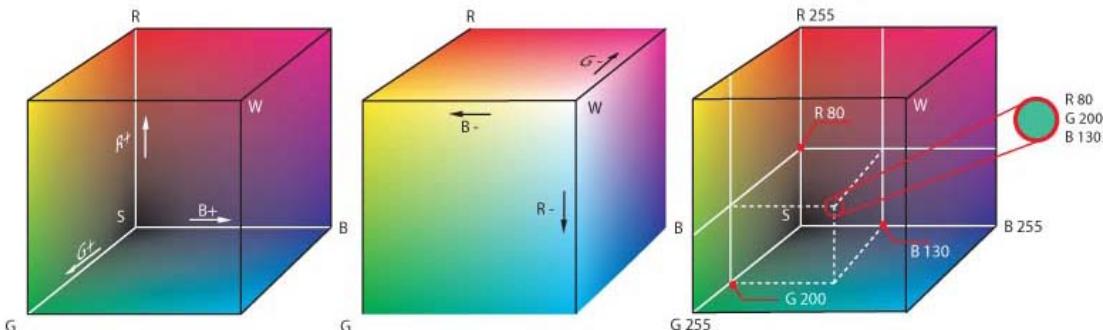
Farben werden in Java mit der Klasse `Color` gemischt. In der Klasse gibt es bereits vordefinierte Konstanten für die Standardfarben, die man z.B. in eine Variable hinein schreiben kann. Beispiel:

```
Color rot = Color.RED; // Schreibt die Farbe Rot in die Variable "rot"
```

Die Variable `rot` kann man dann z.B. innerhalb der `myPaint()`-Methode zum Wechseln der Farbe verwenden:
`g.setColor(rot);`

Farben Mischen

Wenn man eine Farbe mischen möchte, so muss man ein neues `Color`-Objekt erzeugen und dabei als Parameter die RGB-Werte (rot, grün, blau) angeben.



Beispiel:

```
Color hellrot = new Color(255,180,180);
```

Die Variable `hellrot` kann man dann z.B. innerhalb der `myPaint()`-Methode zum Wechseln der Farbe verwenden:

```
g.setColor(hellrot);
```

9.2 Zufallszahlen

- a) Um Zufallszahlen zu verwenden, benötigt man oben in der Datei die folgende Import-Anweisung:

```
import java.util.Random;
```

- b) Bei den globalen Variablen muss man eine Variable der Klasse `Random` anlegen und für diese Variable ein Objekt erzeugen. Dies ist quasi der „Zufallsgenerator“ mit dem man sich später die Zufallszahlen generieren kann.

```
Random zufallsgenerator = new Random();
```

- c) Ganzzahlige Zufallszahlen erhält man, wenn man auf das Zufallsgenerator-Objekt die folgende Methode anwendet:

```
public int nextInt(int n) // Erzeugt eine Zufallszahl von 0 bis n-1
```

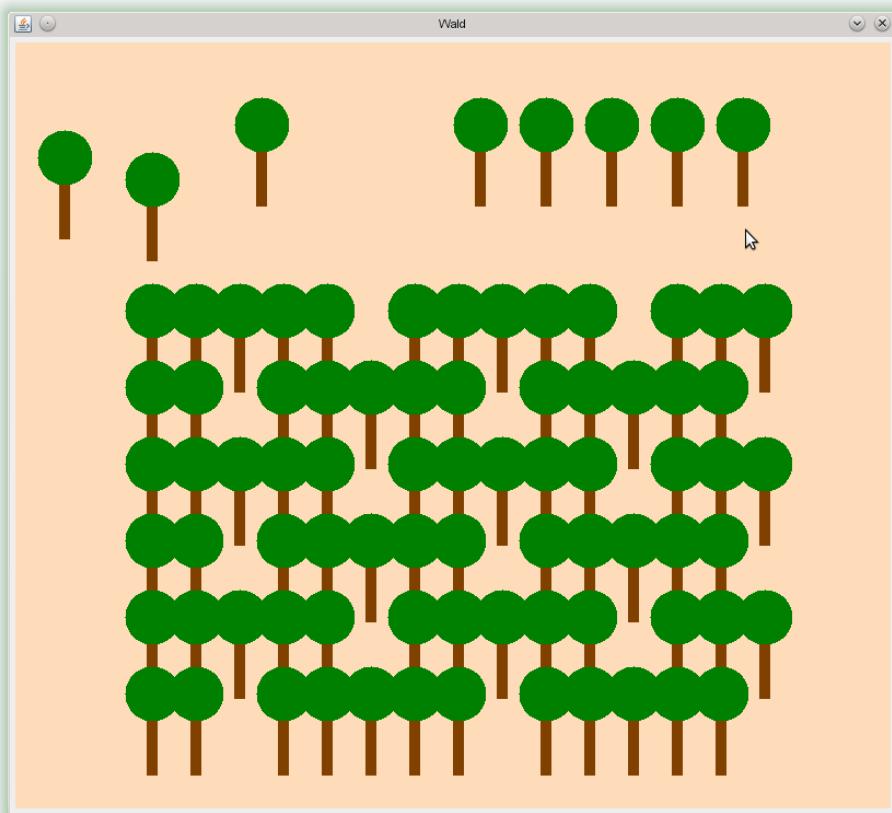
Beispiele:

```
int zufallA = zufallsgenerator.nextInt(5);      // Zufallszahl zwischen 0 und 4
int zufallB = zufallsgenerator.nextInt(5) + 2;  // Zufallszahl zwischen 2 und 6
```

9.3 Farben und Zufallszahlen – Übungen

Aufgabe 1: Wald

- Erzeuge ein Anwendungsfenster mit einer Breite von 800 und einer Höhe von 700 Pixeln. Das Anwendungsfenster soll einen hellbraunen oder beigen Hintergrund erhalten. Die Hintergrundfarbe musst du selber mischen.
- Programmiere eine Methode `baum()`, die einen Baum an einer bestimmten Stelle des Fensters malen kann. Die Methode erhält das `Graphics`-Objekt und die x- und y-Koordinate der linken oberen Ecke übergeben, die in der Zeichnung unten mit einem Kreuz markiert ist. Zeichne die Baumkrone als ausgefüllten dunkelgrünen Kreis von 50 Pixeln Durchmesser und den Baumstamm als ausgefülltes braunes Rechteck von 50 Pixeln Höhe und 10 Pixeln Breite.
- Erzeuge mit Hilfe der Methode aus Teil b) drei einzelne Bäume an den Positionen (20,80), (100,100) und (200,50).
- Erzeuge mit Hilfe der Methode aus Teil b) in einer Schleife eine Zeile mit fünf Bäumen an der y-Position 50. Der erste Baum soll die x-Position 400 erhalten. Die x-Positionen der nachfolgenden Bäume sollen jeweils um 60 Pixel weiter nach rechts verschoben werden, so dass zwischen den Bäumen jeweils 10 Pixel Freiraum ist.
- Erzeuge mit Hilfe der Methode aus Teil b) in einer Schleife eine Zeile mit fünfzehn Bäumen an der y-Position 220. Der erste Baum soll die x-Position 100 erhalten. Die x-Positionen der nachfolgenden Bäume sollen jeweils um 40 Pixel weiter nach rechts verschoben werden, so dass sich die Baumkronen der Bäume leicht überlappen.
- Füge um die Schleife aus Teil e) eine weitere Schleife herum, die die Baumzeile sechs Mal wiederholt. Die y-Position der weiteren Baum-Zeilen soll jeweils um 70 Pixel nach unten verschoben werden, so dass die Baumkronen der unteren Bäume den Stamm des über ihnen liegenden Baumes etwas überdecken.
- Erweitere die Doppelschleife aus Teil f) um einen Zähler, der sämtliche Bäume durchzählt (über die Zeile hinweg). Damit der Wald etwas lebendiger wirkt, soll jeder sechste Baum nicht gezeichnet werden.

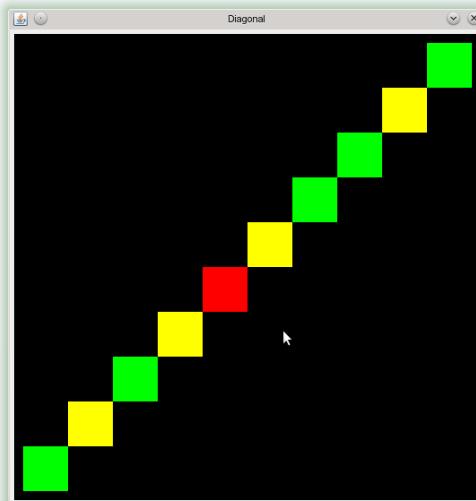


Aufgabe 2: Wald mit Zufallspositionen

- Erzeuge eine neue Klasse für ein Anwendungsfenster mit einer Breite und Höhe von je 500 Pixeln (dies sind die Standardwerte). Kopiere die Methode `baum()` aus Aufgabe 1 in die neue Klasse hinein.
- Erzeuge in einer Schleife 50 Bäume an zufälligen Positionen. Für die x-Position eines Baumes sollen jeweils Zufallswerte zwischen 0 und 449 erzeugt werden. Für die y-Position eines Baumes sollen Zufallswerte zwischen 30 und 399 erzeugt werden.

Aufgabe 3: Diagonale

- Erzeuge ein neues Anwendungsfenster mit einer Breite von 520 Pixeln und einer Höhe von 520 Pixeln. Das Anwendungsfenster soll eine schwarze Hintergrundfarbe erhalten.
- Programmiere eine Diagonale von 10 grünen Quadraten (dazu kannst du die grüne Standardfarbe nehmen), die von links unten nach rechts oben hoch geht. Die Quadrate besitzen alle eine Breite von 50 Pixeln. Das erste Quadrat links unten hat die Koordinaten (10,460). Zwischen den Eckpunkten der Quadrate soll sich keine Lücke befinden.
- Jedes einzelne Quadrat soll zufällig entweder die Farbe rot, grün oder gelb erhalten (dazu kannst du die Standardfarben verwenden). Erzeuge dazu für jedes Quadrat eine Zufallszahl zwischen 0 und 2 und setze die Zahl dann mit einer `switch`-Anweisung in einen Zufallswert um.



Aufgabe 4: Farbsäule

Erstelle ein neues Anwendungsfenster mit einer schwarzen Hintergrundfarbe.

Zeichne in dem Anwendungsfenster eine Säule mit acht untereinander liegenden Rechtecken, die alle einen anderen Rot-Ton erhalten. Das oberste Quadrat erhält ein tiefes dunkelrot, dass folgendermaßen gemischt wird:

```
Color rot = new Color(50,0,0);
```

Bei den nachfolgenden Rechtecken wird der Rot-Anteil der Farbe jeweils um 25 erhöht, so dass die Rechtecke immer heller und heller werden.

Jedes Rechteck soll eine Breite von 100 Pixeln und eine Höhe von 50 Pixeln besitzen. Alle Rechtecke liegen ohne Lücke direkt untereinander. Das oberste Rechteck hat die Position (200,70).

10 Klassen und Objekte

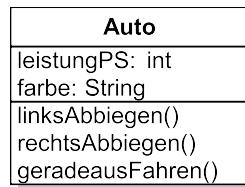
Eine *Klasse* bezeichnet eine Gruppe gleichartiger *Objekte*.

Beispiele:

| Klasse | Objekte der Klasse |
|-----------------|---|
| Sportart | Handball, Badminton, Rudern, Leichtathletik, Boxen |
| Baum | Erle, Birke, Kastanie, Pappel, Buche |
| Bundespräsident | Theodor Heuss, Heinrich Lübke, Richard von Weizsäcker, Johannes Rau |

10.1 Darstellung einer Klasse mit der Modellierungssprache UML

Beispiel: Darstellung der Klasse **Auto**



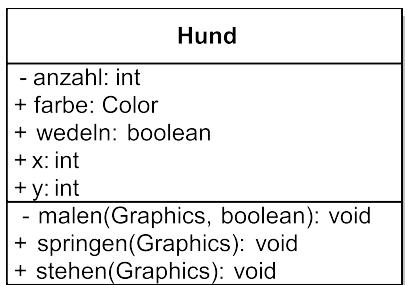
Das Diagramm einer Klasse ist in drei Abschnitte geteilt. In den obersten Abschnitt wird der Name der Klasse geschrieben.

In den mittleren Abschnitt trägt man die Eigenschaften der Klasse ein, die für das Programm, das generiert werden soll, von Bedeutung sind. Die Eigenschaften bezeichnet man als Attribute. Wenn man die Klasse programmiert, wird für jedes Attribut eine Variable angelegt. Der Datentyp der Variablen steht durch einen Doppelpunkt getrennt hinter dem Namen des Attributs.

In den unteren Abschnitt schreibt man die Methoden, die auf die Objekte der Klasse angewendet werden können.

In der Programmiersprache Java beschreibt eine Klasse das gemeinsame Schema für alle ihre Objekte. Der Code wird nur einmal programmiert und automatisch auf jedes Objekt angewendet, das zu der Klasse gehört.

10.2 Beispiel: Klasse Hund



Sichtbarkeit

private (-) : Variable/Methode kann nicht von anderer Klasse aufgerufen werden.

public (+) : Variable/Methode kann von jeder anderen Klasse aufgerufen werden.

Eine erste Version der Klasse Hund

(Datei HundVorversion.java im Kurs-Repository)

```

import java.awt.*;

public class Hund {
    public Color farbe = Color.LIGHT_GRAY;
    public boolean wedeln = false;
    public int x = 0;          // x- und y-Koordinate der
    public int y = 0;          // linken oberen Ecke
    // Klassenvariable: zählt die Anzahl aller Hunde - aber wann???
    static private int anzahl = 0;

    public void stehen(Graphics g) {      // malt den Hund stehend
        malen(g, true);
    }

    public void springen(Graphics g) {    // malt den Hund springend
        malen(g, false);
    }

    // "interne" Programmierung: malen() ist aus anderen Klassen heraus nicht sichtbar!
    private void malen(Graphics g, boolean stehen) {
        Color c = g.getColor();
        if (wedeln) {
            g.drawLine(x+0,y+0,x+10,y+20);
        } else {
            g.drawLine(x+0,y+40,x+10, y+20);
        }
        if (stehen) {
            g.drawLine(x+18, y+40, x+18, y+60);
            g.drawLine(x+20, y+40, x+20, y+60);
            g.drawLine(x+44, y+40, x+44, y+60);
            g.drawLine(x+46, y+40, x+46, y+60);
        } else {
            g.drawLine(x+18, y+40, x+0, y+52);
            g.drawLine(x+20, y+40, x+2, y+52);
            g.drawLine(x+44, y+40, x+56, y+52);
            g.drawLine(x+46, y+40, x+58, y+52);
        }
        g.setColor(farbe);
        g.fillRoundRect(x+10, y+15, 45, 25, 12, 12);
        g.fillOval(x+49,y+0,21,21);
        g.setColor(c);
        g.drawRoundRect(x+10, y+15, 45, 25, 12, 12);
        g.drawOval(x+49,y+0,21,21);
    }
}

```

Was bedeutet static?

Alle normalen (nicht **static**) Variablen und Methoden beziehen sich immer auf ein spezielles Objekt, dessen Namen man angeben muss.

static Variable: Es gibt die Variable nur einmal pro Klasse.

static Methode: Die Methode bezieht sich nicht auf ein einzelnes Objekt und darf keine Objekt-Variablen verwenden.

Um Objekte der Klasse **Hund** in einem eigenen Programm zu nutzen würde man jetzt wie folgt vorgehen. Zunächst würde man eine Variable vom Typ **Hund** deklarieren:

```
Hund felix;
```

Anschließend könnte man ein entsprechendes **Hund**-Objekt erzeugen:

```
felix = new Hund();
```

Dazu wurde der sogenannte *Konstruktor* der Klasse mit dem **new**-Operator aufgerufen. Ein Konstruktor hätte in der Klasse **Hund** programmiert werden können. Aber in der ersten Version der Klasse **Hund** ist davon nichts zu sehen.

Tatsächlich *muss* man keinen Konstruktor programmieren. In diesem Fall gibt es immer einen parameterlosen Konstruktor, der ein neues Objekt der Klasse erzeugt. Dabei werden alle Objektvariablen so initialisiert, wie es in der Klasse vorgegeben ist.

Das Hund-Objekt **felix** hätte somit zu Beginn folgende Attributwerte:

```
farbe      Color.LIGHT_GRAY
wedeln     false
x          0
y          0
```

Da diese Attribute alle als **public** deklariert wurden, lassen sich diese Werte nun aus der Anwendungsklasse heraus problemlos ändern:

```
felix.x = 100;
felix.y = 250;
felix.farbe = Color.BLACK;
felix.wedeln = true;
```

Um unser Hund-Objekt nun im Anwendungsprogramm auch sehen zu können, müssen wir noch eine der beiden Methoden **springen()** oder **stehen()** benutzen:

```
felix.stehen();
```

Der **Graphics**-Kontext, den wir in **myPaint()** automatisch haben, ist der benötigte Parameter **g**. Die Methoden zum Zeichnen der Objekte werden also immer innerhalb von **myPaint()** aufgerufen (bzw. in Methoden, die ihrerseits in **myPaint()** aufgerufen wurden und den **Graphics**-Kontext als Parameter mit übernommen haben).

So weit so gut. Spätestens wenn du eine Hand voll Hunde erzeugt und mit individuellen Attributwerten ausgestattet hast, wirst du dir jedoch eine bequemere Methode zum Erzeugen von solchen Objekten wünschen. Wenn dir das nicht sofort einleuchtet, dann solltest du es ausprobieren!

Klasse Hund mit selbstdefinierten Konstruktoren

Die Lösung besteht darin einen (oder auch mehrere!) Konstruktor in der Klasse **Hund** zu programmieren (Datei **Hund.java** im Kurs-Repository).

Konstruktoren erkennst du daran, dass sie exakt den selben Namen wie die Klasse tragen (und deshalb im Unterschied zu normalen Methoden auch mit einem Großbuchstaben beginnen). Außerdem haben sie keinen Rückgabewert. Nicht einmal das Schlüsselwort **void**, welches normalerweise verwendet wird um bei Methoden anzugeben, dass sie keinen Rückgabewert liefern, wird hier benutzt!

Wenn – wie in unserem Beispiel hier – mehrere Konstruktoren programmiert werden, dann müssen sich diese anhand der Datentypen der Parameterliste unterscheiden lassen. So kann es beispielsweise nicht zwei verschiedene Konstruktoren in der selben Klasse geben, die beide als Parameter zwei Integer Werte erwarten. Anhand der unterschiedlichen Parameterlisten kann der Java-Compiler dann entscheiden, welcher der vorhandenen Konstruktoren im konkreten Fall zu benutzen ist.

In unserer Klasse **Hund** werden vier Konstruktoren definiert. Wenn man sich die Datentypen der Parameterlisten dieser vier Konstruktoren ansieht, erkennt man, dass diese eindeutig voneinander unterscheidbar sind:

Liste der Konstruktoren der Klasse **Hund**:

```
Hund()
Hund(int, int, Color)
Hund(int, int, boolean)
Hund(Color, boolean)

import java.awt.*;

public class Hund {
    public Color farbe = Color.LIGHT_GRAY;
    public boolean wedeln = false;
    public int x = 0;          // x- und y-Koordinate der
    public int y = 0;          // linken oberen Ecke
    static private int anzahl = 0; // Klassenvariable: Anzahl aller Hunde

    public Hund() {
        wedeln = true;
        anzahl++;
    }

    public Hund(int xPos, int yPos, Color farbe) {
        x = xPos;
        y = yPos;
        this.farbe = farbe;
        wedeln = true;
        anzahl++;
    }

    public Hund(int x, int y, boolean wedeln) {
        this.x = x;
        this.y = y;
        farbe = Color.YELLOW;
        this.wedeln = wedeln;
        anzahl++;
    }

    public Hund(Color f, boolean w) {
        farbe = f;
        wedeln = w;
        anzahl++;
    }

    static public int getAnzahlHunde() {
        return anzahl;
    }

    public void stehend(Graphics g) {      // malt den Hund stehend
        malen(g, true);
    }

    public void springend (Graphics g) {   // malt den Hund springend
        malen (g, false);
    }

    private void malen (Graphics g, boolean stehend) {
        ...
    }
}
```

10.3 Objekte erzeugen

Eine Klasse gibt das Schema für eine Gruppe gleichartiger Objekte vor. Ein konkret existierendes Objekt bezeichnet man auch als Instanz.

Zum Erzeugen eines Objektes (bzw. einer Instanz) einer Klasse muss man zunächst eine Variable mit dem Typ der Klasse deklarieren, zum Beispiel:

```
Hund hund1;
```

Schema:

```
Klasse variablenname;
```

Diese Variable selbst enthält jedoch noch kein Objekt. Die Variable dient dazu die Speicheradresse abzuspeichern, an der sich die Daten des Objektes im Hauptspeicher befinden. Solange die Variable noch nicht auf ein Objekt zeigt, hat sie den Wert `null`. `null` bedeutet „kein Objekt“.

Das Anlegen eines Objektes und das Abspeichern der Speicheradresse in der Variablen geschieht zum Beispiel mit folgenden Befehlen:

```
hund1 = new Hund();
hund2 = new Hund(50, 50, Color.YELLOW);
```

Schema:

```
Variablename = new Klasse(Parameter);
```

Man kann die Deklaration und das Erzeugen des Objektes auch in einem Befehl zusammen fassen:

```
Hund hund1 = new Hund();
```

Wenn es für das Programm nützlich ist, kann man weitere Variablen auf dasselbe Objekt zeigen lassen:

```
Hund hilfe = hund1; // hilfe zeigt auch auf hund1
```

Ein Objekt lebt solange, wie es Variablen gibt, die auf das Objekt verweisen. Sobald es auf ein Objekt keinen Verweis mehr gibt, wird das Objekt vom System automatisch vernichtet. Wenn man explizit sagen möchte, dass eine Variable auf kein Objekt zugreift, so kann man ihr den Wert `null` zuweisen:

```
hund1 = null;
```

10.4 Variablen einer Klasse

Es gibt drei verschiedene Arten von Variablen, die in dem folgenden Beispiel verdeutlicht werden:

```
class Beispiel {
    static int klassenVariable; // 1 mal pro Klasse
    int objektVariable; // 1 mal pro Objekt

    void methode() {
        int lokaleVariable; // nur innerhalb von methode()
    }
}
```

Klassenvariablen beziehen sich auf alle Objekte der Klasse und sind deshalb nur einmal vorhanden. Beispiel: Ein Zähler, der zählt, wie viele Objekte der Klasse angelegt werden.

Objektvariablen speichern die Daten eines speziellen Objektes, z.B. die Farbe eines Hundes. Jedes Objekt besitzt seine eigenen Objektvariablen.

Lokale Variablen werden innerhalb einer Methode deklariert und werden am Ende der Methode automatisch vom System wieder gelöscht.

Klassen- und Objektvariablen werden vom System automatisch mit 0 bzw. `false` oder `""` (leerer String) initialisiert. Lokale Variablen dagegen haben zu Beginn einen unbestimmten Wert.

Zugriff auf Variablen

a) von außen

Von außen (d.h. von einer anderen Klasse aus) greift man auf Klassenvariablen nach folgendem Schema zu:

```
Klasse.attribut
```

Beispiel:

```
Beispiel.klassenVariable = 10;
```

Auf eine Objektvariable kann man nur dann zugreifen, wenn man ein Objekt besitzt. Schema:

```
objekt.attribut
```

Beispiel:

```
Beispiel bsp = new Beispiel();  
bsp.objektVariable = 5;
```

b) innerhalb einer Klasse

Innerhalb einer Klasse kann man auf alle Klassen und Objektvariablen direkt zugreifen ohne vorher den Klassen- oder Objektnamen anzugeben. Tatsächlich kann man gar keinen „richtigen“ Objektnamen angeben, da die Klasse ja den Code für alle Objekte darstellt. Um eine Objektvariable direkt anzusprechen (z.B. um sie von einer lokalen Variablen mit dem gleichen Namen zu unterscheiden), kann man das Objekt `this` verwenden. `this` bezeichnet das aktuelle Objekt (analog zu dem menschlichen Ausdruck „ich“). In der Klasse Beispiel könnte beispielsweise stehen:

```
this.objektVariable = 10;
```

10.5 Methoden

Methoden, die mit dem Schlüsselwort `static` gekennzeichnet sind, können jederzeit auch ohne Existenz eines Objektes aufgerufen werden. Sie dürfen jedoch nur Klassenvariablen und lokale Variablen verwenden.

Alle nicht statischen Methoden, werden immer auf ein Objekt der Klasse angewendet. Mit ihnen kann man die Objektvariablen des aktuellen Objektes verändern.

10.6 Konstruktor

Der Konstruktor ist eine spezielle Methode, die vom System bei der Erzeugung eines Objektes mit dem `new` Operator aufgerufen wird. Er ist dazu gedacht, Initialisierungen für das Objekt vorzunehmen (also Anfangswerte für die Variablen des Objektes zu setzen). Das besondere:

- Der Methodename des Konstruktors ist mit dem Namen der Klasse identisch
- Der Konstruktor besitzt keinen Rückgabewert (auch nicht `void!`). Aber er kann beliebig viele Parameter besitzen.
- Der Konstruktor darf niemals explizit aus einer anderen Methode heraus aufgerufen werden.

10.7 Animationen erzeugen

Mit Hilfe der Klasse `javax.swing.Timer` kann man dafür sorgen, dass die `myPaint()`-Routine in regelmäßigen Abständen immer wieder aufgerufen wird.

Dies ermöglicht es einfache Animationen zu schreiben, indem man eine Figur, wie z.B. einen Ball, bei jedem neuen Aufruf von `myPaint()` an einer anderen Position malt. Es entsteht dann der Eindruck einer Bewegung. Die Klasse `Timer` besitzt die folgenden Methoden:

| | |
|--|---|
| <code>Timer(int millisec, ActionListener frame)</code> | Über den Konstruktor wird dem Timer mitgeteilt, in welchem zeitlichen Abstand ein ActionEvent erzeugt werden soll. Der zweite Parameter benennt die Anwendung die auf den Event reagieren soll. In der Klasse <code>HJFrame</code> ist dies bereits fertig implementiert, so dass bei jedem so empfangenen ActionEvent das Fenster neu gezeichnet wird. Als zweiter Parameter wird deshalb von euch typischerweise <code>this</code> verwendet. |
| <code>void start()</code> | Mit dieser Methode wird der Timer gestartet. |
| <code>void stop()</code> | Durch Aufruf der Methode <code>stop()</code> wird der wiederholte Aufruf der <code>myPaint()</code> -Methode beendet. Nachdem <code>stop()</code> aufgerufen wurde, ist ein erneuter Start möglich. |

Beispiel für eine Animation

Du findest die Datei `Animation.java` im Kurs-Repository.

```
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Graphics;
import javax.swing.Timer;
import hilfe.*;

public class Animation extends HJFrame {
    // globale Variablen
    private static final int WIDTH = 500;
    private static final int HEIGHT = 500;
    private static final Color BACKGROUND = Color.WHITE;
    private static final Color FOREGROUND = Color.BLACK;
    private int zaehler = 0;

    public Animation(final String title) {
        super(WIDTH, HEIGHT, BACKGROUND, FOREGROUND, title);
        // eigene Initialisierung
        Timer timer = new Timer(10, this);
        timer.start();
    }

    @Override
    public void myPaint(Graphics g) {
        // wird aufgerufen, wenn das Fenster neu gezeichnet wird
        zaehler++;
        g.drawLine(0, 0, zaehler, zaehler);
        g.drawLine(WIDTH, HEIGHT, WIDTH-zaehler, HEIGHT-zaehler);
        g.drawLine(0, HEIGHT, zaehler, HEIGHT-zaehler);
        g.drawLine(WIDTH, 0, WIDTH-zaehler, zaehler);
    }
}
```

```
public static void main(final String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                Animation anwendung = new Animation("Animation");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

10.8 Klassen aus dem Package hilfe

Das Package `hilfe` enthält neben der Klasse `HJFrame` weitere Vereinfachungen für Programmieraufgaben, die für euch mit den Standard-Java-Klassen aus dem JRE im Moment noch zu schwer zu programmieren sind. Damit Klassen aus dem Package `hilfe` verwendet werden können, muss am Anfang einer Java-Datei folgende `import`-Anweisung eingefügt werden:

```
import hilfe.*;
```

Zeichnen von Dreiecken

Zum Zeichnen von Dreiecken steht die Klasse `HZeichnen` zur Verfügung, die folgende Methoden besitzt:

- `static void drawDreieck(Graphics g, int x1, int y1, int x2, int y2, int x3, int y3)`
`drawDreick()` malt ein hohles Dreieck mit den drei Eckpunkten (x_1, y_1) , (x_2, y_2) und (x_3, y_3) . Als erster Parameter muss ein Objekt der Klasse `Graphics` übergeben werden, da die Methode dieses Objekt zum Malen braucht.
- `static void fillDreieck(Graphics g, int x1, int y1, int x2, int y2, int x3, int y3)`
`fillDreick()` malt ein gefülltes Dreieck mit den drei Eckpunkten (x_1, y_1) , (x_2, y_2) und (x_3, y_3) .

Beispiel für die Verwendung der `fillDreieck()`-Methode:

```
HZeichnen.fillDreieck(g, 10, 90, 50, 90, 30, 60);
```

10.9 Klassen und Objekte – Übungen

Aufgabe 1: Hunde

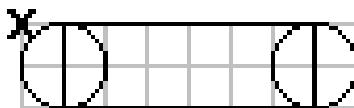
- Erzeuge mindestens vier verschiedene Objekte der Klasse **Hund** und zeichne sie entweder stehend oder springend im Frame. Jeder Hund soll nur an einer Position gemalt werden, denn schließlich sieht man ja auch einen Hund in der realen Welt nicht doppelt (es sei denn man hat zuviel getrunken). Gib außerdem die Anzahl der Hunde im Frame aus. Benutze hierfür die statische Methode `getAnzahlHunde()`.
- Programmiere einen der Hunde so, dass er bei jedem neuen Aufruf der Methode `myPaint()` abwechselnd den Schwanz senkt oder hebt.
- Benutze ein Objekt der Klasse **Timer** um dafür zu sorgen, dass `myPaint()` in regelmäßigen Abständen aufgerufen wird. Nun sollte der in b) programmierte Hund intensiv wedeln.
- Lass einen der Hunde von links nach rechts über den Bildschirm laufen. Wenn er rechts aus dem Bild herausgelaufen ist, soll er wieder von links in das Bild hinein laufen. Damit der Lauf möglichst realistisch wirkt, soll der Hund abwechselnd in stehender und in springender Position gezeichnet werden.

Aufgabe 2: Raupe

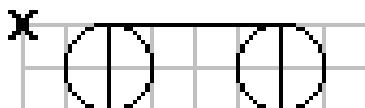
- Erzeuge eine neue, leere Datei (kein Anwendungsfenster!!!) und programmiere in der Datei die Klasse **Raupe**. Eine Raupe besitzt zu Anfang die Attribute x-Position, y-Position, Farbe und Geschwindigkeit in x-Richtung. Alle Attribute sollen **private** sein.

Die Startwerte für die vier Attribute werden im Konstruktor als Parameter übergeben.

Schreibe eine Methode `zeichnen()`, die die Raupe an ihrer aktuellen Position in der für sie gewählten Farbe zeichnet. Die Raupe soll durch zwei ausgefüllte Kreise und ein ausgefülltes Rechteck dargestellt werden. Das Kreuz markiert die linke obere Ecke, die durch den x- und y-Wert beschrieben wird (1 Kästchen entspricht 10 Pixeln):



- Programmiere ein Anwendungsfenster und erzeuge im Anwendungsfenster mehrere Raupen-Objekte mit unterschiedlichen Attributen und zeichne sie in der Methode `myPaint()`.
- Erzeuge im Anwendungsfenster ein Objekt der Klasse **Timer**, damit die Methode `myPaint()` wiederholt aufgerufen wird.
- Die Raupe soll sich wiederholt von links nach rechts über den Bildschirm bewegen. Programmiere dazu in der Klasse **Raupe** eine parameterlose Methode `bewegen()`, die die x-Position der Raupe entsprechend ihrer Geschwindigkeit nach rechts verschiebt. Nachdem die Raupe aus dem Bild „gelaufen“ ist, soll ihre neue x-Position so gewählt werden, dass sie langsam in das Bild hinein gleitet.
Rufe im Anwendungsfenster in der Methode `myPaint()` die Methode `bewegen()` für jede Raupe auf.
- Die Laufbewegung einer Raupe soll auf einfache Weise simuliert werden, in dem die Raupe ihre Größe verändert.
 - Leichte Variante: Die Raupe soll immer abwechselnd einmal mit normaler Länge gezeichnet werden und beim nächsten Mal wie abgebildet verkürzt dargestellt werden.
 - Schwere Variante: Die Raupe soll immer abwechselnd drei mal hintereinander mit normaler Länge gezeichnet werden und dann drei mal hintereinander auf folgende Weise verkürzt dargestellt werden:



Aufgabe 3: Strichmännchen

a) Überlege dir, welche Attribute die Klasse **Strichmaennchen** besitzen muss. Damit es nicht zu langweilig wird, sollen sich die Objekte der Klasse **Strichmaennchen** in folgenden Eigenschaften unterscheiden:

- unterschiedliche Kleidung (Farbe)
- Laufen in unterschiedlichen Geschwindigkeiten
- unterschiedliche y-Positionen (einige Männchen laufen oben andere weiter unten)

Ein Strichmännchen soll zunächst zwei Methoden erhalten:

- Eine Methode **zeichnen()**, die ein Strichmännchen an seiner aktuellen Position auf dem Bildschirm malt.
- Eine Methode **laufen()**, die ein Strichmännchen entsprechend seiner aktuellen Geschwindigkeit eine Position nach vorne setzt.



b) Entwirf die Figur eines Strichmännchens auf kariertem Papier.

c) Programmiere die Klasse **Strichmaennchen** und erzeuge mehrere Strichmännchen-Objekte, die über den Bildschirm laufen. Alle Strichmännchen sollen unterschiedliche Attribut-Werte besitzen. Bei jedem Aufruf der **myPaint()**-Methode werden die Strichmännchen durch Aufruf der Methode **laufen()** um eine Position weiter nach rechts bewegt. Anschließend werden sie durch Aufruf der Methode **zeichnen()** an ihrer aktuellen Position auf dem Bildschirm gemalt. Verwende die Klasse **Timer**, damit die Methode **myPaint()** in regelmäßigen Abständen immer wieder aufgerufen wird.

d) Das Strichmännchen soll jetzt etwas lebendiger gestaltet werden, in dem es zyklisch in verschiedenen Positionen oder mit verschiedenen Bildern gemalt wird. Zum Beispiel könnte man vier verschiedene Bilder des Strichmännchens entwerfen, bei denen die Arme und Beine sich jeweils in unterschiedlichen Laufstellungen befinden. Wenn man diese Bilder dann der Reihe nach immer wiederholt, entsteht für den Betrachter der Eindruck einer Bewegung.

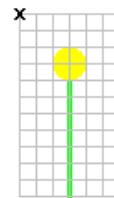
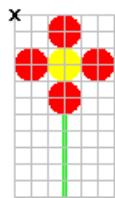
e) Vermutlich habt ihr alle ein typisch europäisches Strichmännchen gezeichnet. Ein großer Teil der Weltbevölkerung besteht aber aus Afrikanern, Asiaten, usw.. Damit auch diese Personen zu ihrem „Strichmännchen-Recht“ kommen, wird die Klasse **Strichmaennchen** so abgewandelt, dass jedes dritte Strichmännchen automatisch mit „fremdländischen“ Attributen versehen wird, z.B. eine dunklere Hautfarbe. Dazu wird in der Klasse **Strichmännchen** ein Zähler benötigt, der als Klassenvariable realisiert wird.

Aufgabe 4: Blumen

a) Programmiere in einer neuen Datei eine Klasse **Blume**.

Eine Blume besitzt zu Anfang die Attribute x-Position, y-Position und Farbe. Weitere Attribute dürfen später nach Bedarf hinzugefügt werden. Die Startwerte für die x- und y-Position werden im Konstruktor als Parameter übergeben (in dieser Reihenfolge). Die Farbe wird automatisch auf rot gestellt.

Schreibe eine Methode **zeichnen()**, die die Blume an ihrer aktuellen Position zeichnet. Das Kreuz markiert die linke obere Ecke, die durch den x- und y-Wert beschrieben wird (1 Kästchen entspricht 10 Pixeln):



Anmerkung: Zu Beginn gibt es nur „blühende“ Blumen (linkes Bild). Die vier äußeren Kreise werden in der eingestellten Blumen-Farbe ausgefüllt. Der mittlere Kreis wird gelb gefüllt. Der Stängel wird als ausgefülltes, grünes Rechteck mit drei Pixel Breite gezeichnet und geht von der Blume bis zum Boden (also bis zum unteren Rand des Fensters). Der Einfachheit halber kann man den Stängel auch über den Rand des Fensters hinaus zeichnen.

b) Erzeuge ein Anwendungsfenster mit mindestens sechs Blumen-Objekten. Füge außerdem in das Anwendungsfenster ein Objekt der Klasse **Timer** ein.

- c) Erweitere die Methode `zeichnen()` so, dass die Blume bei jedem Aufruf um ein Pixel wächst bis sie die y-Position 100 erreicht hat. Das heißt die y-Position wird immer um ein Pixel nach oben verschoben bis die angegebene Höhe erreicht ist.
- d) Sorge im Konstruktor dafür, dass die Blumen automatisch unterschiedliche Farben erhalten und zwar immer abwechselnd `Color.RED`, `Color.BLUE` oder `Color.CYAN`. Die erste und die vierte Blume müssen also rot werden. Die zweite und die fünfte Blumen werden blau und die dritte und die sechste Blume werden cyan. Selbstverständlich muss der Code so geschrieben sein, dass er auch für weitere Blumenobjekte funktioniert.
- e) Jede Blume verwelkt zwei Sekunden nachdem sie ihre maximale Höhe erreicht hat und wird dann wie oben rechts abgebildet gezeichnet.

Aufgabe 5: Verhältnis von Klassen und Objekten

Erkläre die Begriffe Klasse und Objekt anhand der folgenden Worte: Hund, Hasso, Lassie, Katze, Miezi.

Aufgabe 6: Fachbegriffe

Finde jeweils das richtige Fachwort.

| | |
|--|--|
| Festlegung von Namen und Datentyp für eine Variable | |
| Art einer Variable | |
| Variable, die im Kopf einer Methode deklariert wird | |
| Anfangswert für eine Variable setzen | |
| Wiederholungsanweisung | |
| Eigenschaft, die ein Objekt einer Klasse beschreibt | |
| Tätigkeit, die Objekte einer Klasse ausführen können | |

Aufgabe 7: Variablen-deklaration

- a) Wie deklariert man eine Variable in Java?
- b) Welches Schlüsselwort muss vor einer Variablen-deklaration stehen, wenn eine Variable ...

| | |
|--|--|
| ... nicht verändert werden darf (also konstant ist) | |
| ... von einer anderen Klasse benutzt werden darf. | |
| ... von einer anderen Klasse nicht benutzt werden darf. | |
| ... nur einmal pro Klasse vorhanden ist (also nicht für jedes Objekt). | |

Aufgabe 8: Entwurf einer Methode (1)

Programmiere auf Papier eine Methode `treffer()`, die eine Fließkommazahl als Parameter erhält und einen booleschen Wert zurück gibt. Es wird der Wert `true` zurückgegeben, wenn eine der Zahlen 1.5, 4 oder 5.5 angegeben wird. Andernfalls wird der Wert `false` zurück gegeben.

Aufgabe 9: Entwurf einer Methode (2)

Programmiere auf Papier eine Methode `summe()`, die eine ganze Zahl `x` als Parameter erhält und eine (eventuell sehr große) ganze Zahl als Rückgabewert zurück gibt. Die Methode soll alle ganzen Zahlen von zehn bis zu der übergebenen Zahl `x` aufaddieren.

Aufgabe 10: Programmierung von Klassen und Objekten

- a) Mit welchem Schlüsselwort kennzeichnet man beim Programmieren eine Klasse?
- b) Angenommen es existiert eine Klasse **Fisch**. Wie erzeugt man im Anwendungsfenster eine Variable, die auf ein Objekt dieser Klasse verweisen kann?
- c) Wie erzeugt man ein neues Objekt der Klasse im Speicher?
- d) Wann wird das Objekt wieder aus dem Speicher gelöscht?

Aufgabe 11: Verweis auf sich selbst

Eine Klasse beschreibt das Schema für eine ganze Gruppe von Objekten. Wie verweist man beim Programmieren innerhalb der Klasse auf das eigene Objekt (also quasi auf sich selber)?

Aufgabe 12: Konstruktor

- a) Was ist ein Konstruktor?
- b) Wann wird der Konstruktor einer Klasse aufgerufen?
- c) Wie programmiert man einen Konstruktor?

Aufgabe 13: Leseübung

Lies den folgenden Programmtext sorgfältig und beantworte die unten gestellten Fragen.

```
class Demo {
    public String name = "Demo";
    public double gewicht = 55.5;
    public int groesse;
    public static int alter;

    public Demo(String name, double gewicht) {
        this.name = name;
        gewicht = 87.2;
        alter = 50;
    }

    public Demo(String name, int a) {
        this.name = name;
        groesse = 183;
        alter = a;
    }

    public void drucken() {
        // Ausgabe der Daten eines Objektes auf die Konsole
        System.out.println("Name: "+name);
        System.out.println("Gewicht: "+gewicht);
        System.out.println("Groesse: "+groesse);
        System.out.println("Alter: "+alter);
        System.out.println(" ---");
    }

    public static void main(final String[] args) {
        Demo a = new Demo("Mona", 51.3);
        Demo b;
```

```

    Demo c = a;
    Demo d = new Demo ("Simson", 19);
    c.gewicht = 45.0;
    a.drucken();
    c.drucken();
    d.drucken();
}
}

```

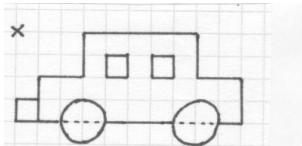
- Markiere im Code die Deklaration einer Klassenvariablen (K), einer Objektvariablen (O) und einer lokalen Variablen (L).
- Welche Methoden können benutzt werden, ohne dass es ein Objekt gibt?
- Ermittle, welche Ausgabe das folgende Programm erzeugt. Zeichne dazu den Inhalt der Objekte und Variablen im Hauptspeicher auf, um den Ablauf des Programms nachzuvollziehen.
- Was passiert wenn man `b.drucken();` aufruft?
- Was passiert bei der Anweisung: `b = new Demo();` ?

Hinweis: Du findest die Datei `Demo.java` im Kurs-Repository. Du solltest die Aufgaben aber zunächst wirklich beantworten, ohne das Programm in Eclipse auszuführen. Das kannst du dann anschließend tun um deine eigenen Antworten zu überprüfen.

Aufgabe 14: Autorennen

- Programmiere in einer neuen Datei eine Klasse `Auto`.
- Ein Auto besitzt zu Beginn die Attribute `x-Position`, `y-Position` und `Farbe`. Weitere Attribute dürfen später nach Bedarf hinzugefügt werden. Die Startwerte für den `y-Wert` und die Farbe werden im Konstruktor als Parameter übergeben (in der angegebenen Reihenfolge). Die `x-Position` wird automatisch auf den Wert 10 gesetzt.

Schreibe eine Methode `zeichnen()` (mit dem `Graphics`-Objekt als Parameter), die das Auto an seiner aktuellen Position wie abgebildet zeichnet. Zunächst werden zwei große ausgefüllte Rechtecke in der eingestellten Autofarbe gezeichnet. Anschließend werden die Fenster mit zwei ausgefüllten gelben Quadraten darüber gemalt. Danach wird der Bereich für den Auspuff und die Räder schwarz ausgefüllt. Das Kreuz markiert die linke obere Ecke, die durch den `x-` und `y-Wert` beschrieben wird. Ein Kästchen in der Abbildung entspricht zehn Pixeln:

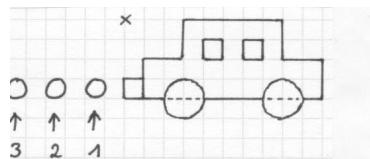


- Erzeuge ein Anwendungsfenster mit einer Breite von 800 Pixeln und einer Höhe von 500 Pixeln. Zeichne im Anwendungsfenster mehrere Objekte der Klasse `Auto`. Füge auch ein Objekt der Klasse `Timer` ein, das die `myPaint()`-Methode im Abstand von 100 Millisekunden wiederholt aufruft.
- Die Autos sollen sich nun bewegen. Füge dazu ein Attribut `speed` mit dem Datentyp `int` in die Klasse ein, und setze das Attribut im Konstruktor auf einen Zufallswert zwischen 5 und 14.

Um Zufallswerte erzeugen zu können, musst du in die Klasse `Auto` auch noch eine Objekt der Klasse `Random` einfügen. Beachte, dass dieses Objekt eine Klassenvariable sein muss, denn wenn jedes Auto seinen eigenen Zufallsgenerator hätte, würden die meisten Autos exakt dieselbe Geschwindigkeit erhalten, weil ihr Zufallsgenerator jeweils mit dem selben Anfangswert gestartet wird.

Erweitere anschließend die Methode `zeichnen()` so, dass die `x-Position` des Autos bei jedem Aufruf um den Wert `speed` nach rechts verschoben wird. Wenn die `x-Position` größer als 800 ist, soll seine `x-Position` links vor das Fenster gesetzt werden, so dass das Auto langsam in das Fenster hinein gleitet.

- e) Die Abgase des Autos sollen mit grau ausgefüllten Kreisen animiert werden. Dazu werden vier Zustände unterschieden (durchnummert von 0 bis 3), die das Auto der Reihe nach wiederholt durchläuft. Im Zustand 0 werden keinerlei Abgase gezeichnet. Im Zustand 1 wird nur der mit 1 bezeichnete Kreis gezeichnet. Im Zustand 2 werden die mit 1 und 2 bezeichneten Kreise gezeichnet. Im Zustand drei werden alle drei Kreise gezeichnet.



- f) Die Autos sollen in der Reihenfolge ihrer Erzeugung durchnummeriert werden. Das erste Auto erhält die Nummer 1. Dazu musst du eine Klassenvariable erzeugen, mit der die Autos im Konstruktor durchgezählt werden. Außerdem benötigt man eine Objektvariable, in der jedes Auto seine eigene Nummer abspeichert. In der Methode `zeichnen()` soll jedes Auto seine Nummer mit der `Graphics`-Methode `drawString()` an der Position ($x+50$, $y+35$) mit schwarzer Schrift ausgeben.
- g) Ein Autorennen dauert drei Runden. Nachdem ein Auto dreimal von links nach rechts durch das Fenster gefahren ist, soll es nicht mehr gezeichnet werden. Außerdem soll die Nummer des Autos, das als erstes „das Ziel“ erreicht, im Anwendungsfenster an der Position (300, 480) mit einem kleinen Text ausgegeben werden, z.B. „Der Sieger ist Auto 4“. Welche Klasse diese Ausgabe übernimmt, darfst du selber entscheiden.

Aufgabe 15: Sternen-Himmel

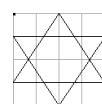
Es soll ein Sternen-Himmel programmiert werden.

- Wähle im Anwendungsfenster-Fenster die Farbe schwarz als Hintergrundfarbe aus.
- Programmiere eine Klasse `Stern` mit folgenden Attributien:
 - der Farbe des Sterns
 - der linken oberen Ecke des Sterns (x- und y-Position)
 - der Geschwindigkeit des Sterns (Veränderung der x-Richtung in Pixel pro Aufruf vom `myPaint()`)

Alle Attribute der Klasse dürfen von außen (d.h. von einer anderen Klasse aus) nicht verändert werden. Weise den Variablen in der Klasse die entsprechende Eigenschaft zu. Zur Programmierung der im nachfolgenden beschriebenen Funktionalitäten dürfen weitere Klassen- und Instanzvariablen eingefügt werden. Auch diese Variablen dürfen jedoch nicht von außen veränderbar sein.

Die Klasse `Stern` soll nur einen einzigen Konstruktor haben, in dem als Parameter die x- und y- Position der linken oberen Ecke des Sterns angegeben werden. Die Geschwindigkeit wird innerhalb des Konstruktors automatisch für alle Sterne auf den Wert 3 gesetzt. Auch die Farbe wird automatisch generiert. Jeder zweite Stern erhält die Farbe orange. Alle anderen Sterne erhalten die Farbe gelb.

Die Klasse `Stern` besitzt die öffentliche Methode `zeichnen()`, die den Stern an seiner aktuellen Position malt und ihn anschließend entsprechend der eingestellten Geschwindigkeit in der x-Position verschiebt. Entscheide selbst, ob die Methode `zeichnen()` einen Parameter benötigt oder nicht. Ein Stern wird durch zwei ausgefüllte Dreiecke gemalt, wie in der nebenstehenden Abbildung zu sehen ist. Die linke obere Ecke, die durch die x- und y-Werte beschrieben ist, ist im Bild durch einen dicken Punkt markiert. Dieser Punkt soll natürlich nicht mit gezeichnet werden. Wähle beim Zeichnen pro Kästchen eine Größe von 10 Pixel. Wenn ein Stern rechts aus dem Bild gewandert ist, soll die Methode `zeichnen()` ihn wieder auf der linken Seite ins Bild setzen.



Damit ein Stern harmonisch in das Bild hinein gleitet, wird seine neue x-Position nicht auf den Wert 0 gesetzt, sondern soweit nach links verschoben, das zunächst nur seine rechten Spalten sichtbar sind.

- Erzeuge im Anwendungsfenster mindestens vier verschiedene Objekte der Klasse `Stern` an verschiedenen

Positionen. Sorge mit Hilfe der Klasse **Timer** dafür, dass das Anwendungsfenster und die Sterne alle 100 Millisekunden neu gezeichnet werden.

- d) Die Klasse **Stern** soll um das boolesche Attribut **blinkend** erweitert werden. Auch dieses Attribut darf von außen nicht verändert werden. Der Standard-Wert für das Attribut ist **false** (ausgeschaltet). Füge in die Klasse einen zweiten Konstruktor ein, mit dem neben der x- und y-Position der Wert des Attributs **blinkend** eingestellt werden kann. Wenn **blinkend** auf **true** gesetzt wird, wird das Objekt nur bei jedem zweiten Aufruf der Methode **zeichnen()** tatsächlich gezeichnet, so dass ein Blink-Effekt entsteht. Erzeuge im Anwendungsfenster ein weiteres Objekt der Klasse **Stern**, bei dem das Attribut **blinkend** eingeschaltet ist.
- e) Hin und wieder ziehen am Himmel Wolken vorbei und verdecken die Sterne. Wenn eine Wolke da ist, sind alle Sterne nicht sichtbar und werden daher nicht gezeichnet. Ergänze die Klasse **Stern** um ein öffentliches boolesches Attribut **wolke_da**, das nur einmal für die gesamte Klasse existieren soll. Wenn eine Wolke vorhanden ist, werden die Sterne in der Methode **zeichnen()** nicht gezeichnet. Achte darauf, dass der Wolken-Effekt auch für blinkende Sterne korrekt funktioniert.
Stelle das boolesche Attribut **wolke_da** des Anwendungsfensters so ein, dass immer abwechselnd 1200 ms lang keine Wolke da ist und dann 600 ms lang eine Wolke vorüber zieht.

Aufgabe 16: Bälle (alte Klausuraufgabe)

Hinweis: In kursiver Schrift wird bei einigen Teilaufgaben eine Alternative für diejenigen vorgeschlagen, die Probleme haben, die Teilaufgabe zu lösen. Der Alternativ-Vorschlag soll euch dazu befähigen, mit den anderen Teilaufgaben weiterzumachen, damit ihr euch nicht an einem Problem fest beißt. Wer den Alternativ- Vorschlag wählt, erhält für die jeweilige Teilaufgabe natürlich nicht die volle Punktzahl.

- a) Erzeuge ein Anwendungsfenster mit einem schwarzen Hintergrund und weißer Vordergrundfarbe. Das Anwendungsfenster soll eine Breite und Höhe von je 500 Pixel besitzen.
- b) Programmiere in einer zweiten Datei eine Klasse **Ball**. Alle Attribute der Klasse **Ball** sollen vor dem Zugriff von außen (d.h. vom Anwendungsfenster aus) geschützt sein. Alle Methoden sind öffentlich und dürfen vom Anwendungsfenster aus benutzt werden.

Ein Ball besitzt zu Anfang die Attribute x-Position, y-Position, Breite und Farbe. Weitere Attribute dürfen später nach Bedarf hinzugefügt werden. Die x-Position wird automatisch auf den Wert 25 festgelegt, die y-Position auf den Wert 250 und die Breite auf den Wert 50. Nur die Farbe kann vom Anwendungsfenster aus eingestellt werden und wird im Konstruktor als Parameter übergeben.

Schreibe eine Methode **zeichnen()**, die den Ball an seiner aktuellen Position und in der für ihn gewählten Farbe als ausgefüllten Kreis zeichnet. Entscheide selbst welche Parameter und welchen Rückgabewert die Methode **zeichnen()** benötigt.

- c) Erzeuge im Anwendungsfenster ein Objekt der Klasse **Ball**, das eine rote Farbe besitzt.
- d) Füge im Anwendungsfenster ein Objekt der Klasse **Timer** ein, und stelle den Timer so ein, dass die **myPaint()**-Methode alle zehn Millisekunden aufgerufen wird.
- e) Erweitere in der Klasse **Ball** die Methode **zeichnen()**, so dass in der Methode die x-Position des Balls bei jedem Aufruf um zwei Pixel verschoben wird. Der Ball soll immer abwechselnd zunächst nach rechts rollen bis zur x-Position 425 und anschließend wieder zurück nach links bis zur x-Position 25.

Falls es dir nicht gelingt den Ball abwechselnd nach rechts und wieder nach links zu steuern, versuche den Ball – wie das Strichmännchen – immer von links nach rechts rollen zu lassen. Das heißt, das der Ball nach Erreichen der x-Position 425 wieder auf die x-Position 25 gesetzt wird. (halbe Punktzahl)

- f) Erweitere die Klasse **Ball** so, dass jedes weitere Ball-Objekt automatisch in seiner anfänglichen x- Position um 50 Pixel nach rechts verschoben wird. Das erste Ball-Objekt besitzt bei Erzeugung die x- Position 25, das zweite Ball-Objekt die x-Position 75, das dritte Ball-Objekt die x-Position 125, usw.

Erzeuge im Frame acht verschiedene Ball-Objekte, die unterschiedliche Farben besitzen sollen. Die Farben darfst du frei wählen. Wenn du alles richtig gemacht hast, bilden die Ball-Objekte eine Art „Schlange“.

Falls dir die Lösung dieser Aufgabe nicht gelingt, überspringe die Aufgabe und arbeite einfach mit einem Ball-Objekt weiter.

- g) Die Klasse `Ball` soll so erweitert werden, dass sich der Ball im Uhrzeigersinn im Kreis bewegt. Wenn der Ball sich nach rechts bewegt, wird die y-Koordinate in Abhängigkeit von der x-Koordinate folgendermaßen berechnet:

```
y = 250 - (int) Math.sqrt(40000-(x-225)*(x-225));
```

Dadurch bewegt sich der Ball im Halbkreis nach oben. Wenn sich der Ball nach links bewegt, wird die y-Koordinate mit der Formel

```
y = 250 + (int) Math.sqrt(40000-(x-225)*(x-225));
```

berechnet. Dadurch bewegt sich der Ball im Halbkreis nach unten. Beachte, dass der Ball bei dieser Technik an den Seiten links und rechts wesentlich schneller rollt als in der Mitte. Das ist kein Programmierfehler.

Vorgabe: Berechne die y-Koordinate für den Ball mit Hilfe einer Methode, die die x-Koordinate als Parameter erhält und die y-Koordinate als Rückgabewert zurück gibt. Schreibe entweder für jede der beiden Formeln eine Methode oder programmiere eine einzige Methode, die die aktuelle Richtung des Balls berücksichtigt.

Falls du nicht weißt, wie man die Methode programmiert, löse die Aufgabe ohne Verwendung einer neuen Methode. (halbe Punktzahl)

- h) Erweitere die Klasse `Ball` so, dass ein Ball zyklisch immer zweimal im Kreis rollt und dann einmal wie in Teilaufgabe (e) mit fester y-Koordinate nach rechts und wieder zurück nach links rollt. Das heißt, die Berechnung der y-Koordinate mit den Kreis-Formeln entfällt bei jedem dritten Durchgang.

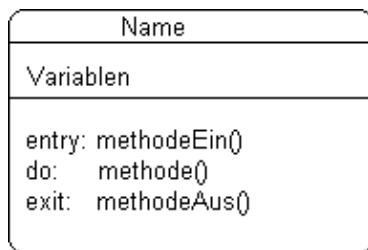
11 UML-Zustandsdiagramme

11.1 Notation

Ein UML-Zustandsdiagramm beschreibt die Zustände eines Objektes. Bei der Umsetzung des Zustandsdiagramms in ein Programm nummeriert man die einzelnen Zustände durch und fügt im Code eine Integer-Variable ein, in der die Nummer des aktuellen Zustands gespeichert wird.

Darstellung der Zustände

Schema:

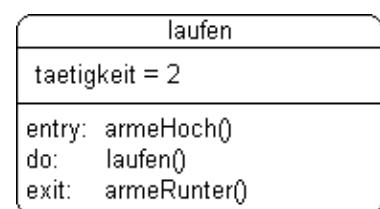
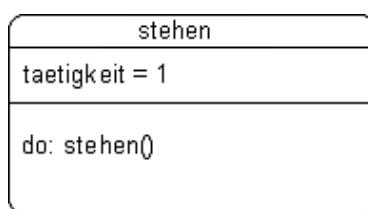


Hinter **entry** werden alle Methoden aufgelistet, die beim Eintritt in den Zustand ausgeführt werden.

Hinter **do** werden alle Methoden aufgelistet, die solange ausgeführt werden, wie der Zustand andauert.

Hinter **exit** werden alle Methoden aufgelistet, die beim Verlassen des Zustands ausgeführt werden.

Beispiele:

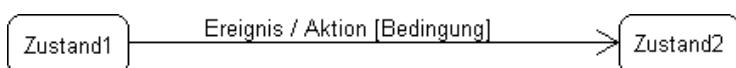


Besondere Zustände: ● Startzustand

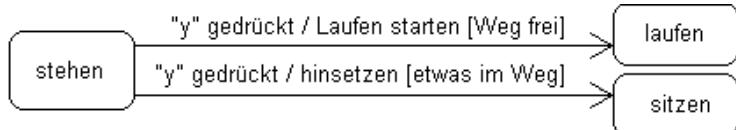
○ Endzustand

Darstellung von Zustandsübergängen

Schema:



Beispiel:



Beschriftung der Zustandsübergänge

| | |
|--------------------|--|
| Ereignis | Das System wird von außen angestoßen seinen Zustand zu ändern (in der Regel durch den Benutzer). Beispiele: Tastendruck, Klick auf einen Button, Signal von der Systemuhr |
| /Aktion | Tätigkeit, die das System während des Zustandsübergangs ausführt (z.B. Aufruf einer Methode). |
| [Bedingung] | Das System entscheidet aufgrund von eigenen Informationen (z.B. Variablenwerten), in welchen Zustand es übergeht. Eine Bedingung ist nur dann sinnvoll, wenn verschiedene Zustände zur Auswahl stehen. |

Zustand oder Zustandsübergang?

Zustandsübergänge sind extrem kurze Momente. Eine Tätigkeit, die eine längere Zeitspanne umfasst, muss als Zustand modelliert werden.

11.2 Beispiele

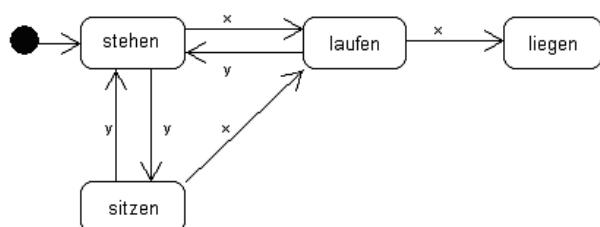
Zustandsdiagramme für das Strichmännchen

Im Kurs-Repository findest du die Dateien `Maennchen.java` und `Main.java`. Kopiere die beiden Dateien in dein eigenes Java-Projekt und starte das Programm. Du kannst das Männchen mit den Tasten 'x' und 'y' steuern.

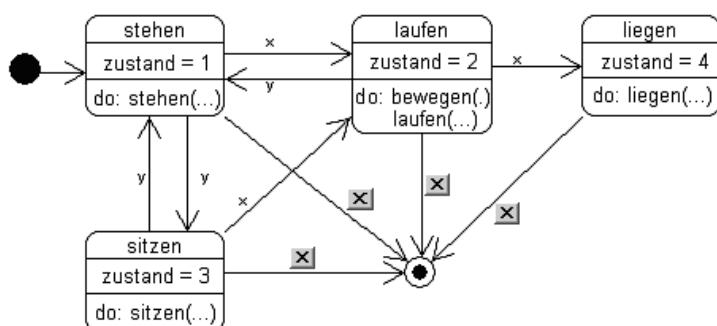


Das Verhalten des Männchens lässt sich in einem Zustandsdiagramm abbilden.

a) Einfaches Zustandsdiagramm

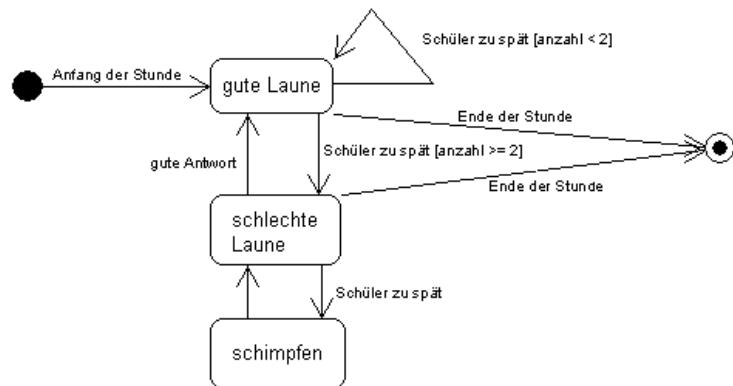


b) Zustandsdiagramm mit Details für die Programmierung



Komplexes Beispiel

Zustände eines Objekts der Klasse „Lehrer“:



11.3 UML-Zustandsdiagramme – Übungen

Aufgabe 1: Windows



Unter dem Betriebssystem Windows lässt sich der Zustand eines Programmfensters durch Knöpfe verstellen, die sich rechts oben am Fenster befinden (siehe Abbildung). Beschreibe in einem Zustandsdiagramm, wie man den Zustand eines Programmfensters verändern kann. Berücksichtige in deinem Zustandsdiagramm auch die Möglichkeit den Zustand eines Fensters über das Kontextmenü in der Taskleiste, oder auch durch einfachen Klick auf den entsprechenden Eintrag in der Taskleiste, zu verändern.

Nimm dabei an, dass das Fenster direkt nach dem Programmstart nur einen Teil des Bildschirms belegt.

Aufgabe 2: Fähre

Es soll eine Computersimulation für eine Fähre erstellt werden. Zeichne dafür ein UML-Zustandsdiagramm, das die verschiedenen Zustände der Fähre darstellt:

Die Fähre wartet zehn Minuten am östlichen Ufer bis alle Personen und Transportmittel an Bord sind. Dann fährt sie in westlicher Richtung bis zum anderen Ufer. Am West-Ufer wartet die Fähre wieder zehn Minuten, bis alle Personen und Transportmittel aus- beziehungsweise eingestiegen sind. Anschließend fährt sie in östlicher Richtung bis sie wieder den Landeplatz am Ost-Ufer erreicht hat. Dieser Zyklus wiederholt sich den ganzen Tag lang. Nachts legt die Fähre am Ost-Ufer an. Dem entsprechend startet sie am frühen Morgen vom Ost-Ufer aus.



Aufgabe 3: Seehund

In einem Computerspiel soll ein Seehund dargestellt werden. Zeichne dafür ein UML-Zustandsdiagramm, das die verschiedenen Zustände veranschaulicht, in denen sich der Seehund des Computerprogramms befinden kann:

Zu Beginn schwimmt der Seehund ziellos im Wasser herum. Sobald ein Fisch erscheint, verfolgt er diesen. Nachdem er den Fisch gefangen hat, isst er ihn (längerer Vorgang). Falls dies der dritte Fisch war, den er gefangen hat, ist er satt und ruht sich auf einer Sandbank aus. Die Seehund-Simulation ist damit zu Ende. Andernfalls schwimmt er weiter, um erneut einen Fisch zu fangen.

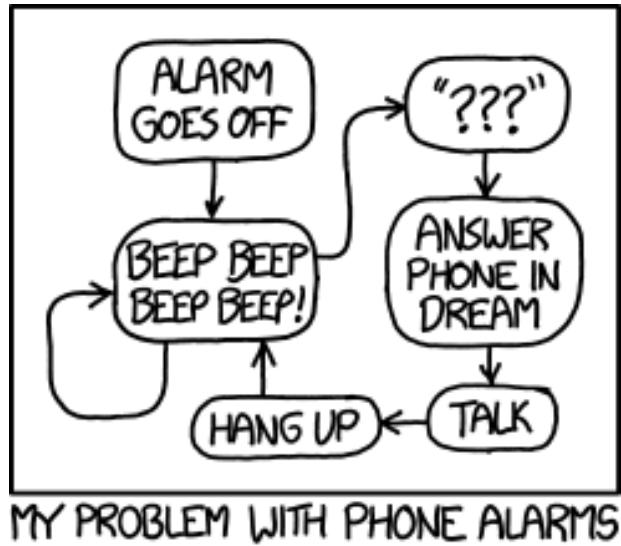
Aufgabe 4: Getränke-Automat

Schreibe ein Zustandsdiagramm, das die Funktionsweise eines Cola-Automaten graphisch darstellt:

Eine Dose kostet einen Euro. Man kann jedoch auch andere Münzen in den Automaten einwerfen. Der Automat gibt gegebenenfalls Wechselgeld zurück. Nach dem Einwurf der Münzen muss man einen Knopf drücken, damit der Automat weiß, dass die Eingabe der Münzen abgeschlossen ist. Wenn der Automat leer ist oder wenn der Benutzer zu wenig Geld eingeworfen hat, wird das Geld wieder zurück gegeben. Andernfalls gibt der Automat zuerst eine Cola-Dose aus. Anschließend gibt der Automat das Wechselgeld zurück, falls der eingeworfene Geldbetrag höher als ein Euro war.

Aufgabe 5: xkcd: My Problem With Phone Alarms

Interpretiere das folgende „Zustandsdiagramm“:



(Quelle: <http://xkcd.com/1359/>)

Werden die Zustände und ihre Übergänge eindeutig beschrieben?

12 Datenkapselung

12.1 Problembeschreibung

Gegeben ist eine Klasse `Tisch` mit folgendem Code:

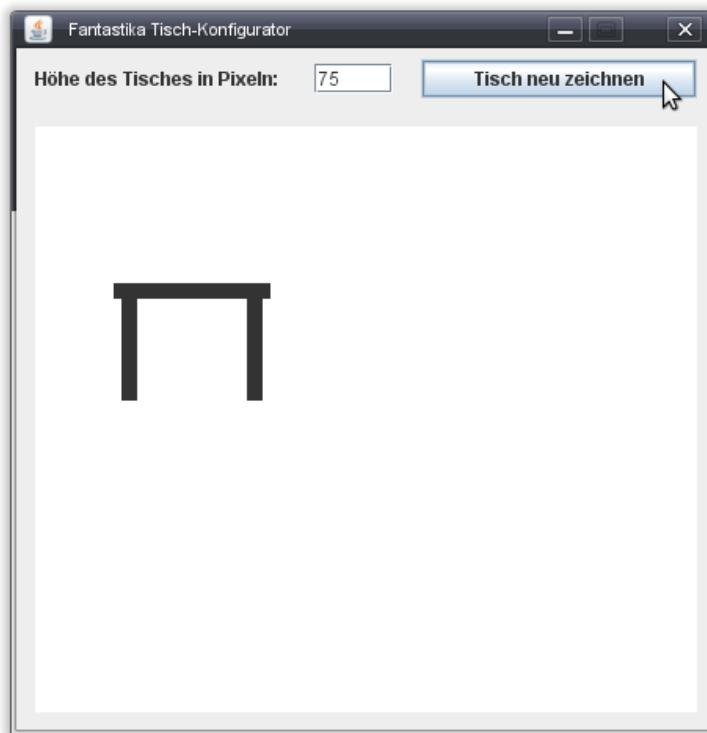
```
import java.awt.*;

public class Tisch {
    private int x, y;           // linke obere Ecke
    public int hoehe = 50;
    public int breite = 100; // Wichtig: Die Breite muss immer das Doppelte der Höhe sein!

    public Tisch(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void zeichnen(Graphics g) {
        g.fillRect(x, y, breite, 10);
        g.fillRect(x + 5, y, 10, hoehe);
        g.fillRect(x + breite - 15, y, 10, hoehe);
    }
}
```

Für alle Tische des Herstellers „Fantastika“ besteht die Vorgabe, dass die Breite immer das Doppelte der Höhe betragen muss. Ein unachtsamer Programmierer, der die Aufgabe hatte, für die Anwendung eine hübsche Bedienungsoberfläche zu schreiben, hat diese Vorgabe jedoch nicht beachtet. Die Bedienungsoberfläche bietet dem Benutzer die Möglichkeit, die Höhe des Tisches unabhängig von der Breite zu ändern:



Die Höhe wird von außen nach Eingabe des Benutzers z.B. mit folgender Anweisung geändert:

```
tisch1.hoehe = 100; // tisch1 ist ein Objekt der Klasse Tisch
```

Natürlich wird jetzt ordentlich gemeckert und der Programmierer muss die Oberfläche umschreiben. Aber es wäre ja viel schöner gewesen, wenn diese Fehlbenutzung von vornherein durch geschickte Programmierung verhindert worden wäre.

Wie kann man innerhalb der Klasse Tisch dafür sorgen, dass die Anforderung „die Breite ist das Doppelte der Höhe“ nicht verletzt werden kann, auch wenn der ahnungslose Benutzer der Klasse ausschließlich die Höhe verstellt?

12.2 Lösung: Datenkapselung

Wenn mehrere Programmierer zusammen ein großes Programm schreiben, müssen sie sich sorgfältig absprechen, damit die einzelnen Programmteile korrekt zusammen arbeiten. Jeder Programmierer entwirft für seine Programmteile eine sogenannte *Schnittstelle*. Die Schnittstelle besteht aus den öffentlichen (`public`) Methoden einer Klasse, die die anderen Programmierer benutzen können. Eine Schnittstelle sollte möglichst übersichtlich und für andere leicht verständlich sein. Bewährt hat sich dabei das Prinzip der *Datenkapselung*:

Datenkapselung hat zwei Aspekte:

1. Die bereits im Kapitel 10 beschriebene Verknüpfung von Daten und Methoden: In einer Klasse werden zum einen die Daten selbst als Attribute der Klasse definiert. Zum anderen werden in der Klasse auch alle Operatoren (Methoden) definiert, die auf Objekte dieser Klasse anzuwenden sind.
2. Das so genannte *Geheimnisprinzip*: Dass alle Attribute einer Klasse (also alle ihre Daten) vor dem Zugriff von außen versteckt werden (d.h. sie sind `private`). Andere Programmierer können die Objekte der Klasse nur über eine Reihe sorgfältig definierter Methoden verändern. Auf diese Weise wird dafür gesorgt, dass die Benutzer der Klasse nicht versehentlich oder absichtlich die Funktionalität der Klasse unterwandern. In den öffentlichen Methoden der Klasse wird sichergestellt, dass die Variablen keine verbotenen Werte annehmen können. Außerdem wird darauf geachtet, dass alle Seiteneffekte, die die Veränderung einer Variablen mit sich ziehen kann, berücksichtigt werden.

Neben der **Sicherstellung der korrekten Arbeitsweise** der Klasse bietet die Datenkapselung noch weitere Vorteile:

Benutzbarkeit: Die Schnittstelle ist für andere Programmierer übersichtlich. Sie brauchen sich nicht mit der internen Arbeitsweise der Klasse auseinander setzen (Black-Box-Prinzip).

Wartbarkeit: Wenn die interne Funktionalität der Klasse verändert wird, braucht der Code, in dem die Klasse verwendet wird, nicht ebenfalls umgeschrieben werden, sofern die Schnittstelle unverändert bleibt. Änderungen und Erweiterungen der Klasse sind dadurch problemlos möglich.

12.3 Datenkapselung – Übungen

Die folgenden Übungsaufgaben sollen nach dem Prinzip der Datenkapselung programmiert werden.

Aufgabe 1: Auf ein Haus zufahren

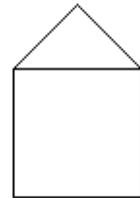
- a) Schreibe eine Klasse, die ein einfaches Haus darstellt.

Vorgaben :

Das Haus soll durch folgende private Variablen beschrieben werden:

```
private int x, y;      // x und y Position der linken unteren Ecke
private int hoehe;    // Höhe des Hauses
```

Das Hinzufügen weiterer Hilfsvariablen ist erlaubt. Auch die Hilfsvariablen müssen selbstverständlich versteckt werden.



Folgende Methoden sind notwendig:

- 1) Schreibe einen Konstruktor, durch den die Position des Hauses und die anfängliche Höhe gesetzt werden können.
 - 2) Schreibe eine Methode zum Zeichnen des Hauses mit den aktuellen Einstellungen.
 - 3) Die Höhe des Hauses soll nachträglich verändert werden können. Dabei soll jedoch die linke untere Ecke des Hauses unverändert bleiben. Außerdem soll die Breite des Hauses proportional mit verändert werden.
Schreibe eine Methode zum Auslesen der aktuellen Höhe und eine Methode zum Verändern der Höhe.
- b) Erzeuge in einem Anwendungsfenster ein Objekt der Klasse **Haus**. Zeichne das Haus in der **myPaint()**-Methode und erhöhe anschließend die Höhe z.B. um 50 Pixel. Wenn man bei der laufenden Anwendung das Anwendungsfenster kurz verdeckt und anschließend wieder sichtbar macht, ruft das System die **myPaint()**-Methode zum zweiten Mal auf und das Haus muss nun größer erscheinen.
- c) Füge in das Anwendungsfenster ein Objekt der Klasse **Timer** ein. Verändere die **myPaint()**-Methode des Anwendungsfenster so, dass das Haus nach und nach immer größer wird, so als würde man in einem Auto auf das Haus zufahren. Lese dazu bei jedem Aufruf der **myPaint()**-Methode die aktuelle Höhe des Hauses mit der entsprechenden **Haus**-Methode aus und erhöhe den Wert um 1.

Aufgabe 2: Lampe

Programmiere eine Klasse **Lampe**, die anschließend in den beiden nachfolgenden Aufgaben verwendet wird. Eine Lampe besitzt Attribute für die x-Position, die y-Position und die Farbe. Die Breite der Lampe wird mit einer Konstanten festgelegt. Außerdem besitzt die Lampe eine boolesche Variable, die angibt ob die Lampe an oder aus ist. Alle Attribute sind **private**.

Im Konstruktor der Lampe kann man die Farbe und die Position der Lampe festlegen. Es gibt Methoden zum Anschalten und Ausschalten der Lampe, die den Wert der internen Variablen entsprechend verändern. Es gibt außerdem eine Methode, mit der man abfragen kann, ob die Lampe an oder aus ist. Die Methode **zeichnen()** zeichnet die Lampe als ausgefüllten Kreis. Wenn das Licht an ist, wird die Lampe in ihrer Farbe gezeichnet. Andernfalls wird die Lampe mit grauer Farbe gezeichnet.

Erzeuge zum Test in einem Anwendungsfenster ein Objekt der Klasse **Lampe** und schalte die Lampe in einer kleinen Animation immer abwechselnd an und aus.

| Lampe | |
|--------------|---------------------------|
| ■ | farbe : Color |
| ■ | isAn : boolean |
| ■ | xPos : int |
| ■ | yPos : int |
| ◎ | Lampe(Color, int, int) |
| ◎ | an() : void |
| ◎ | aus() : void |
| ◎ | getAn() : boolean |
| ◎ | zeichnen(Graphics) : void |

Aufgabe 3: Ampel

Es ist eine Ampel zu programmieren, die in regelmäßigen Zeitabständen ihre Farben ändert: rot, rot/gelb, grün, gelb, rot, usw.. Eine Ampel besitzt drei verschiedene Objekte der Klasse **Lampe**: eine rote Lampe, eine gelbe Lampe und eine grüne Lampe. Das Ein- und Ausschalten der Lampen wird von der Ampel aus gesteuert. Die Ampel besitzt eine Methode `umschalten()`, die vom Anwendungsfenster im Sekunden-Takt aufgerufen wird. Sie schaltet dann die Farben um. Abgesehen von der Taktgebung soll die Haupt-Anwendung nichts über die Interna einer Ampel wissen müssen.



Erzeuge ein Anwendungsfenster, dass ein Objekt der Klasse **Ampel** darstellt.

Zusatzaufgabe: Erzeuge mehrere Objekte der Klasse **Ampel** und programmiere die Ampel so, dass die Ampeln mit unterschiedlichen Anfangszuständen anfangen.

Aufgabe 4: Lichterkette

Programmiere eine Klasse **Lichterkette**. Eine Lichterkette besteht aus einer Reihe verschiedenfarbiger Lampen-Objekte, die in regelmäßigen Abständen an und aus gehen. Wichtig: Die Lampen sind in der Klasse **Lichterkette** „versteckt“, d.h. sie sind private. Die Lichterkette ist für die Haupt-Anwendung eine „Black Box“, über deren Inneres sie nichts weiß.

Die Klasse **Lichterkette** besitzt einen Konstruktor, in dem die Position der Lichterkette festgelegt werden kann. Außerdem gibt es (öffentliche) Methoden zum Zeichnen der Lichterkette und zum An- und Ausschalten der Lichter.

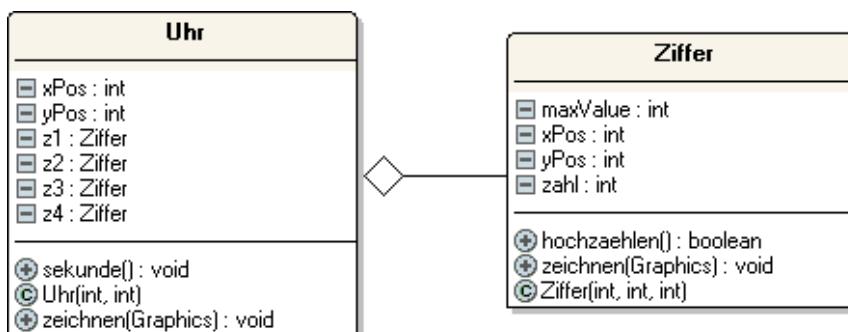
Erzeuge im Anwendungsfenster mehrere Objekte der Klasse **Lichterkette**.

Aufgabe 5: Digitaluhr

Programmiere eine Digitaluhr mit vier Ziffern: zwei Ziffern zum Zählen der Minuten und zwei Ziffern zum Zählen der Sekunden. Programmiere für die Ziffern eine eigene Klasse und erzeuge in der Klasse Digitaluhr vier Objekte der Klasse **Ziffer**.

01:40

Entwurf für die Digitaluhr



Beschreibung der Klasse **Ziffer**:

a) Variablen

```

private int xPos, yPos;      // linke untere Ecke in Pixel
private int zahl;            // die aktuelle Ziffer
private int maxValue;        // höchster Wert, den die Ziffer annehmen kann
                            // (je nach Stellung der Ziffer 5 oder 9)

```

b) Methoden

```
public Ziffer(int x, int y, int maxValue) // Konstruktor
public void zeichnen(Graphics g)           // zeichnet die Ziffer mit aktueller Zahl
public boolean hochzaehlen()                /* Zählt die Ziffer um 1 hoch. Wenn die
                                               höchste Zahl (maxValue) überschritten
                                               wird, wird die Ziffer auf 0 gesetzt.
                                               Der Rückgabewert gibt an, ob die Ziffer
                                               auf 0 gesetzt wurde (true) oder nicht
                                               (false). */
```

Beschreibung der Klasse Uhr:

a) Variablen

```
private int xPos, yPos;                  // linke obere Ecke in Pixel
private Ziffer z1, z2, z3, z4;           // die 4 Ziffern der Uhr
```

b) Methoden

```
public Uhr(int x, int y)                 // Konstruktor
public void zeichnen(Graphics g)         // zeichnet die Uhr
public void sekunde()                   // erhöht die Zeit um eine Sekunde
```


13 Grafik-Dateien

13.1 Grafik-Dateien in eigenen Java-Programmen nutzen

Unterstützte Bildformate

Java unterstützt die Bildformate GIF, JPEG und PNG. Bei GIF-Bildern kann man mit einem Bildbearbeitungsprogramm wie z.B. Gimp oder Photoshop einen transparenten Hintergrund setzen, der beim Zeichnen nicht mit gemalt wird.

Erzeugung eines Image-Objektes

Für Bildern gibt es die Klasse `Image`.

Um ein Objekt der Klasse `Image` zu erzeugen, benötigt man zunächst das `Toolkit` der aktuellen Umgebung. Die Klasse `JFrame` (das Anwendungsfenster) besitzt eine Methode, um das `Toolkit` zu beschaffen:

```
public Toolkit getToolkit()
```

Das `Toolkit` wiederum besitzt eine Methode zum Erstellen eines `Image`-Objektes:

```
public Image getImage(String datei)
```

Ein `Image`-Objekt für die Datei `bild.gif` kann man im Anwendungsfenster z.B. folgendermaßen erzeugen:

```
Image bild1;
Toolkit kit = getToolkit();
bild1 = kit.getImage("bild.gif");
```

Verkürzt kann man auch schreiben:

```
Image bild2;
bild2 = getToolkit().getImage("name.gif");
```

Zu bevorzugen ist jedoch eine zweite Variante von `getImage()` – dabei wird statt einen Strings eine URL übergeben:

```
public Image getImage(URL url)
```

Die benötigte URL bekommt man auf den ersten Blick umständlich – dafür in der Praxis sehr verlässlich – über den folgenden Aufruf:

```
bild2 = getToolkit().getImage(getClass().getResource("name.gif"));
```

Zur Erklärung: `getClass()` liefert die Klasse des aufrufenden Objektes zurück. Über `getResource()` wird die Datei, die als String angegeben wurde, als URL-Objekt zurück gegeben. Das funktioniert selbst dann, wenn diese Dateien aus einer .jar Datei geladen werden. Wenn die Bild-Datei in einem Unterverzeichnis relativ zu der aufrufenden Java-Klasse liegt, dann kann dies mit angegeben werden. Beispielsweise:

```
bild2 = getToolkit().getImage(getClass().getResource("images/name.gif"));
```

Da das Laden einer Bitmap sehr lange dauern kann, sollte die Bitmap nur einmal im Konstruktor geladen werden. Um sicher zu stellen, dass die Bilder vollständig geladen sind ehe die Anwendung zum ersten Mal gezeichnet wird, sollte man anschließend im Konstruktor noch ein Objekt der Klasse `MediaTracker` benutzen:

```
MediaTracker mt = new MediaTracker(this);      // this ist das Anwendungsfenster
mt.addImage(bild1, 0);
mt.addImage(bild2, 1);
try {
    mt.waitForAll();
} catch (Exception e) {
    e.printStackTrace();
}
```

Um zu überprüfen, ob es beim Laden eines der Bilder zu einem Fehler kam, sollte man anschließend noch genau dies mit der Methode `isErrorAny()` tun:

```
if (mt.isErrorAny()) {
    System.out.println("Problem beim Laden eines Bildes!");
}
```

Malen eines Image-Objektes

Objekte der Klasse `Image` können mit der Methode `drawImage()` der Klasse `Graphics` angezeigt werden:

```
public boolean drawImage(Image img, int x, int y, ImageObserver observer)
```

Neben dem `Image`-Objekt wird die Position der linken oberen Ecke (`x, y`) angegeben. Als letzter Parameter wird ein Objekt des Fensters angegeben, das den Ladezustand der Bitmap überwacht. Bei uns ist dies immer das Objekt des Anwendungsfensters.

Beispiel-Programm

Datei HausAnwendung.java:

```
import java.awt.*;
import hilfe.*;

public class HausAnwendung extends HJFrame {
    private static final int WIDTH = 500;
    private static final int HEIGHT = 500;
    private static final Color BACKGROUND = new Color(199, 231, 203); // blassgrün
    private static final Color FOREGROUND = Color.BLACK;
    private Haus h1, h2, h3, h4;

    public HausAnwendung(final String title) {
        super(WIDTH, HEIGHT, BACKGROUND, FOREGROUND, title);
        h1 = new Haus(100, 50, this);
        h2 = new Haus(300, 300, this);
        h3 = new Haus(50, 250, this);
        h4 = new Haus(200, 100, this);
    }

    public void myPaint(Graphics g) {
        h1.zeichnen(g);
        h2.zeichnen(g);
        h3.zeichnen(g);
        h4.zeichnen(g);
    }

    public static void main(final String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    HausAnwendung anwendung = new HausAnwendung("HausAnwendung");
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```
 }  
 }
```

Datei Haus.java:

```
import java.awt.Graphics;  
import java.awt.Image;  
import java.awt.MediaTracker;  
  
public class Haus {  
    private HausAnwendung anwendung;  
    private int x, y;  
    private Image hausBild;  
  
    public Haus (int x, int y, HausAnwendung anwendung) {  
        this.x = x;  
        this.y = y;  
        this.anwendung = anwendung;  
        hausBild = anwendung.getToolkit().getImage(getClass().getResource("haus.gif"));  
        MediaTracker mt = new MediaTracker(anwendung);  
        mt.addImage(hausBild, 0);  
        try {  
            mt.waitForAll();  
        } catch (Exception e) {}  
        if (mt.isErrorAny()) {  
            System.out.println("Problem beim Laden eines Bildes!");  
        }  
    }  
  
    public void zeichnen (Graphics g) {  
        g.drawImage(hausBild, x, y, anwendung);  
    }  
}
```



Anmerkung:

Die Methode `getToolkit()` kann in diesem Beispiel nicht direkt aufgerufen werden, weil sie nur auf Fenster (genauer: Komponenten) anzuwenden ist. Die Klasse, die bei uns das Fenster liefert ist die Klasse `HausAnwendung`. Deshalb wird in diese Beispiel das Objekt der Anwendungsklasse gebraucht, um die Methode `getToolkit()` benutzen zu können. Wir brauchen ja das Toolit der Anwendungsklasse!

13.2 Grafik-Dateien – Übungen

Aufgabe 1: Himmel mit Vögeln und Wolken

a) Erstelle ein Anwendungsfenster mit einem blauen Hintergrund. Das Anwendungsfenster soll eine Breite von 800 Pixeln und eine Höhe von 600 Pixeln erhalten.

b) Suche dir aus dem Kursverzeichnis ein Wolkenbild aus, das dir gefällt, und kopiere es in dein Arbeitsverzeichnis. Lade dieses Wolkenbild im Anwendungsfenster. Programmiere eine eigene Klasse Wolke, die eine einzelne Wolke an einer bestimmten Stelle zeichnet.

Im Konstruktor der Klasse werden als Parameter die x- und y-Position der Wolke und ein Verweis auf das Anwendungsfenster übergeben. Programmiere eine Methode `zeichnen()`, die die Wolke zeichnet.

Erzeuge im Anwendungsfenster mehrere Objekte der Klasse `Wolke`.

c) Füge in das Anwendungsfenster ein Objekt der Klasse `Timer` ein und stelle eine Wiederholungsrate von 150 Millisekunden ein.

d) Kopiere alle Gans-Bilder aus dem Kursverzeichnis in dein Arbeitsverzeichnis und lade die Bilder im Anwendungsfenster.

Programmiere eine Klasse `Gans`, die eine einzelne Gans wiederholt von links nach rechts durch das Fenster fliegen lässt. Die Klasse erhält im Konstruktor die x-Position, die y-Position, die Geschwindigkeit und einen Verweis auf das Anwendungsfenster als Parameter. Sie besitzt außerdem eine Methode `zeichnen()`, die die Gans zeichnet und bewegt. Verwende zum Zeichnen immer abwechselnd eines der vier verschiedenen Gänse-Bilder. Dann wirkt es so, als würde die Gans mit den Flügeln schlagen.

Erzeuge im Anwendungsfenster mehrere Objekte der Klasse `Gans`.

e) Kopiere die beiden Vogel-Bilder aus dem Kursverzeichnis in dein Arbeitsverzeichnis und lade die Vogel- Bilder im Anwendungsfenster.

Programmiere eine Klasse `Vogel`. Im Konstruktor wird als Parameter nur ein Verweis auf das Anwendungsfenster übergeben. Die anfängliche x-Position wird automatisch auf den Wert 500 eingestellt. Die anfängliche y-Position wird auf 500 gestellt. In der Methode `zeichnen()` wird immer abwechselnd eines der beiden Vogel-Bilder angezeigt, so dass es so wirkt, als würde der Vogel mit den Flügeln schlagen. Verschiebe außerdem die x-Position bei jedem Aufruf von `zeichnen()` um 15 Pixel nach links und die y-Position um drei Pixel nach oben. Dann fliegt der Vogel nach links oben. Wenn der Vogel links aus dem Fenster heraus geflogen ist, soll seine x-Position auf den Wert 800 gesetzt werden, so dass er erneut in das Fenster hinein fliegen kann. Die y-Position wird nicht verändert.

Erzeuge im Anwendungsfenster ein einzelnes Objekt der Klasse `Vogel`.

Aufgabe 2: Schiffe auf dem Meer

a) Kopiere die drei Schiff-Bilder aus dem Kursverzeichnis in dein Arbeitsverzeichnis. Erstelle ein Anwendungsfenster mit weißem Hintergrund und lade die drei Schiff-Bilder.

Programmiere eine Klasse `Schiff`. Im Konstruktor erhält die Klasse die x- und y-Position eines Schiffes, die Geschwindigkeit, das Anwendungsfenster und ein bestimmtes Schiff-Bild als Parameter übergeben. Programmiere eine `zeichnen()`-Methode, die das Schiff mit seinem speziellen Bild zeichnet (zunächst noch ohne Bewegung).

Erzeuge im Anwendungsfenster mindestens drei verschiedene Objekte der Klasse `Schiff`, die unterschiedliche Bilder besitzen. Die Schiffe, die nach links zeigen, erhalten einen negativen Geschwindigkeitswert.

b) Erweitere die Klasse `Schiff` so, dass sich die Schiffe je nach Ausrichtung entweder nach links oder nach rechts bewegen. Wenn ein Schiff aus dem Fenster hinaus fährt, soll es auf der gegenüberliegenden Seite wieder in das Fenster hinein gleiten.

c) Erweitere die Klasse `Schiff` so, dass die Schiffe Wellenbewegungen ausführen. Dies kann man simulieren in dem man die Schiffe pixelweise zuerst zehn Pixel nach oben verschiebt und anschließend wieder zehn Pixel nach unten. Dadurch fahren die Schiffe in einer Zickzacklinie, die wie eine Wellenbewegung wirkt.

14 Sound

Analog zum Laden von Bildern können in Java auch Sound-Dateien geladen werden.

14.1 Wiedergabe von *.wav Dateien

```
import java.applet.Applet;
import java.applet.AudioClip;
AudioClip sound = Applet.newAudioClip(getClass().getResource("ok.wav"));
```

Anschließend kann man das `AudioClip` Objekt wahlweise mit `play()` einmalig, oder mit `loop()` sich ständig wiederholend abspielen lassen.

Mit `stop()` lässt sich die Wiedergabe stoppen. Der Programmablauf wartet nicht auf `play()` bzw. `loop()` sondern wird sofort fortgesetzt.

Komplettes Beispiel:

```
import java.applet.Applet;
import java.applet.AudioClip;
import java.awt.BorderLayout;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

public class PlayWAV extends JFrame {
    private JPanel contentPane;

    public PlayWAV() {
        super("PlayWAV");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new BorderLayout(0, 0));
        setContentPane(contentPane);
        AudioClip sound = Applet.newAudioClip(getClass().getResource("ok.wav"));
        sound.play();
        //sound.loop();
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    PlayWAV frame = new PlayWAV();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

14.2 Wiedergabe von *.mp3 Dateien

```
import java.awt.BorderLayout;
import java.awt.EventQueue;
import java.util.concurrent.CountDownLatch;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javafx.embed.swing.JFXPanel;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;

public class PlayMP3 extends JFrame {
    private JPanel contentPane;

    public PlayMP3() {
        super("PlayMP3");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new BorderLayout(0, 0));
        setContentPane(contentPane);
        playClip();
    }

    public void playClip() {
        Media clip = new
            Media(getClass().getResource("Every_OS_Sucks.mp3").toString());
        MediaPlayer mediaPlayer = new MediaPlayer(clip);
        mediaPlayer.play();
    }

    public static void main(String[] args) {
        final CountDownLatch latch = new CountDownLatch(1);
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                new JFXPanel(); // initializes JavaFX environment
                latch.countDown();
            }
        });
        try {
            latch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    PlayMP3 frame = new PlayMP3();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

Für die MP3-Wiedergabe muss auf das noch recht neue JavaFX-Framework zurückgegriffen werden. Die nötige Laufzeitumgebung wird erst seit JDK 7 Update 6 standardmäßig mit installiert. Die Bibliothek `jfxrt.jar` muss in Eclipse noch in das Projekt eingebunden werden. Seit JDK 8 ist das nicht mehr nötig.

So können übrigens nicht nur `*.mp3` sondern auch `*.wav` Dateien wiedergegeben werden. Wenn man vorab also noch nicht weiß welches Dateiformat später zum Einsatz kommt, ist man so auf der sicheren Seite.

Wenn es andererseits klar ist, dass man mit `*.wav` Dateien auskommt, dann ist der zuerst vorgestellte Weg über `AudioClip` der deutlich einfachere.

15 UML-Klassendiagramme

15.1 Allgemeine Beziehungen: Assoziation

Beispiel für die Beziehung zwischen der Klasse **Auto** und der Klasse **Person**:



Eine Beziehung zwischen zwei Klassen kennzeichnet man durch eine Verbindungsline. Über die Verbindungsline schreibt man den Beziehungsnamen (im Beispiel „besitzen“). Das UML-Fachwort für eine einfache Verbindung ist *Assoziation*.

Multiplizität

Die sogenannte Multiplizität gibt an, wie viele Objekte einer Klasse mit einem Objekt der gegenüberliegenden Klasse verbunden sein können. Die Multiplizität steht oberhalb der Verbindungsline direkt neben der zugehörigen Klasse.

Notation

- 1 bedeutet genau 1
- 1..4 bedeutet 1 bis 4
- * bedeutet beliebig viele (auch 0)
- 1..* bedeutet beliebig viele aber mindestens 1

Beispiel

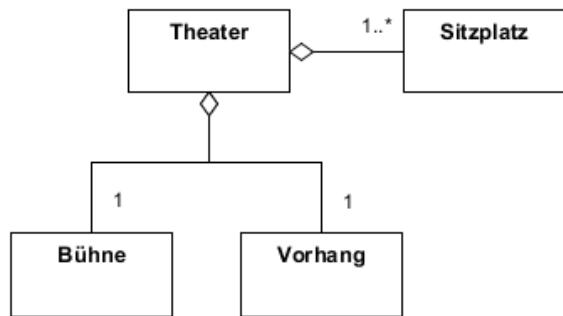


Die Angabe der Multiplizität ist folgendermaßen zu interpretieren: Linke Seite: „Ein Kunde kann beliebig viele Konten besitzen (auch 0).“ Rechte Seite: „Jedes Konto gehört einem oder zwei Personen.“

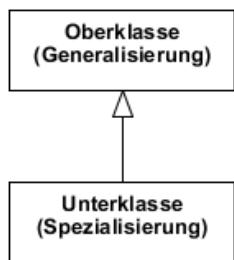
15.2 „Ist-Teil-von“-Beziehung: Aggregation

Sehr häufig sind „Ist-Teil-von“-Beziehungen. Zum Beispiel sind Sitzplätze, Vorhang und Bühne Teile eines Theaters. Ein anderes Beispiel ist eine Adresse, die man in die Bestandteile Name, Straße und Ortsangabe zerlegen kann. Die „Ist-Teil-von“ Beziehungen bezeichnet man als Aggregation.

Weil die „Ist-Teil-von“-Beziehung so häufig vorkommt, gibt es für Aggregationen eine spezielle Notation. Die Beziehungslinie wird an der Seite des „Ganzen“ mit einer Raute versehen.



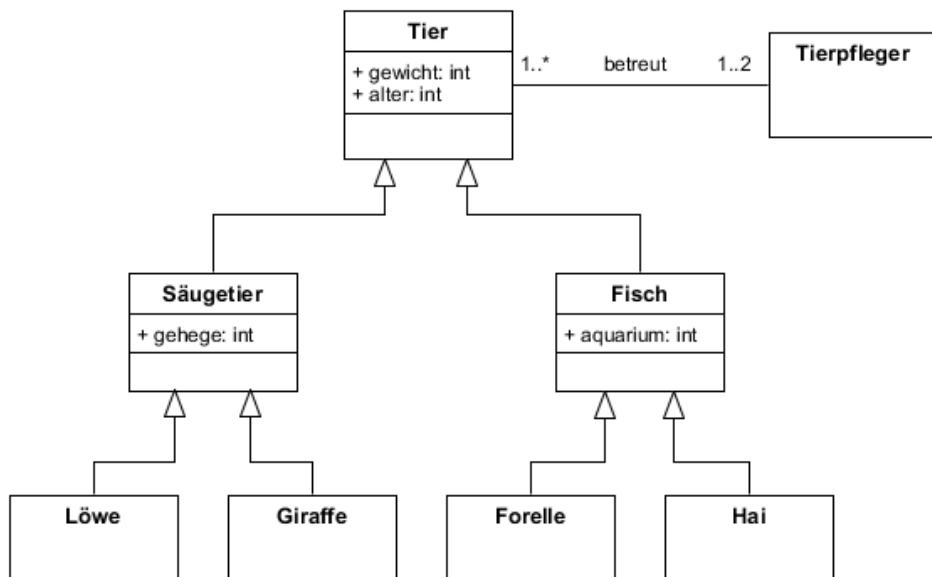
15.3 „Ist-ein“-Beziehung: Vererbung



Häufig kommt es vor, dass eine Klasse einen Spezialfall einer anderen Klasse darstellt. Z.B. sind die Klassen „Säugetier“ und „Fisch“ Spezialfälle der Klasse „Tier“. Die Klassen „Löwe“ und „Giraffe“ sind wiederum Spezialisierungen der Klasse „Säugetier“. Gleichzeitig sind sie natürlich auch Spezialisierungen der darüber liegenden Klasse „Tier“. Zwischen der Spezialklasse und der allgemeineren Klasse besteht also eine „ist-ein“-Beziehung. In UML kennzeichnet man eine solche Beziehung, indem man einen Pfeil von der spezielleren Klasse zu der allgemeineren Klasse zieht. Die spezielle Klasse bezeichnet man als *Unterklasse*, die allgemeine Klasse als *Oberklasse*.

Klassenhierarchie

Beispiel für die *Klassenhierarchie* der Tiere in einem Zoo:



In der Oberklasse *Tier* sind alle Attribute, Methoden und Assoziationen eingetragen, die für alle Tiere gemeinsam gelten. In den Unterklassen brauchen diese gemeinsamen Eigenschaften nicht noch einmal eingetragen werden. In den Unterklassen werden nur die Eigenschaften eingetragen, die für den entsprechenden Spezialfall zutreffen. Z.B. besitzen alle Tiere ein Gewicht und ein Alter. Jedes Tier wird von einem Tierpfleger betreut. Säugetiere

und Fische unterscheiden sich jedoch dadurch, dass sie in unterschiedlichen „Unterkünften“ leben. Ein Säugetier besitzt daher das zusätzliche Attribut „Gehege“, während ein Fisch das Attribut „Aquarium“ besitzt. Ein Löwe erbt von der Oberklasse „Tier“ die Attribute Gewicht und Alter und die Assoziation mit dem Tierpfleger. Er besitzt auch das Attribut „Gehege“, das er durch seine direkte Oberklasse „Säugetier“ erbt. Ein Löwe hat jedoch nicht das Attribut „Aquarium“, da er kein Fisch ist.

Da Attribute, Methoden und Beziehungen von der Oberklasse an die Unterklasse vererbt werden, bezeichnet man „ist-ein“-Beziehungen auch als *Vererbungsbeziehungen*.

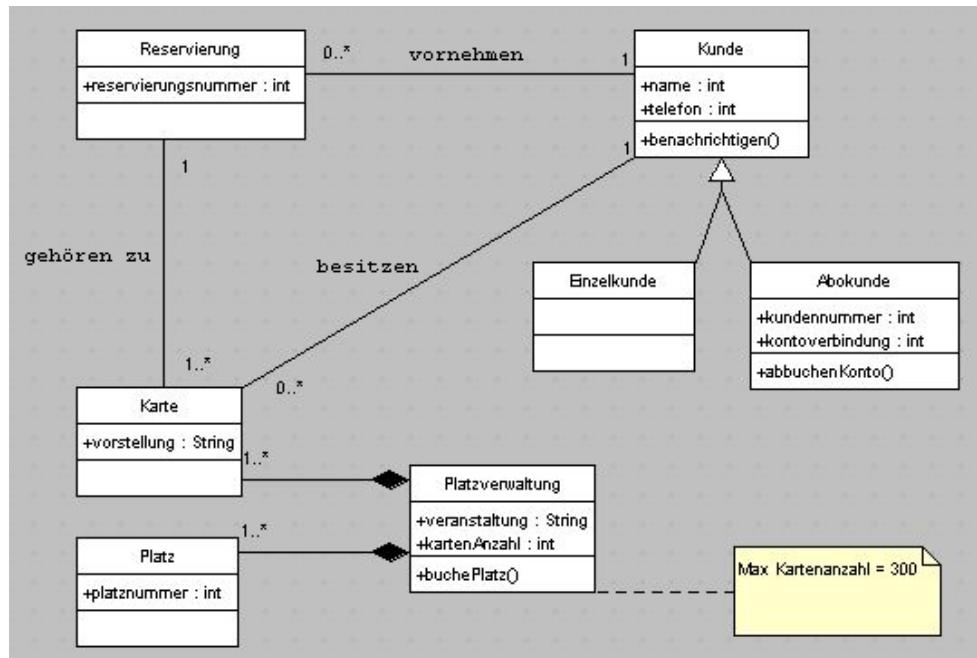
Abstrakte Klassen

Ein UML-Diagramm wird dadurch übersichtlich, dass man die Gemeinsamkeiten von verschiedenen Klassen in Oberklassen zusammen fasst. Häufig gibt es jedoch keine konkreten Objekte von den angelegten Oberklassen. In dem Zoo-Beispiel gibt es z.B. kein Objekt von der Klasse Säugetier, weil die konkreten Objekte immer den spezielleren Klassen „Löwe“, „Giraffe“, usw. zugeordnet werden müssen. Wenn keine konkreten Objekte für eine Oberklasse existieren können(!), sagt man die Klasse ist abstrakt.

15.4 UML-Klassendiagramme – Übungen

Aufgabe 1: Kartenverkauf im Theater

Interpretiere das folgende Klassendiagramm:



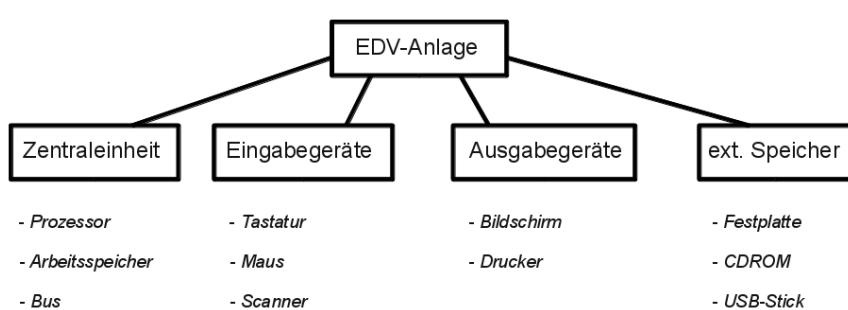
Aufgabe 2: Assoziationen

Zeichne die Assoziationen zwischen folgenden Klassen (Attribute und Methoden brauchen nicht angegeben werden):

- a) Ampel, Lampe
 - b) Baum, Birke, Buche
 - c) Auto, Mensch, Reifen, Lenkrad, BMW
 - d) Tisch, Stuhl, Tafel, Raum
 - e) Schule, Schüler, Lehrer, Klassenraum, Direktor (ein Direktor ist der Lehrer, der die Schule leitet)

Aufgabe 3: Bestandteile eines Computers

Beschreibe in einem Klassendiagramm den Aufbau eines Computers. Die nachfolgende Abbildung soll dafür als Gedächtnissstütze dienen.



Aufgabe 4: Möbelfirma

Es soll ein EDV-System für eine Möbelfirma entwickelt werden. Das EDV-System soll sowohl die Kunden wie auch die zur Verfügung stehenden Artikel verwalten.

- a) Ein Mitarbeiter der Möbelfirma gibt die nachfolgende Erklärung zu den Anforderungen an das benötigte EDV-System:

Jeder Artikel wird durch seine Artikelnummer gekennzeichnet. Das EDV-System muss abspeichern, wie viele Stücke eines Artikels zur Zeit im Lager verfügbar sind. Kunden können einen oder mehrere Artikel (in beliebiger Stückzahl) bestellen. Wenn der Artikel nicht im Lager vorhanden ist, wird er bei einem Lieferanten bestellt. Damit dies automatisch geschehen kann, muss das EDV-System abspeichern, welcher Artikel von welchem Lieferanten geliefert wird. Jeder Artikel wird von nur einem Lieferanten geliefert. Von den Lieferanten müssen Firmenname, Adresse und Telefonnummer bekannt sein. Zur Bestellung von Artikeln muss das System automatisch ein Fax an einen Lieferanten schicken können. Von den Kunden werden der Name, die Adresse und die Telefonnummer abgespeichert. Zu Werbezwecken muss automatisch ein Brief an einen Kunden geschrieben werden können.

Entwirf ein Klassendiagramm, das die Beschreibung grafisch darstellt. Gehe dabei in folgenden Teilschritten vor:

1. Entscheide, welche Klassen es gibt.
2. Trage die Attribute und die Methoden jeder Klasse ein.
3. Beschreibe die Beziehung zwischen den Klassen.

- b) Zu den Artikeln erklärt der Mitarbeiter der Möbelfirma noch weitere Details. Erweitere das Klassendiagramm entsprechend:

Die Artikel werden nach Holzmöbeln und Polstermöbel sortiert. Für Holzmöbel muss die Holzart abgespeichert werden. Bei Polstermöbeln ist dagegen zwischen verschiedenen Stoffmustern zu unterscheiden. Zu den Holzmöbeln zählt z.B. ein Tisch. Für einen Tisch werden die Länge und die Breite abgespeichert. Ein weiteres Holzmöbelstück ist ein Stuhl. . . . Ein spezielles Polstermöbelstück ist das Sofa. Für ein Sofa werden die Länge und die Breite angegeben. . . .

Aufgabe 5: Bank

Es soll ein EDV-System für eine Bank entwickelt werden. Ein Mitarbeiter der Bank gibt die nachfolgende Erklärung zu der Verwaltung von Konten für die Kunden der Bank. Identifizierte anhand der Beschreibung Klassen, Attribute, Methoden, Assoziationen und Vererbungshierarchien und zeichne sie in ein Klassendiagramm ein:

Eine Person wird Kunde, wenn sie ein Konto eröffnet. Ein Kunde kann beliebig viele weitere Konten eröffnen. Für jeden neuen Kunden werden dessen Name, Adresse und das Datum der ersten Kontoeröffnung erfasst. Bei der Kontoeröffnung muss der Kunde gleich eine erste Einzahlung vornehmen. Wir unterscheiden Girokonten und Sparkonten. Girokonten dürfen bis zu einem bestimmten Betrag überzogen werden. Für jedes Konto wird ein individueller Habenzins, für Girokonten auch ein individueller Sollzins festgelegt; außerdem besitzt jedes Konto eine eindeutige Kontonummer. Für jedes Sparkonto wird die Art des Spars - z.B. Festgeld - gespeichert. Ein Kunde kann Beträge einzahlen und abheben. Des Weiteren werden Zinsen gutgeschrieben und bei Girokonten Überziehungszinsen abgebucht. Um die Zinsen zu berechnen, muss für jede Kontobewegung das Datum und der Betrag notiert werden. Die Gutschrift/Abbuchung der Zinsen erfolgt bei Sparkonten jährlich und bei den Girokonten quartalsweise. Ein Kunde kann jedes seiner Konten wieder auflösen. Bei der Auflösung des letzten Kontos hört er auf, Kunde zu sein.

Aufgabe 6: Werkstatt

Es soll ein EDV-System für eine Werkstatt entwickelt werden. Das EDV-System soll sowohl die Kunden wie auch die in Reparatur gegebenen Fahrzeuge verwalten. Entwirf ein Klassendiagramm, das die nachfolgende Beschreibung grafisch darstellt:

Eine Person wird Kunde, wenn sie zum ersten Mal ein Fahrzeug zur Reparatur bringt. Ein Kunde kann beliebig viele Fahrzeuge zur Reparatur bringen. Für jeden neuen Kunden werden dessen Name, Adresse und das Datum der ersten Reparatur erfasst.

Wir unterscheiden bei den Mitarbeitern zwischen Meister, Büroangestellten und Mechanikern und bei den Fahrzeugen zwischen Autos und Motorrädern. Für jedes Fahrzeug wird die Marke, die Farbe und das Kennzeichen gespeichert. Für Autos wird zusätzlich auch die Anzahl der Türen festgehalten. Kunden, die Fahrzeuge zur Reparatur bringen, bekommen standardmäßig ein halbes Jahr Garantie, sie können sich aber auch eine längere Garantiezeit kaufen. Für die Reparatur muss ein Termin vereinbart werden. Die Reparaturkosten variieren, während die Garantiekosten gestaffelt sind.

16 Vererbung

16.1 Wie man eine Klasse ableitet

Bei der Klassendeklaration kann man mit dem Schlüsselwort `extends` eine Oberklasse angeben, von der die neue Klasse alle Variablen und Methoden erbt:

```
[public] class KlassenName [extends Super-Klasse] {
    // ...
}
```

[...] bedeutet, dass die Angaben zwischen den eckigen Klammern optional sind.

Wenn man keine Oberklasse angibt, dann wird die Klasse automatisch von der Klasse `Object` aus dem Package `java.lang` abgeleitet. Diese Klasse stellt einige Java-Standardfunktionalitäten bereit.

16.2 Überschreiben von Methoden

Die neue Klasse erbt automatisch alle Methoden der Oberklasse und kann neue Methoden dazu definieren. Sie kann jedoch auch die Arbeitsweise von Methoden der Oberklasse verändern, in dem sie sie überschreibt. Eine Methode wird überschrieben, in dem in der abgeleiteten Klasse exakt dieselbe Methode noch einmal deklariert wird. Name, Parameter und Rückgabewert müssen dabei übereinstimmen.

Werden nun beide Methoden ausgeführt (die Methode in der Oberklasse und die Methode in der abgeleiteten Klasse)?

Nein. Es wird nur die Methode in der abgeleiteten Klasse ausgeführt. Wenn ein Programmierer eine Methode auf ein Objekt anwendet, dann sucht das Java-System zunächst in der Klasse des Objektes nach einer entsprechenden Methode. Wenn es diese findet, wird die Methode ausgeführt und die Aufgabe ist erledigt. Wenn die Methode jedoch in der eigentlichen Klasse nicht vorhanden ist, dann sucht das System in der Oberklasse nach einer entsprechenden Methode und führt diese aus. Ist auch dort keine Methode definiert, wird die Ober-Oberklasse durchsucht, usw.

Was ist aber, wenn man die Methode der Oberklasse nur erweitern möchte? Muss man den ganzen Code neu schreiben, oder kann man die Methode der Oberklasse irgendwie aufrufen?

Wenn man die Methode der Oberklasse nicht völlig ändern sondern nur erweitern möchte, sollte man auf jeden Fall die Methode aus der Oberklasse aufrufen. Wir haben ja gelernt, dass derselbe Code nicht an mehreren Stellen im Programm stehen sollte. Mit folgendem Befehl kann man explizit eine Methode aus der Oberklasse aufrufen:

```
super.methode(parameter);
```

Noch ein paar Besonderheiten

- Methoden mit dem Schlüsselwort `static` können nicht überschrieben werden, da sie sich ja auf die gesamte Klasse und nicht auf einzelne Objekte beziehen.
- Man kann eine Methode in der Oberklasse mit `final` markieren, um zu verbieten, dass sie in abgeleiteten Klassen überschrieben wird.

Konstruktor

Mit folgendem Befehl kann man innerhalb eines Konstruktors den Konstruktor der Oberklasse aufrufen:

```
super(parameter);
```

Konstruktoren werden nicht wie andere Methoden vererbt. Wenn man in der Oberklasse einen Konstruktor mit Parametern definiert hat, so ist dieser nicht automatisch in der abgeleiteten Klasse vorhanden.

Die Regeln für Konstruktoren lauten folgendermaßen:

1. Wenn in einer Klasse kein Konstruktor definiert ist, dann erzeugt das System automatisch einen Standardkonstruktor ohne Parameter, der folgendermaßen aussieht:

```
Klasse() {
    super();
}
```

2. Falls man einen oder mehrere Konstruktoren selbst geschrieben hat, dann existiert der parameterlose Konstruktor (Standardkonstruktor) nicht, wenn man ihn nicht explizit geschrieben hat.
3. Wenn man einen eigenen Konstruktor schreibt, dann muss man in der ersten Zeile des Konstruktors den gewünschten Konstruktor der Oberklasse aufrufen. Wenn man dies nicht tut, dann wird automatisch der parameterlose Konstruktor der Oberklasse aufgerufen.

Achtung: Falls in der Oberklasse gar kein parameterloser Konstruktor definiert ist, erhält man einen Compiler-Fehler!

16.3 Datenkapselung: vollständiger Überblick

Bisher kennen wir für die Datenkapselung nur die Schlüsselworte `public` und `private`. Mit `private` gekennzeichnete Methoden können jedoch in einer abgeleiteten Klasse nicht benutzt werden. Wie kann man dafür sorgen, dass eine Methode zwar von abgeleiteten Klassen benutzt werden kann, aber für andere Klassen (zumindest aus anderen Packages) versteckt ist? Antwort: man benutzt das Schlüsselwort `protected`.

Die folgende Tabelle gibt einen vollständigen Überblick über die Möglichkeiten der Datenkapselung in Java:

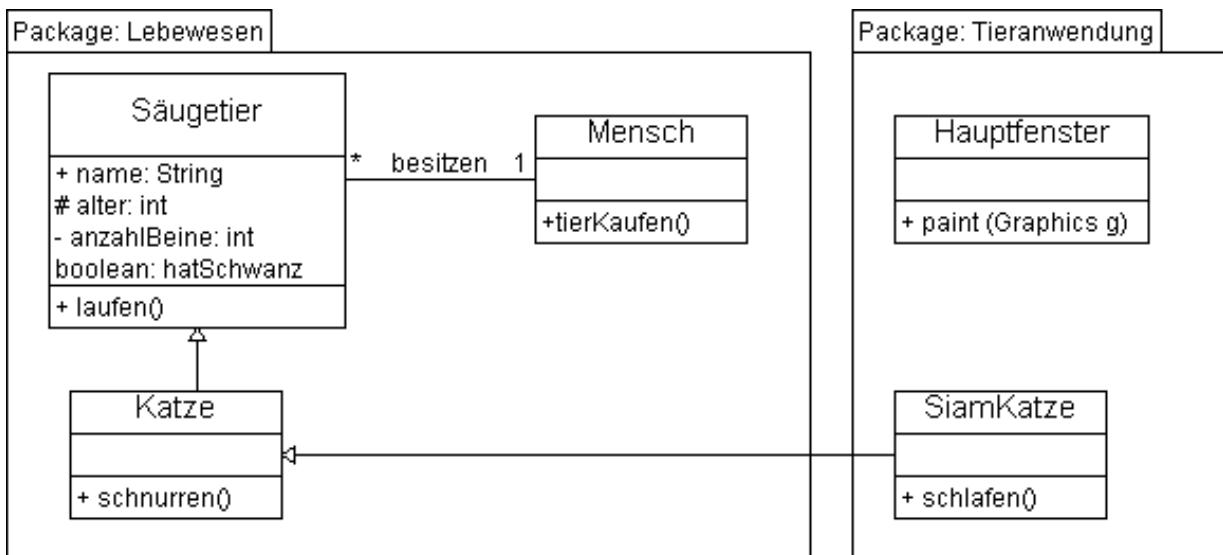
| Attribut (UML-Zeichen) | Klasse | abgeleitete Klasse | Package | andere Packages |
|----------------------------------|--------|--------------------|---------|-----------------|
| <code>private (-)</code> | + | - | - | - |
| <code>protected (#)</code> | + | + (*) | + | - |
| <code>public (+)</code> | + | + | + | + |
| <code><kein Angabe></code> | + | - | + | - |

(*) Ableitung in fremdem Paket: Zugriff nur bei Objekten vom Typ der abgeleiteten Klasse nicht in Objekten vom Typ der Basisklasse.

16.4 Vererbung – Übungen

Aufgabe 1: Übung zur Datenkapselung

Das UML-Klassendiagramm zeigt mehrere Klassen, die sich in zwei verschiedenen Packages befinden. Vereinfacht kann man sagen, dass ein Package aus allen Dateien in einem Verzeichnis besteht. Wenn man auf Klassen aus einem fremden Package zugreifen möchte, muss man die Klassen mit Hilfe einer import- Anweisung in die Datei einbinden.

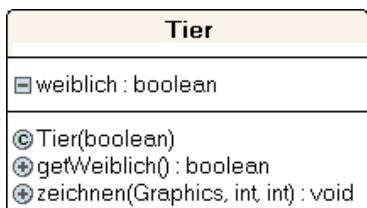


Auf welche der Variablen aus der Klasse **Säugetier** kann man von den anderen Klassen aus zugreifen?

| Variable | Methode <code>schnurren()</code> für ein Katzen- Objekt | Methode <code>tierKaufen()</code> für ein Mensch- Objekt | Methode <code>paint(g)</code> für ein Haupt- fenster-Objekt | Methode <code>schlafen()</code> für ein SiamKatze- Objekt |
|--------------------------|--|---|--|--|
| <code>name</code> | | | | |
| <code>alter</code> | | | | |
| <code>anzahlBeine</code> | | | | |
| <code>hatSchwanz</code> | | | | |

Aufgabe 2: Leseübung

Es existiert eine Klasse **Tier**, die folgendermaßen aussieht:



Von der Klasse **Tier** wird eine Klasse **Hund** abgeleitet. Fülle die Lücken im Code geeignet aus:

```
public class _____ {
```

```
public void zeichnen (Graphics g, int x, int y) {  
    // Nur männliche Hunde sollen gezeichnet werden. Weibliche Hunde  
    // verstecken sich. Rufe die Methode zeichnen aus der Superklasse  
    // Tier auf, wenn der Hund männlich ist.  
  
}  
}
```

Fragen

- Wieso gibt es in der überschriebenen Methode `zeichnen(...)` eine Fehlermeldung, wenn man auf die Variable `weiblich` zugreift?
- Wieso gibt der Compiler eine Fehlermeldung aus, wenn in der Klasse `Hund` kein Konstruktor steht? Schreibe zur Beantwortung der Frage den Konstruktor auf, den das System automatisch erzeugt.
- Füge in die Klasse `Hund` einen Konstruktor ein, der als Parameter einen boolschen Wert erhält. Der boolsche Wert gibt an, ob der Hund weiblich ist oder nicht. Dies ist derselbe Konstruktor wie ihn die Superklasse `Tier` besitzt.
- Schreibe zusätzlich einen parameterlosen Konstruktor für die Klasse `Hund`, der immer männliche Hunde erzeugt.

Aufgabe 3: Zwei spezielle Lampen

- Verändere die Sichtbarkeit der Attribute in der Klasse `Lampe` (du findest die Datei `Lampe.java` im Kurs-Repository im Ordner `16_Java_Vererbung`) von `private` nach `protected`. Ansonsten bleibt die Klasse `Lampe` unverändert.

```
public class Lampe {  
    protected Color farbe;  
    protected boolean isAn = false;  
    protected int xPos, yPos;  
    protected static final int BREITE = 50;  
  
    ...  
}
```

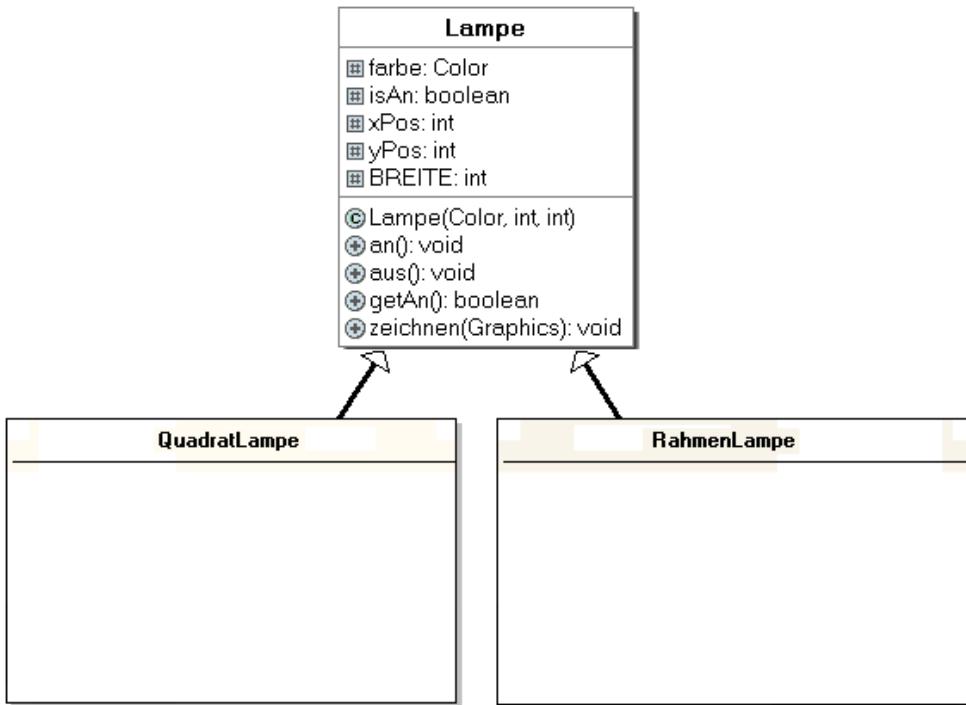
protected:
Außerhalb des eigenen Packages Zugriff nur für abgeleitete Klassen.

- In einem neuen Package (nicht das, in welchem die Klasse `Lampe` bereits existiert!) sollen zwei spezielle Lampentypen programmiert werden, die von der Klasse `Lampe` abgeleitet werden:

- die `QuadratLampe`, die statt eines runden ein quadratisches Licht anzeigt.
- die `RahmenLampe`, die um den Lichtkreis einen schwarzen Rand zeichnet.

Welche Attribute und Methoden benötigen die abgeleiteten Klassen?

- Erstelle im gleichen Package eine Klasse für das Anwendungsprogramm. Erzeuge im Anwendungsprogramm je ein Objekt der drei Lampen-Klassen, und lass jede Lampe blinken.
- Eine Variable der Superklasse `Lampe` kann auch Objekte abgeleiteter Klassen speichern.



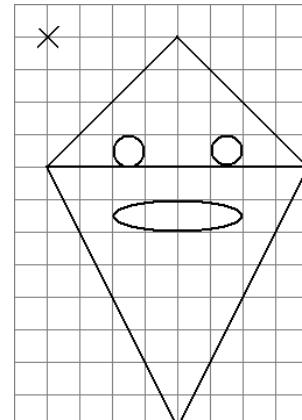
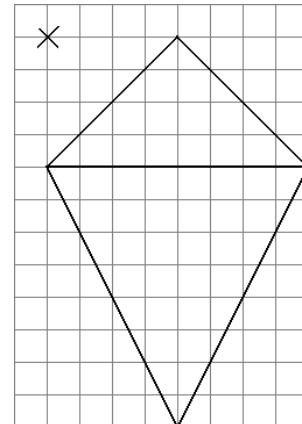
Erzeuge im Anwendungsprogramm eine weitere Variable vom Typ der Superklasse **Lampe**. Weise dieser Variable im Konstruktor des Anwendungsfensters über eine Zufallszahl ein Objekt von einem der drei Lampen-Typen zu ($0 \rightarrow \text{Lampe}$, $1 \rightarrow \text{QuadratLampe}$, $2 \rightarrow \text{RahmenLampe}$). Zeichne auch diese Lampe im Anwendungsfenster und lass sie blinken.

Aufgabe 4: Drachen

- a) Programmiere eine Klasse **Drachen** (siehe Abbildung). Alle Attribute der Klasse **Drachen** sollen vor dem Zugriff von außen (d.h. vom Anwendungsfenster aus) geschützt sein. Alle Methoden sind öffentlich und dürfen vom Anwendungsfenster aus benutzt werden. Im Konstruktor der Klasse werden die x-Position, die y-Position (linke obere Ecke: in der Zeichnung mit einem Kreuz markiert) und die Farbe des Drachen als Parameter übergeben.

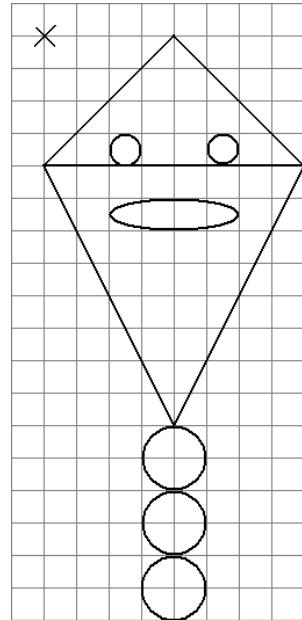
Der Drachen besitzt eine Methode **zeichnen()**, die den Drachen mit der gewählten Farbe an der eingestellten Position malt. Dazu werden zwei ausgefüllte Dreiecke gezeichnet. Ein Kästchen in der Abbildung soll einer Breite von zehn Pixeln entsprechen.

Erzeuge im Anwendungsfenster zwei verschiedene Objekte der Klasse **Drachen**.



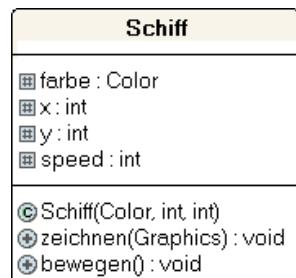
- b) Programmiere in einem anderen Package eine Klasse **GesichtDrachen**, die sich von der Klasse **Drachen** ableitet (siehe Abbildung). Überschreibe die Methode **zeichnen()** und ergänze den Drachen um zwei blaue Augen und einen roten Mund. Erweitere die Methode so, dass du den Code zum Zeichnen des Drachen nicht noch einmal neu programmieren musst. Erstelle im gleichen Package auch eine Anwendungsklasse und erzeuge im Anwendungsfenster ein oder mehrere Objekte der Klasse **GesichtDrachen**.

- c) Im selben Package sollst du nun auch noch eine Klasse **SchwanzDrachen** programmieren, die sich von der Klasse **GesichtDrachen** ableitet (siehe Abbildung). Der Konstruktor dieser Klasse soll einen zusätzlichen Parameter haben, der die Farbe für den Schwanz des Drachen angibt. Überschreibe die Methode `zeichnen()` und hänge unten an den Drachen drei farbige Kreise an. Erweitere die Methode `zeichnen()` so, dass du den Code zum Zeichnen des GesichtDrachen nicht noch einmal neu programmieren musst.
- Erzeuge im Anwendungsfenster ein oder mehrere Objekte der Klasse **SchwanzDrachen**.

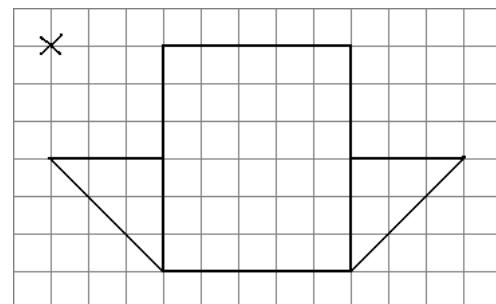


Aufgabe 5: Schiffe

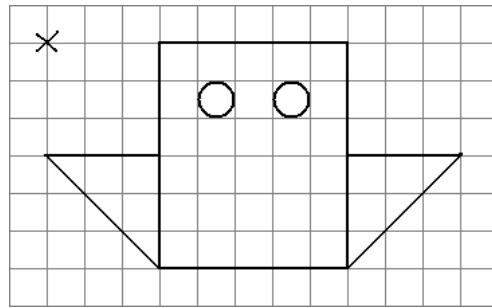
- a) Erzeuge ein Anwendungsfenster mit einem blauen Hintergrund. Das Anwendungsfenster soll eine Breite und Höhe von je 500 Pixel besitzen. Füge in das Anwendungsfenster ein Objekt der Klasse **Timer** ein und stelle eine Wiederholungsrate von 20 Millisekunden ein.
- b) Programmiere in einem anderen Package eine Klasse **Schiff**. Alle Attribute der Klasse **Schiff** sollen vor dem Zugriff von außen (d.h. vom Anwendungsfenster aus) geschützt sein. Alle Methoden sind öffentlich und dürfen vom Anwendungsfenster aus benutzt werden. Der Klasse **Schiff** wird im Konstruktor eine Farbe, die anfängliche y-Position und ein Wert für die Geschwindigkeit (`speed`) übergeben. Die anfängliche x-Position wird automatisch auf 0 gesetzt. Programmiere eine Methode `bewegen()`, die das Schiff jeweils um den in der Variablen `speed` stehenden Wert nach rechts verschiebt. Sobald die x-Position größer als 500 ist, wird das Schiff 110 Pixel vor den linken Rand gesetzt, damit es wieder langsam in das Bild hineingleitet.



Schreibe eine Methode `zeichnen()`, die das Schiff an seiner aktuellen Position in der gewählten Farbe zeichnet. Das Schiff wird mit einem ausgefüllten Rechteck und zwei angrenzenden ausgefüllten Dreiecken gezeichnet. Die Position, die durch die x- und die y-Koordinate beschrieben wird, ist in der Abbildung durch ein Kreuz markiert. Ein Kästchen soll eine Breite von 10 Pixel besitzen.



- c) Erzeuge im Anwendungsfenster ein oranges (`Color.ORANGE`) Schiff mit der y-Position 200 und einer Geschwindigkeit (`speed`) von 2. Erzeuge ein zweites Schiff mit der Farbe PINK, der y-Position 50 und einer Geschwindigkeit von 3. Rufe für beide Schiffe in der `myPaint()`-Methode die beiden Methoden `bewegen()` und `zeichnen()` auf.
- d) Leite von der Klasse **Schiff** die Klasse **Fensterschiff** ab. Die Klasse **Fensterschiff** soll im selben Package liegen, wie die Klasse des Anwendungsfensters. Das Fensterschiff besitzt zusätzlich zwei Fenster, die als ausgefüllte, gelbe Kreise gezeichnet werden (siehe Abbildung). Füge diese Erweiterung so ein, dass die Zeichen-Funktionalität der Oberklasse wieder verwendet wird.



Erzeuge im Anwendungsfenster ein Objekt der Klasse **FensterSchiff** mit der Farbe `LIGHT_GRAY`, der y-Position 150 und einer Geschwindigkeit von 4.

- e) Zeichne im Anwendungsfenster an der Position (200, 400) ein ausgefülltes Rechteck mit einer Breite von 110 und einer Höhe von 100 Pixel ein. Das Rechteck soll die Farbe `LIGHT_GRAY` besitzen. Dieses Rechteck soll einen Steg darstellen, an dem Schiffe anlegen können.
- f) Programmiere eine Klasse **AnlegeSchiff** (wiederum im selben Package), die sich von **FensterSchiff** ableitet. Im Konstruktor des AnlegeSchiffs wird zusätzlich zu den geerbten Parametern eine Integer-Variable übergeben, in der die Position des Stegs steht. Wenn das Schiff mit seiner x-Position auf den Steg trifft, soll es 30 Aufrufe der Methode `bewegen` lang stehen bleiben. Danach fährt es mit der alten Geschwindigkeit weiter. Programmiere diese Erweiterung so, dass die Bewegungsänderung nicht neu geschrieben werden muss.
Erzeuge im Anwendungsfenster ein AnlegeSchiff mit der Farbe `PINK`, der y-Position 340, der Geschwindigkeit 2 und der Stegposition 200.

Aufgabe 6: Billardkugeln

Verschiedene Billardkugeln sollen über den Bildschirm rollen. Wenn eine Kugel an den Rand des Fensters stößt, prallt sie vom Rand ab und bekommt eine neue Richtung nach dem Gesetz „Einfallwinkel gleich Ausfallwinkel“. Dafür muss die Kugel die aktuelle Fenstergröße kennen.

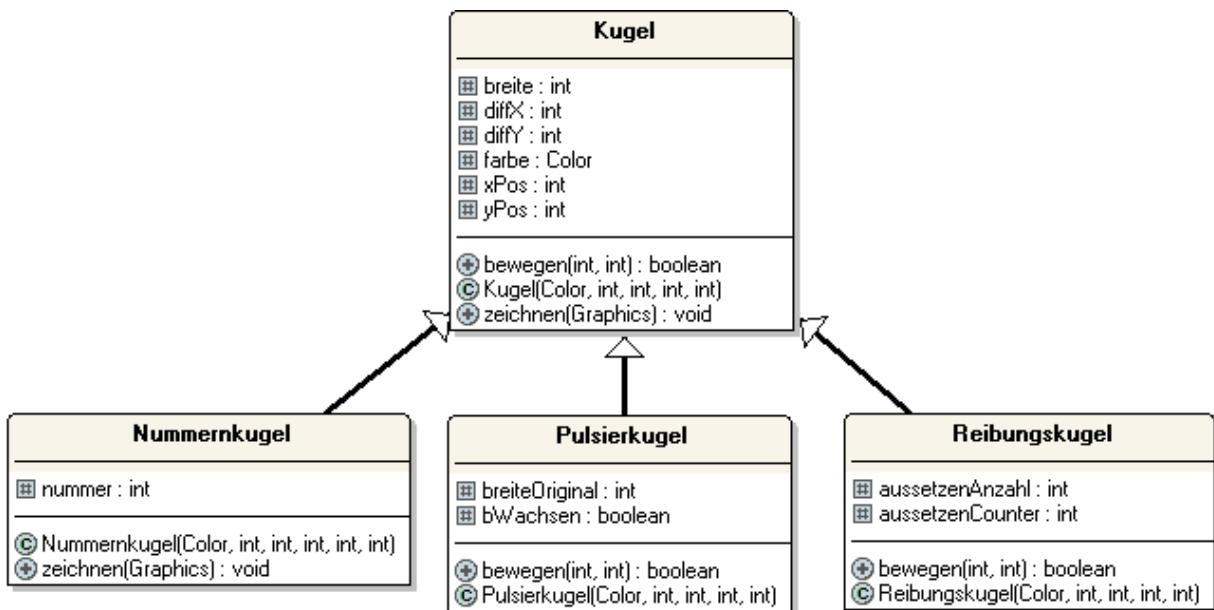


Neben den „einfachen“ Billardkugeln existieren eine Reihe von speziellen Kugeln:

- Nummernkugel : Die Kugel zeigt in ihrem Innern eine Nummer an.
- Pulsierkugel: Die Kugel soll langsam größer und dann wieder kleiner werden. Das Wachsen und Schrumpfen soll ständig passieren.
- Reibungskugel : Die Kugel reagiert auf Reibung und wird nach jedem Stoß gegen die Fensterwand langsamer.

Programmiere zunächst eine „einfache“ Kugel und teste sie aus (siehe dazu den letzten Abschnitt „Die Billard-Anwendung“). Programmier dann nach und nach die Spezialkugeln und teste sie.

Entwurf für die Billardkugeln

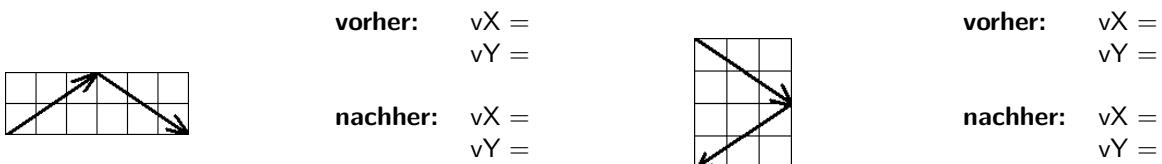


Klasse Kugel

Die linke obere Ecke einer Kugel wird mit den Integer-Variablen `xPos` und `yPos` bezeichnet. Die `breite` entspricht dem Durchmesser der Kugel. Die Geschwindigkeit und die Richtung werden über die Integer-Variablen `vX` und `vY` ausgedrückt. `vX` gibt die Änderung der Geschwindigkeit in x-Richtung an, während `vY` die Änderung in y-Richtung angibt. Beide Werte werden bei jedem Aufruf der Methode `bewegen()` zu `xPos` bzw. `yPos` dazu addiert. Im Konstruktor werden die Farbe sowie die Variablen `xPos`, `yPos`, `vX` und `vY` übergeben. Die `breite` wird automatisch auf 20 Pixel gesetzt (oder einen anderen geeigneten Wert).

Die Methode `zeichnen()` zeichnet die Kugel an ihrer aktuellen Position. Die Methode `bewegen()` bewegt die Kugel wie oben beschrieben weiter. Als Parameter erhält `bewegen()` die Breite und die Höhe des Fenster. Als Hilfe für die Reibungskugel gibt sie einen boolschen Wert zurück, der angibt, ob die Kugel gegen die Wand gestoßen ist (`true`) oder nicht (`false`).

Programmierung des Abprallens vom Rand



Klasse Nummernkugel

Die Klasse **Nummernkugel** wird um ein zusätzliches Attribut `nummer` erweitert, in dem die Zahl steht, die in der Kugel angezeigt werden soll. Der Konstruktor wird um einen Parameter erweitert, in dem die Zahl übergeben wird. Die Methode `zeichnen()` wird geeignet überschrieben.

Klasse Pulskugel

Die Klasse **Pulskugel** besitzt denselben Konstruktor wie die Oberklasse.

Sie überschreibt die Methode `bewegen()`, und erweitert sie um die Änderung der Kugelgröße. Die Originalbreite wird in einer Variablen abgespeichert. Die Kugel wächst zunächst schrittweise um einen Pixel bis sie die Originalbreite um fünf Pixel überschritten hat. Anschließend schrumpft sie schrittweise um einen Pixel bis sie die

Originalbreite um fünf Pixel unterschritten hat, usw. Eine boolesche Variable speichert die aktuelle Richtung der Größenänderung (wachsen oder schrumpfen).

Klasse Reibungskugel

Die Klasse **Reibungskugel** besitzt denselben Konstruktor wie die Oberklasse und überschreibt die Methode `bewegen()`. Die Verlangsamung der Bewegung wird programmiert, indem die Kugel bei der Bewegungsänderung „Aussetzer“ hat. Nach dem ersten Stoß gegen die Wand wird die Position der Kugel nur noch bei jedem zweiten Aufruf der Methode `bewegen()` tatsächlich verändert. Nach dem nächsten Stoß wird einmal die Bewegung geändert und dann zweimal ausgesetzt, nach dem darauf folgenden Stoß wird einmal die Bewegung geändert und dann dreimal ausgesetzt, usw.

Es gibt eine Variable `aussetzenAnzahl`, die angibt, nach wie viel `bewegen()`-Aufrufen die Bewegung verändert wird. Zu Anfang ist der Wert 0. Nach dem ersten Stoß gegen die Wand ist der Wert 1, dann 2 usw. Die Variable `aussetzenCounter` zählt die Anzahl der `bewegen()`-Aufrufe bis die `aussetzenAnzahl` überschritten ist und tatsächlich eine Bewegung ausgeführt wird.

Die Billard-Anwendung

Erzeuge in einem anderen Package eine Klasse **Billard**, die von **HJFrame** abgeleitet ist. Das Anwendungsfenster soll eine Breite und Höhe von je 500 Pixeln und einen grünen Hintergrund haben.

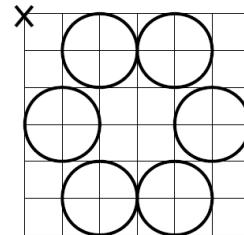
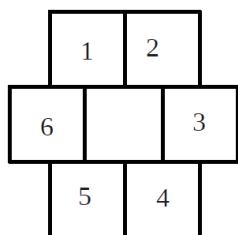
Erzeuge von jeder Kugelart je ein Objekt mit geeigneten Startwerten (gib jeder Kugel eine eigene Farbe und benutze unterschiedliche Anfangspositionen und Geschwindigkeiten). Erzeuge im Konstruktor der Anwendungsklasse ein geeignetes Timer-Objekt um die Billardanwendung zu animieren.

17 Wiederholung

17.1 Programmierung

Aufgabe 1: Kette

- a) Erzeuge ein Anwendungsfenster mit einem schwarzen Hintergrund. Das Anwendungsfenster soll eine Breite und Höhe von je 500 Pixel besitzen.
- b) Programmiere eine Klasse **Kreis**. Ein Kreis besitzt private Attribute für Farbe, x-Position und y-Position. Alle drei genannten Eigenschaften werden im Konstruktor als Parameter übergeben. Die Breite des Kreises wird mit einer Konstanten auf 40 Pixel festgesetzt. Es gibt eine Methode `zeichnen()`, die einen ausgefüllten Kreis an der (x,y)-Position mit der eingestellten Farbe zeichnet. Überlege dir selber, welche Parameter und welcher Rückgabewert für die Methode `zeichnen()` sinnvoll sind.
 Zur Lösung der weiteren Aufgaben darf die Klasse **Kreis** zusätzliche Attribute erhalten, die jedoch vor einem Zugriff von außen geschützt werden müssen.
- c) Programmiere eine Klasse **Kette**. Eine Kette besteht aus sechs Objekten der Klasse **Kreis**, die wie folgt angeordnet werden sollen:



Die rechte Abbildung zeigt wie die Kette aussehen soll. Die linke Abbildung zeigt die Quadrate, durch die die Kreise beschrieben werden. Die Ziffern innerhalb der Quadrate geben die Reihenfolge an, in der die Kreis-Objekte erzeugt werden sollen. Bitte halte dich an diese Reihenfolge, weil sie für die weiteren Aufgaben von Bedeutung ist. Ein Karo-Kästchen entspricht einer Breite von 20 Pixel. Das Kreuz markiert die linke obere Ecke der Kette und soll nicht mit gezeichnet werden.

Im Konstruktor der Kette werden die Farbe und die linke obere Ecke der Kette übergeben (x- und y-Position). Alle Kreis-Objekte erhalten die in der Kette übergebene Farbe. Es gibt eine Methode `zeichnen()`, die die Kette an der angegebenen Position zeichnet.

Erzeuge im Anwendungsfenster drei Objekte von der Kette:

- Das erste Objekt erhält die Farbe rot und die linke obere Ecke (180 | 100).
 - Das zweite Objekt erhält die Farbe blau und die linke obere Ecke (50 | 250).
 - Das dritte Objekt erhält die Farbe grün und die linke obere Ecke (300 | 250).
- d) Füge in das Anwendungsfenster ein Objekt der Klasse **Timer** ein und sorge dafür, dass das Fenster alle 500 Millisekunden neu gezeichnet wird.
 Erweitere anschließend die Klasse **Kreis**. Bei jedem dritten Aufruf der Methode `zeichnen()` wird das **Kreis**-Objekt nicht in seiner eingestellten Farbe sondern mit der Farbe orange gezeichnet. Dadurch erhält die gesamte Kette bei jedem dritten Aufruf von `zeichnen()` die Farbe orange.
- e) Erweitere die Klasse **Kreis** noch einmal. Die Kreis-Objekte sollen jetzt nicht mehr alle zur gleichen Zeit die Farbe orange anzeigen. Die Kreise werden beim Erzeugen der Objekte zyklisch durchgezählt: 1, 2, 3, 1, 2, 3, 1, 2, 3,
 Jedes erste **Kreis**-Objekt hat den Rhythmus: farbe, farbe, orange, farbe, farbe, orange, ...
 Jedes zweite **Kreis**-Objekt hat den Rhythmus: farbe, orange, farbe, farbe, orange, farbe ...
 Jedes dritte **Kreis**-Objekt hat den Rhythmus: orange, farbe, farbe, orange, farbe, farbe, ...
 Dadurch gibt es in jeder Kette immer zwei orange Kugeln zur Zeit. Die orange Farbe wandert von einer Kugel zur anderen.

- f) Füge in die Klasse **Kette** ein Objekt der Klasse **Random** ein. Das **Random**-Objekt darf nur in der gesamten Klasse **Kette** nur einmal vorkommen.

Erweitere die Klasse **Kette** nun folgendermaßen: Jede Kette wird abwechselnd eine zufällige Zeitspanne lang ausgeschaltet (das heißt sie ist nicht sichtbar) und danach wieder eine zufällige Zeitspanne lang angeschaltet. Die Zufallswerte werden für jeden „An“ bzw. „Aus“-Zyklus neu erzeugt und sollen zwischen 5 und 15 liegen. Die Zufallszahl 6 bedeutet beispielsweise, dass die Kette über 6 Aufrufe von `zeichnen()` nicht zu sehen ist.

Aufgabe 2: Kreisende Bälle

- a) Erzeuge ein Anwendungsfenster mit einem schwarzen Hintergrund und weißer Vordergrundfarbe. Das Anwendungsfenster soll eine Breite und Höhe von je 500 Pixel besitzen.
- b) Programmiere in einer zweiten Datei eine Klasse **Ball**. Alle Attribute der Klasse **Ball** sollen vor dem Zugriff von außen (d.h. vom Anwendungsfenster aus) geschützt sein. Alle Methoden sind öffentlich und dürfen vom Anwendungsfenster aus benutzt werden.
- c) Ein Ball besitzt zu Anfang die Attribute x-Position, y-Position, Breite und Farbe. Weitere Attribute dürfen später nach Bedarf hinzugefügt werden. Die x-Position wird automatisch auf den Wert 25 festgelegt, die y-Position auf den Wert 250 und die Breite auf den Wert 50. Nur die Farbe kann vom Anwendungsfenster aus eingestellt werden und wird im Konstruktor als Parameter übergeben.
Schreibe eine Methode `zeichnen()`, die den Ball an seiner aktuellen Position und in der für ihn gewählten Farbe als ausgefüllten Kreis zeichnet. Entscheide selbst welche Parameter und welchen Rückgabewert die Methode `zeichnen()` benötigt.
- d) Erzeuge im Anwendungsfenster ein Objekt der Klasse **Ball**, das eine rote Farbe besitzt.
- e) Füge im Anwendungsfenster ein Objekt der Klasse **Timer** ein, und stelle den Timer so ein, dass die `myPaint()`-Methode alle 10 Millisekunden aufgerufen wird.
- f) Erweitere in der Klasse **Ball** die Methode `zeichnen()`, so dass in der Methode die x-Position des Balls bei jedem Aufruf um zwei Pixel verschoben wird. Der Ball soll immer abwechselnd zunächst nach rechts rollen bis zur x-Position 425 und anschließend wieder zurück nach links bis zur x-Position 25.
- g) Erweitere die Klasse **Ball** so, dass jedes weitere Ball-Objekt automatisch in seiner anfänglichen x-Position um 50 Pixel nach rechts verschoben wird. Das erste Ball-Objekt besitzt bei Erzeugung die x-Position 25, das zweite Ball-Objekt die x-Position 75, das dritte Ball-Objekt die x-Position 125, usw.
Erzeuge im Anwendungsfenster acht verschiedene Ball-Objekte, die unterschiedliche Farben besitzen sollen. Die Farben darfst du frei wählen. Wenn du alles richtig gemacht hast, bilden die Ball-Objekte eine Art „Schlange“.
- h) Die Klasse **Ball** soll so erweitert werden, dass sich der Ball im Uhrzeigersinn im Kreis bewegt. Wenn der Ball sich nach rechts bewegt, wird die y-Koordinate in Abhängigkeit von der x-Koordinate folgendermaßen berechnet:

```
y = 250 - (int) Math.sqrt(40000-(x-225)*(x-225));
```

Dadurch bewegt sich der Ball im Halbkreis nach oben. Wenn sich der Ball nach links bewegt, wird die y-Koordinate mit der Formel

```
y = 250 + (int) Math.sqrt(40000-(x-225)*(x-225));
```

berechnet. Dadurch bewegt sich der Ball im Halbkreis nach unten. Beachte, dass der Ball bei dieser Technik an den Seiten links und rechts wesentlich schneller rollt als in der Mitte. Das ist kein Programmierfehler.

Vorgabe: Berechne die y-Koordinate für den Ball mit Hilfe einer Methode, die die x-Koordinate als Parameter erhält und die y-Koordinate als Rückgabewert zurück gibt. Schreibe entweder für jede der beiden Formeln eine Methode oder programmiere eine einzige Methode, die die aktuelle Richtung des Balls berücksichtigt.

- i) Erweitere die Klasse **Ball** so, dass ein Ball zyklisch immer zweimal im Kreis rollt und dann einmal wie in Teilaufgabe e) mit fester y-Koordinate nach rechts und wieder zurück nach links rollt. Das heißt, die Berechnung der y-Koordinate mit den Kreis-Formeln entfällt bei jedem dritten Durchgang.

17.2 Theorie

Aufgabe 1: Schlüsselworte

Was bedeuten die Schlüsselworte public, private, protected, final und static vor einer Variablen-deklaration?

Aufgabe 2: Datenkapselung

- Was versteht man unter dem Begriff Datenkapselung?
- Welche Vorteile bringt es, wenn man nach dem Prinzip der Datenkapselung programmiert?

Aufgabe 3: Vererbung

Was versteht man beim Programmieren unter „Vererbung“?

17.3 UML

Aufgabe 1: UML-Klassendiagramme

Beschreibe die Beziehungen zwischen den Klassen in einem UML-Klassendiagramm (soweit angebracht mit Beziehungsname und Multiplizität):

- Hund, Schwanz, Mensch, Dackel
- Kanarienvogel, Käfig, Gitterstab
- Bürgerweide, Fahrgeschäft, Karussell, Achterbahn, Sitzplatz, Mensch

Aufgabe 2: Model-Agentur

Eine Model-Agentur hat dich beauftragt, ein Computersystem zu entwickeln. Erstelle ein UML-Klassendiagramm, das die Informationen, die dir die Chefin der Agentur gibt, geeignet abbildet:

Für unsere Agentur arbeiten weibliche und männliche Models. Von jedem Model speichern wir die Adresse, die Größe, das Gewicht, die Haarfarbe und ein Foto. „Normale“ Models werden von uns für jeden Auftrag einzeln bezahlt. Mit Models, die besonders erfolgreich sind, machen wir jedoch häufig Exklusiv-Verträge, in denen vereinbart wird, dass sie ausschließlich für unsere Agentur arbeiten dürfen. Zum Exklusiv-Vertrag gehört auch ein monatliches Grundgehalt, das wir an das Model zahlen. Zusätzlich zum Grundgehalt erhalten auch die Models mit Exklusiv-Verträgen für jeden Auftrag eine Bezahlung.

Wir verwalten natürlich auch eine Liste von Kunden (wie z.B. Modehäuser), die regelmäßig bei uns Models anfordern. Für jeden Kunden arbeitet mindestens ein Model. Die Models dagegen können für beliebig viele Kunden arbeiten. Weniger begehrte Models werden häufig auch gar nicht angefordert.

Von den Kunden speichern wir die Adresse und die Anzahl unserer Models, die schon für diesen Kunden gearbeitet haben. Wir verwalten auch für jeden Kunden eine Liste von Großereignissen (wie z.B. Modeschauen), bei denen Models benötigt werden. Für die Großereignisse speichern wir die Art der Veranstaltung und die Anzahl der benötigten Models. Die Kunden veranstalten zwischen null und fünf Großereignisse im Jahr.

Anmerkung: Die Adresse braucht nicht in einzelne Attribute aufgegliedert zu werden.

Aufgabe 3: Weihnachtsmann-Roboter

Zu Weihnachten verkleiden sich häufig Vater, Onkel oder freundliche Bekannte als Weihnachtsmann, um den Kindern die Geschenke zu bringen. Jetzt hat eine Firma für diese Zwecke einen Weihnachtsmann-Roboter entwickelt. Eine Anwendung, die in den kommenden Jahren sicher große Erfolge feiern wird.

Schreibe ein Zustandsdiagramm für den im folgenden Text beschriebenen Weihnachtsmann-Roboter:

Weihnachtsmann-Roboter (Preiswerte Version für nur 1 Kind)

Der Weihnachtsmann-Roboter wartet geduldig (ohne zu murren auch stundenlang!) draußen vor der Tür, bis ihm ein Mitglied der Familie die Tür öffnet. Nachdem er das Wohnzimmer betreten hat, sagt er zunächst ein Gedicht auf („Draußen vom Walde komm ich her..“). Danach wendet er sich an das Kind und stellt ihm eine Quizfrage (die Frage „Warst du auch schön brav“ ist heutzutage schließlich total veraltet). Wenn das Kind die Quizfrage korrekt beantwortet, erhält es ein großes Geschenk. Falls es eine falsche Antwort gibt, darf es noch ein zweites Mal versuchen, die Frage zu beantworten. Wenn die zweite Antwort richtig war, erhält das Kind ebenfalls das große Geschenk. Andernfalls droht der Weihnachtsmann ihm – nach alter Tradition – mit der Rute und überreicht ihm dann ein kleines Geschenk. Zum Schluss singt der Weihnachtsmann der Familie noch ein Weihnachtslied vor und verlässt dann das Haus.

Aufgabe 4: Weihnachts-Lieferservice

Der Weihnachts-Lieferservice muss besonders gut organisiert werden, da jedes Jahr zu Weihnachten Millionen von Kunden zu beliefern sind. Deshalb möchte jetzt auch der Weihnachtsmann modernisieren und sein Büro „auf Computer umstellen“. Du hast den Auftrag, das Computersystem für den Weihnachtsmann zu entwickeln. Schreibe ein Klassendiagramm, das die Abläufe im Weihnachts-Lieferservice darstellt:

Die Annahme, das es nur einen einzigen Weihnachtsmann gibt, ist natürlich eine kindliche Illusion. Ein Mann alleine könnte ja nie die ganze Arbeit bewältigen, die an Heiligabend anfällt. Es gibt hunderte von Weihnachtsmännern und -frauen, die jeweils für einen bestimmten Bezirk zuständig sind. Die Weihnachts“männer“ unterscheiden sich durch verschiedene Vornamen, das Geschlecht und ihr Alter voneinander. Jeder Weihnachtsmann (und jede Weihnachtsfrau) besitzt vier Rentiere, die er während der Sommermonate hart trainiert, damit sie für die großen Anforderungen am Weihnachtsabend fit sind. Jedes Rentier hat seinen eigenen Namen. Für die Bestellannahme ist eine große Schar von Engeln zuständig. 15 bis 30 Engel arbeiten jeweils als Hilfskräfte für einen Weihnachtsmann. Einfache Engel können nur normale Bestellungen annehmen. Oberengel können auch Fern- Aufträge für Lieferungen in fremde Länder bearbeiten. Die eigentliche Auslieferung der Geschenke erfolgt jedoch einzig und allein durch die Weihnachtsmänner.

18 Arrays

Zur Verwaltung großer Mengen von Daten gibt es sogenannte *Arrays* (Datenfelder). Mit einer einzigen Array-Variablen kann man ein ganzes Feld von Daten eines Datentyps verwalten.

18.1 Arrays von primitiven Datentypen(boolean, int, double, char)

Eindimensionale Arrays

Deklaration (Anlegen der Variablen)

```
int[] zahl;
boolean[] wahr;
```

Erzeugung des Datenfeldes

```
zahl = new int[200];           // erzeugt ein Feld mit Index 0 bis 199
wahr = new boolean[2];         // erzeugt ein Feld mit Index 0 bis 1
```

Deklaration und Erzeugung gleichzeitig

```
int[] zahlen = new int[15];      // erzeugt 15 int-Zahlen
int[] liste = {10,3,12,3,5};    // erzeugt ein Array mit 5 int-Zahlen
boolean[] b = {true, true, false}; // erzeugt ein Array mit 3 boolean-Werten
```

Verwendung des Datenfeldes (einfache Datentypen)

Bei einfachen Datentypen wie `int` oder `boolean`, kann man die erzeugten Feldelemente über den Index ansprechen und Werte hinein schreiben:

```
for (int i = 0; i < 200; i++) {
    zahl[i] = 10 + i;
}
wahr[0] = false;
wahr[1] = true;
```

Übergabe eines Arrays an eine Methode

Aufruf der Methode:

```
int summe1 = addieren(zahlen);
int summe2 = addieren(liste);
```

Deklaration der Methode:

```
public int addieren (int[] feld) {
// addiert alle Zahlen des Feldes und gibt die Summe zurück
    int summe = 0;
    for (int i = 0; i < feld.length; i++) {
        summe += feld[i];
    }
    return summe;
}
```

Der Parameter `feld` ist ein Array vom Typ `int` von beliebiger Länge. Es wird die Speicheradresse des Arrays übergeben, so dass sich Änderungen im Array durch die Methode auch auf die übergebene Array-Variable auswirken!

Das im Schleifenkopf benutzte Attribut `feld.length` gibt die Länge des Arrays an.

Mehrdimensionale Arrays

Beispiele:

```
boolean[][] zweidimensional = new boolean[4][3];
zweidimensional[0][0] = true;
zweidimensional[1][0] = false;
...
zweidimensional[3][2] = false;

int[][][] dreidimensional = new int[20][3][4];
dreidimensional[19][2][3] = -97;
...
```

18.2 Arrays von Objekten

Deklaration

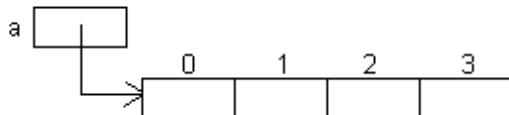
```
Auto[] a; // Variable vom Typ einer Klasse "Auto"
```

Es wird ein Speicherplatz angelegt, der später die Speicheradresse des Datenfeldes speichern soll:



Erzeugung des Datenfeldes

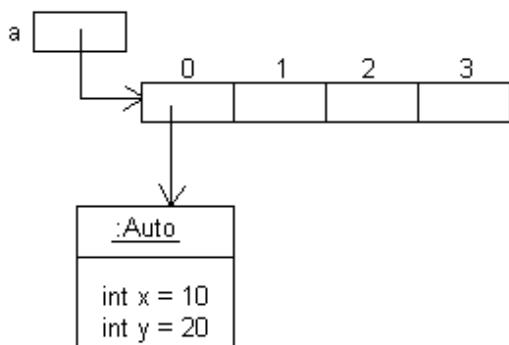
```
a = new Auto[4]; // erzeugt Felder mit Index 0 bis 3
```



Erzeugung der Objekte

Bei Feldern, die als Typ eine Klasse haben, muss man die einzelnen Objekte explizit erzeugen:

```
a[0] = new Auto(10,20);
```

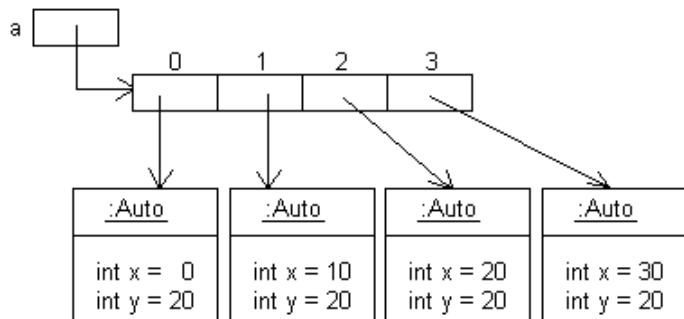


Anschließend kann man Methoden und Variablen des Objektes aufrufen:

```
a[0].x = 30; // Verändert die Variable x des Objektes
a[0].fahren(500); // ruft die Methode "fahren" der Klasse Auto auf
```

In einer Schleife kann man alle Objekte des Feldes auf einmal erzeugen:

```
for (int x = 0; x < 4; x++) {  
    a[x] = new Auto(x*10, 20);  
}
```



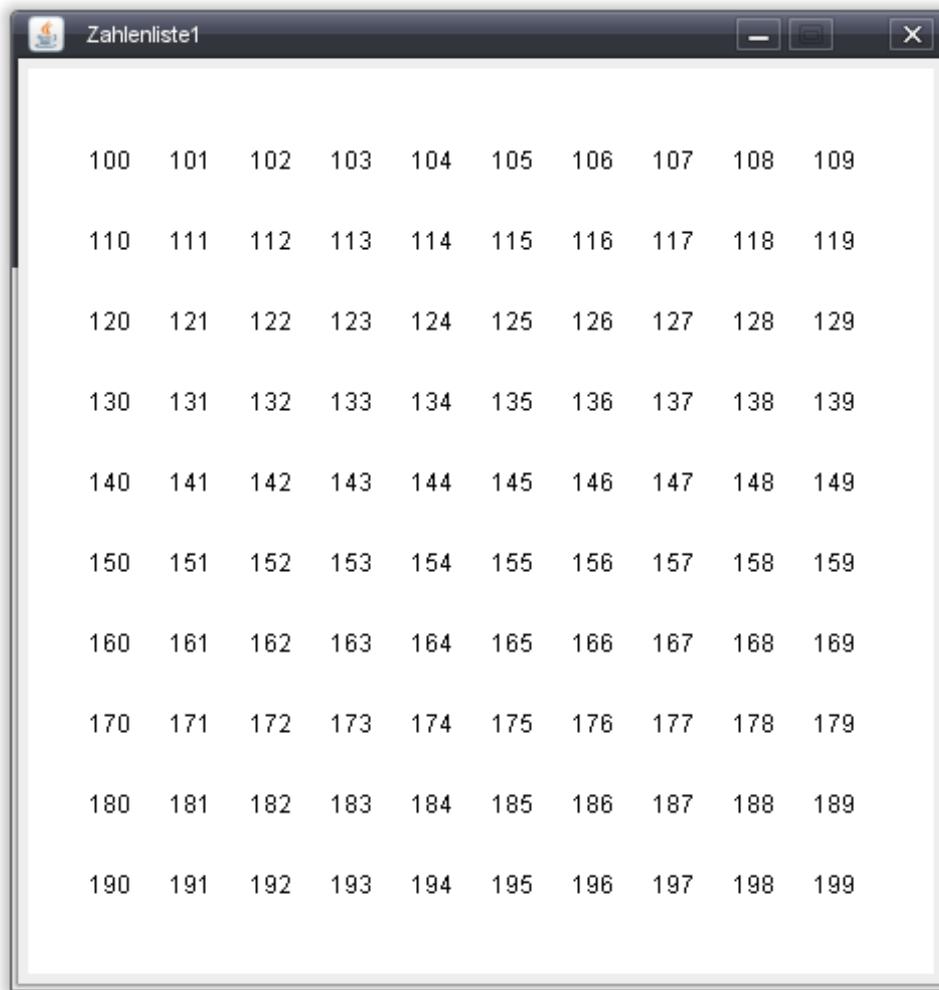
18.3 Arrays – Übungen

Aufgabe 1: Zahlen

- a) Erstelle ein neues Java-Anwendungsfenster. Lege eine globale Variable für ein Array von `int`-Werten an und erzeuge ein Feld mit 100 Elementen. Füll das Zahlenarray im Konstruktor mit Zahlen von 100 bis 199.
- b) Gib in einer Zeile im Anwendungsfenster die ersten 10 Zahlen des Arrays aus:

```
100 101 102 103 104 105 106 107 108 109
```

- c) Erweitere die Ausgabe so, dass alle Zahlen des Arrays zeilenweise untereinander ausgegeben werden. Jeweils 10 Zahlen sollen in einer Zeile stehen.

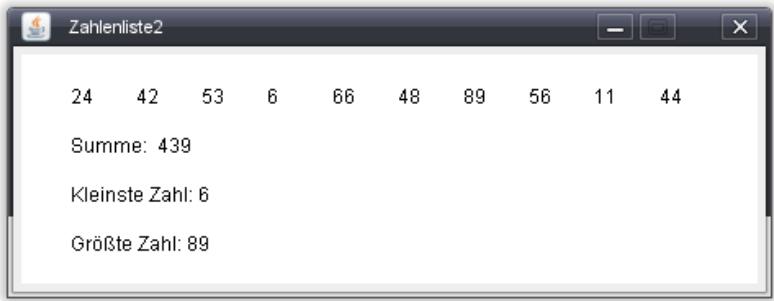


- d) Wenn du willst kannst du nun Versuchen diese Aufgabe mit einem zweidimensionalen Array statt mit einem eindimensionalen Array zu lösen.

Aufgabe 2: Summe und Extremwerte von Zahlen

- a) Erstelle ein neues Java-Anwendungsfenster. Erzeuge ein `int`-Array mit 10 Elementen und schreibe im Konstruktor das Anwendungsfensters in das Array Zahlen von 1 bis 10. Gib die 10 Zahlen in einer Zeile des Anwendungsfensters aus.
- b) Verändere das Programm so, dass in jedes Feld des Arrays im Konstruktor mit einer Zufallszahl zwischen 1 und 100 gefüllt wird.

- c) Berechne die Summe aller Zahlen und gib sie im Anwendungsfenster aus.
- d) Ermittle die kleinste Zahl des Arrays und gib sie im Anwendungsfenster aus.
- e) Ermittle die größte Zahl des Arrays und gib sie im Anwendungsfenster aus.



Aufgabe 3: Strichmännchen

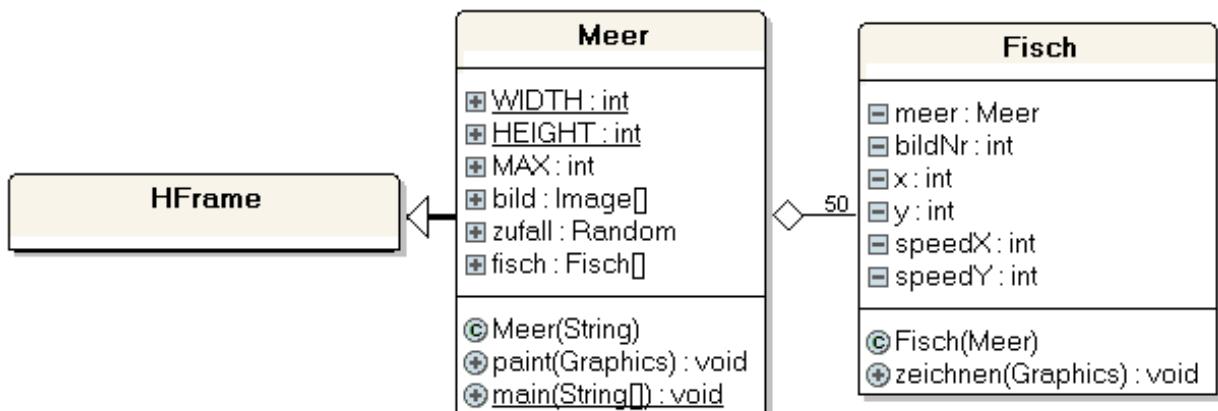
Vor einiger Zeit hast du eine Klasse **Strichmaennchen** programmiert und eigene Strichmännchen über den Bildschirm laufen lassen. Erweitere das alte Anwendungsprogramm so, dass du ein Array von 20 Strichmännchen erzeugst. Generiere dabei für die y-Position und für die Geschwindigkeit des Strichmännchens Zufallswerte, damit die Strichmännchen an unterschiedlichen Positionen erscheinen und unterschiedlich schnell laufen.

Aufgabe 4: Billardkugeln

Benutze die Billardkugeln aus dem Kapitel über Vererbung. Erzeuge ein Array von 50 oder mehr Billardkugeln, die über den Bildschirm rollen. Generiere für die x-Position, die y-Position und die Geschwindigkeitswerte der Billardkugeln geeignete Zufallswerte.

Aufgabe 5: Meer mit Fischen

Ein Schwarm von Fischen schwimmt wild durcheinander. Dazu findest du im Kursrepository verschiedene animierte Bilder von Fischen.



- a) Erzeuge ein großes Anwendungsfenster mit der Hintergrundfarbe cyan. Generiere im Anwendungsfenster einen Zufallsgenerator (ein Objekt der Klasse **Random**) und lade alle Fisch-Bilder, die du verwenden möchtest, in ein Array vom Typ **Image**. Außerdem soll ein Array von 50 Objekten der Klasse **Fisch** erzeugt werden, die in Teil b) programmiert wird.

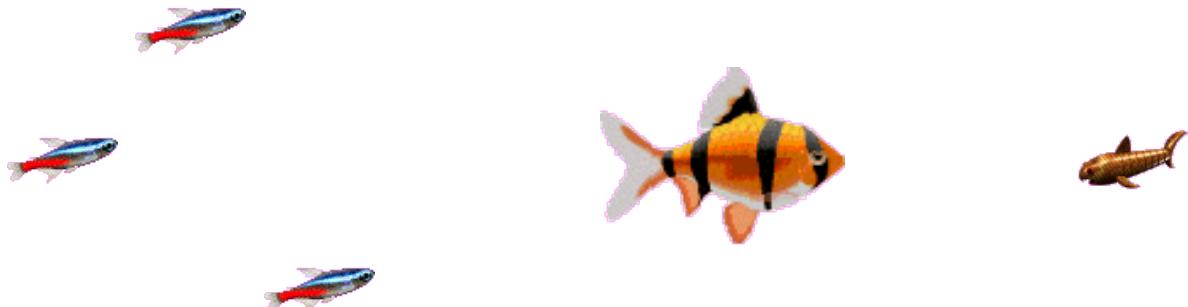
Anmerkung: Auf ein Objekt der Klasse **Timer** kann man verzichten, weil das Fenster durch die animierten Bilder automatisch regelmäßig neu gezeichnet wird.

- b) Programmiere eine Klasse **Fisch**. Der einzige Parameter, der im Konstruktor übergeben wird, ist ein Objekt des Anwendungsfensters. Über dieses Objekt kann man beim Zeichnen auf die Bilder zugreifen und man kann die Breite und Höhe des Fensters abfragen (**WIDTH**, **HEIGHT**).

Im Konstruktor werden die Eigenschaften eines Fisches mit Zufallswerten festgelegt. Verwende dazu den Zufallsgenerator aus dem Anwendungsfenster:

- **bildNr**: legt fest welches der Fisch-Bilder verwendet wird
- **x**: anfängliche x-Position (Wert zwischen 0 und **WIDTH**)
- **y**: anfängliche y-Position (Wert zwischen 20 und **HEIGHT**)
- **speedX**: Geschwindigkeit in x-Richtung. Wenn der Fisch nach rechts guckt, ein Wert zwischen 1 und 5. Wenn der Fisch nach links guckt, ein Wert zwischen -1 und -5. Am Anfang kannst du erst mal alle Fisch in eine Richtung schwimmen lassen, damit es nicht gleich zu schwierig wird.
- **speedY**: Geschwindigkeit in y-Richtung: Wert zwischen -2 und +2.

In der Methode zeichnen wird der Fisch mit dem Bild gezeichnet, das seiner **bildNr** entspricht. Außerdem wird er entsprechend seiner Geschwindigkeit in x- und in y-Richtung weiter bewegt. Wenn ein Fisch rechts aus dem Bild heraus schwimmt, wird seine x-Position 100 Pixel vor den linken Rand gesetzt, damit er langsam in das Bild hinein schwimmt. Wenn er links heraus schwimmt, wird er an den rechten Rand gesetzt. Außerdem wird für die y-Position ein neuer Zufallswert bestimmt.



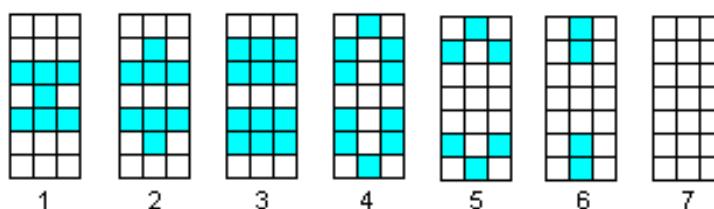
Aufgabe 6: Game of Life

Das „Lebens-Spiel“ wurde 1970 von dem englischen Mathematiker John Conway entwickelt. Es simuliert die Population von einfachen Lebewesen, die gegenseitig von einander abhängig sind. Der Lebensraum der Lebewesen wird durch ein Gitternetz aus quadratischen Zellen dargestellt. In jeder Zelle kann sich ein Lebewesen befinden (oder auch nicht).

Nach einem Lebenszyklus überlebt ein Lebewesen, wenn es zwei oder drei Nachbarn hat. Wenn es weniger als zwei Nachbarn hat, stirbt es aus Einsamkeit. Wenn es mehr als drei Nachbarn hat, stirbt es wegen Überbevölkerung.

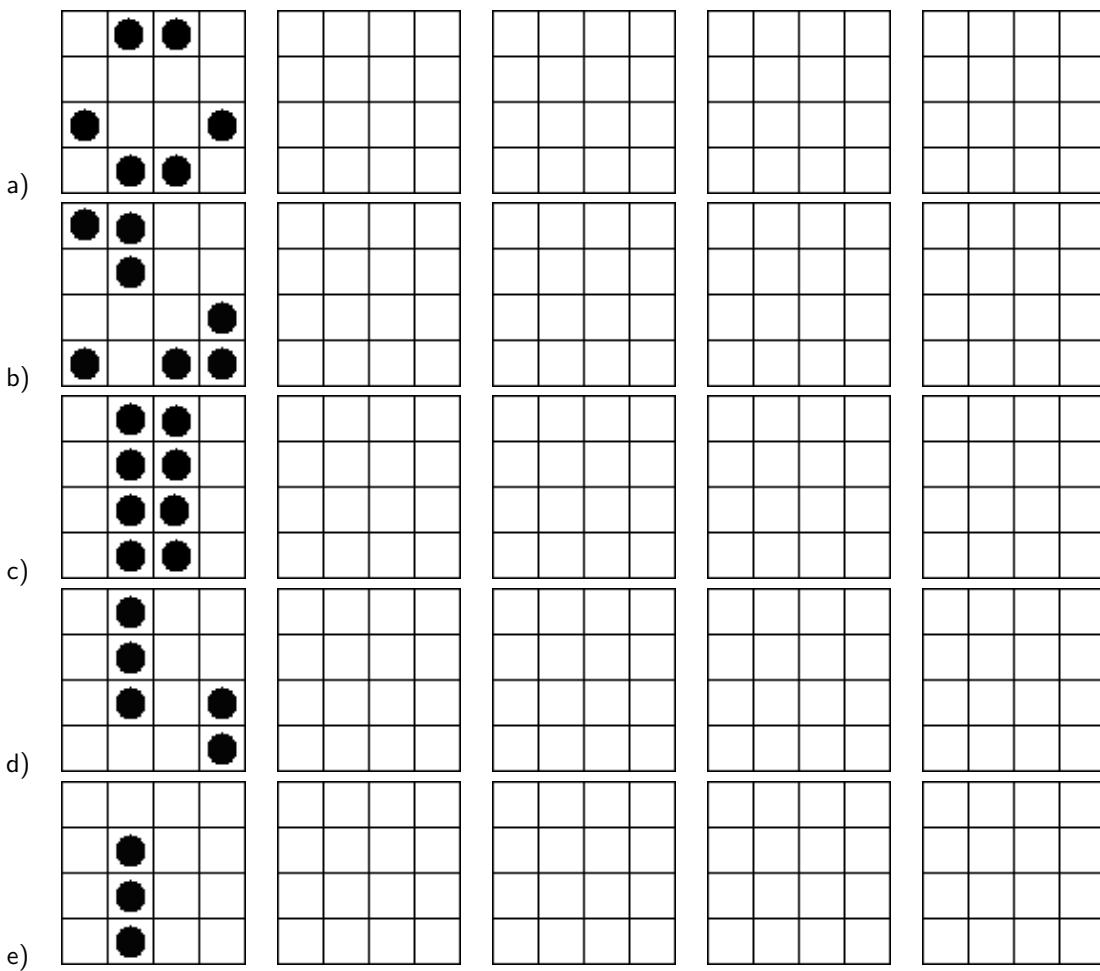
Wenn eine leere Zelle genau drei lebende Nachbarn hat, entsteht in der Zelle ein neues Leben.

In dem nachfolgenden Beispiel beginnt die Simulation mit einer Population von sieben Lebewesen. Die Population wächst bis auf zwölf Lebewesen an und stirbt nach sechs Generationen aus:



Weitere Informationen und (teilweise animierte) Beispiele findest in der Datei `Game_of_Life.html` im Kurs-Repository. Du kannst die Datei direkt in Eclipse öffnen.

Führe zunächst einige einfache Simulationen per Hand durch:



Programmieraufgabe

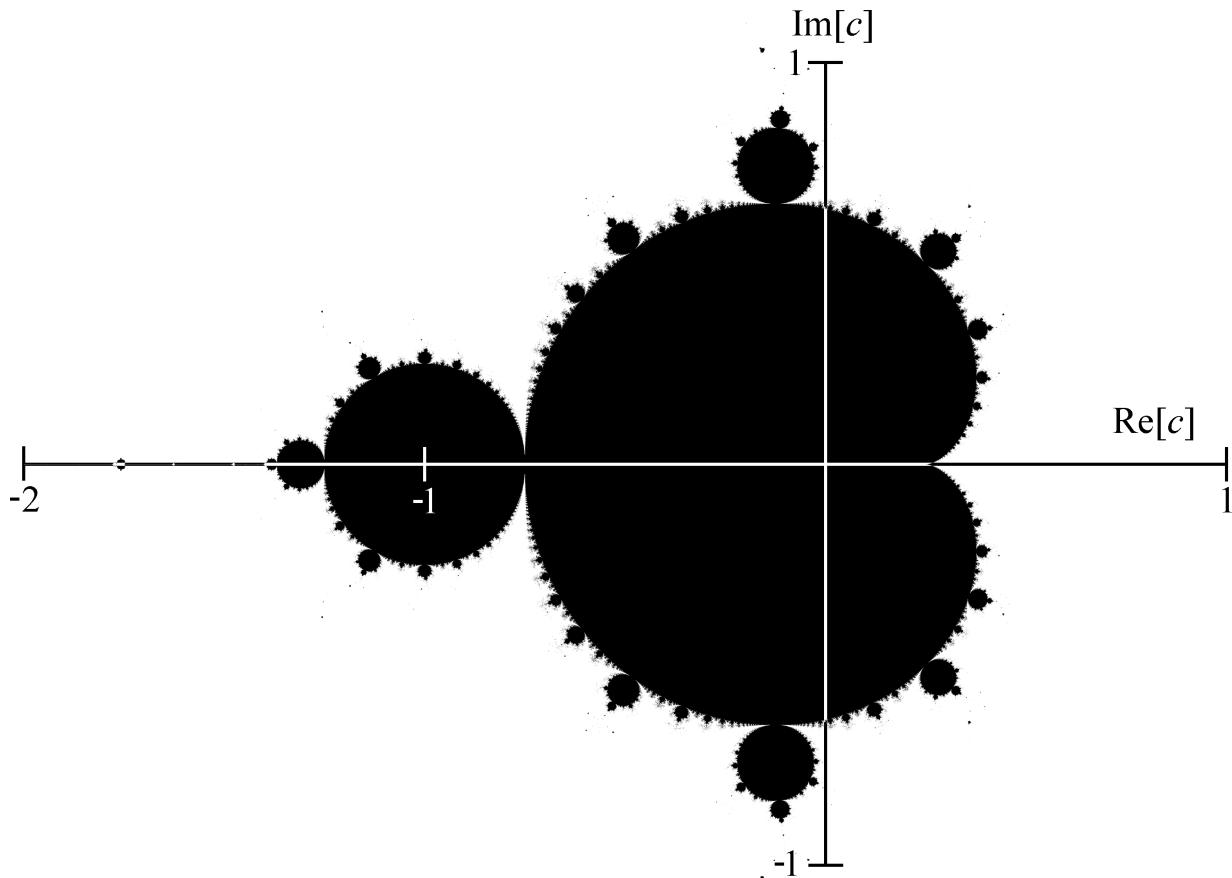
Schreibe eine Java-Anwendung, die das *Game of Life* simuliert. Gehe in folgenden Teilschritten vor:

- Zum Testen ist es hilfreich, wenn man anfangs mit wenigen Zellen arbeitet (z.B. 3*3 Zellen). Wenn alles funktioniert, kann man dann ein großes Feld erzeugen (z.B. 150*100 Zellen). Definiere deshalb Konstanten, die die Anzahl der Zellen in x- und in y-Richtung festlegen. Lege in einer weiteren Konstanten die Breite einer Zelle fest.
- Erzeuge ein zweidimensionales boolesches Array, das für jede Zelle abspeichert ob sie lebendig (`true`) oder tot (`false`) ist. Fülle das Array mit zufälligen Anfangswerten.
- Zeichne die Zellen. Eine lebendige Zelle wird als ausgefülltes Quadrat gezeichnet. Eine tote Zelle wird als hohles Quadrat gezeichnet.
- Programmiere eine Methode, die die Anzahl der lebenden Nachbarn einer Zelle zählt. Dabei wird zunächst davon ausgegangen, dass sich die Zelle in der Mitte des Feldes befindet (das heißt sie besitzt alle acht Nachbarn). Die Methode erhält die Koordinaten der Zelle (x- und y-Position) als Parameter übergeben und gibt die Anzahl der Nachbarn zurück. Teste die korrekte Funktionsweise der Methode gründlich aus, ehe du weiter arbeitest. Rufe dazu die Methode für irgendeine Zelle auf (z.B. Zelle (2|1)) und überprüfe den Rückgabewert in dem du die Anzahl der Nachbarn von Hand abzählst.
- Erweitere die Methode aus Aufgabe (d) so, dass sie auch für Zellen funktioniert, die sich am Rand des Gitternetzes befinden. Dazu musst du durch `if`-Abfragen sicherstellen, dass beim Zählen der lebenden Nachbarn niemals Koordinaten von Zellen aufrufen werden, die nicht existieren. Teste die Methode für verschiedene Rand-Zellen aus (linker Rand, rechter Rand, oberer Rand und unterer Rand).

- f) Programmiere eine Methode, die entsprechend der Regeln von Conway berechnet, ob eine bestimmte Zelle im nächsten Zyklus lebt oder nicht. Die Methode erhält als Parameter die x- und y-Koordinate der Zelle übergeben sowie die Anzahl ihrer Nachbarzellen (die mit der Methode aus Aufgabe e) ermittelt wird). Der Rückgabewert ist ein boolescher Wert, der angibt, ob die Zelle im nächsten Zyklus lebt (`true`) oder nicht (`false`).
- g) Rufe in der `myPaint()`-Methode des Anwendungsfensters die Methoden aus Aufgabe e) und f) für jede Zelle des Feldes einmal auf (durch zwei ineinander geschachtelte Schleifen), um den nächsten Lebenszyklus zu berechnen. Wichtig ist, dass die neuen „Lebenswerte“ nicht sofort in das boolesche Array hineingeschrieben werden, weil dann die Berechnung der Nachbarzellen nicht mehr korrekt erfolgen würde. Deshalb muss ein zweites Array von booleschen Werten angelegt werden, das die neuen „Lebenswerte“ vorübergehend abspeichert. Nachdem die neuen „Lebenswerte“ aller Zellen berechnet wurden, werden die booleschen Werte des Hilfs-Arrays in das echte Array hinüber kopiert. Dazu benötigt man wieder zwei ineinander geschachtelte Schleifen um jeden Wert einzeln zu kopieren. Anschließend wird das veränderte Feld mit dem in c) programmierten Code erneut gezeichnet.
- h) Füge in das Anwendungsfenster ein Objekt der Klasse `Timer` ein, damit die `myPaint()`-Methode in regelmäßigen Zeitabständen aufgerufen wird, um einen neuen Lebenszyklus zu berechnen.

Aufgabe 7: Mandelbrot-Menge

Ende der achtziger und Anfang der neunziger Jahre des letzten Jahrhunderts erzeugte nicht zuletzt dank eines Bremer Wissenschaftlers (Heinz-Otto Peitgen, seit 2013 Präsident der Jacobs University Bremen) die sogenannte „Chaos-Forschung“ viel Aufmerksamkeit – auch in der breiteren Öffentlichkeit. Nicht zuletzt dürfte das an den faszinierenden Bildern gelegen haben, die veröffentlicht wurden.



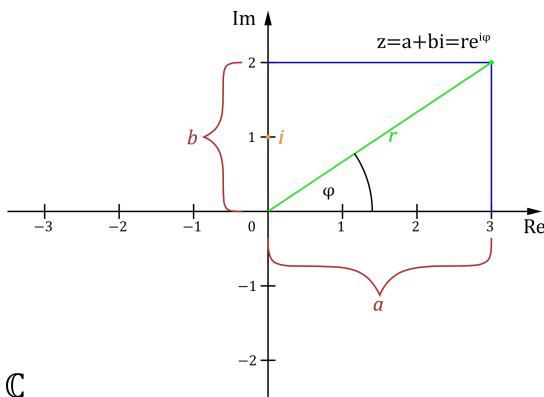
Das die Chaos-Forschung damals solch einen Aufschwung genommen hat, hängt eng mit der Entwicklung immer leistungsfähigerer Computer und – in diesem Zusammenhang bestimmt ebenso wichtig – auch geeigneter

Ausgabegeräte (hochauflösende Grafikbildschirme) zusammen. Noch zehn Jahre zuvor hätten die technischen Möglichkeiten noch nicht gereicht.

Die Mandelbrot-Menge (deren Darstellung oft als „Apfelmännchen“ bezeichnet wird) war dabei der populärste Betrachtungsgegenstand.

Das auch viele Hobby-Programmierer sich seit damals gerne dem Apfelmännchen widmen, liegt neben der Schönheit der resultierenden Grafiken sicher auch daran, dass die Berechnungen ziemlich einfach zu implementieren sind.

Wenn man verstehen will, was bei der Berechnung passiert (was nicht unbedingt nötig ist), dann muss man zunächst wissen was komplexe Zahlen sind. Die Zahlen, die ihr aus der Schule kennt sind ganze Zahlen, rationale oder auch irrationale Zahlen. All diesen Zahlen ist gemein, dass man sie auf einem Zahlenstrahl – also in einer Dimension – darstellen kann. Bei den komplexen Zahlen kommt eine zweite Dimension hinzu. Neben dem uns bereits bekannten reellen Anteil (das was wir als Zahlen auf dem Zahlenstrahl bereits kennen), kommt ein imaginärer Anteil hinzu. Die Menge der komplexen Zahlen wird nicht auf einem Zahlenstrahl, sondern in einer Ebene Dargestellt:



Die komplexe Zahlen c besteht aus dem Realteil x und dem Imaginärteil y . Man kann schreiben

$$c = x + i \cdot y$$

Die Komplexe Zahl ist also die Summe aus ihrem Realteil und dem reellen Vielfachen der imaginären Einheit i . Mit $i = \sqrt{-1}$.

Unsere „normalen“ Zahlen sind damit ein Sonderfall der komplexen Zahlen, nämlich die komplexen Zahlen mit dem Imaginärteil 0.

Die Ebene der komplexen Zahlen lässt sich problemlos in einem uns bereits gut vertrauten Koordinaten-System abbilden: Auf der x-Achse wird dann der Realteil und auf der y-Achse der Imaginärteil der komplexen Zahl aufgetragen. Da jedes Pixel auf dem Bildschirm durch eine x- und eine y-Koordinate beschrieben wird kann man auch jedem Pixel einer komplexen Zahl zuordnen. Lediglich die Wahl des Ausschnitts aus der komplexen Zahlebene muss noch verabredet werden um eine eindeutige Zuordnung von Bildschirmkoordinaten zu komplexen Zahlen zu bekommen.

Das Apfelmännchen berechnet man, in dem man für jeden Bildpunkt (genauer: für die komplexe Zahl, die dem Bildpunkt entspricht) die Folge

$$z_{n+1} = z_n^2 + c \quad (18.1)$$

und dem Anfangsglied $z_0 = 0$ berechnet. Die Folgeglieder z_n sind komplexe Zahlen. c ist die komplexe Zahl, die der jeweiligen Bildschirmkoordinate entspricht.

Für jeden Bildpunkt muss diese Folge berechnet werden und die Frage beantwortet werden: strebt die Folge ins Unendliche oder bleibt sie beschränkt? Dazu betrachtet man den Betrag der Folgeglieder.

Der Betrag einer komplexen Zahl ist definiert als

$$\sqrt{x^2 + y^2}$$

Soviel zum mathematischen Hintergrund.

In Java rechnen wir nicht mit komplexen Zahlen, sondern mit reellen Zahlen (`double`). Für jede komplexe Zahl brauchen wir zwei reelle Zahlen: eine für den Realteil und die andere für den Imaginärteil. Oder anders ausgedrückt: eine für die x-Koordinate und eine für die y-Koordinate.

Der Realteil der Folgeglieder berechnet sich nach

$$x_{n+1} = x_n^2 + y_n^2 + c_x$$

und der Imaginärteil nach

$$y_{n+1} = 2 \cdot x_n \cdot y_n + c_y$$

Die Frage lautete ja: Ist die Folge für den gewählten Punkt in der komplexen Zahlenebene beschränkt oder strebt sie ins Unendliche?

Man hat heraus gefunden, dass 18.1 nach unendlich strebt, wenn der Betrag der Folgeglieder den Wert 4 übersteigt. Die Berechnung der Folgeglieder kann also abgebrochen werden, sobald der Betrag größer als 4 geworden ist. Das spart viel Rechenzeit!

Aber was ist, wenn die Folge diesen Grenzwert noch nicht überschritten hat. Es ist durchaus möglich, dass sie es später noch tun wird! Um diese Frage einwandfrei zu klären, müsste man im Zweifelsfall unendlich viele Folgeglieder berechnen. Unendlich viele Folgeglieder sind aber selbst für sehr schneller Computer zu viel. Also müssen wir die Berechnung auf eine endliche Anzahl von Folgegliedern beschränken. Dabei gilt es einen Kompromiss zu finden: je mehr Folgeglieder berechnet werden, desto genauer wird das Ergebnis. Andererseits steigt der Rechenaufwand.

In Java übersetzt lässt sich die Folge 18.1 so berechnen:

```
public int punktIteration(double cx, double cy, double maxBetragQuadrat, int maxIter) {
    double betragQuadrat = 0.0;
    int iter = 0;
    double x = 0.0;
    double y = 0.0;
    double xTemp;

    while ((betragQuadrat <= maxBetragQuadrat) && (iter < maxIter)) {
        xTemp = x * x - y * y + cx;
        y = 2 * x * y + cy;
        x = xTemp;
        iter++;
        betragQuadrat = x * x + y * y;
    }

    if (betragQuadrat > maxBetragQuadrat) {
        iter = 0;
    }
    return iter;
}
```

Dabei ist `maxBetragQuadrat` der Grenzwert, ab dessen Erreichen wir davon ausgehen, dass die Folge ins Unendliche strebt. Und `maxIter` ist die Anzahl der Iterationen nach denen die Berechnung abgebrochen werden soll.

Als Ergebnis liefert die Methode die Anzahl der Iterationen zurück, die sie durchgeführt hat. Das Ergebnis kann Werte zwischen 0 und `maxIter` annehmen. Je größer der Rückgabewert ist, desto länger hat die Folge gebraucht um ins Unendliche zu entweichen. Der Rückgabewert 0 ist ein Sonderfall: Wenn du dir die Methode `punktIteration()` anschaust, dann siehst du, dass dieser Wert genau dann zurück gegeben wird, wenn die Folge es auch nach `maxIter` Iterationen noch nicht geschafft hat den Grenzwert `maxBetragQuadrat` zu erreichen – wir also davon ausgehen, dass die Folge für diesen Startwert beschränkt bleibt (nicht ins Unendliche strebt).

Programmieraufgabe

Erzeuge ein zweidimensionales Array von `int`-Werten von 800 mal 600. Dieses entspricht dann einer Zeichenfläche von 800 mal 600 Pixeln.

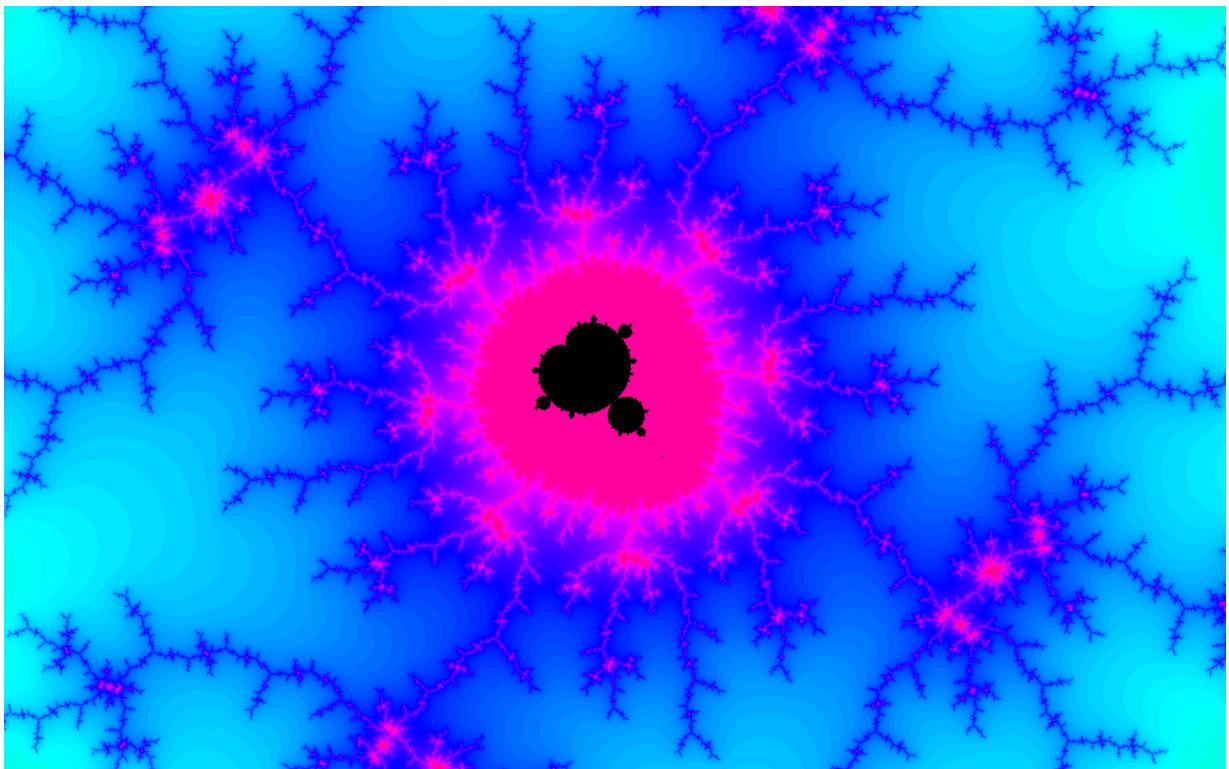
Die Zeichenfläche soll dem Bereich der komplexen Zahlen von -2.2 bis 1.1 (Realteil) und -1.25 bis 1.25 (Imaginärteil) entsprechen. Du musst also später für die Berechnung Pixelkoordinaten in Real- und Imaginärteil der komplexen Zahl c „übersetzen“.

Nun muss für jedes Element des zweidimensionalen Arrays die Methode `punktIteration()` aufgerufen werden und der Rückgabewert in dem Arrayelement gespeichert werden.

In der `myPaint()`-Methode liest du die Array-Elemente aus und weist dem zugehörigen Bildpunkt auf der Zeichenfläche einen Farbwert in Abhängigkeit vom Wert des Array-Elements zu. Im einfachsten Fall lässt du für alle Array-Elemente, die den Wert 0 enthalten einen schwarzen Punkt an der entsprechenden Koordinate auf der Zeichenfläche zeichnen (etwa mit der `Graphics`-Methode `fillRect()`, wobei du als Breite und Höhe jeweils genau ein Pixel angibst), und alle anderen Bildpunkte lässt du weiß.

Wenn du es hübscher willst, dann dann weist du den Punkten, die größere Werte als 0 haben (die also ins Unendliche divergiert sind), einen Farbwert zu, der abhängig vom tatsächlichen Wert ist. Etwa Grauwerte, die umso dunkler sind, je länger der Punkt gebraucht hat um ins Unendliche zu entkommen. Natürlich geht das auch farbig.

Wenn es dir so gelungen ist ein Apfelmännchen auf den Bildschirm zu zeichnen kannst du anfangen zu experimentieren (andere Bildausschnitte, andere Rechentiefen, andere Farbgebung).



Hinweis: `maxIter` sollte mindestens den Wert 100 haben. Je tiefer du in die Mandelbrot-Menge hinein zoomst (je kleiner der Ausschnitt ist, den du betrachtest), desto höheren Rechenaufwand musst du betreiben um die

dann sichtbar werdenden feinen Details noch auflösen zu können. Der Wert für `maxIter` muss also größer werden, je stärker du „vergrößern“ möchtest.

Aufgabe 8: Clifford-Attraktor

Betrachte die Gleichungen

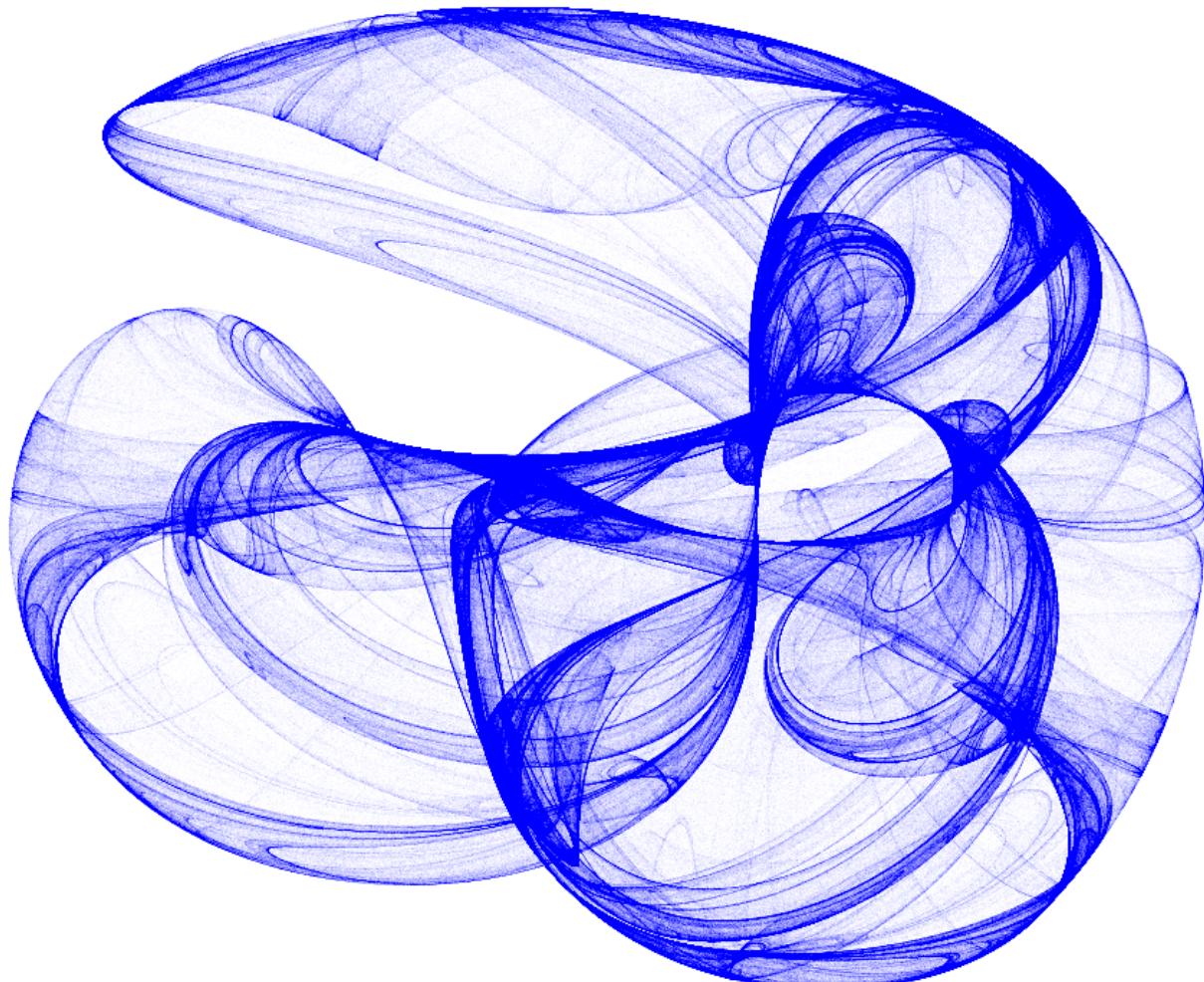
$$x_{n+1} = \sin(a \cdot y_n) + c \cdot \cos(a \cdot x_n) \quad (18.2)$$

$$y_{n+1} = \sin(b \cdot x_n) + d \cdot \cos(b \cdot y_n) \quad (18.3)$$

Du siehst hier zwei Folgen, die zum einen voneinander abhängig sind (y_{n+1} ist abhängig von x_n und x_{n+1} ist abhängig von y_n). Außerdem gibt es noch vier Parameter: a , b , c und d .

Wenn man den Verlauf der Folgenglieder betrachtet, so stellt man fest, dass diese sich (bei gegebenen Parametern a , b , c und d) – unabhängig vom gewählten Startpunkt – auf festen, wenn auch teilweise recht chaotischen Bahnen bewegen. Wobei der Begriff „Bahn“ etwas irreführend ist, da er suggeriert die Folgenglieder würden Schritt für Schritt diese Bahn abschreiten. Dem ist jedoch keineswegs so: Vielmehr springen die Folgenglieder „wild“ umher, landen bei diesen Sprüngen aber immer wieder auf dieser „Bahn“.

Solch ein „anziehendes“ Verhalten wird als *Attraktor* bezeichnet.



Die Abbildung zeigt einen nach Ihrem „Entdecker“ Clifford Pickover benannten Clifford-Attraktor (manchmal auch Pickover-Attraktor genannt) der auf den Gleichungen 18.2 und 18.3 basiert. Als Parameter wurden dabei gewählt: $a = 1.1$, $b = -1.0$, $c = 1.8$ und $d = 1.5$

Programmieraufgabe

Gehe vor wie für das Apfelmännchen beschrieben: Berechne eine große (aber endliche) Anzahl von Folgegliedern.

Für einen ersten Eindruck reichen eine halbe Million Iterationen. Für ein schönes Ergebnis sollten es schon zehn Millionen Iterationen sein.

Zähle dabei mit, wie oft Folgeglieder auf den einzelnen x- und y- Koordinaten landen, in dem du in einem zweidimensionalen Array vom Typ `int` die Trefferanzahl hochzählst. Jedes Element dieses Treffer-Arrays gibt anschließend also Auskunft darüber, wie oft es von der Folge (den x- und y-Koordinaten, die sich durch die Folgeglieder von 18.2 und 18.3 definieren) getroffen wurde.

In der `myPaint()`-Methode übersetzt du dann die Werte im Treffer-Array in Farbwerte für die entsprechenden Bildschirmkoordinaten.

Interessant ist die Wahl der Startwerte für die Folgen x und y : Du kannst entweder darauf vertrauen, dass das oben gesagte stimmt, und die Folgeglieder tatsächlich unabhängig vom Startwert auf dem Attraktor landen, oder du überprüfst dies, in dem du beispielsweise alle zehntausend Iterationen neue Startwerte für x und y per Zufallsgenerator erzeugen lässt. Beide Verfahren sollten zum gleichen Ergebnis führen!

19 Sortierverfahren

Dieses Thema soll zunächst in Gruppen erarbeitet werden. Dazu wird die Klasse in drei etwa gleich große Gruppen eingeteilt. Jede Gruppe erhält ein anderes Sortierverfahren und soll dieses Verfahren verstehen. Anschließend sollen die Gruppenmitglieder vier Zahlen mit ihrem Verfahren sortieren.

Zum Ausprobieren bekommt jede Gruppe ein Kartenset!

Jede Gruppe beschreibt ihr Sortierverfahren schriftlich (Die Beschreibungen werden am Schluss der Übung für alle ausgedruckt). Außerdem sollen sie einen Vortrag über ihr Sortierverfahren für die anderen Schüler vorbereiten.

Die folgenden zwei Aufgaben sollen von jeder der drei Gruppen bearbeitet werden:

- a) Erklärt das Sortierverfahren.
- b) Sortiert mit euren Regeln die Zahlenreihe „8 3 1 7“. Schreibt dabei die Zahlenreihe nach jedem Schritt auf.

Nach den Gruppen-Vorträgen sollen die Übungen in Einzelarbeit gelöst werden.

19.1 Gruppe 1: Sortieren durch Einfügen (Insertion Sort)

Nach welchen Regeln werden die Karten sortiert?

Tipp: Gehe von unten nach oben. Markiere die Kartenmenge, die sortiert ist. Der sortierte Teil wird von oben her immer größer.

| | | | | | | | | |
|----------------------|--|--|--|--|--|--|--|--|
| Ausgangssituation: | | | | | | | | |
| Nach dem 1. Schritt: | | | | | | | | |
| Nach dem 2. Schritt: | | | | | | | | |
| Nach dem 3. Schritt: | | | | | | | | |
| Nach dem 4. Schritt: | | | | | | | | |
| Nach dem 5. Schritt: | | | | | | | | |
| Nach dem 6. Schritt: | | | | | | | | |
| Nach dem 7. Schritt: | | | | | | | | |

19.2 Gruppe 2: Sortieren durch Auswählen (Selection Sort)

Nach welchen Regeln werden die Karten sortiert?

Tipp: Ab welchem Schritt steht die Karte an der richtigen Position?

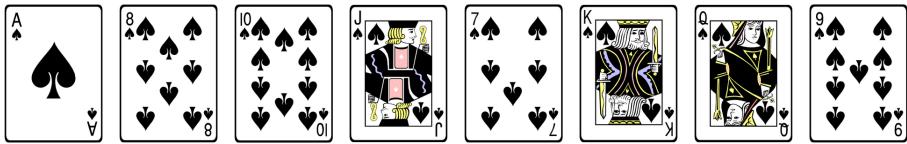
| | | | | | | | | |
|----------------------|--|--|--|--|--|--|--|--|
| Ausgangssituation: | | | | | | | | |
| Nach dem 1. Schritt: | | | | | | | | |
| Nach dem 2. Schritt: | | | | | | | | |
| Nach dem 3. Schritt: | | | | | | | | |
| Nach dem 4. Schritt: | | | | | | | | |
| Nach dem 5. Schritt: | | | | | | | | |
| Nach dem 6. Schritt: | | | | | | | | |
| Nach dem 7. Schritt: | | | | | | | | |

19.3 Gruppe 3: Bubble Sort

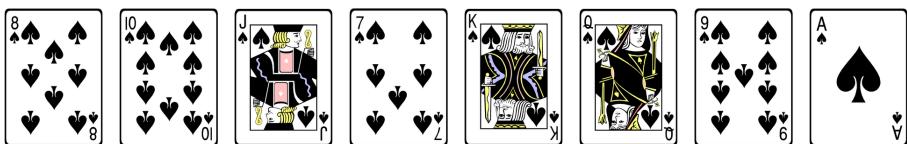
Nach welchen Regeln werden die Karten sortiert?

Tipp: Beachte, welche Karten nach rechts wandern. Achte auch darauf wie weit die Karten nach rechts wandern!

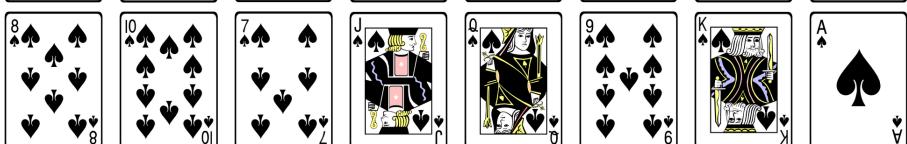
Ausgangssituation:



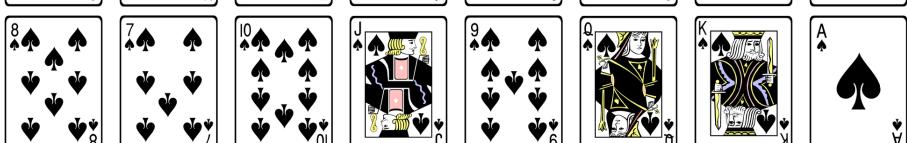
Nach dem 1. Schritt:



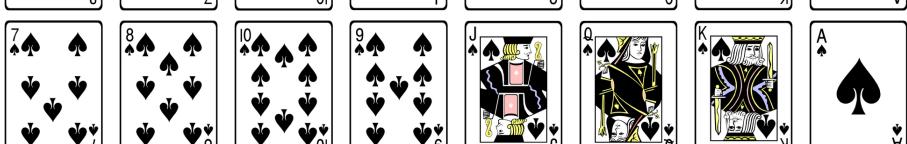
Nach dem 2. Schritt:



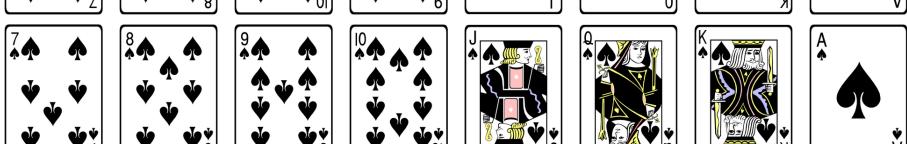
Nach dem 3. Schritt:



Nach dem 4. Schritt:



Nach dem 5. Schritt:



Nach dem 6. Schritt:



19.4 Sortierverfahren – Übungen

Aufgabe 1: Sortieren der Zahlenreihe „8 3 2 1“

Sortieren durch Einfügen (Insertion Sort)

Zuerst wird überprüft, ob das zweite Element vor das erste geschoben werden muss.

Nachdem die ersten beiden Elemente in der richtigen Reihenfolge sind, wird das dritte Element in die bereits sortierte Teilliste eingefügt.

Nachdem die ersten drei Elemente in der richtigen Reihenfolge sind, wird das vierte Element in die bereits sortierte Teilliste eingefügt.

Und so weiter.

Sortieren durch Auswählen (Selection Sort)

Es wird zuerst das kleinste Element herausgesucht und mit dem Element an der ersten Stelle vertauscht. Damit ist das vorderste Element korrekt sortiert.

Anschließend wird aus der unsortierten Restliste das zweit kleinste Element herausgesucht und mit dem Element an der zweiten Position vertauscht. Damit sind die beiden vordersten Elemente korrekt sortiert.

Danach wird aus der Restliste das dritt kleinste Element herausgesucht und mit dem Element an der dritten Position vertauscht, und so weiter.

Bubble Sort

Zuerst wird das erste Element mit dem zweiten Element verglichen und gegebenenfalls vertauscht. Dann wird das zweite Element mit dem dritten Element verglichen und gegebenenfalls vertauscht. Anschließend wird das dritte Element mit dem vierten Element verglichen und gegebenenfalls vertauscht, und so weiter bis das vorletzte Element mit dem letzten Element verglichen und (bei Bedarf) vertauscht wurde.

Das oben beschriebene Verfahren wird so lange wiederholt, bis bei einem Zeilendurchgang keine Vertauschungen mehr durchgeführt werden mussten. Dann befinden sich alle Zahlen in der richtigen Reihenfolge.

Aufgabe 2: Namen sortieren

Du sollst die folgenden Namen alphabetisch ordnen: Meier, Zeller, Müller, Egger

Beginne mit dieser Reihenfolge. Schreibe für alle drei Sortierverfahren die Reihen nach jedem Sortierschritt auf.

Aufgabe 3: Welches Verfahren sortiert am schnellsten?

Um die Zeit für ein Sortierverfahren exakt zu bestimmen, müsste man die Anzahl der Vergleiche zwischen zwei Elementen zählen und mit der dafür benötigten Rechenzeit multiplizieren. Des Weiteren müsste die Anzahl der Vertauschungen von zwei Elementen gezählt werden und wiederum mit der dafür benötigten Rechenzeit multiplizieren werden. Wenn man beide Werte zusammenaddiert, kann man die Rechenzeit ziemlich exakt bestimmen.

Wir wollen vereinfacht davon ausgehen, dass ein Verfahren dann besonders schnell ist, wenn nur wenige „Zeilen“ für die Lösung benötigt werden. Für jede der unten stehenden Zahlenreihen gibt es nach dieser vereinfachten Betrachtungsweise ein Sortierverfahren, das optimal ist. Welches ist es jeweils?

- a) 4 1 2 3
- b) 3 2 1 4
- c) 2 1 3 4

Aufgabe 4: Programmierübung

- a) Erstelle ein neues Java-Anwendungsfenster. Erzeuge im Konstruktor der Anwendung eine Zufallszahl zwischen eins und zehn, die die Länge der Zahlenliste festlegt. Gib die Zahl zum Test in einem Dialog aus.
- b) Erzeuge im Konstruktor ein Array mit Zufallszahlen vom Typ Integer. Das Array soll die unter a) festgelegte Länge erhalten. Schreibe in die Felder des Arrays Zufallszahlen zwischen 10 und 99.
- c) Schreibe eine Methode, die ein Zahlen-Array in einer bestimmten Zeile auf dem Bildschirm ausgibt. Die Methode erhält als Parameter das **Graphics**-Objekt, das Zahlen-Array und die y-Koordinate der Zeile. Rufe die Methode in der **myPaint()**-Methode des Anwendungsfensters auf.
- d) Schreibe eine Methode, die die größte Zahl der Liste ermittelt. Die Methode erhält als Parameter das Array und gibt als Rückgabewert die größte Zahl zurück. Rufe die Methode im Konstruktor auf und gib das Ergebnis der Methode in einer Messagebox aus.
- e) Schreibe eine Methode, die ein Array von Integer-Zahlen als Parameter erhält und die Zahlen-Liste mit dem Bubble-Sort Verfahren sortiert. Die Methode gibt keinen Rückgabewert zurück. Ein Rückgabewert ist nicht nötig, da nur die Speicheradresse der Zahlenliste übergeben wird und das als Parameter übergebene Array deshalb automatisch mit verändert wird.
Rufe in der **myPaint()**-Methode des Anwendungsfensters die Sortier-Methode für das in Teil b) erzeugte Array auf und gib die sortierte Liste mit der Ausgabe-Methode aus Teilaufgabe c) aus. Du kannst die Ausgabe-Methode auch in die Sortermethode einbauen, um Teilschritte des Sortierverfahrens auszutesten.

Zusatzaufgaben:

- f) Programmiere eine Methode, die eine Zahlenliste nach dem Verfahren Sortieren durch Auswählen sortiert.
- g) Programmiere eine Methode, die eine Zahlenliste nach dem Verfahren Sortieren durch Einfügen sortiert.

20 Abstrakte Klassen und Interfaces

20.1 Abstrakte Klassen

Abstrakte Klassen kennst du bereits aus dem UML-Design. Mit einer abstrakten Klasse kann man Gemeinsamkeiten einer Reihe verwandter Klassen zusammen fassen (Variablen, Methoden und Assoziationen). Zum Beispiel ist Fahrzeug eine abstrakte Oberklasse für die Klassen Auto, Fahrrad und Zug.

Abstrakte Klassen werden in Java durch das Schlüsselwort `abstract` gekennzeichnet. Der Compiler sorgt dafür, dass für eine abstrakte Klasse keine Objekte erzeugt werden können.

Eine abstrakte Klasse kann nicht nur fertig ausprogrammierte Methoden enthalten. Es dürfen auch reine „Methodeköpfe“ geschrieben werden. Diese müssen mit dem Schlüsselwort `abstract` gekennzeichnet sein. Jede abgeleitete Klasse ist verpflichtet, alle aufgelisteten abstrakten Methoden zu programmieren.

Beispiel:

```
abstract public class Fahrzeug {
    protected int geschwindigkeit = 0;

    public Fahrzeug() {
        geschwindigkeit = 50;
    }

    public abstract void bewegen();
    public abstract void zeichnen(Graphics g);

    public int getGeschwindigkeit() {
        return geschwindigkeit;
    }
}
```

20.2 Interfaces

Wenn man von einer Klasse aus auf die Methoden einer fremden Klasse zugreifen will, muss man wissen, welche Methoden die fremde Klasse bereit stellt.

Aber wie kann man ein Programm erstellen, das den Code einer fremden Klasse verwendet, die zum aktuellen Zeitpunkt noch gar nicht existiert?

Beispiele:

- Das Betriebssystem muss die Druckertreiber unterschiedlicher Hersteller ansprechen können. Nach Fertigstellung des Betriebssystems werden eventuell neue Drucker hergestellt, für die es auch geeignete Druckertreiber geben muss.
- Das Java-System soll eine bestimmte Methode eines Anwendungsprogramms aufrufen, wenn der Benutzer eine Taste gedrückt hat. Das Anwendungsprogramm wird jedoch erst nach Fertigstellung der Java-Entwicklungsumgebung programmiert.

Lösung:

Es werden eine Reihe von Methodenköpfen definiert, die beide Seiten kennen. Wenn man nur den Kopf einer Methode hinschreibt, bezeichnet man die Methode als *abstrakt*. Die Methodenköpfe bilden die *Schnittstelle* (engl. *Interface*), die beide Seiten kennen müssen, um miteinander kommunizieren zu können. Zum Beispiel legen Methodenköpfe fest, wie das Betriebssystem einen Druckertreiber ansprechen muss, und sie sagen dem Hersteller des Druckertreibers, welche Methoden er programmieren muss.

Formal sieht eine Schnittstelle so ähnlich aus wie eine Klasse. Ein Interface wird mit dem Schlüsselwort `interface` gekennzeichnet. Im Unterschied zu Klassen dürfen Interfaces nur Methodenköpfe und Konstanten (Variablen mit dem Schlüsselwort `final`) enthalten und sie besitzen keinen Konstruktor.

Alle aufgelisteten Methoden sind automatisch **abstract** und **public**. Deshalb kann man diese beiden Schlüsselwörter bei der Definition der Methoden auch weglassen.

Beispiel für ein Interface:

```
public interface Drucker {  
    public abstract String hersteller();  
}
```

Beispiel für eine Klasse, die dieses Interface implementiert:

```
public class Epson_V12 implements Drucker {  
  
    @Override  
    public String hersteller() {  
        return "Firma Epson 19.01.2007";  
    }  
}
```

Eine Klasse, die ein Interface implementiert (zum Beispiel der Druckertreiber, der das Drucker-Interface implementiert), muss alle definierten Methoden in Code umsetzen (Fachwort: implementieren). Wenn eine Klasse ein Interface umsetzen möchte, gibt sie dies durch das Schlüsselwort **implements** an. Der Compiler achtet darauf, dass die Klasse alle Methoden des Interfaces besitzt. Vor allen Methoden, die aus dem Interface „geerbt“ werden, muss zwingend das Schlüsselwort **public** stehen.

Eine Klasse darf gleichzeitig ein oder mehrere Interfaces implementieren und sich von einer anderen Klasse ableiten. Schema:

```
public class Name extends Superklasse implements Interface1, Interface2
```

20.3 Abstrakte Klassen und Interfaces – Übungen

Aufgabe 1: Ableitung von Abstrakte Klassen

Es soll in Gruppenarbeit ein Zug mit verschiedenen Waggons entwickelt werden. Jeder programmiert einen Waggon oder eine Lokomotive (Es sollte eine Lokomotive mit Blickrichtung nach links und eine mit Blickrichtung nach rechts geben.). Hinterher kann sich jeder einen oder mehrere Züge aus den vorhandenen Waggons zusammenbauen und sie über den Bildschirm fahren lassen.

1. Waggonbau

Wichtig ist, dass alle Waggons zusammenpassen und dass alle Waggons einheitlich angesteuert werden können. Deshalb muss zunächst ein Grundentwurf gemacht werden, der für alle Waggons gilt:

- Es müssen verbindliche Richtlinien für die Maße des Waggons festgelegt werden. Zum Beispiel müssen sich die Puffer aller Waggons auf derselben Höhe befinden.
- Damit jeder jeden Waggon benutzen kann, müssen die Waggons alle dieselben Methoden bereitstellen. Dazu wird gemeinsam eine abstrakte Klasse **Waggon** entwickelt. Jeder leitet seinen eigenen Waggon von der Oberklasse **Waggon** ab.

2. Zusammenbau der Züge

Damit nicht jeder Waggon einzeln vom Anwendungsfenster aus gesteuert werden muss, sollte eine Klasse Zug geschrieben werden, die die Steuerung der einzelnen Waggons eines Zuges übernimmt.

Auch hier ist es vorteilhaft, wenn verschiedene Züge gleich angesteuert werden können. Deshalb wird auch für die Züge zunächst eine abstrakte Oberklasse geschrieben.

3. Steuerung über die Tastatur

Die Züge sollen durch Tastendruck gesteuert werden (diesen Aufgabenteil könnt ihr erst im Anschluss an das nächste Kapitel lösen). Für jeden Zug wird eine Taste festgelegt, bei deren Drücken der Zug beschleunigt wird und eine weitere Taste, durch die der Zug abgebremst wird.

Aufgabe 2: Implementierung eines Interfaces

Gegeben ist ein Interface für geometrische Figuren:

```
public interface Geometrie {
    boolean istEckig();           // gibt zurück, ob die Figur eckig ist oder nicht
    int anzahlEcken ();          // gibt die Anzahl der Ecken zurück
}
```

Die geometrischen Figuren Kreis, Dreieck, Rechteck, Trapez und Raute sollen dieses Interface implementieren. Suche dir eine Klasse aus, und schreibe für diese Klasse den Code.

21 Tastaturereignisse

21.1 Ereignisbehandlung

Wenn ein Benutzer eine Taste drückt während das Fenster aktiviert ist, dann findet im Fenster ein sogenanntes *Ereignis* (engl. *event*) statt.

Wenn der Programmierer von Tastaturereignissen unterrichtet werden möchte, muss er eine Klasse schreiben, die das Interface **KeyListener** implementiert. In diesem Interface sind eine Reihe von Methoden definiert, die das System bei Tastaturereignissen aufruft:

| Methoden des Interfaces KeyListener | Wir aufgerufen wenn ... |
|--|---|
| <code>public void keyPressed(KeyEvent e)</code> | ... eine Taste gedrückt wurde. |
| <code>public void keyReleased(KeyEvent e)</code> | ... eine Taste losgelassen wurde. |
| <code>public void keyTyped(KeyEvent e)</code> | ... eine Taste gedrückt und wieder losgelassen wurde. |

Im Normalfall benötigt man nur die Methode `keyPressed()`. Die beiden anderen Methoden müssen jedoch zumindest mit leeren Methodenrumpfen vorhanden sein.

Damit die Tastaturereignisse an die Klasse mit dem implementierten Interface weitergeleitet werden, muss man die Klasse beim Anwendungsfenster „ anmelden“. Dazu besitzt die Klasse **JFrame** folgende Methode:

```
public void addKeyListener(KeyListener l)
```

Beim Aufruf einer Methode aus dem Interface **KeyListener** übergibt das System ein Objekt der Klasse **KeyEvent**. Mit einer Methode dieses Objektes kann man in Erfahrung bringen, welche Taste vom Benutzer betätigt wurde:

```
public char getKeyChar()
```

Die Methode `getKeyChar()` gibt den zur Taste gehörenden Buchstaben zurück. Der Buchstabe ist vom Datentyp **char**.

Alternativ bekommt man mit

```
public int getKeyCode()
```

zurückgemeldet, welche Taste gedrückt wurde. Zu jeder Taste gibt es eine vordefinierte Konstante der Klasse **KeyEvent** – etwa `VK_UP` für Pfeiltaste nach oben oder `VK_N` für die Taste mit dem Buchstaben N. Im Unterschied zu `getKeyChar()` spielt es dabei keine Rolle welche Modifier-Taste(n) zusätzlich gedrückt wurden (unter Modifier-Tasten versteht man die Tasten `<Strg>`, `<Alt>`, `<AltGr>` und die Hochstelltaste zum Umschalten auf Großbuchstaben). Zur Unterscheidung zwischen großen und kleinen Buchstaben ist `getKeyCode()` folglich ungeeignet.

Achtung: `getKeyCode()` liefert nur in `keyPressed()` ein sinnvolles Ergebnis, nicht aber in `keyTyped()`!

Umgekehrt empfiehlt es sich `getKeyChar()` nur in `keyTyped()` zu benutzen: Bereits das Drücken der Shift-Taste erzeugt einen Aufruf von `keyPressed()`. `getKeyChar()` kann in diesem Moment aber noch gar keinen Buchstaben liefern (statt dessen wird der Wert `KeyEvent.CHAR_UNDEFINED` – welcher dem Integer Wert 65536 entspricht – zurück geliefert). Diesen Sonderfall müsstet ihr also in eurem Programm zusätzlich behandeln.

Einfache Faustregel: Wer `getKeyCode()` benutzen will, tut dies in `keyPressed()`. Und wer `getKeyChar()` benutzen will, tut dies in `keyTyped()`.

21.2 Datentyp **char**

Buchstaben werden in Java in zwei Byte großen Speicherbereichen abgespeichert, die mit dem Datentyp **char** bezeichnet werden. Intern werden die Buchstaben als Zahlen von 0 bis 65536 kodiert. Die Unicode-Tabelle weist jeder Zahl einen entsprechenden Buchstaben zu.

Man kann mit Charactern wie mit Zahlenvariablen arbeiten und ihnen z.B. den Zahlenwert eines Buchstabens zuweisen:

```
char x = 69;           // Buchstabe 'E'
```

Da man meistens die Buchstabencodes nicht im Kopf hat, ist es auch möglich, einen Buchstaben direkt anzugeben. Dazu muss man den Buchstaben in Hochkommata setzen:

```
char x = 'E';          // Zahlencode 69
```

21.3 Tastaturereignisse – Übungen

Aufgabe 1: Leseübung

Hier siehst du ein Beispielprogramm, das Tastatureingaben verarbeitet. Kannst du die Bedeutung der hervorgehobenen Methoden und Schlüsselwörter erklären?

```

import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import hilfe.*;

public class Zeichnen extends HJFrame implements KeyListener {
    private static final final int WIDTH = 500;
    private static final int HEIGHT = 500;
    private static final Color BACKGROUND = Color.WHITE;
    private static final Color FOREGROUND = Color.BLACK;
    private boolean bKreisZeichnen = false;

    public Zeichnen(final String title) {
        super(WIDTH, HEIGHT, BACKGROUND, FOREGROUND, title);
        addKeyListener(this);
    }

    @Override
    public void myPaint(Graphics g) {
        if (bKreisZeichnen) {
            g.drawString("Wenn du 'L' drückst, lösche ich den Kreis.", 20, 50);
            g.drawOval(50,100, 100,100);
        } else {
            g.drawString("Wenn du 'M' drückst, male ich einen Kreis.", 20, 50);
        }
    }

    @Override
    public void keyPressed(KeyEvent e) {
        int c = e.getKeyCode();
        switch (c) {
            case KeyEvent.VK_M:           // Taste 'M' (egal ob groß oder klein)
                bKreisZeichnen = true;
                repaint();               // Bildschirm neu malen
                break;
            case KeyEvent.VK_L:           // Taste 'L'
                bKreisZeichnen = false;
                repaint();               // Bildschirm neu malen
        }
    }

    @Override
    public void keyReleased(KeyEvent e) {}

    @Override
    public void keyTyped(KeyEvent e) {}

    public static void main(final String[] args) {
        EventQueue.invokeLater(new Runnable() {

```

```

        public void run() {
            try {
                Zeichnen anwendung = new Zeichnen("Zeichnen");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    );
}
}

```

Aufgabe 2: Game of Life

Das „Game of Life“ soll mit Hilfe von Tasturbefehlen gesteuert werden können.

a) Neustart

Wenn der Benutzer ein kleines oder großes 'N' drückt, soll das zweidimensionale boolesche Array mit neuen Zufallswerten versehen werden, so dass die Simulation quasi neu startet.

b) Anhalten und weiter machen

Der Ablauf der Simulation kann angehalten und anschließend wieder fortgesetzt werden. Wenn der Benutzer ein kleines oder großes 'A' drückt, soll der Timer ausgeschaltet werden, damit das Bild eingefroren wird. Wenn der Benutzer ein kleines oder großes 'W' drückt wird der Timer wieder gestartet.

c) Geschwindigkeit variieren: langsam, mittel oder schnell

Der Ablauf der Simulation kann in verschiedenen Geschwindigkeiten geschehen. Dazu gibt es drei Stufen: 'l' für langsam, 'm' für mittel und 's' für schnell. Zu Beginn startet die Simulation mit einer mittleren Geschwindigkeit. Zum Ändern der Geschwindigkeit kann die Methode `setDelay()` des `Timer`-Objektes benutzt werden. So wird mit

```
timer.setDelay(10)
```

der bereits existierende Timer so verändert, dass die `myPaint()`-Methode alle 10 Millisekunden aufgerufen wird.

Achte darauf, dass die Simulation nach einer Pause (die in Teil b programmiert wurde) mit der Geschwindigkeit des zuletzt gewählten Levels weiter läuft.

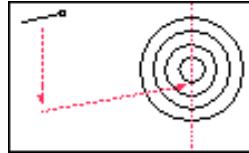
Aufgabe 3: Stoppuhr

Vor einiger Zeit haben wir eine Klasse `Uhr` programmiert. Leite eine Unterklasse von der Klasse `Uhr` ab, und erweitere sie zur Stoppuhr. Die Stoppuhr soll mit dem Messen der Zeit beginnen, wenn die Taste 'S' gedrückt wird (für „Start“). Wenn die Taste ,E' gedrückt wird (für „Ende“), soll die Zeitmessung beendet werden. Das Drücken auf 'R' (für „Reset“) setzt die Uhrzeit auf 00:00 zurück.

Die ursprüngliche Klasse `Uhr` soll unverändert erhalten bleiben, damit man sie weiterhin benutzen kann. Die Klasse `Ziffer` darf jedoch um eine Methode zum Zurückstellen der Ziffer auf Null ergänzt werden.

Aufgabe 4: Dartspiel

Ein Pfeil wird auf eine Dartscheibe abgeschossen. Der Benutzer muss versuchen, die Mitte der Zielscheibe zu treffen.



Auf dem Bildschirm werden eine fest positionierte Zielscheibe und ein Pfeil dargestellt. Der Pfeil ist leicht nach oben geneigt, damit das Spiel nicht zu einfach wird. Das Spiel beginnt, sobald der Benutzer die Taste 'N' drückt (für „Neues Spiel“). Der Pfeil fällt senkrecht nach unten. Der Benutzer muss den Pfeil zum richtigen Zeitpunkt durch Drücken der Taste 'S' (für „Schuss“) abschießen. Der Pfeil fliegt dann in der eingeschlagenen Richtung auf die Scheibe zu. Er landet, sobald er die imaginäre Mittellinie der Scheibe erreicht hat. Drückt man die Taste 'N' erneut, so beginnt ein neues Spiel. Der Pfeil wird wieder an die Anfangsposition gesetzt und fällt sofort senkrecht nach unten.

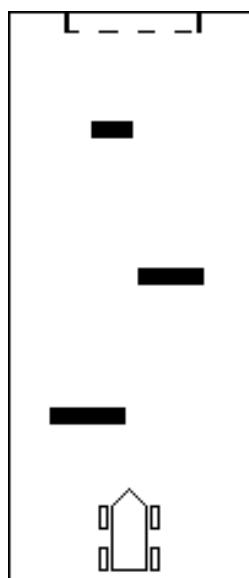
Tipps: Für die Zielscheibe braucht man keine eigene Klasse programmieren. Sie kann in der `myPaint()`-Methode des Anwendungsfensters gezeichnet werden. Eine eigene Klasse muss nur für den Pfeil geschrieben werden. Überlege dir zunächst, welche verschiedenen Zustände ein Pfeil besitzt. Zeichne dazu ein UML-Zustandsdiagramm. Beachte auch, dass der Pfeil die x-Position der Scheiben-Mitte kennen muss, damit er weiß, wann er sein Ziel erreicht hat. Die Mittellinie und die Geschwindigkeit des Pfeils in x-Richtung sollten so aufeinander abgestimmt werden, dass der Pfeil die Mittellinie genau trifft.

Aufgabe 5: Autorennen

Ein Auto muss in möglichst kurzer Zeit um eine Reihe von Hindernissen herum navigiert werden. Es gibt eine Taste zum Erhöhen der Geschwindigkeit und eine Taste zum Abbremsen. Zwei weitere Tasten ermöglichen es, das Auto während der Fahrt nach links oder rechts zu steuern (es reicht, einfach die „x-Position“ zu verschieben). Wenn das Auto gegen ein Hindernis prallt, hat der Benutzer das Spiel verloren. Das Programm misst die Zeit des Spielers und zeigt sie am Ende an. Die Zeitmessung kann mit der statischen Methode `currentTimeMillis()` aus der Klasse `System` erfolgen:

```
long t1, t2;
t1 = System.currentTimeMillis();
... hier passiert was auch immer du zeitlich messen möchtest
t2 = System.currentTimeMillis();
System.out.println("Es sind " + (t2 - t1) / 1000.0 + " Sekunden vergangen");
```

Es empfiehlt sich, das Rennen aus der Vogelperspektive zu zeichnen (siehe Skizze).



Mögliche Erweiterung:

Zwei Spieler spielen gleichzeitig gegeneinander. Jeder Spieler hat sein eigenes Auto, das er mit einer eigenen Tastenkombination steuert.

Aufgabe 6: Streifen

- a) Programmiere eine Klasse **Streifen**. Im Konstruktor der Klasse **Streifen** soll zunächst nur die y-Position eines Streifens als Parameter übergeben werden. Die x-Position wird fest auf 100 gesetzt und die Breite und die Höhe werden jeweils auf 30 Pixel eingestellt. Alle Variablen der Klasse sollen vor dem Zugriff von außen versteckt sein. Die Klasse **Streifen** enthält eine Methode **zeichnen()**, die den Streifen als gefülltes rotes Rechteck mit den eingestellten Koordinaten malt.

Erzeuge im Anwendungsfenster in einer Schleife ein Array von zehn Streifen. Der erste Streifen soll die y-Position 40 besitzen. Der zweite Streifen besitzt die y-Position 80, der dritte die y-Position 120, der vierte die y-Position 160, und so weiter.

- b) Die Breite der Streifen soll durch Tastendrücke verstellt werden können. Dazu erhält jeder Streifen in alphabetischer Reihenfolge jeweils einen Buchstaben in Klein- und in Großschreibung zugewiesen. Der erste Streifen erhält 'a' und 'A', der zweite 'b' und 'B', und so weiter. Der letzte Streifen wird durch 'j' und 'J' gesteuert.

Wenn man den Buchstaben als Großbuchstaben drückt (z.B. 'C'), soll die Breite des entsprechenden Streifens um zehn Pixel wachsen. Wenn man den Buchstaben als Kleinbuchstaben drückt (z.B. 'c'), soll die Breite um zehn Pixel verringert werden.

Sie soll allerdings nie unter zehn Pixel fallen. Wenn man bei einer Breite von zehn Pixeln den Kleinbuchstaben drückt, wird die Eingabe ignoriert. Beachte, dass das Anwendungsfenster nach jedem Tastendruck neu gezeichnet werden muss. Erweitere den Konstruktor der Klasse **Streifen** so, wie es für die Lösung dieser Aufgabe erforderlich ist.



22 Mausereignisse

22.1 Normale Mausereignisse

Ebenso wie Tastaturereignisse können in einem Frame auch Mausereignisse abgefangen werden. Wenn man Mausereignisse abfangen möchte, muss man das Interface `MouseListener` implementieren:

| Methoden des Interfaces <code>MouseListener</code> | Wir aufgerufen wenn ... |
|--|---|
| <code>public void mouseClicked(MouseEvent e)</code> | ... eine Maustaste gedrückt und wieder losgelassen wurde. |
| <code>public void mousePressed(MouseEvent e)</code> | ... eine Maustaste gedrückt wurde. |
| <code>public void mouseReleased(MouseEvent e)</code> | ... eine Maustaste losgelassen wurde. |
| <code>public void mouseEntered(MouseEvent e)</code> | ... der Mauszeiger das Fenster „betreten“ hat. |
| <code>public void mouseExited(MouseEvent e)</code> | ... der Mauszeiger das Fenster verlassen hat. |

Eine Klasse, die das Interface `MouseListener` implementiert, muss beim Frame durch Aufruf der folgenden Methode registriert werden:

```
public void addMouseListener(MouseListener l)
```

22.2 Mausbewegungen

Für Mausbewegungen gibt es ein eigenes Interface `MouseMotionListener`:

| Methoden des Interfaces <code>MouseMotionListener</code> | Wir aufgerufen wenn ... |
|--|---|
| <code>public void mouseDragged(MouseEvent e)</code> | ... die Maus bei gedrückter Maustaste bewegt wurde. |
| <code>public void mouseMoved(MouseEvent e)</code> | ... die Maus ohne gedrückte Maustaste bewegt wurde. |

Eine Klasse, die das Interface `MouseMotionListener` implementiert, muss beim Frame durch Aufruf der folgenden Methode registriert werden:

```
public void addMouseMotionListener(MouseMotionListener l)
```

22.3 Details über Mausereignisse erfahren

Alle diese Methoden erhalten als Parameter ein Objekt der Klasse `MouseEvent`, in dem Details über das Mausereignis gespeichert sind. Die Klasse `MouseEvent` besitzt unter anderem folgende Methoden:

| Methoden der Klasse <code>MouseEvent</code> (Auswahl) | Beschreibung |
|---|--|
| <code>public int getButton()</code> | Gibt an, welcher Knopf gedrückt wurde. <code>MouseEvent.BUTTON1</code> , <code>MouseEvent.BUTTON2</code> oder <code>MouseEvent.BUTTON3</code> . |
| <code>public int getClickCount()</code> | Anzahl der Mausklicks. |
| <code>public int getX()</code> | x-Position der Maus. |
| <code>public int getY()</code> | y-Position der Maus. |

22.4 Adapterklassen

Wenn man ein Interface implementiert, muss jede einzelne Methode des Interfaces vorhanden sein, auch wenn man vielleicht nur eine einzige Methode benötigt. Das kann auf die Dauer recht nervig sein. Deshalb stellt die Bibliothek für alle Interfaces eine sogenannte *Adapterklasse* bereit. Eine Adapterklasse ist eine Klasse, die ein Interface mit leeren Methodenrumpfen implementiert. Zum Beispiel gibt es für das Interface `MouseListener` die Klasse

MouseAdapter, für das Interface **MouseMotionListener** die Klasse **MouseMotionAdapter** und für das Interface **KeyListner** die Klasse **KeyAdapter**. Alle Adapterklassen befinden sich im Package `java.awt.event`.

Anstatt das Interface vollständig zu implementieren kann man sich einfach von der entsprechenden Adapterklasse ableiten und die benötigten Methoden überschreiben. Der Nachteil dabei ist, dass man sich dann von keiner anderen Klasse mehr ableiten kann, da eine Java-Klasse immer nur eine einzige Superklasse haben darf (in einigen anderen Programmiersprachen gibt es diese Regel nicht).

22.5 Mausereignisse – Übungen

Aufgabe 1: Markierungen setzen

Wenn der Benutzer mit der Maus in das Fenster klickt, wird die Stelle, auf die er geklickt hat, mit einem kleinen roten Kreis markiert. Wenn der Benutzer einen Doppelklick macht, wird statt eines Kreises ein Quadrat gezeichnet.

Anmerkung: Da der Bildschirmhintergrund vor dem Zeichnen gelöscht wird, ist immer nur eine Markierung zur Zeit sichtbar.

Aufgabe 2: Zeichenbrett

Es soll eine Art „Zeichenbrett“ programmiert werden.



- Erzeuge dazu ein Frame mit einem weißen Hintergrund. Wenn der Benutzer mit der Maus in das Fenster klickt, wird das Pixel an der Stelle, auf die geklickt wurde, schwarz gezeichnet (Anmerkung: mit der **Graphics**-Methode **drawLine()** kann man auch Punkte zeichnen). Wenn man mit der gedrückten Maus über das Fenster fährt, werden ebenfalls alle Pixel, über die die Maus fährt, schwarz eingefärbt. Alle „alten“ Punkte, die der Benutzer zuvor gezeichnet hat, sollen immer wieder mit gezeichnet werden. Das Programm muss sich also die Farbe aller Pixel merken. Lege dazu ein zweidimensionales Array vom Typ **Color** an, das in seiner Größe der Breite und Höhe des Fensters entspricht. Nachdem du das Array erzeugt hast, haben alle Einträge zu Beginn den Wert **null** („kein Color-Objekt vorhanden“). Wenn der Benutzer einen Punkt anklickt, erhält der entsprechende Eintrag im Array die Farbe **Color.BLACK** zugewiesen. Da der Hintergrund weiß ist, brauchen beim Zeichnen nur die Pixel tatsächlich gemalt werden, bei denen der Array-Eintrag ungleich **null** ist. Wenn man jedes Mal alle Pixel neu malen würde, würde das das System zeitlich zu sehr belasten.
- Wenn der Benutzer ein kleines oder großes 'L' drückt, wird das bisher gezeichnete Bild gelöscht und der Bildschirm ist wieder weiß. Setze dazu alle Einträge im Array auf den Wert **null** zurück.
- Erweitere das Zeichenbrett so, dass man auch farbige Zeichnungen machen kann. Der Benutzer kann die Farbe des Zeichenstiftes über die Tastatur verstellen, z.B:
 'r' oder 'R': rot
 'b' oder 'B': blau
 'g' oder 'G': grün

's' oder 'S': schwarz

Zu Beginn ist die Farbe schwarz ausgewählt.

- d) Über die Tasten '1' bis '9' kann man die Stift-Breite einstellen. Bei einem Maus-Klick wird immer ein rechteckiges Kästchen mit der angegebenen Breite eingefärbt, bei einer Stiftbreite von 4 werden zum Beispiel 4*4 Pixel gefärbt.

In der Methode `keyTyped()` kann man die Stiftbreite mit folgendem Code-Auszug geschickt einstellen:

```
if (e.getKeyChar() >= '1' && e.getKeyChar() <= '9') {
    breite = e.getKeyChar() - '0';
}
```

Dieser Code nutzt die Tatsache aus, dass die Buchstaben von '0' bis '9' in der Zeichentabelle fortlaufende ASCII-Codes besitzen. '0' wird durch die Zahl 48 kodiert, '1' durch die Zahl 49, ... '9' durch die Zahl 57.

Aufgabe 3: Ameisen fangen

Grundidee: Im Anwendungsfenster wird an irgendeiner zufällig ausgewählten (festen) Position eine Ameise angezeigt, die der Benutzer möglichst schnell durch einen Mausklick einfangen muss. Wenn er die Ameise getroffen hat, verschwindet sie und eine andere Ameise wird an anderer, zufällig ausgewählter Stelle angezeigt. Der Benutzer muss in einer vorgegebenen Zeitspanne (zum Beispiel 15 Sekunden) möglichst viele Ameisen fangen. Damit die Anwendung interessanter wird, gibt es acht verschiedene Ameisenbilder, von denen immer eines zufällig ausgewählt wird. Die Ameisenbilder findest du im Kursrepository. Alle Ameisenbilder sind 13 Pixel breit und 13 Pixel hoch.

Im Konstruktor der Anwendung werden alle acht Ameisenbilder in ein Array geladen. Während des Spiels wird immer nur eine Ameise an einer zufälligen Position angezeigt. Ein Objekt der Klasse `Timer` wird gestartet, um den Zeitablauf überprüfen zu können. Die abgelaufene Zeit wird im Programm durch Zählen der `myPaint()`-Aufrufe ermittelt. Wenn die Zeitspanne abgelaufen ist, wird dem Benutzer in der `myPaint()`-Methode mit `drawString()` angezeigt, wie viele Ameisen er fangen konnte.

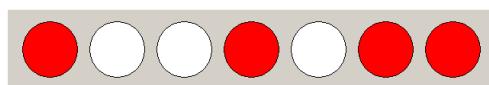
Hinweis: Neben dem Anwendungsfenster braucht keine weitere Klasse programmiert werden.

Aufgabe 4: Das Lampenproblem

Im Anwendungsfenster wird eine Reihe von Lampen positioniert. Der Benutzer hat die Aufgabe, die Lampen auszuschalten. Wenn er eine Lampe anklickt, wird jedoch nicht die Lampe selber an- oder ausgeschaltet, sondern es werden die beiden Nachbarlampen umgeschaltet.

Wenn der Benutzer im abgebildeten Beispiel die zweite Lampe von links anklickt, geht die erste Lampe aus und die dritte Lampe geht an. Die zweite Lampe selber verändert sich nicht.

Wenn der Benutzer eine der Lampen am Rand anklickt, wird nur die eine Nachbar verändert, den die Lampe hat. Klickt man im abgebildeten Beispiel die erste Lampe, geht die zweite Lampe aus.



Gehe folgendermaßen vor:

- a) Vor einiger Zeit haben wir im Unterricht eine Klasse `Lampe` programmiert. Erzeuge ein neues Anwendungsfenster mit einem Array von sieben Lampen, die wie abgebildet nebeneinander positioniert werden. Zu Beginn sind alle Lampen ausgeschaltet.

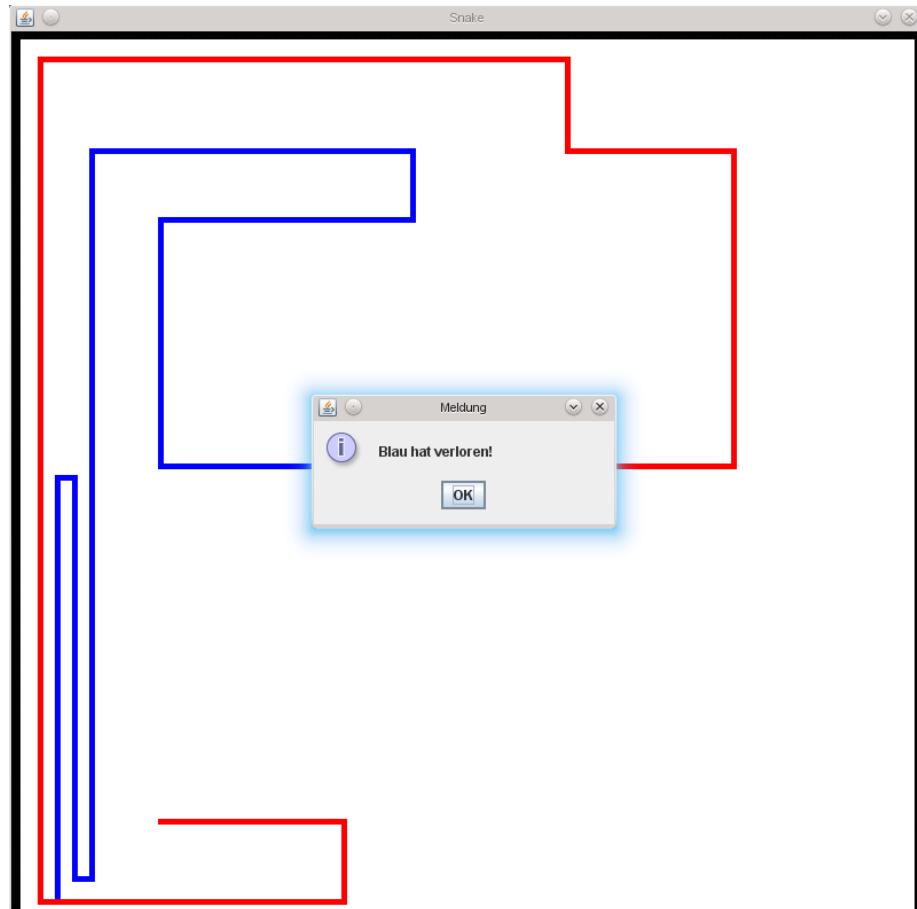
- b) Fang im Anwendungsfenster die Mausereignisse ab. Wenn der Benutzer mit der Maus in das Fenster klickt, muss die Nummer der angeklickten Lampe ermittelt werden. Gehe dazu in einer Schleife alle Lampen durch und vergleiche die Mausposition mit der Position der Lampe. Eine Lampe wird „getroffen“, wenn sich die x-Position der Maus zwischen der linken Seite der Lampe ($x\text{-Position Maus} \geq x\text{-Position Lampe}$) und der rechten Seite der Lampe ($x\text{-Position Maus} \leq x\text{-Position Lampe} + \text{Breite der Lampe}$) befinden. Außerdem muss die y-Position der Maus zwischen der oberen und der unteren Seite der Lampe liegen. Merke dir in einer Variablen, welche Lampe angeklickt wurde. Es kann auch sein, dass der Benutzer gar keine Lampe getroffen hat. Dann sollte die Variable einen ungültigen Wert haben, zum Beispiel -1. Gib den Wert der Variable zum Test auf der Konsole aus, damit du weißt, ob das Programm die getroffene Lampe richtig ermittelt.
- c) Wenn eine Lampe getroffen wurde, wird der Zustand der Nachbarlampen verändert (von an auf aus oder umgekehrt) und anschließend wird das Fenster neu gezeichnet. Beachte, dass die Rand-Lampen nur einen Nachbarn besitzen.

Aufgabe 5: Game of Life erweitern

- a) Erweitere das *Game of Life* so, dass man im angehaltenen Zustand den Zustand einer Zelle verändern kann (von lebendig zu tot oder umgekehrt) indem man mit der Maus auf die Zelle klickt.
- b) Wenn der Benutzer im angehaltenen Zustand mit der Maus über mehrere Zellen fährt, soll der Zustand aller überfahrenen Zellen umgedreht werden. Beachte dabei, dass sich die Maus auf ihrem Weg längere Zeit in derselben Zelle befindet. Der Zustand der Zelle darf natürlich trotzdem nur einmal verändert werden.

Aufgabe 6: Snake

Du sollst den Spiele-Klassiker *Snake* programmieren.



- a) Erzeuge ein Anwendungsfenster mit Höhe und Breite von je 800 Pixeln. Das Spielfeld besteht aus einzelnen Punkten (Koordinaten), die jeweils ein Quadrat von 5x5 Pixeln auf dem Bildschirm darstellen. 160X160 Punkte füllen somit das Anwendungsfenster von 800x800 Pixeln komplett aus. Das Spielfeld sollst du als zweidimensionales Integer-Array anlegen. In den einzelnen Array-Elementen kannst du für dann für jeden Punkt des Spielfeldes den Zustand speichern:

0: Dieser Punkt ist noch frei.
 1: Dieser Punkt wird vom Körper der roten Schlange belegt.
 2: Dieser Punkt wird vom Körper der blauen Schlange belegt.
 -1: Dieser Punkt ist Teil der Wand

Im Konstruktor der Klasse solltest du das Array so initialisieren, dass die beiden äußeren Reihen an jeder Seite die Mauer bilden.

In der Mitte des Spielfeldes sollte der Kopf der roten Schlange und direkt links daneben der Kopf der blauen Schlange sein. Setze die Geschwindigkeit in x-Richtung für die rote Schlange auf 1 und für die blaue Schlange auf -1. Speichere die x- und y-Position sowie die x- und y-Komponenten der Geschwindigkeit von beiden Schlangen in geeigneten Variablen.

- b) In der `myPaint()`-Methode werden zunächst auf Basis der aktuellen Koordinaten und Geschwindigkeiten die neuen x- und y-Positionen der beiden Schlangenköpfe berechnet und anschließend mit entsprechenden Werten im Spielfeld-Array markiert.

Anschließend muss die `myPaint()`-Methode das Spielfeld zeichnen, indem sie für alle Elemente des Arrays überprüft ob und in welcher Farbe ein kleines Quadrat von 5x5 Pixeln gezeichnet werden soll. Die Wand wird schwarz, die blaue Schlange blau (Wunder über Wunder) und die rote Schlange rot gezeichnet.

Damit tatsächlich Bewegung in die Sache kommt muss jetzt nur noch ein Timer gestartet werden. Starte ihn so, dass `myPaint()` alle 50 Millisekunden aufgerufen wird.

- c) Wenn du bis hierher alles richtig gemacht hast, werden die beiden Schlangen beim Programmstart in entgegengesetzte Richtungen los laufen. Rot nach rechts und blau nach links.

Erweitere das Programm jetzt so, dass die beiden Schlangen das nächste Feld nur dann besetzen, wenn dieses zuvor noch frei war. Sollte dies bei einer von beiden Schlangen nicht der Fall sein, sollst du die Geschwindigkeiten in x- und y-Richtung von beiden Schlangen auf 0 setzen und den Timer stoppen.

Anschließend sollst die Spieler über eine MessageBox informieren, welcher Spieler verloren hat (oder wenn es dir lieber ist: welcher Spieler gewonnen hat). Siehe Abbildung.

Sobald diese MessageBox von den Spielern geschlossen wird, soll das Programm mit `System.exit(0)` beendet werden.

- d) Jetzt fehlt noch die (für den Spielspaß nicht ganz unwesentliche) Möglichkeit für die beiden Spieler ihre Schlangen auch tatsächlich zu steuern.

Die rote Schlange soll durch die Tastatur gesteuert werden. Welche Tasten du für die vier Richtungen (oben, unten, rechts und links) wählst bleibt dir überlassen. Beachte, dass man immer nur abbiegen kann. Es ist also beispielsweise nicht möglich direkt umzukehren (und somit Selbstmord zu begehen). Auch kann man nicht seine Geschwindigkeit erhöhen, indem man die aktuelle Richtung noch einmal bestätigt. Wenn der Benutzer solch unmöglichen Steuerbefehle gibt sollen diese einfach ignoriert werden.

- e) Die blaue Schlange soll mit der Maus gesteuert werden. Dabei soll aber auch eine andere Logik benutzt werden als bei der Tastatursteuerung für die rote Schlange. Während dort immer absolut nach oben, unten, rechts oder links gesteuert wurde, soll der zweite Spieler seine blaue Schlange immer relativ zur aktuellen Bewegungsrichtung beeinflussen: Rechter Mausklick führt zu einer Drehung um 90° im Uhrzeigersinn und linker Mausklick zu einer Drehung um 90° gegen den Uhrzeigersinn.

23 Fenster

23.1 Klassen-Hierarchie in AWT und Swing

Wer in Java grafische Benutzerschnittstellen (GUI) programmieren will hat die Wahl zwischen AWT und Swing. Sowohl AWT (`java.awt.*`) als auch Swing (`javax.swing.*`) sind als Bibliotheken Teil jeder Java Runtime Environment (JRE) – stehen also auf allen Systemen, die Java anbieten zur Verfügung.

Swing ist etwas moderner und bietet auch mehr Funktionalität. Wir werden uns deshalb auf Swing konzentrieren. Aus dem (stark vereinfachten) Klassendiagramm rechts kann man erkennen, dass man auch bei der Swing-Programmierung nicht ohne AWT auskommt, da die Swing Klassen aus AWT-Klassen abgeleitet sind.

Swing-Klassen erkennt man an dem führenden 'J' im Klassennamen. Wie etwa `JLabel` oder `JMenu`.

`java.awt.Component`

- abstrakte Klasse
- ein „GUI-Element“ mit Größe und Position
- kann Ereignisse senden und darauf reagieren

`java.awt.Container`

- abstrakte Klasse
- kann andere Komponenten aufnehmen
- Positionierung und Anordnung von Dialog Elementen in Zusammenarbeit mit LayoutManager-Klassen

`javax.swing.JFrame`

- Hauptfenster (Top-Level-Fenster) mit Rahmen und Titelleiste
- optionales Menü
- dem Fenster kann ein Icon zugeordnet werden
- Aussehen der Maus kann verändert werden

`javax.swing.JWindow`

- Hauptfenster (Top-Level-Fenster) ohne Rahmen, Titelleiste und Menü
- Anwendung übernimmt das Zeichnen der Rahmen selbst

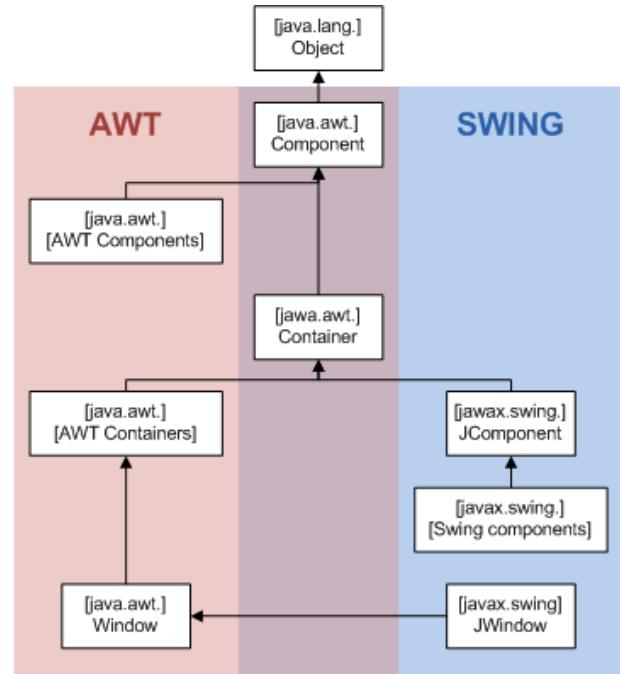
`javax.swing.JDialog`

- Hauptfenster (Top-Level-Fenster) zum Anzeigen von Dialogen

`javax.swing.JPanel`

- einfachste konkrete Ableitung von JComponent und Container
- wird verwendet, um eine Sammlung von Dialog-Elementen mit einem vorgegebenen Layout zu definieren

Als Merksatz: Die Hauptfenster-Klassen aus Swing sind direkt aus den entsprechenden Hauptfenster-Klassen des AWT abgeleitet. Alle anderen Swing-Klassen leiten sich aus `JComponent` ab.



Für uns von besonderer Bedeutung sind zunächst die Klassen `JFrame` und `JPanel`. Unsere Anwendungsfenster leiten wir aus der Klasse `JFrame` ab. Um innerhalb eines Frames Komponenten wie Buttons oder Textfelder positionieren zu können, brauchen wir aber zunächst einen Container, der diese Komponenten aufnehmen kann. Genau dies leistet `JPanel`.

23.2 Erstellung eines eigenständigen Java-Programms

Ein eigenständiges Programm muss die folgende Methode besitzen, die beim automatisch Programmstart aufgerufen wird:

```
public static void main(String[] args)
```

23.3 Erzeugung eines Programmfensters

Es muss eine eigene Klasse von der Klasse `JFrame` abgeleitet werden. Dabei ist folgendes zu beachten:

1. `JFrame`-Objekt erzeugen
2. Mit der `JFrame`-Methode `setDefaultCloseOperation()` wird definiert, was beim Schließen des Fensters geschehen soll. Wir verwenden hier typischerweise `JFrame.EXIT_ON_CLOSE` als Argument.
3. Einen Container (`JPanel`) erzeugen und auf der sogenannten „contentPane“ platzieren.
4. Komponenten hinzufügen und/oder mit `paintComponent()` zeichnen.
5. `JFrame`-Methode `setVisible(true)` macht das Fenster sichtbar.

23.4 Methoden von `JFrame` (Auswahl)

| | |
|--|--|
| <code>JFrame()</code> <code>JFrame(String title)</code> | im Konstruktor kann der Titel angegeben werden |
| <code>void setSize(int width, int height)</code> | setzt die Fenstergröße in Pixeln |
| <code>void setLocation(int x, int y)</code> | Positioniert das Fenster. Angegeben wird die linke obere Fensterecke in Pixeln. Position (0, 0) ist die linke obere Ecke des Monitors. |
| <code>void setVisible(boolean b)</code> | macht das Fenster sichtbar oder unsichtbar |
| <code>void setTitle(String title)</code> | verändert den Titel des Fensters |
| <code>void repaint()</code> | Das System löscht den Bildschirm und ruft dann die <code>paintComponent()</code> -Methoden der betroffenen Komponenten auf |

23.5 Fenster-Ereignisse

Mögliche Ereignisquellen:

`JWindow, JDialog, JFrame`

Registrierungsmethode:

`void addWindowListener(WindowListener l)`

Interface für Empfänger:

`WindowListener` [Adapterklasse: `WindowAdapter`]

Methoden des Interfaces WindowListener

| | |
|--|---|
| <code>void windowOpened(WindowEvent e)</code> | Fenster wurde geöffnet |
| <code>void windowClosing(WindowEvent e)</code> | Fenster soll geschlossen werden |
| <code>void windowClosed(WindowEvent e)</code> | Fenster wurde geschlossen |
| <code>void windowIconified(WindowEvent e)</code> | Fenster wurde auf Symbolgröße verkleinert |
| <code>void windowDeiconified(WindowEvent e)</code> | Fenster wurde wiederhergestellt |
| <code>void windowActivated(WindowEvent e)</code> | Benutzer arbeitet jetzt in dem Fenster |
| <code>void windowDeactivated(WindowEvent e)</code> | Benutzer arbeitet nicht mehr in dem Fenster |

Wichtige Methode von WindowEvent

| | |
|---------------------------------|--|
| <code>Window getWindow()</code> | gibt das Fenster zurück, in dem das Ereignis passierte |
|---------------------------------|--|

23.6 Beispiel

```

1 import java.awt.*;
2 import javax.swing.*;
3 import javax.swing.border.EmptyBorder;
4
5 public class MyJFrame extends JFrame {
6     // globale Variablen
7     private static final int WIDTH = 500;
8     private static final int HEIGHT = 500;
9     private static final Color BACKGROUND = Color.WHITE;
10    private static final Color FOREGROUND = Color.BLACK;
11    private JLabel zeichenflaeche;
12
13    public MyJFrame(final String title) {
14        super(title);
15        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16        JPanel contentPane = new JPanel();
17        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
18        contentPane.setLayout(new BorderLayout(0, 0));
19        setContentPane(contentPane);
20        zeichenflaeche = new JLabel() {
21            @Override
22            protected void paintComponent(Graphics g) {
23                super.paintComponent(g);
24                myPaint(g);
25            }
26        };
27        zeichenflaeche.setPreferredSize(new Dimension(WIDTH, HEIGHT));
28        zeichenflaeche.setOpaque(true);
29        zeichenflaeche.setBackground(BACKGROUND);
30        zeichenflaeche.setForeground(FOREGROUND);
31        zeichenflaeche.setFont(new Font("Arial", Font.PLAIN, 12));
32        contentPane.add(zeichenflaeche);
33        pack();
34        setLocationRelativeTo(null);
35        setResizable(false);
36        setVisible(true);
37    }
38

```

```

39     public void myPaint(Graphics g) {
40         // wird aufgerufen, wenn das Fenster neu gezeichnet wird
41         g.drawString("Guten Tag!", 100, 200);
42     }
43
44     public static void main(final String[] args) {
45         EventQueue.invokeLater(new Runnable() {
46             public void run() {
47                 try {
48                     MyJFrame anwendung = new MyJFrame("MyJFrame");
49                 } catch (Exception e) {
50                     e.printStackTrace();
51                 }
52             }
53         });
54     }
55 }
```

Erläuterungen zum Beispiel

Zeile 14: Im Konstruktor wird zunächst der Konstruktor der Super-Klasse (`JFrame`) aufgerufen. Der Titel des Fensters wird übergeben.

Zeile 15: Hier wird festgelegt, dass das Programm beendet werden soll, sobald der Benutzer den Schließen-Knopf des Fensters benutzt.

Zeile 16-17: Ein `JPanel`-Container wird mit Rahmen erzeugt.

Zeile 18: Der LayoutManager für diesen Container wird bestimmt.

Zeile 19: Der Container wird als `contentPane` benutzt.

Zeile 20-26: Ein neues `JLabel`-Objekt wird erzeugt. Die `paintComponent()`-Methode der `JLabel`-Klasse wird dabei überschrieben, so dass es nun möglich ist beliebige Zeichenoperationen auf dieser Komponente auszuführen (ausgelagert in die Methode `myPaint()`, die in den Zeilen 39-42 definiert wird).

Zeile 27: Die Größe des neuen `JLabel`-Objekts (nicht die des Fensters!) wird festgelegt.

Zeile 28: Die Deckkraft des `JLabel`-Objekts wird eingestellt. `true` bedeutet, dass das Objekt deckend gezeichnet wird. Mit `false` würde es durchsichtig gezeichnet.

Zeile 29-31: Vorder- und Hintergrundfarbe sowie der verwendete Font werden eingestellt.

Zeile 32: Das `JLabel`-Objekt wird auf der `contentPane` plaziert.

Zeile 33: Mit `pack()` wird die Größe des Fensters so gesetzt, dass alle Komponenten darin den nötigen Platz finden. (Hinweis: `pack()` macht nur Sinn, wenn ein LayoutManager benutzt wird! Da ihr auch des öfteren ohne LayoutManager arbeiten werdet, wird es bei euch auch oft Beispiele geben, in denen `pack()` nicht benutzt wird.)

Zeile 34: Das Fenster wird in der Bildschirmmitte geöffnet.

Zeile 35: Es kann durch den Anwender nicht in seiner Größe verändert werden.

Zeile 36: Jetzt erst wird das fertige Fenster auf dem Desktop angezeigt.

Zeile 44-55: In der `main()`-Methode unseres Programms wird der Konstruktor unseres Anwendungsfensters innerhalb eines `try-catch`-Blocks in den sogenannten Event Dispatch Thread (EDT) eingestellt.

23.7 WindowListener verwenden

Statt es uns einfach zu machen und wie in Zeile 15 des Beispielprogramms auf die gegebene Funktionalität von Swing zurückzugreifen, können wir auch selbst dafür sorgen, dass das Schließen des Fensters durch den Benutzer auch tatsächlich zur Beendigung des Programms führt.

Dazu erzeugen wir zunächst eine neue Klasse `FensterSchliesser`, die wir von der Klasse `WindowAdapter` ableiten:

```
import java.awt.*;
import java.awt.event.*;

class FensterSchliesser extends WindowAdapter {
    public void windowClosing (WindowEvent event) {
        MyJFrame frame = (MyJFrame) event.getWindow();
        frame.setVisible(false);
        frame.dispose();
        System.exit(0);
    }
}
```

Und ersetzen anschließend Zeile 15 in unserem Beispiel durch:

```
setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
FensterSchliesser close = new FensterSchliesser();
addWindowListener(close);
```

23.8 Auf das Objekt des Anwendungsfensters zugreifen

Die Methode `getWindow()` gibt ein Objekt der Superklasse `Window` zurück:

```
Window w = event.getWindow();
```

Solange man mit dem Datentyp `Window` arbeitet, kann man jedoch nur auf Variablen und Methoden zugreifen, die in der Klasse `Window` definiert sind.

Wenn man auf Variablen und Methoden zugreifen möchte, die in abgeleiteten Klassen definiert sind (z.B. eine Variable, die man selbst in seinem eigenen Anwendungsfenster definiert hat), muss man den Datentyp umwandeln. Das geht so:

```
MeinFrame frame = (MeinFrame) w;
```

23.9 Programmierung von komplexen Dialogen

Ein allgemeiner Dialog wird ganz ähnlich programmiert wie ein Frame. Der Haupt-Unterschied ist, dass man seine eigene Klasse nicht von der Klasse `JFrame` sondern von der Klasse `JDialog` ableitet. Darüber hinaus sind folgende Details zu beachten:

- In einem Dialog gibt es natürlich keine `main()`-Methode, da der Dialog nicht wie das Anwendungsfenster vom System selbst gestartet wird. Der Code, der beim Frame zur Erzeugung des Objektes in der `main()`-Methode steht, wird in den Event-Handler gepackt, der beim Klick des entsprechenden Buttons im Hauptfenster aufgerufen wird (Näheres dazu im nächsten Kapitel).
- Im Konstruktor sollte mit der Anweisung `super()` der folgende Konstruktor der Superklasse `Dialog` aufgerufen werden:

```
public Dialog(Dialog owner, String title, boolean modal)
```

Für `modal` übergibt man den Wert `true` um einen sogenannten *modalen Dialog* zu erzeugen. Bei einem modalen Dialog kann der Benutzer das Hauptfenster nicht benutzen solange der Dialog geöffnet ist.

- Wie im Frame muss das Event `windowClosing` abgefangen werden, um das Fenster zu schließen. Während beim Frame jedoch die komplette Anwendung geschlossen wird (`EXIT_ON_CLOSE`) darf bei einem Unterdialog nur das aktuelle Fenster geschlossen werden:

```
dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
```

23.10 Fenster – Übungen

Aufgabe 1: `windowClosing()`

- Erzeuge in Eclipse über Rechtsclick auf das aktuelle Package im Package Explorer und *New → Class* eine Anwendung mit einem eigenen Programmfenster, in dem du dein Programm von der Klasse `JFrame` ableitest. Die Hilfsklasse `JJFrame` darf von heute an nicht mehr verwendet werden!!! Fange das Fenster-Ereignis `windowClosing` ab, damit die Anwendung vom Benutzer geschlossen werden kann (Damit Swing nicht mit seinem Standard-Verhalten dazwischen-funkt musst du im Konstruktor `setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE)` benutzen).
- Erweitere deinen Eventhandler für das Ereignis `windowClosing` so, dass der Benutzer zuerst mit einem `showConfirmDialog()` der Klasse `JOptionPane` (siehe unten) gefragt wird, ob er die Anwendung wirklich schließen möchte. Wenn er bejaht, wird die Anwendung geschlossen, andernfalls nicht.
- Fange die Ereignisse `windowActivated` und `windowDeactivated` ab. Wenn das Anwendungsfenster aktiviert ist (das erkennt man an der blau markierten Titelleiste), soll die Hintergrundfarbe des Fensters gelb sein. Wenn das Fenster deaktiviert ist (graue Titelleiste), ist die Hintergrundfarbe blau.
- Wenn das Fenster aktiviert ist, wird in der Titelzeile der Text „Schön, dass du da bist!“ ausgegeben. Wenn das Fenster deaktiviert ist wird der Text „Warum hast du mich verlassen?“ ausgegeben.

Bestätigungs-Dialoge

Die Klasse `JOptionPane` aus der Swing-Bibliothek besitzt die Methode `showConfirmDialog()` zur Abfrage einer Benutzerbestätigung:

```
public static int showConfirmDialog(Component parentComponent, Object message)
```

Als Parameter werden die Vater-Komponente (das ist das Anwendungsfenster) und ein Text angegeben.

Der Rückgabewert zeigt an, welchen Button der Benutzer gedrückt hat. Es sind folgende Rückgabewerte definiert:

| | |
|--|-----------------------------|
| <code>JOptionPane.YES_OPTION</code> | (Ja-Button gedrückt) |
| <code>JOptionPane.NO_OPTION</code> | (Nein-Button gedrückt) |
| <code>JOptionPane.CANCEL_OPTION</code> | (Abbrechen-Button gedrückt) |

Swing ist nicht Thread-Safe. Das bedeutet, dass der Programmierer selbst dafür sorge tragen muss, dass nicht mehrere Threads gleichzeitig versuchen auf ein Objekt zuzugreifen. Es ist im Zweifelsfall sinnvoll, Swing-Methoden nicht direkt auszuführen, sondern diese an den sogenannten *Event Dispatch Thread* (EDT) zu übergeben.

Im Anwendungsfenster könnte zum Beispiel folgender Code stehen:

```
EventQueue.invokeLater(new Runnable() {
    public void run() {
        int erg = JOptionPane.showConfirmDialog(null,
                "Anwendung wirklich beenden?");
        if (erg == JOptionPane.YES_OPTION) {
            // ...
        }
    }
});
```

Hinweis: Wenn du `showConfirmDialog()` in einer anonymen inneren Klasse (wie oben) benutzt, dann musst du als erstes Argument `null` anstelle von `this` benutzen!

Aufgabe 2: Beleidigtes Fenster

- a) Programmiere ein `JFrame` mit einem eigenen Eventhandler für Fenster-Ereignisse. Die Fenster-Ereignisse können entweder in der `JFrame`-Klasse abgefangen werden (indem das entsprechende Interface implementiert wird) oder in einer eigenen Klasse, die sich von der Klasse `WindowAdapter` ableitet.

Das Fenster soll zu Beginn eine Breite und eine Höhe von 500 Pixel haben. Es hat einen gelben Hintergrund und eine schwarze Schriftfarbe. Im Fenster wird mit Hilfe der `paintComponent()`-Methode der Text „Wehe du gehst weg.“ angezeigt.

Im ersten Schritt wird der Eventhandler so programmiert, dass das Fenster bei einem Klick auf den „Schließen“-Knopf geschlossen wird.

- b) Fang die Fenster-Ereignisse `windowActivated` und `windowDeactivated` ab.

Wenn das Fenster deaktiviert wird, nimmt es eine rote Hintergrundfarbe an (es wird sozusagen rot vor Ärger). Wenn es aktiviert wird, nimmt es wieder eine gelbe Hintergrundfarbe an.

- c) Wenn das Fenster deaktiviert ist, wird der Text im Fenster durch den neuen Text „Verräter, wieso gehst du einfach weg?“ ersetzt. Wenn das Fenster aktiviert ist wird wieder der alte Text „Wehe du gehst weg.“ angezeigt.

- d) Wenn das Fenster deaktiviert wird, schrumpft es vor Ärger zusammen. Reduziere die aktuelle Fensterbreite und Fensterhöhe bei jeder Deaktivierung jeweils um 100 Pixel. Wenn die Breite des Fensters dadurch kleiner als 100 Pixel wird, beendet sich die Anwendung selbstständig.

Hinweis: Es darf vereinfacht davon ausgegangen werden, dass der Benutzer die Fenstergröße nicht manuell verstellt.

Aufgabe 3: JFrame-Template von Eclipse benutzen

Eclipse erstellt das Grundgerüst für eine Anwendung, die von einem `JFrame` abgeleitet wird, wenn man sich statt `New → Class` für `New → Other ... → Window Builder → Swing Designer → JFrame` entscheidet.

Erzeuge auf diese Art und Weise eine neue Anwendung und versuche den Programmcode zu verstehen.

24 GUI-Komponenten

24.1 Struktur eines Swing UI

Hauptfenster

Ein Swing User Interface (UI) besteht immer aus (mindestens) einem Hauptfenster (Klassen `JFrame`, `JWindow`, `JDialog` und `JApplet`).

Container

Dann gibt es Container wie `JPanel`, die dazu dienen andere Komponenten aufzunehmen (auch wieder andere Container) und mit Hilfe eines Layout-Managers anzugeordnen. Methoden dazu: `setLayout()`, `setBorder()`.

Jedes Hauptfenster hat eine sogenannte `ContentPane` (Methoden dazu: `getContentPane()` und `setContentPane()`), dass ist der Teil des Fensters, der die weiteren Elemente des UI aufnimmt. Die `ContentPane` ist somit der erste Container eines Hauptfensters. Man kann keine UI-Elemente (Komponenten) außerhalb der `ContentPane` platzieren.

Komponenten

Jeder Container kann seinerseits weitere Container und eben auch Komponenten (z.B. `JLabel`, `JButton`, `JTextField`, `JCheckBox`, `JList` oder `JComboBox`) enthalten. Diese werden mit Hilfe der Methode `add()` einem bestimmten Container hinzugefügt. Wenn die Platzierung der Komponenten im Container über einen Layout-Manager kontrolliert wird, dann sollte man die gewünschte Größe der einzelnen Komponenten über die Methode `setPreferredSize()` angeben. Für den Fall, dass man ohne Layout-Manager (`setLayout(null)`, auch als *Absolute-Layout* bezeichnet) arbeitet, gibt man Größe und Position innerhalb des Containers mit der Methode `setBounds()` an.

Layout-Manager

Eine große Bedeutung kommt dem Layout-Manager zu. Es gibt mehrere zur Auswahl. Bereits genannt wurden `BorderLayout` (Elemente können oben, unten, links, mittig und rechts im Container platziert werden) und das `FlowLayout`, welches die Elemente einfach der Reihe nach – von links nach rechts und von oben nach unten – in den Container einfüllt. Oft benutzt sind außerdem auch `GridLayout`, welches eine frei-wählbare (aber starre) Matrix von einer festzulegenden Anzahl Spalten und Zeilen verwaltet, in die die Komponenten der Reihe nach platziert werden. Es gibt aber noch mehr. Der schnellste (wenn auch nicht schönste) Weg zum Ziel wird für euch oft über das Absolute-Layout gehen. Sprich: über den Verzicht auf einen Layout-Manager. Schön ist es nicht, weil ihr dann jede Komponente selber absolut platzieren müsst (es muss für jede Komponente die genauen x- und y-Position angegeben werden). Das ist wie gesagt nicht schön, aber es geht schnell auch ohne viel Übung. Und in Klausuren kommt es für euch in erster Linie darauf an mit dem GUI keine Zeit zu verlieren. Denn für ein schönes Layout bekommt ihr keine Punkte.

Arbeiten mit einem GUI-Designer

Für die Erstellung eines UIs wird üblicherweise ein sogenannter *GUI-Designer* – in Eclipse heißt er *Window Builder* – benutzt. In diesem kann man sich die grafische Benutzerschnittstelle „zusammen-klicken“. Ohne ein Verständnis über die Funktionen und Aufgaben der einzelnen Elemente eines GUIs wird man davon allerdings nicht viel haben!

Um in Eclipse eine Java-Datei im Window Builder zu öffnen wählt man im Kontext-Menü (Rechts-Klick) der Datei: *Open With → WindowBuilder Editor*.

24.2 Einfache Dialogelemente

Gemeinsamkeiten aller Komponenten (geerbt von JComponent)

Im Package `javax.swing` werden eine ganze Reihe fertiger Komponenten bereitgestellt, die einem die Programmierung erleichtern. Für jeden Komponenten-Typ gibt es eine eigene Klasse. Alle Komponenten haben die gemeinsame Superklasse `JComponent`, von der sie einige Grund-Eigenschaften erben.

Methoden

Alle Komponenten erben von der Superklasse `JComponent` folgende Methoden (Auswahl):

| Methode | Erläuterung |
|---|---|
| <code>public void setEnabled(boolean b)</code> | Aktiviert oder deaktiviert die Komponente. Wenn die Komponente deaktiviert ist, sind keine Eingaben möglich. |
| <code>public void setFont(Font f)</code> | Bestimmt die Schriftart für die Komponente. |
| <code>public void setBackground(Color c)</code> | Bestimmt die Hintergrundfarbe der Komponente. |
| <code>public void setForeground(Color c)</code> | Bestimmt die Vordergrundfarbe der Komponente (die Farbe, mit der Text und andere Sachen „gezeichnet“ werden). |
| <code>public void setPreferredSize(Dimension d)</code> | Legt die gewünschte Größe der Komponente fest für den Fall, dass der Container einen Layout-Manager benutzt (nicht Absolute- bzw. NULL-Layout). |
| <code>public void setBounds(int x, int y, int width, int height)</code> | Positioniert die Komponente an der Stelle (x,y) mit der angegebenen Breite und Höhe (für das Absolute- bzw. NULL-Layout). |

Ereignisbehandlung

Wenn in der Komponente eine Tastatureingabe gemacht wird oder auf die Komponente mit der Maus geklickt wird, so wird ein Ereignis ausgelöst. Dieses Ereignis kann man als Programmierer abfangen. Jede Komponente besitzt ein eigenes Interface, das zum Abfangen ihrer Ereignisse implementiert werden muss. In dem Interface sind eine Reihe von Methoden beschrieben, die das System bei Eintreffen eines Ereignisses aufruft. Ein Objekt der Klasse, die das Interface implementiert, muss bei der jeweiligen Komponente mit einer speziellen Methode registriert werden. Ein „Ereignisüberwachungs“-Objekt kann mehrere Komponenten-Objekte gleichzeitig überwachen (zum Beispiel drei verschiedene Buttons).

Anordnung von Komponenten

Bei den meisten Programmiersprachen wird die Anordnung der Komponenten durch Angabe absoluter Koordinaten pixelgenau festgelegt. Java-Programme sollen jedoch auf vielen unterschiedlichen Betriebssystemen laufen, auf denen es zum Beispiel verschiedene Komponentengrößen gibt. Deshalb gibt es in Java sogenannte Layout-Manager, die die Anordnung der Komponenten automatisch übernehmen. Der Programmierer kann zwischen verschiedenen Layouts wählen, indem er dem `JFrame` mit der Methode `setLayout()` den gewünschten Layout-Manager zuweist.

Beispiele:

a) `setLayout(new FlowLayout());`

Die Komponenten werden zeilenweise nebeneinander angeordnet.

b) `setLayout(new GridLayout(4, 2));`

Ordnet die Komponenten in einem rechteckigen Gitter mit 4 Zeilen und 2 Spalten an. Im Beispiel werden also $4 * 2 = 8$ Komponenten angeordnet.

```
c) setLayout(null);
```

Ein Null-Layout (in Eclipse auch Absolute-Layout genannt) wird erzeugt, indem die Methode `setLayout()` mit dem Argument `null` aufgerufen wird. In diesem Fall verwendet das Fenster keinen Layout-Manager, sondern überlässt die Positionierung der Komponenten der Anwendung. Dazu muss der Programmierer für jede Komponente die Methode `setBounds()` aufrufen.

In Swing werden Komponenten nie direkt in einem Hauptfenster platziert sondern in der sogenannten *ContentPane* (oder einem darin liegenden Container).

Nach der Auswahl des Layout-Managers, übergibt man dem `JFrame` bzw. dem enthaltenden Container mit der Methode `add()` die Objekte der gewünschten Komponenten zur Anordnung. Beispiel:

```
 JButton btnOK = new JButton("OK");
btnOK.setBounds(48, 88, 75, 25);      // nur beim NULL-Layout
panel.add(btnOK);
```

(`panel` ist hier ein willkürlich gewählter Name für den Container, in dem der Button platziert werden soll)

Ein Button ist eine beschriftete Schaltfläche, die dazu verwendet wird, auf Knopfdruck des Anwenders Aktionen auszulösen.

Einen Button erzeugen

Beim Erzeugen eines Objekts der Klasse `JButton`, gibt man als Parameter die Beschriftung der Schaltfläche an:

```
JButton btnNeu;
btnNeu = new JButton("Neues Spiel");
```

Damit der Button im `JFrame` bzw. im enthaltenden Container sichtbar wird, muss man das angelegte Objekt über die Methode `add()` hinzufügen. Im Konstruktor des `JFrame` steht also:

```
panel.add(btnNeu);
```

Auf Knopfdruck reagieren

Um von dem Ereignis „Knopfdruck“ des Buttons benachrichtigt zu werden, muss man das Interface `ActionListener` implementieren. Die Klasse, die das Interface implementiert, wird beim Button mit folgender Methode registriert:

```
public void addActionListener(ActionListener listener)
```

Beispiel:

```
btnNeu.addActionListener(this);
```

Das Interface ActionListener

Das Interface `ActionListener` hat nur eine Methode, die immer bei Knopfdruck aufgerufen wird:

```
public void actionPerformed(ActionEvent e)
```

Das übergebene Objekt der Klasse `ActionEvent` enthält Informationen über die Art des Knopfdrucks. Wenn man nur einen Button hat, muss das Objekt nicht ausgewertet werden. Wenn man jedoch mehrere Buttons in derselben Klasse verwalten will, kann man durch das Objekt herausfinden, welcher Button gedrückt wurde. Dazu gibt es unter anderem folgende Methoden in der Klasse `ActionEvent`:

| Methode | Erläuterung |
|---|--|
| <code>public Object getSource()</code> | Gibt das Objekt (Klasse <code>Object</code>) zurück, auf das geklickt wurde. Das Objekt muss zur Auswertung in ein <code>JButton</code> -Objekt konvertiert werden. |
| <code>public String getActionCommand()</code> | Gibt den Titel des Knopfes zurück, z.B. "Neues Spiel". |

JLabel Hier steht mein Text

Ein Label ist eine Komponente, die nur zur Ausgabe von Text dient. Man kann sie zum Beispiel zur Beschriftung von Dialogboxen verwenden. Wie beim Button gibt man im Konstruktor den Text an, der in der Komponente angezeigt werden soll. Im Konstruktor des Frames könnte z.B. stehen:

```
JLabel lblText;
lblText = new JLabel("Hier steht mein Text");
panel.add(lblText);
```

Da ein Benutzer den Inhalt eines Labels nicht verändern kann, ist eine Ereignisbehandlung für ein Label-Objekt nicht nötig. Das Programm kann den Text eines Labels nachträglich mit folgender Methode verändern:

```
public void setText(String text)
```

JCheckBox Ich stimme dem Vertrag zu

Eine Checkbox ist ein Eingabeelement, dass zwischen den Werten `true` und `false` umgeschaltet werden kann. Der Ja-Fall wird durch ein Häkchen angezeigt.

Im Konstruktor der Checkbox können der Titel der Checkbox und der Anfangszustand (Häkchen: ja oder nein) angegeben werden:

```
JCheckBox cbZustimmung;
cbZustimmung = new JCheckBox("Ich stimme dem Vertrag zu", true);
panel.add(cbZustimmung);
```

Im laufenden Programm kann der aktuelle Zustand der Checkbox mit folgender Methode abgefragt werden:

```
public boolean isSelected()
```

Ereignisbehandlung

Wenn im Programm auf das Setzen oder Löschen des Häkchens reagiert werden soll, muss das Interface `ItemListener` implementiert werden. Ein Objekt, das dieses Interface implementiert, wird durch Aufruf folgender Methode bei der Checkbox registriert:

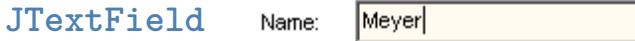
```
public void addItemListener(ItemListener l)
```

Das Interface `ItemListener` besitzt nur eine Methode, die bei Änderung des Zustands aufgerufen wird:

```
public void itemStateChanged(ItemEvent e)
```

Beispiel:

```
JCheckBox cbAntwort = new JCheckBox("Alle Vögel fliegen hoch", false);
cbAntwort.addItemListener(new ItemListener() {
    @Override
    public void itemStateChanged(ItemEvent e) {
        if (cbAntwort.isSelected()) {
            System.out.println("Alle Vögel fliegen hoch!");
        }
    }
});
```



Ein **JTextField** dient zur Darstellung und zur Eingabe von Text. Sowohl der Anwender als auch das Programm können den dargestellten Text auslesen und verändern.

Die Beschriftung vor einem **JTextField** gehört nicht zum Textfeld dazu, sondern muss mit einem **JLabel** erzeugt werden:

```
JLabel lblName = new JLabel("Name:");
panel.add(lblName);
```

Bei der Erzeugung des Textfeldes gibt man als Parameter die Anzahl der Zeichen an, die im **JTextField** dargestellt werden sollen. Dadurch wird die Länge der Komponente beeinflusst. Wenn der Benutzer mehr als die angegebenen Zeichen eingibt, ist ein Teil des Textes nicht mehr sichtbar.

```
JTextField tfName;
tfName = new JTextField(20);
panel.add(tfName);
```

Mit folgenden Methoden kann der Text der Komponente gelesen und verändert werden:

```
public void setText(String t)
public String getText()
```

Mit

```
public void setEditable(boolean b)
```

kann festgelegt werden, ob der Inhalt des Textfeldes durch den Anwender editiert werden kann oder nicht.

Ereignisbehandlung

Es gibt zwei verschiedene Möglichkeiten, Ereignisse des Textfeldes zu behandeln:

- Wenn man sich nur dann benachrichtigen lassen möchte, wenn der Anwender innerhalb des Textfeldes die ENTER-Taste drückt, implementiert man das Interface **ActionListener**, das auch in Buttons verwendet wird. Mit folgender Methode wird ein Objekt, das dieses Interface implementiert, bei einem **JTextField** registriert:

```
public void addActionListener(ActionListener l)
```

Die einzige Methode des Interfaces **ActionListener** sieht folgendermaßen aus:

```
public void actionPerformed(ActionEvent e)
```

- Möchte man hingegen von jeder Änderung im Textfeld unterrichtet werden, muss man einen **DocumentListener** auf das Modell des Textfeldes registrieren:

```
JTextField tfEingabe = new JTextField(20);
tfEingabe.getDocument().addDocumentListener(new DocumentListener() {
    /**
     * insertUpdate() meldet, dass es eine Änderung gab
     */
    public void insertUpdate(DocumentEvent e) {}
    /**
     * removeUpdate() meldet, dass ein Teil des Textes gelöscht wurde
     */
    public void removeUpdate(DocumentEvent e) {}
    /**
     * changedUpdate() meldet, dass Attribute geändert wurde
     * --> NICHT relevant für JTextField
     */
    public void changedUpdate(DocumentEvent e) {}
});
```

JTextArea

Analog zu `JTextField` dient `JTextArea` zur Darstellung und zur Eingabe von Text. Allerdings mehrzeilig. Es gibt mehrere Konstruktoren. Die drei folgenden sind für uns besonders nützlich:

```
public JTextArea()  
public JTextArea(int rows, int columns)  
public JTextArea(String t, int rows, int columns)
```

Man gibt im Konstruktor also die Größe (als Anzahl Zeilen und Anzahl Spalten), sowie optional auch bereits den darzustellenden Text an. Beispiel:

```
JTextArea taBrief;  
taBrief = new JTextArea(20, 80);  
panel.add(taBrief);
```

Der Konstruktor ohne Parameter erzeugt eine Komponente, die sich in den aufnehmenden Container einpasst. Die wichtigsten Methoden sind wie schon bei `JTextField`

```
public void setText(String t)  
public String getText()
```

Außerdem gibt es noch eine Methode, um Text an den bereits vorhandenen Text anzuhängen:

```
public void append(String t)
```

`JTextArea` kümmert sich nicht darum einen zu breiten oder zu langen Text über entsprechende Scrollbars erreichbar zu machen. Wer diese Funktionalität braucht, muss die `JTextArea` in eine `JScrollPane`-Komponente einbetten. Wie dies geht ist im Abschnitt zu den komplexeren Komponenten im Zusammenhang mit `JList` erklärt.

24.3 Einfache Dialogelemente – Übungen

Aufgabe 1: Layout-Manager austesten

Teste die beiden wichtigsten Layout-Manager und das NULL-Layout aus, in dem du jeweils eine Anwendung erzeugst, die fünf verschiedene Buttons (mit den Aufschriften A, B, C, D und E) mit dem jeweiligen Layout-Manager anordnet. Beim Klick auf die Buttons soll in dieser einfachen Vorübung noch nichts geschehen.

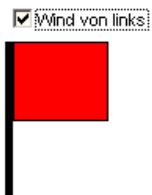
- a) `FlowLayout`
- b) `GridLayout` mit zwei Zeilen und drei Spalten
- c) `NULL-Layout`

Aufgabe 2: Knopfdrücke zählen

Programmiere ein `JFrame` mit einem `JButton` und einem `JLabel`. Gib im Label einen Text aus, der angibt, wie oft der Button gedrückt wurde. Bei jedem Drücken des Buttons wird der Text angepasst. Verwende dazu einen Layout-Manager. Ich empfehle das `FlowLayout`. Du darfst dich aber auch für einen anderen Layout-Manager entscheiden (nicht `NULL-Layout`).

Aufgabe 3: Fahne im Wind

Zeichne eine Fahne, die je nach Windrichtung nach rechts oder nach links ausgerichtet ist. Die Windrichtung wird mit einer `JCheckBox` ausgewählt. Wenn der Benutzer die Checkbox anklickt, dreht sich die Fahne sofort in die neue Richtung. Damit die Fahne neu gezeichnet wird, muss im Frame die Methode `repaint()` aufgerufen werden.



Aufgabe 4: Anrede

Programmiere ein `JFrame`, das folgendes Aussehen hat:



Der Benutzer kann im Textfeld einen Namen eingeben. Wenn man auf den Button drückt, wird der Benutzer höflich begrüßt. Dabei wird die Checkbox ausgewertet, um zu erkunden, ob der Benutzer mit „Frau“ oder mit „Herr“ angeredet werden soll. Wenn der Benutzer zum Beispiel den Namen „Meyer“ eingegeben hat und die Checkbox nicht selektiert ist, wird ausgegeben: „Guten Tag Herr Meyer!“.

Aufgabe 5: RGB-Farben mischen

Programmiere ein `JFrame`, in dem man die RGB-Werte als Zahlen eingeben kann. Wenn man auf einen Button drückt wird mit den eingegebenen RGB-Werten ein Objekt der Klasse `Color` erzeugt. Im Konstruktor der Klasse `Color` werden als Parameter die drei Farbwerte übergeben. Anschließend wird die Hintergrundfarbe des `JFrame`-Fensters und der `JLabel`-Komponenten auf den gemischten Farbwert gesetzt.

**Konvertierung von String nach int**

Beispiel:

```
String text = "77";
int zahl = Integer.parseInt(text);
```

24.4 Allgemeines zur Implementierung von Event-Listenern (Ereignisbehandlung)

In den Musterlösungen findet ihr zwei verschiedene Ansätze um Ereignisse zu behandeln:

1. (Beispiele: Fahne.java und Anrede.java) Implementierung eines Interfaces

Dies entspricht dem Vorgehen, welches auch auf im Abschnitt „Einfache Dialogelemente“ beschrieben ist: Ein zur jeweiligen Komponente passender EventListener wird implementiert. Dass sieht man dann schon im Kopf der Klasse

```
public class ... implements ...
```

Wie immer bei der Verwendung eines Interfaces müssen dann alle Methoden des Interfaces implementiert werden (in den Beispielen hier ist dies praktischer Weise jeweils nur eine einzige Methode).

Damit die Komponente dann etwas von diesem EventListener hat, muss er auf die jeweilige Komponente noch „angesetzt“ werden:

Beispiel aus Anrede.java:

```
btnAuswerten.addActionListener(this);
```

Beispiel aus Fahne.java:

```
cbWind Von Links.addItemListener(this);
```

2. (Beispiel: FarbenMischen.java): Ableiten von einer passenden Wrapper-Klasse bzw. einem Interface in einer anonymen Klasse:

```
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent evt) {
        // Was auch immer getan werden soll ...
    }
});
```

Auf den ersten Blick etwas verwirrend, aber sehr praktisch:

In der unter 1. beschriebenen Methode wurde der EventListener auf die Klasse der Anwendung (`this`) registriert. Wenn die Anwendungsklasse aber das entsprechende Interface nicht implementiert, kann der EventListener auch nicht auf diese Klasse registriert werden. Vielmehr muss er auf eine andere Klasse registriert werden, die solch einen EventListener zur Verfügung stellt. Da es diese Klasse noch nicht gibt, muss sie noch erschaffen werden. Die sehr kompakte Schreibweise mit der anonymen Klasse oben ist eine Abkürzung für

```
button.addActionListener(MyActionListener());
...
class MyActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent evt) {
        // Was auch immer getan werden soll ...
    }
});
```

In der kompakten Schreibweise mit der anonymen Klasse spart man es also überhaupt eine neue Klasse zu benennen (im gegebenen Beispiel: `MyActionListener`). Wenn diese Klasse nur ein einziges Mal benutzt werden soll (also nicht mehrere Objekte dieser Klasse erzeugt werden müssen), dann bietet die Verwendung von anonymen Klassen die kompakteste Schreibweise.

Diese Art der Implementierung eines EventListeners für eine Komponente erhält man auch, wenn den WindowBuilder in Eclipse benutzt und dort per Rechtsklick auf eine Komponente → *Add event handler* → *action* → *actionPerformed* wählt.

24.5 Typkonvertierung

In Java besitzen alle Variablen einen Datentyp. Neben den primitiven (im Sprachkern fest implementierten) Datentypen `int`, `short`, `long`, `float`, `double`, `boolean`, `char` und `byte` gibt es weitere Datentypen, die in Klassen definiert sind bzw. werden.

Java erlaubt dabei die Umwandlung (*type casting*) von einem Datentyp in einen anderen. Allerdings nicht beliebig. Dabei wird unterschieden zwischen impliziten und expliziten Typumwandlungen.

Beispiel

In einer Ereignis-Behandlungsroutine wird immer ein `Event`-Objekt übergeben, das Informationen über die Art des aufgetretenen Ereignisses besitzt. Mit Hilfe der Methode

```
public Object getSource()
```

kann man erfragen, in welcher Komponente das Ereignis aufgetreten ist. Als Rückgabewert erhält man ein Objekt, das keinen speziellen Typen besitzt. Wenn man weiß, dass das Ereignis zum Beispiel in einem Button aufgetreten sein muss, kann man das System zwingen, das allgemeine Objekt vom Typ `Object` in ein `JButton`-Objekt umzuwandeln. Die Umwandlung geschieht jedoch „auf eigene Gefahr“. Falls das Ereignis doch in einer anderen Komponente ausgelöst wurde, z.B. in einem `JTextField`, gibt es einen Programmabsturz.

Der folgende Code demonstriert, wie man in einem `JFrame` mit drei Buttons herausfinden kann, welcher der Buttons gedrückt wurde:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.EmptyBorder;

public class WelcherButton extends JFrame implements ActionListener {
    private static final int WIDTH = 300;
    private static final int HEIGHT = 120;
    private JButton eins, zwei, drei;
    private JLabel text, text2;

    public WelcherButton(final String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel contentPane = new JPanel();
        contentPane.setLayout(new FlowLayout());
        setContentPane(contentPane);
        setSize(WIDTH, HEIGHT);
        eins = new JButton("1");
        contentPane.add(eins);
        eins.addActionListener(this);
        zwei = new JButton("2");
        contentPane.add(zwei);
        zwei.addActionListener(this);
        drei = new JButton("3");
        contentPane.add(drei);
        drei.addActionListener(this);
        text = new JLabel("Es wurde Button _ gedrückt.");
        contentPane.add(text);
        text2 = new JLabel("Es wurde Button _ gedrückt.");
        contentPane.add(text2);
        setVisible(true);
    }
}
```

```

public void actionPerformed(ActionEvent e) {
    JButton button = (JButton) e.getSource();

    // 1. Alternative: lange Variante
    if (button == eins) {
        text.setText("Es wurde Button 1 gedrückt.");
    }
    if (button == zwei) {
        text.setText("Es wurde Button 2 gedrückt.");
    }
    if (button == drei) {
        text.setText("Es wurde Button 3 gedrückt.");
    }

    // 2. Alternative: kurze Variante
    text2.setText("Es wurde Button " + button.getText() + " gedrückt.");
}

public static void main(final String[] args) {
    new WelcherButton("WelcherButton");
}
}
}

```

Regeln für die Typkonvertierungen

Schema

Eine Variable `var1` vom Typ `Typ1` wird mit folgendem Befehl explizit in eine Variable `var2` vom Typ `Typ2` umgewandelt (explizite Typumwandlung bzw. *explicit type cast*):

```
Typ2 var2 = (Typ2) var1;
```

In einigen wenigen Fällen erledigt Java diese Typumwandlung für uns sogar automatisch (implizite Typumwandlung bzw. *implicit type cast*). Dann kann auf den *type cast operator* (im Beispiel oben (`Typ2`)) verzichtet werden.

In Berechnungen, in denen Ganzzahl- und Fließkommazahl-Variablen gemischt verwendet werden geschieht dies häufig (und oft mit „überraschenden“ Ergebnissen).

Beispiel für die implizite Typumwandlung

Schon bei der Umwandlung von „ähnlichen“ Datentypen (etwa `int` ↔ `long`, oder auch `float` ↔ `double`) kann es Probleme geben: Die Umwandlung des „kleineren“ in den „größeren“ Datentyp ist problemlos und kann deshalb implizit erfolgen:

```
int i = 12345;
long l = i;
```

Für den umgekehrten Weg ist aber ein expliziter Type Cast notwendig, da hier ein Informationsverlust stattfinden kann (der Wertebereich des „größeren“ Datentyps lässt sich nicht komplett auf den „kleineren“ Datentyp abbilden). Das kann nicht automatisch passieren, sondern muss durch den Programmierer explizit angewiesen (und damit verantwortet) werden:

```
long l = 12345678901L;
int i = (int) l;           // ergibt -539222987, weil die oberen Bytes abgeschnitten werden
```

Aber auch die Umwandlung von nicht-ähnlichen Datentypen geschieht teils implizit:

```
int i = 5;
int j = 4;
float x = 0.2f;
System.out.println("" + x * i);           // ergibt 1.0
System.out.println("" + i / j);           // ergibt 1
```

Das Ergebnis der ersten Berechnung überrascht nicht. Und trotzdem war hierfür eine implizite Typumwandlung nötig, um die Datentypen der beiden Operanden aneinander anzupassen. Das Ganzzahl-Objekt wurde dabei in ein Fließkomma-Objekt umgewandelt.

Für die zweite Berechnung hätten wir intuitiv eine Typumwandlung erwartet. Aber sie findet nicht statt. Da beide Operanden Ganzzahl-Objekte sind, sieht Java keine Notwendigkeit für eine Umwandlung. Und das Ergebnis einer Ganzzahl-Division ist dann eben wieder eine Ganzzahl und nicht etwa eine Fließkommazahl (der Nachkommateil des von uns erwarteten Ergebnisses wird dabei einfach ignoriert).

Um bei der zweiten Berechnung das von uns erwartete Ergebnis zu bekommen, müssen wir per explizitem Type Cast zumindest einen der beiden Operanden in eine Fließkommazahl konvertieren:

```
System.out.println("" + ((float) i) / j);    // ergibt 1.25
```

Regeln

- Konvertierungen zwischen `int` und `boolean` sind nicht möglich. Weder implizit noch explizit.
- Konvertierung zwischen `int` und `char` sind möglich:

```
c = (char) i;      // explicit type cast ist nötig
i = c;            // explicit type cast ist nicht nötig
```

Dies folgt der zuvor beschriebenen Logik, dass eine Umwandlung vom „größeren“ zum „kleineren“ Datentyp einen expliziten Type Cast benötigt (möglicher Informationsverlust!), der umgekehrte Weg aber implizit möglich ist (keine Gefahr von Informationsverlust).

- Konvertierung zwischen `int` und `String` sind mit Type Cast nicht möglich. Weder implizit noch explizit.
- Konvertierung zwischen `char` und `String` sind mit Type Cast nicht möglich. Weder implizit noch explizit.
- Konvertierung zwischen `int` und einer Klasse (zum Beispiel `JButton`) sind mit Type Cast nicht möglich. Weder implizit noch explizit.
- Eine Konvertierung zwischen zwei verschiedenen Klassen (zum Beispiel `Component` und `JTextField`) ist erlaubt, wenn die eine Klasse eine Oberklasse der anderen Klasse ist. Falls die Objekte nicht zueinander passen, gibt es während des Programmablaufs einen Absturz.

Sonderrolle von String

Die Klasse `String` nimmt in Java eine Sonderrolle ein: Einerseits gehört sie nicht zu den primitiven Datentypen sondern ist als Klasse implementiert. Andererseits sind ihre Eigenschaften im Java-Kern bekannt und erlauben so einige Sachen, die mit anderen Objekten „normaler“ Klassen nicht möglich wäre. So ist es indirekt doch möglich, Zahlenwerte in Strings umzuwandeln:

```
int i = 10;
String text = "";
text = (String) i;                  // Direkte Typumwandlung nicht(!) möglich
text = "Du hast " + i + " Finger."; // ergibt: "Du hast 10 Finger."
text = "";
text += i;                         // ergibt: den String "10"
text = "Der Schreiner hat nur noch " + (i - 1) + " Finger.;"
```

24.6 Typkonvertierungen – Übungen

Aufgabe 1: Typkonvertierungen

Kopiere in den nachfolgenden Teilaufgaben jeweils die obere Variable in die untere Variable, in dem du den Typ der Variablen geeignet konvertierst. Welcher Wert wird nach der Konvertierung in der zweiten Variablen stehen? Welche Konvertierungen sind erlaubt, welche führen zu einem Systemfehler? Teste deine Lösung in Eclipse aus.

a) `char c = 'A';
int zahl =`

b) `int zahl = 65;
char c =`

c) `boolean b = true;
int zahl =`

d) `String text = "Hallo";
int zahl =`

e) `int zahl = 65;
String text =`

f) `char c = 'A';
String text =`

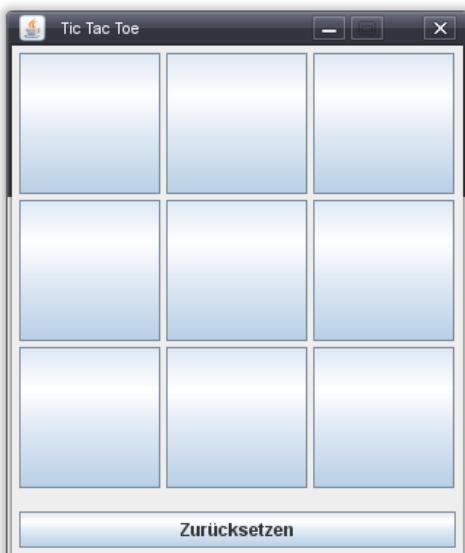
g) `JButton btn = new JButton("Drück mich");
JLabel lbl =`

h) `JButton btn = new JButton("Drück mich");
Object obj = (Object) b;
 JTextField tf =`

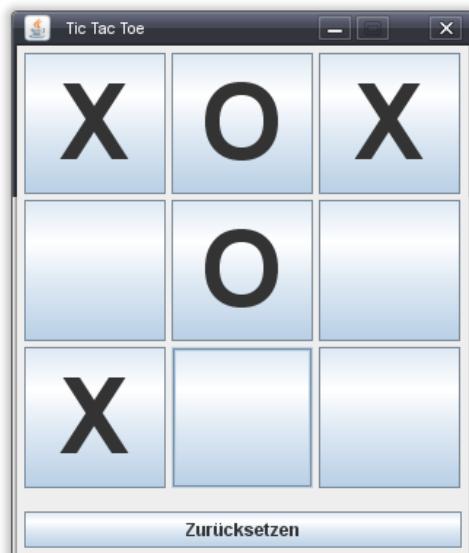
Aufgabe 2: Tic Tac Toe

Programmiere eine einfache Version des Spiels Tic Tac Toe.

Oberfläche zu Beginn des Spiels:



Oberfläche während des Spiels:



Das Spielfeld wird als ein Array von neun Buttons dargestellt, die zu Beginn keinen Text enthalten. Wenn man auf einen Button drückt, erhalten die Buttons immer abwechselnd den String „X“ oder „O“ als Aufschrift.

Wenn der Zurücksetzen-Button gedrückt wird, erhalten alle Buttons einen leeren String als Aufschrift, damit das Spiel von vorne beginnen kann. Mehr braucht die einfache Spiel-Version nicht zu können.

Zusatzaufgabe:

- Überprüfe zusätzlich die Korrektheit der Eingabe und gib eine Fehlermeldung aus, wenn ein Button gedrückt wird, der schon einen Text enthält.
- Überprüfe, ob einer der Spieler gewonnen hat und gib gegebenenfalls eine entsprechende Meldung aus.

Aufgabe 3: Taschenrechner

Programmiere einen sehr vereinfachten Taschenrechner.



- Im ersten Schritt sollen nur Zahlen zwischen 1 und 9 addiert werden können. Der Taschenrechner besteht aus folgenden Komponenten:

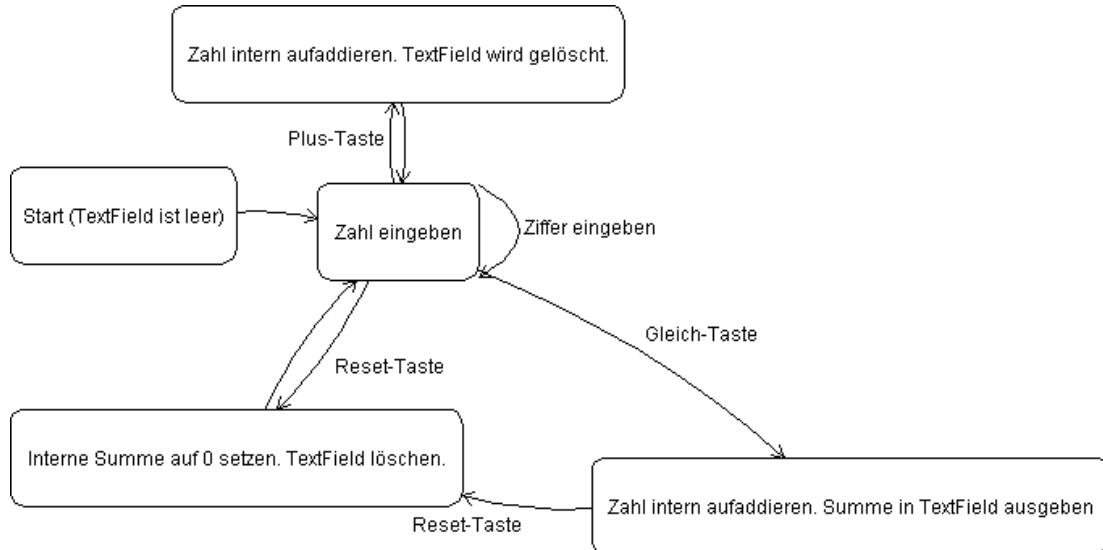
- Ein JTextField zur Ausgabe der aktuellen Summe. Es soll dem Benutzer unmöglich sein, in dem Textfeld Eingaben vorzunehmen.
- Ein Array von zehn Buttons zur Eingabe der Zahlen von 0 bis 9 (Es ist einfacher die Null mit einzubeziehen, auch wenn sie in Teilaufgabe a) noch nicht benötigt wird).
- Einen Reset-Button, mit dem die Summe auf 0 zurück gesetzt werden kann.

Wenn der Benutzer einen der Zahlen-Button drückt, wird die Summe im Textfeld um den Wert der entsprechenden Zahl erhöht.

- Erweitere den Taschenrechner so, dass auch zusammengesetzte Zahlen addiert werden können (z.B. 46 + 333). Füge Buttons für eine Plus-Taste (+) und eine Gleich-Taste (=) hinzu.

Wenn Zahlen-Tasten gedrückt werden, sollen die Ziffern (wie bei einem normalen Taschenrechner) an die aktuell angezeigte Zahl angehängt werden. Beim Drücken der Plus-Taste wird die angezeigte Zahl zu der intern abgespeicherten Summe dazu addiert und das Textfeld wird für die nächste Zahl gelöscht. Wenn der Benutzer die Gleich-Taste drückt, wird die aktuelle Zahl zu der internen Summe dazu addiert und die Summe ausgegeben.

Das nachfolgende Zustandsdiagramm veranschaulicht die Funktionsweise des Taschenrechners:



- c) Erweitere den Taschenrechner um eine Minus-Taste (-), eine Multiplikations-Taste (x) und eine Divisions-Taste (\div). Die drei Tasten arbeiten analog zu der Additions-Taste.

24.7 Arbeiten mit Zeichenketten (Strings)

Im Package `java.lang` wird die Klasse `String` zur Arbeit mit Zeichenketten definiert. Das Package `java.lang` wird immer automatisch eingebunden und braucht deshalb nicht explizit importiert werden.

Mit Objekten der Klasse `String` haben wir schon oft gearbeitet. Jede in doppelte Anführungszeichen gesetzte Zeichenkette (zum Beispiel "Hallo") ist ein Objekt der Klasse `String`. Im Gegensatz zu anderen Klassen brauchen Objekte der Klasse `String` nicht mit `new` erzeugt werden. Wie bei `int` und `boolean` Variablen kann man einer `String` Variablen einfach einen Wert durch ein Gleichheitszeichen zuweisen:

```
String s1 = "Hallo";
String s2 = "Guten Tag";
```

Mit dem Plus-Zeichen können zwei Strings aneinander gehängt werden:

```
s1 = s1 + s2;           // in s1 steht jetzt: "HalloGuten Tag"; s2 bleibt unverändert
s1 = s2 + " Otto";      // in s1 steht jetzt: "Guten Tag Otto"
s1 += "!";              // in s1 steht jetzt: "Guten Tag Otto!"
```

Der Nachteil bei der Arbeit mit Strings ist, dass bei jeder Veränderung der Zeichenkette ein neues Objekt im Speicher erzeugt wird. Wenn beispielsweise `s1` einen neuen Inhalt bekommt, bleibt der alte Inhalt im Speicher „als Leiche“ liegen bis irgendwann der sogenannte *Garbage Collector* („Müll-Aufsammler“) der Java Virtual Machine Zeit hat, diesen Speicher wieder frei zu geben. Wenn man eine effizientere Speichernutzung haben will, muss man mit der Klasse `StringBuilder` arbeiten. Das lohnt sich aber nur, wenn man ein Programm schreibt, in dem große Mengen von Zeichenketten verwaltet werden müssen.

Methoden der Klasse String (Auswahl)

| | |
|--|--|
| <code>int length()</code> | Länge des Strings |
| <code>String toLowerCase()</code> <code>String toUpperCase()</code> | Umwandlung in Klein- bzw. Großschreibung |
| <code>char charAt(int index)</code> | liefert das Zeichen an Position <code>index</code> |
| <code>String replace(char oldChar, char newChar)</code> | ersetzt überall im String Zeichen <code>oldChar</code> durch <code>newChar</code> |
| <code>String trim()</code> | entfernt alle Leerzeichen am Anfang und Ende |
| <code>int indexOf(int ch)</code> <code>int indexOf(String str)</code> | Anfangsposition des Zeichens oder Teil-Strings. Falls das Zeichen nicht im String vorkommt, wird -1 zurückgegeben. |
| <code>String substring(int beginIndex)</code> <code>String substring(int beginIndex, int endIndex)</code> | liefert den Teil-String von Position <code>beginIndex</code> bis eine Position vor <code>endIndex</code> (oder bis zum Ende) |
| <code>boolean contains(String str)</code> | <code>true</code> falls <code>str</code> im gegebenen String-Objekt enthalten ist. Sonst <code>false</code> |

Vergleich von Strings

| | |
|---|--|
| <code>==</code> | vergleicht die Speicheradressen der Objekte |
| <code>boolean equals(Object anObject)</code> | vergleicht Zeichenketten (überschriebene Methode der Superklasse <code>Object</code>) |
| <code>boolean equalsIgnoreCase(String anotherString)</code> | vergleicht Zeichenketten ohne Beachtung von Groß- und Kleinschreibung |
| <code>int compareTo(String anotherString)</code> | Lexikografischer Vergleich: <code>this < anotherString</code> : negativer Wert <code>this = anotherString</code> : 0 <code>this > anotherString</code> : positiver Wert |

24.8 Methoden der Klasse Character

Die Klasse **Character** besitzt folgende Funktionen, die für dich nützlich sind:

| | |
|---|-------------------------|
| <code>public static boolean isDigit(char ch)</code> | Ziffer von '0' bis '9'? |
| <code>public static boolean isLetter(char ch)</code> | Buchstabe? |
| <code>public static boolean isUpperCase(char ch)</code> | Großbuchstabe? |
| <code>public static boolean isLowerCase(char ch)</code> | Kleinbuchstabe? |

24.9 Strings – Übungen

Aufgabe 1: Vergleich von Strings

a) Was wird in dem folgenden Code ausgegeben?

```
public void paintComponent(Graphics g) {
    String s1 = "Hallo";
    String s2 = "Hallo";
    String s3 = "Hal";
    s3 += "lo";
    if (s1 == s2) {
        g.drawString("s1 und s2 sind gleich", 20,100);
    } else {
        g.drawString("s1 und s2 sind nicht gleich", 20,100);
    }
    if (s1 == s3) {
        g.drawString ("s1 und s3 sind gleich", 20,150);
    } else {
        g.drawString ("s1 und s3 sind nicht gleich", 20,150);
    }
}
```

b) Wie muss der Code abgeändert werden, damit ein echter Vergleich der Zeichenketten durchgeführt wird?

Aufgabe 2: String-Operationen

Wie ändert sich die Zeichenkette `result` in dem folgenden Code? Welcher Text wird am Ende ausgegeben?

```
public void paintComponent (Graphics g) {
    String original = "software";
    String result;

    result = " hallo ";                                // Schritt 1
    result = result.replace('l',original.charAt(6));   // Schritt 2
    result = result.trim();                            // Schritt 3
    int index = original.indexOf('t');
    result = result.substring(0,index);                // Schritt 4
    result += "d";                                    // Schritt 5
    result = result.toUpperCase();                     // Schritt 6
    result += original.substring(4);                  // Schritt 7

    g.drawString(result, 20, 30);
}
```

Aufgabe 3: Noch mehr String-Operationen

Füge in die nachfolgenden Code-Auszüge jeweils geeignete Aufrufe von `String`-Methoden ein.

a)

```
String s1 = "hallo";
String s2 = " HALLO ";

/* Abschneiden der Leerzeichen von String s2 */
```

```
/* Vergleich der Strings ohne Berücksichtigung von Groß- und
 Kleinschreibung */

if (           ) {
    System.out.println("Die Strings sind gleich");
}
```

- b) Wandle die beiden Zahlen in Strings um und hänge beide Strings zu einem Text aneinander (Ergebnis soll ein String mit dem Inhalt "1122" sein).

```
int zahl1 = 11;
int zahl2 = 22;
```

Aufgabe 4: Initialen

Schreibe ein Programm, das die Initialen eines Namens ermittelt.

Der **JFrame** enthält ein Textfeld zur Eingabe des Namens, einen Button, mit dem die Berechnung gestartet wird, und ein nicht-editierbares Textfeld zur Ausgabe der Initialen.

Der Benutzer gibt in dem ersten Textfeld einen Namen ein. Zum Beispiel:

Emil Anton Müller-Bruckner

Wenn er auf den Button drückt, schreibt das Programm in das zweite Textfeld die Initialen des Namens. In diesem Beispiel:

EAMB

Aufgabe 5: Rechnen mit versteckten Zahlen

Im **JFrame** gibt es zwei Textfelder, in die der Benutzer Zahlen eingeben soll.

Wenn der Benutzer auf einen Button drückt, werden die Zahlen addiert.

In einem **JTextField** kann der Benutzer natürlich auch falsche Werte eintragen. Zum Beispiel könnte er versehentlich in eines der Textfelder Buchstaben eingeben. Wenn man einen Text, der Buchstaben enthält, mit der Methode **parseInt()** der Klasse **Integer** in eine Zahl konvertieren will, erzeugt das Programm einen Laufzeitfehler vom Typ **NumberFormatException**.

Ein gutes Programm muss den Fehler also abfangen. Schreibe ein Programm, das alle Buchstaben und Sonderzeichen aus den eingegebenen Zeichenketten entfernt und nur die eingegebenen Ziffern auswertet. Wenn der Benutzer keine Ziffer in ein **JTextField** eingegeben hat, soll mit der Zahl 0 gerechnet werden.



Aufgabe 6: Geheimbotschaft

Es wurde vereinbart in einem ganz normalen Text eine geheime Botschaft zu übermitteln. Die Geheimbotschaft erhält man, wenn man alle Buchstaben, die hinter einem kleinen oder großen 'e' stehen, aneinander fügt. Beispieldateien:

1. Tante Paula und Tante Anna sind erstaunlich nett. Gruß Meybritt.
2. Bitte fang im Februar ein Lama für mich!
3. Ehrlich sein ist schön. Meld dich bitte früh. Gruß Meerit.

4. Emran und Anne Ohlmann sind mächtig. Erkundige dich mal.

Tipp zur Entschlüsselung: Wandle zunächst alle Buchstaben des Textes in Kleinbuchstaben um und lösche alle Zeichen aus dem Eingabe-String, die keine Buchstaben sind.



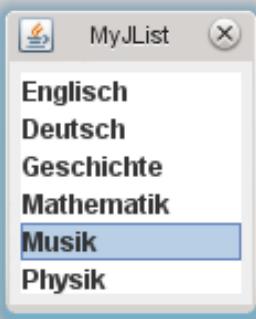
24.10 Komplexere Dialogelemente

Allgemein

Swing unterscheidet zwischen den Daten und ihrer Darstellung. Für die Swing-Komponenten gibt es jeweils ein Datenmodell (Model). Das ist uns bisher bei den einfachen Komponenten, wie `JLabel` und `JTextField` noch nicht begegnet¹. Aber jetzt ...

JList

Die Klasse `JList` ermöglicht es eine Liste von Daten in einem Dialogelement anzuzeigen. Das kann im einfachsten Fall eine Liste von `Strings` sein. Es könnten aber auch komplexere Datensätze – etwa eine Kombination von Bildern und Text – sein.



Im einfachsten Fall, nämlich der Präsentation von statischen (unveränderlichen) Listen kommt man noch ohne Model aus. Man muss bei der Erzeugung der `JList`-Komponente nur festlegen, welcher Datentyp abgebildet werden soll. Etwa so:

```
String[] faecher = {"Englisch", "Deutsch", "Geschichte",
                    "Mathematik", "Musik", "Physik"};
JList<String> listFaecher = new JList<String>(faecher);
```

In Spalten Klammern wird dabei der Datentyp angegeben (ab JDK7). Mit der Methode `getSelectedValue()` aus der Klasse `JList` bekommt man das vom Benutzer ausgewählte Element der Liste zurück:

```
String gewaehltesFach = listFaecher.getSelectedValue();
```

ListSelectionModel

Der Benutzer kann mit der Maus Einträge aus der Liste auswählen. Dabei kann festgelegt werden, ob er nur einzelne oder auch mehrere Einträge gleichzeitig auswählen kann:

```
listFaecher.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

legt fest dass nur jeweils ein Listen-Eintrag ausgewählt werden kann. Alternativ kann man auch

```
SINGLE_INTERVAL_SELECTION und MULTIPLE_INTERVAL_SELECTION
```

angeben. Damit kann der Benutzer später einzelne oder eben auch mehrere zusammenhängende Bereiche aus der Liste auswählen.

DefaultListModel

Oft ist es jedoch nötig, die Daten in der Liste zu verändern (etwa Daten hinzufügen oder löschen). Diese Flexibilität wird über die Klasse `DefaultListModel` erreicht: Im Model wird festgelegt um was für Daten es sich handelt. Das Model ist auch dafür zuständig mit diesen Daten zu arbeiten. Ohne Datenmodell auch keine dynamischen (veränderliche) Listen! Die Definition des Datenmodells ist also oft der erste Schritt um mit einer `JList`-Komponente arbeiten zu können.

```
String[] faecher = {"Englisch", "Deutsch", "Geschichte",
                    "Mathematik", "Musik", "Physik"};
DefaultListModel<String> faecherliste = new DefaultListModel<String>();
JList<String> listFaecher = new JList<String>(faecherliste);
```

¹Das stimmt nicht ganz: Im Abschnitt zur Ereignisbehandlung für die Klasse `JTextField` wurde bereits einmal dessen Datenmodell erwähnt/benutzt.

Wenn man die `JList`-Komponente nun darstellen lassen würde, wäre sie allerdings noch leer! Dank des Models ist unsere Liste ja nun dynamisch. Als erstes werden wir sie also mit den alten Daten befüllen:

```
for (String fach: faecher) {
    faecherliste.addElement(fach);
}
```

Die Methode `addElement()` aus der Klasse `DefaultListModel` erledigt dabei nicht nur das Einfügen der einzelnen Fächer in unsere Liste (es hängt das neue Element einfach an die bestehende Liste an), sondern sorgt gleichzeitig (und automatisch) dafür, dass alle Komponenten, die dieses Model benutzen die aktualisierte Liste darstellen.

Weitere wichtige Methoden aus der Klasse `DefaultListModel` sind:

| Methode | Erläuterung |
|--|--|
| * void add(int index, Object element) | Fügt das Element an der angegebenen Position ein. |
| void clear() | Löscht alle Elemente aus der Liste. |
| * Object get(int index) | Liefert das Element an der angegebenen Position. |
| int getSize() | Liefert die Größe der Liste zurück. |
| boolean isEmpty() | Prüft ob die Liste leer ist. |
| * Object remove(int index) | Löscht das Element an der angegebenen Position (und liefert es als Rückgabewert zurück). |
| * void setElementAt(Object element, int index) | Ändert den Wert des genannten Elements. |

(* throws `ArrayIndexOutOfBoundsException`

Um das vom Benutzer aktuell gewählte Element aus der Liste zu löschen, braucht man die Methode `getSelectedIndex()` aus der Klasse `JList`. Diese liefert den Index des aktuell ausgewählten Eintrags als Integer wert:

```
DefaultListModel<String> faecherliste = new DefaultListModel<String>();
JList<String> listFaecher = new JList<String>(faecherliste);
.
.
.

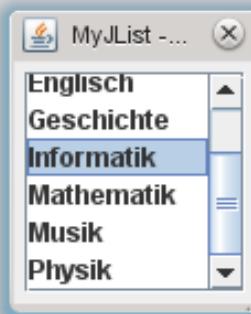
faecherliste.remove(listFaecher.getSelectedIndex());
```

JScrollPane

Was passiert aber, wenn die Liste so groß wird, dass sie nicht mehr in gegeben Platz der `JList`-Komponente hinein passt? Ganz einfach: Nichts. Oder besser gesagt: die zusätzlichen Daten sind einfach nicht sichtbar. `JList` selber bietet dazu keine Funktionalität. Und das `ListModel` selbstverständlich auch nicht, denn das hat mit der Darstellung der Daten ja gar nichts zu tun.

Die Klasse `JScrollPane` hingegen bietet anderen Komponenten wie `JList` oder `JTextArea` an, diese Scrollen für sie zu erledigen.

Dazu müssen wir unsere `JList`-Komponente nur in die `JScrollPane`-Komponente einbetten:



```
JScrollPane scrollPane = new JScrollPane();
scrollPane.setPreferredSize(new Dimension(WIDTH, HEIGHT));
contentPane.add(scrollPane);
JList<String> listFaecher = new JList<String>(faecherliste);
scrollPane.setViewportView(listFaecher);
```

JComboBox

Unter einer JComboBox versteht man eine Drop-Down-Auswahlliste. Im Unterschied zu einer JList-Komponente nimmt die JComboBox vertikal immer nur den Platz ein, der für die Darstellung des aktuell ausgewählten Elements benötigt wird. Es gibt folglich auch keine Notwendigkeit für eine Einbettung in eine JScrollPane. Auch kann man immer nur genau ein Element aus der Liste auswählen.

DefaultComboBoxModel

Genau wie bei JList ist es für dynamische Inhalte nötig, ein Daten-Modell zu benutzen:

```
String[] faecher = {"Englisch", "Deutsch", "Geschichte",
                    "Mathematik", "Musik", "Physik"};
DefaultComboBoxModel<String> faecherliste = new DefaultComboBoxModel<String>();
JComboBox<String> comboboxFaecher = new JComboBox<String>(faecherliste);
```



Die Methoden der Klasse `DefaultComboBoxModel` unterscheiden sich von denen der Klasse `DefaultListModel`:

| Methode | Erläuterung |
|--|--|
| <code>void addElement(Object element)</code> | Fügt das Element am Ende der Liste ein. |
| <code>* Object getElementAt(int index)</code> | Liefert das Element an der angegebenen Position. |
| <code>Object getSelectedItem()</code> | Liefert das vom Benutzer ausgewählte Element. |
| <code>int getSize()</code> | Liefert die Größe der Liste zurück. |
| <code>* void insertElementAt(Object element, int index)</code> | Fügt das Element an der angegebenen Position ein. |
| <code>void removeAllElements()</code> | Löscht alle Elemente aus der Liste. |
| <code>* void removeElementAt(int index)</code> | Löscht das Element an der angegebenen Position. |
| <code>void setSelectedItem(Object element)</code> | Ändert den Wert des aktuell vom Benutzer gewählten Elements. |

(*) throws `ArrayIndexOutOfBoundsException`

Im Besonderen ist es so, dass bei der Verwendung von JComboBox auch das Daten-Modell (`DefaultComboBoxModel`) eine Methode zur Verfügung stellt, um das gewählte Element zu ermitteln, während es bei JList nur über eine entsprechende Methode aus der Klasse JList (und nicht aus der Klasse des Daten-Modells) geht. Man hat also die Wahl zwischen

```
String gewaehltesFach = (String) comboboxFaecher.getSelectedItem();
```

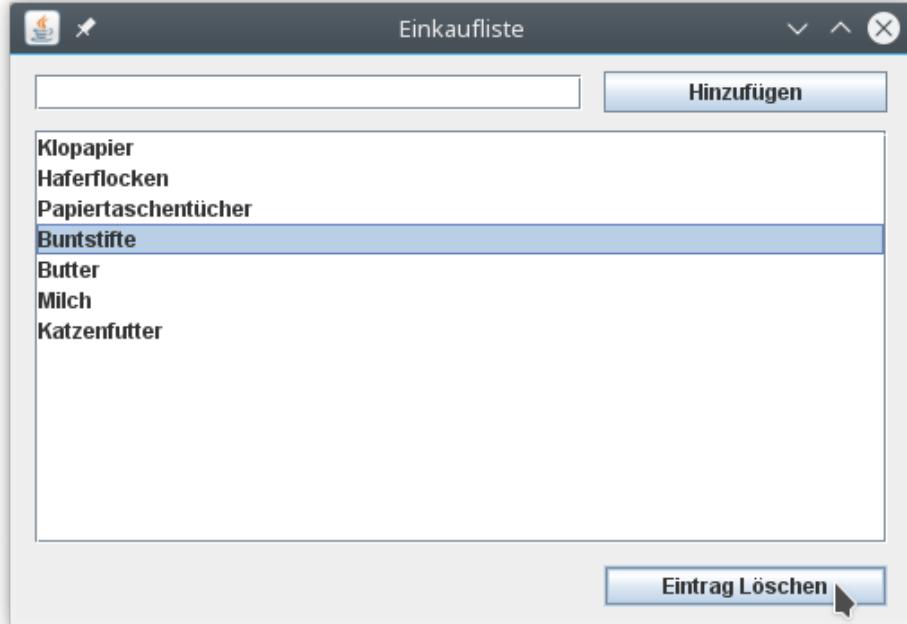
und

```
String gewaehltesFach = (String) faecherliste.getSelectedItem();
```

24.11 Komplexe Dialogelemente – Übungen

Aufgabe 1: Einkaufsliste

Programmiere einen elektronischen Einkaufszettel.



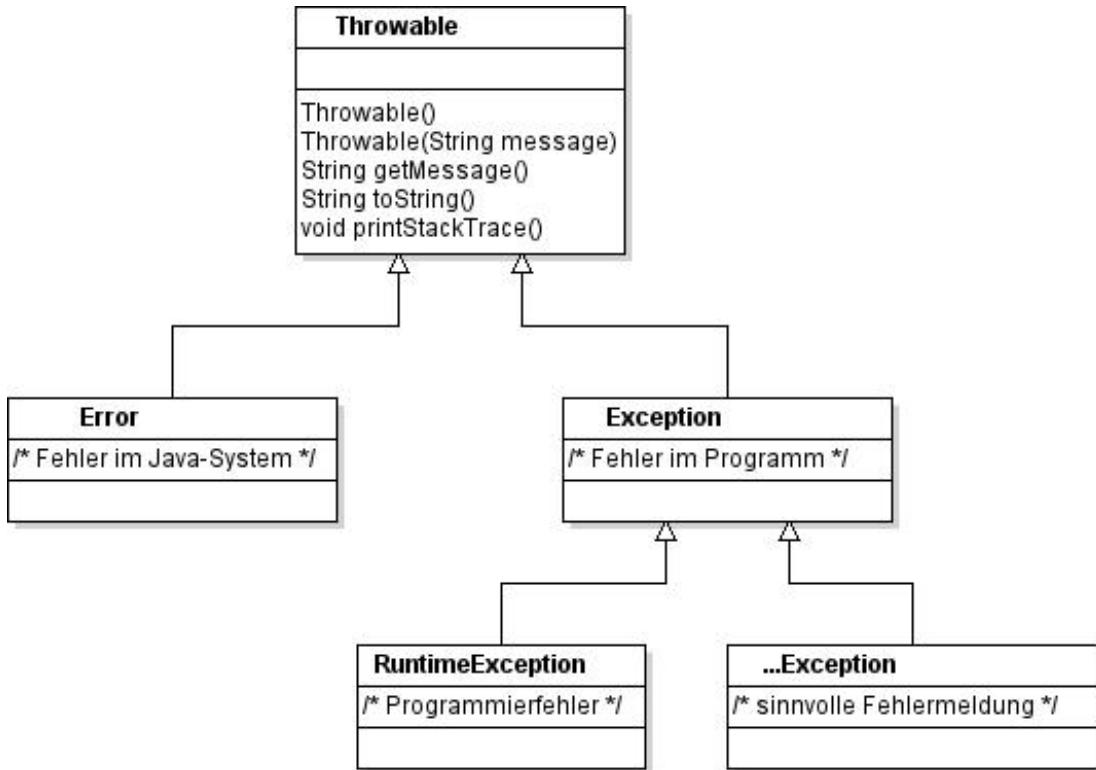
Text, der in dem Textfeld eingegeben wurde, soll durch betätigen des Buttons zum Hinzufügen in das Datenmodell der `JList`-Komponente übernommen werden. Übrigens kannst du auch dem Textfeld einen `ActionListener` spendieren und dort ebenfalls die Methode zum Hinzufügen zur Liste aufrufen. Dann reicht es im Textfeld nach Eingabe eines neuen Eintrags die Eingabetaste zu drücken und schon landet der Eintrag genauso in der Einkaufsliste, als wenn der Button gedrückt worden wäre.

Außerdem soll es möglich sein, den jeweils ausgewählten Listeneintrag aus dem Datenmodell der `JList`-Komponente zu löschen. Geschickter Weise sollte die `JList`-Komponente so im WindwoBuilder eingestellt werden, dass man jeweils nur einen Listeneintrag auswählen kann (`SelectionMode → SINGLE_SELECTION`).

25 Fehlerbehandlung

25.1 Fehler-Klassen

Informationen über aufgetretene Fehler werden in Java in Objekten der Klasse `Exception` (oder einer abgeleiteten Klasse) transportiert. Das folgende UML-Diagramm gibt einen Überblick über die Klassenhierarchie der Fehlermeldungen:



- Die Superklasse `Throwable` wird nie direkt verwendet.
- Errors** sind Fehler im System, die von den Programmierern des Java-Systems verschuldet wurden. Normalerweise sollten niemals Errors auftreten.
- Exception** ist der Sammelbegriff für alle Fehler, die im Programm behandelt werden können.
- RuntimeExceptions** sind Fehler, die durch falsche Programmierung verursacht wurden (z.B. durch eine unerlaubte Typumwandlung). Bei korrekter Programmierung dürfen RuntimeExceptions niemals auftreten.
- Sinnvolle Exceptions** (z.B. `PrinterException` – „im Drucker ist kein Papier“) sollten vom Programm abgefangen und dem Benutzer angezeigt werden. Man kann auch eigene Exception-Klassen erzeugen.

Wichtige Methoden der Super-Klasse `Throwable`

| Methode | Erläuterung |
|--|--|
| <code>Throwable()</code> | Konstruktor: Es kann optional ein Fehlertext angegeben werden. |
| <code>Throwable(String message)</code> | |
| <code>String getMessage()</code> | Liefert den Fehlertext zurück. |
| <code>String toString()</code> | Liefert den Klassennamen der Exception und den Fehlertext zurück. |
| <code>public void printStackTrace()</code> | Gibt auf der Konsole aus, bei welcher Programmzeile das Programm abgestürzt ist. |

25.2 Beispiel

a) Erstellung einer eigenen Exception-Klasse

```
public class DurchNullException extends Exception {

    public DurchNullException() {
        super("Man kann nicht durch 0 teilen.");
    }

    public DurchNullException(String text) {
        super(text);
    }
}
```

b) Exceptions auslösen (throw)

```
public class Formeln {

    public static double division(double x1, double x2) throws DurchNullException {
        if (x2 == 0) {
            DurchNullException fehler;
            fehler = new DurchNullException();
            throw fehler;
            // Oder kürzer:
            // throw new DurchNullException("Teilen durch 0 ist verboten!");
        }
        return (x1 / x2);
    }
}
```

Durch den Zusatz `throws` im Methodenkopf, wird der Methode erlaubt eine auftretende Exception der angegebenen Art an die aufrufende Methode weiter zu reichen. Umgekehrt wird diese Information vom Java-System auch dazu benutzt zu erkennen, dass durch den Aufruf dieser Methode eine entsprechende Exception erzeugt werden kann (und dass in der aufrufenden Methode entsprechende Maßnahmen zum Umgang mit dieser Exception getroffen werden müssen).

Der Befehl `throw` im Methodenkörper schließlich löst diese Exception dann aus.

c) Exception abfangen (catch)

```
public class ExceptionBsp {

    public static void main(final String[] args) {
        try {
            for (int i = 5; i > -5; i--) {
                System.out.println("10 durch " + i + " = "
                    + Formeln.division(10, i));
            }
        } catch (DurchNullException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Exceptions können in einem `try-catch` Block abgefangen und behandelt werden. Im `try` Teil dieses Konstrukts steht der Teil des Codes, in dem eine Exception auftreten kann.

Für den Fall, dass die Exception tatsächlich ausgelöst wird, werden alle noch folgenden Befehle innerhalb des `try`-Blocks übersprungen und statt dessen, die Befehle im `catch`-Block abgearbeitet.

Falls es nicht zu einer Exception kommt, wird der `try`-Block komplett abgearbeitet und der `catch`-Block ignoriert.

25.3 Weiterleiten von Exceptions

Manchmal kann es sinnvoll sein, eine Exception in einem `catch`-Block nicht direkt zu behandeln sondern den Fehler weiterzuleiten, damit die Fehlerbehandlung an zentraler Stelle in einem übergeordneten `catch`-Block ausgeführt werden kann. Auch dies ist möglich. Beispiel (Code-Auszug):

```
public class ExceptionBsp2 {

    public static void test1() throws DurchNullException {
        for (int i = 0; i < 10; i++) {
            System.out.println("10 durch " + i + " = " + Formeln.division(10, i));
        }
    }

    public static void test2() throws DurchNullException {
        try {
            System.out.println("" + Formeln.division(4,
                Integer.parseInt("Hi")));
        } catch (DurchNullException e) {
            // diese Exception soll an anderer Stelle weiter bearbeitet werden
            throw e;
        } catch (Exception e) {
            System.out.println("Unerwarteter Fehler: " + e.toString());
        }
    }

    public static void main(final String[] args) {
        try {
            test1();
            test2();
        } catch (DurchNullException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Erläuterung zum zweiten Beispiel

Vergleicht man die Methoden `test1()` und `test2()`, so fällt auf, dass `throws DurchNullException` in beiden Methodenköpfen deklariert wird.

Es ist ihnen also erlaubt Exceptions von diesem Typ nicht zu behandeln! In `test1()` geschieht dies offensichtlich auch nicht. In `test2()` hingegen gibt es einen `catch`-Block, der die `DurchNullException` abfängt.

Warum? In diesem `catch`-Block wird die gerade empfangene Exception sofort weitergeleitet (`throw`). Das `throw` (also die Weiterleitung des Fehlers bzw. die *Propagation* der Fehlerbehandlung) ist nur erlaubt, weil im Methodenkopf `throws DurchNullException` deklariert wurde. Trotzdem ist dann noch nicht geklärt, warum der erste `catch`-Block in `test2()` nötig ist – immerhin klappt diese Weiterleitung in `test1()` auch ohne `catch`-Block.

In `test2()` hingegen, würde ohne diesen ersten `catch`-Block die `DurchNullException` im folgenden `catch`-Block (`catch (Exception e)`) mit „verarbeitet“.

Und wieder: warum? `DurchNullException` ist zwar nicht das gleiche wie eine generische Exception, aber da `DurchNullException` von `Exception` abgeleitet wurde lässt sie sich auch auf diesen Datentyp abbilden. `Exception` passt also auf alle Fehlerarten!

Merksatz: Exceptions müssen dort wo sie entstehen (können) entweder behandelt oder abgefangen werden („`catch or throw`“).

Ausnahme: Nur Fehlertypen, die von der Klasse `RuntimeException` abgeleitet wurden, müssen nicht unbedingt behandelt oder mit `throws` im Methodenkopf deklariert werden. Als Beispiel sei die `NumberFormatException` genannt, die im zweiten Beispiel auftritt, wenn man `test1();` in `main()` auskommentiert.

25.4 Behandlung verschiedenartiger Exceptions

```
try {  
    // Anweisungen  
}  
catch (<Fehler-Typ> var1) {  
    // Fehler-Behandlung  
}  
catch (<Fehler-Typ2> var2) {  
    // Fehler-Behandlung  
}  
finally {  
    // wird immer ausgeführt  
}
```

- Der erste passende `catch`-Block wird ausgeführt. Wenn ein passender `catch`-Block gefunden wurde, wird kein weiterer `catch`-Block ausgeführt, auch wenn der Exception-Typ passend ist.
- Der `finally`-Block steht optional hinter dem letzten `catch`-Block. Er ist für Aufräumarbeiten gedacht, die grundsätzlich ausgeführt werden müssen, zum Beispiel das Schließen einer Datei (was allerdings seit Java 7 eleganter durch das sogenannte *try-with-resource* gelöst wird).
- Der `finally`-Block wird immer ausgeführt – egal ob eine Exception auftritt (und behandelt wird) oder nicht.

25.5 Fehlerbehandlung – Übungen

Aufgabe 1: Währungsrechner

Schreibe ein Programm, das zwischen zwei Währungen umrechnen kann (zum Beispiel zwischen Euro und Britischen Pfund). Damit das Programm nicht veraltet wird, wird der aktuelle Währungskurs in einem `JTextField` eingegeben. Die Oberfläche könnte z.B. so aussehen:



Das Ausgabe-Textfeld soll nicht editierbar sein. Programmiere zunächst die Oberfläche (ohne Umrechnungsfunktionalität) und gehe dann in folgenden Teilschritten vor:

- Leite drei verschiedene Exception-Klassen von der Superklasse `Exception` ab. Alle Exception-Klassen sollen einen parameterlosen Konstruktor besitzen, der automatisch den in Klammern angegebenen Fehler-Text generiert:
 - `LeerException` („Sie haben die Eingabe in das Textfeld vergessen. Das Textfeld ist leer.“)
 - `KommaException` („Sie haben in der Fließkommazahl ein Komma an Stelle eines Punktes eingegeben.“)
 - `ZeichenException` („Sie haben ein unerlaubtes Zeichen eingegeben. Zulässig sind nur Ziffern und ein Punkt.“)
- Schreibe eine Methode `zahlAuslesen()`, der ein Textfeld als Parameter übergeben wird. Die Methode überprüft die Eingabe im Textfeld auf Korrektheit und wirft bei Eingabefehlern des Benutzer die entsprechende Exception. Wenn der Benutzer den Text korrekt eingegeben hat, konvertiert sie den Text mit der Methode `Double.parseDouble()` in einen `double`-Wert und gibt die so erzeugte Zahl als Rückgabewert zurück.
- Programmiere den Code zur Währungsumrechnung unter Verwendung der Methode `zahlAuslesen()`. Fangt eventuell auftretende Exceptions ab und gib den Fehler-Text (inklusive des Namens der Fehlerklasse) in einer Message-Box aus.

Aufgabe 2: Benzinkosten berechnen

Schreibe ein Programm, das die Benzinkosten für eine bestimmte Strecke (zum Beispiel Bremen – Hamburg) berechnet. Der Benutzer soll folgende Eingaben vornehmen können:

- die Strecke in km
- der Verbrauch des PKWs pro 100 km
- der aktuelle Benzinpreis (in Euro/Liter)

Im Hauptfenster gibt es drei Textfelder zu Eingabe der Werte und einen Button, mit dem die Berechnung gestartet wird. Die vom Benutzer eingegebenen Strings müssen zur Berechnung mit der Methode `Double.parseDouble()` in Fließkommazahlen umgewandelt werden. Falls der Benutzer eine fehlerhafte Eingabe gemacht hat, wirft die Methode `parseDouble()` eine `NumberFormatException`. Fangt diese Exception in einem `catch`-Block ab, und gib dem Benutzer gegebenenfalls eine für ihn verständliche Fehlermeldung aus. (Eigene Exceptions brauchen bei dieser Übung nicht generiert werden).



Konvertierung von String in double

Beispiel:

```
String text = "3.14";
double pi = Double.parseDouble(text);
```

Aufgabe 3: Andere Programme starten

Mit Hilfe der Klasse `Runtime` können von einem Java-Programm aus andere Programme gestartet werden. Ein Objekt der Klasse `Runtime` holt man sich mit einer statischen Methode von der Klasse `Runtime` selbst:

```
public static Runtime getRuntime()
```

Anschließend kann man mit der folgenden Methode der Klasse `Runtime` ein anderes Programm starten:

```
public Process exec(String command) throws IOException
```

Die Methode wirft eine `IOException`, wenn das Programm nicht gefunden werden konnte.

Als Rückgabewert erhält man ein Objekt der Klasse `Process` (Anmerkung: Prozess ist das Fachwort für ein laufendes Programm.). Mit Hilfe dieses Objektes kann man mit dem gestarteten Programm „kommunizieren“, in dem man zum Beispiel die folgenden Methoden der Klasse `Process` aufruft:

```
public void destroy()
```

Mit Hilfe dieser Methode wird das laufende Programm auf harte Weise (ohne Benutzerabfrage) beendet.

```
public int waitFor() throws InterruptedException
```

Mit der Methode `waitFor()` kann man im aktuellen Programm auf die Beendigung des gestarteten Programms warten. Dies ist zum Beispiel sinnvoll, wenn das fremde Programm wichtige Daten berechnet, die zur weiteren Arbeit benötigt werden. Das aktuelle Programm läuft nicht mehr weiter, bis das fremde Programm beendet wurde. Als Rückgabewert erhält man die Zahl, die das fremde Programm im `exit()`-Aufruf zurückgibt. Dies ist normalerweise der Wert 0, der für „erfolgreich beendet“ steht. Die `InterruptedException` kann nur theoretisch auftreten und darf ignoriert werden. Man braucht jedoch einen Exception-Handler ohne Funktionalität, um den Code durch den Compiler zu bekommen.

Übungsaufgabe



Erzeuge ein Programm mit der oben abgebildeten Oberfläche. Wenn man auf den Button „Start mit Warten“ drückt, wird das vom Benutzer angegebene Programm gestartet und anschließend mit `waitFor()` auf seine Beendigung gewartet. Nach der Rückkehr des Aufrufs von `waitFor()` wird eine Meldung ausgegeben, in der der Rückgabewert des fremden Programms mitgeteilt wird. Falls das Programm nicht gestartet werden kann, wird in einem Exception-Handler eine entsprechende Fehlermeldung ausgegeben.

Wenn man auf den Button „Start ohne Warten“ drückt, wird das Programm einfach nur gestartet. Das zurückgegebene `Process`-Objekt wird in einer globalen Variablen gespeichert. Wenn man auf den Stop-Button drückt, wird das Programm, das in der `Process`-Variablen abgespeichert wurde, mit `destroy()` beendet. Falls in der `Process`-Variablen kein fremdes Programm abgespeichert ist, gibt es eine `NullPointerException`. Fangt diese Exception mit einem Exception-Handler ab und gib gegebenenfalls eine geeignete Fehlermeldung aus.

26 Nebenläufige Programmierung: Threads

26.1 Wofür benötigt man Threads?

Threads ermöglichen es ein Programm in mehrere Teile zu unterteilen, die unabhängig voneinander parallel ausgeführt werden. Jeder Thread (engl. für Faden) arbeitet eigenständig in seinem eigenen Tempo. Über Variablen können die Threads Informationen untereinander austauschen. Wenn ein Programm beispielsweise eine lange Berechnung durchführen muss, die mehrere Minuten dauert, so wäre es in einem „normalen“ Programm während dieser Zeit nicht möglich auf Benutzereingaben zu reagieren. Wenn man die Berechnung dagegen in einem zweiten Thread durchführt, kann sich der Haupt-Thread weiterhin um Benutzereingaben kümmern. Weitere Beispiele für den Einsatz von Threads:

- Wenn ein Text gedruckt wird, ist dies eine Aktion die relativ lange dauert. Wird der Druckvorgang in einem eigenen Thread durchgeführt, so steht der Haupt-Thread dem Benutzer während des Druckvorgangs für weitere Arbeitsaufträge zur Verfügung.
- Während auf Eingaben des Benutzers gewartet wird, sollen im Hintergrund aufwändige Berechnungen fortgeführt werden.

Jede von Thread abgeleiteten Klasse muss die Methode `run()` überschreiben. Diese wird automatisch ausgeführt sobald der Thread über die geerbte (und bei Bedarf ebenfalls überschriebene) Methode `start()` aus der Basisklasse `Thread` gestartet wird.

Der Methode `start()` kommt dabei eine besondere Aufgabe zu: Sie sorgt dafür, dass der neue Thread aus Sicht des Betriebssystems als eigenständiger Prozess läuft. Nur so laufen die verschiedenen Threads tatsächlich parallel nebeneinander her. Würde man `run()` hingegen direkt aufrufen (was technisch gesehen durchaus möglich ist), würde diese Methode – wie alle anderen Methoden auch – ganz normal ausgeführt. Und der nächste Programmschritt würde erst dann bearbeitet, wenn `run()` seine Arbeit beendet hat.

Obwohl es in der Basisklasse `Thread` eine Methode `stop()` gibt, sollte man diese nicht verwenden um einen Thread zu beenden! Vielmehr sollte man über eine boolesche Variable, die den Zustand (läuft oder läuft nicht) des Threads festhält steuern, ob die Methode `run()` weiter ausgeführt werden soll. Der Thread als Ganzes wird beendet, sobald die Methode `run()` beendet ist. Siehe dazu das letzte Beispiel in diesem Kapitel.

Merksatz: `run()` muss, `start()` kann überschrieben werden. Und `stop()` sollte man nicht verwenden.

26.2 Beispiel

a) Thread-Klasse

```
import javax.swing.JLabel;

public class ZaehlThread extends Thread {
    JLabel lbl;
    int index;
    int zaehler = 0;

    public ZaehlThread(JLabel lbl, int index) {
        this.lbl = lbl;
        this.index = index;
    }

    @Override
    public void run() {
        while (true) {
            lbl.setText("Zaehler" + index + ":" + zaehler);
            try {

```

```
        Thread.sleep(index * 1000);
    }
    catch (Exception e) {
    }
    zaehler++;
}
}
```

Erläuterungen:

1. Um einen Thread zu programmieren muss man eine Klasse von der Superklasse `Thread` ableiten.
 2. Die Methode `run()` wird vom System automatisch aufgerufen sobald der Thread gestartet wurde. Der Thread existiert bis die Methode `run()` beendet ist.
 3. `sleep()` ist eine statische Methode der Klasse `Thread`, die den aktuellen Thread für die angegebene Anzahl von Millisekunden schlafen legt (blockiert).

b) **Frame**

```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.EmptyBorder;

public class ThreadBeispiel extends JFrame {
    // globale Variablen
    private static final int WIDTH = 200;
    private static final int HEIGHT = 120;
    private JLabel label[];
    private ZaehlThread thread[];

    public ThreadBeispiel(final String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new FlowLayout());
        contentPane.setPreferredSize(new Dimension(WIDTH, HEIGHT));
        setContentPane(contentPane);
        label = new JLabel[3];
        thread = new ZaehlThread[3];
        for (int i = 0; i < 3; i++) {
            label[i] = new JLabel();
            label[i].setFont(new Font("Arial", Font.BOLD, 20));
            contentPane.add(label[i]);
            thread[i] = new ZaehlThread(label[i], i + 1);
            thread[i].start();
        }
        pack();
        setLocationRelativeTo(null);
        setResizable(false);
        setVisible(true);
    }

    public static void main(final String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    new ThreadBeispiel("Thread-Beispiel");
                }
                catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
```

Die Anwendung erzeugt ein oder mehrere Objekte der selbst geschriebenen Thread-Klasse. Anschließend werden die Threads durch den Aufruf der Methode `start()` gestartet.

Frage:

1. Wie viele Threads besitzt dieses Beispiel-Programm?
 2. Wieso darf man die Thread-Methode `run()` nicht direkt aufrufen?

26.3 Animation steuern mit einem eigenen Timer-Thread

Wenn man eine Animation programmieren möchte, muss in regelmäßigen Zeitabständen die Methode `repaint()` aufgerufen werden. Dazu legt man einen eigenen Thread an, dessen einzige Aufgabe es ist die Zeit zu zählen und immer wieder mit `repaint()` das Neu-Zeichnen des Frames und seiner Komponenten anzustoßen. Natürlich ist es nicht nötig, solch einen Timer selbst zu programmieren: Wir haben ja auch bislang schon immer die Klasse `javax.swing.Timer` für diesen Zweck verwendet. Aber es ist ein schönes Beispiel.

Beispiel

```
public class MyTimer extends Thread {  
    private int repaintMillisec;  
    private JFrame frame;  
    private boolean threadBeendet = false;  
  
    MyTimer(int millisec, JFrame app) {  
        repaintMillisec = millisec;  
        frame = app;  
    }  
  
    public void beenden() {  
        threadBeendet = true;  
    }  
  
    @Override  
    public void run() {  
        while (!threadBeendet) {  
            try {  
                frame.repaint();  
                Thread.sleep(repaintMillisec);  
            }  
            catch (Exception e) {  
            }  
        }  
    }  
}
```

Und die Fenster-Klasse, die den Timer benutzt:

```
import java.awt.*;  
import java.awt.event.*;
```

```
import javax.swing.*;
import javax.swing.border.EmptyBorder;

public class SpringendeBaelle extends JFrame {
    private static final int WIDTH = 500;
    private static final int HEIGHT = 500;
    private static final Color BACKGROUND = Color.WHITE;
    private JPanel zeichenflaeche;
    private Ball b1, b2, b3;

    public SpringendeBaelle(final String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new BorderLayout(0, 0));
        setContentPane(contentPane);
        zeichenflaeche = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                myPaint(g);
            }
        };
        zeichenflaeche.setPreferredSize(new Dimension(WIDTH, HEIGHT));
        zeichenflaeche.setBackground(BACKGROUND);
        contentPane.add(zeichenflaeche);
        pack();
        setLocationRelativeTo(null);
        setResizable(false);
        setVisible(true);
        b1 = new Ball(100, 100, 50, -10, Color.RED, false);
        b2 = new Ball(200, 200, 150, -5, Color.GREEN, true);
        b3 = new Ball(400, 50, 20, 7, Color.BLUE, true);
        MyTimer timer = new MyTimer(30, this);
        timer.start();
    }

    public void myPaint(Graphics g) {
        // wird aufgerufen, wenn das Fenster neu gezeichnet wird
        b1.zeichnen(g);
        b2.zeichnen(g);
        b3.zeichnen(g);
    }

    public static void main(final String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    new SpringendeBaelle("Springende Bälle");
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

26.4 Threads – Übungen

Aufgabe 1: Rennendes Pferd

Programmiere eine Animation, in der ein Pferd wiederholt von links nach rechts über den Bildschirm galoppiert.

Erzeuge dazu ein neues Anwendungsfenster (in Eclipse: Rechtsklick auf das Package im Package Explorer → *New* → *Other ...* → *Window Builder* → *Swing Designer* → *JFrame*) und erzeuge eine Komponente **zeichenflaeche** (die du von **JPanel** ableitest). Lade im Anwendungsfenster die beiden Bilder **horse1.gif** und **horse2.gif**. Überschreibe die **paintComponent()**-Methode von **zeichenflaeche**, und zeichne in dieser Methode das Pferd immer abwechselnd einmal mit dem ersten und anschließend mit dem zweiten Bild. Verschiebe außerdem die x-Position des Pferdes bei jedem Aufruf von **paintComponent()** um 10 Pixel nach rechts.

Damit die **paintComponent()** wiederholt aufgerufen wird, sollst du eine zweite Klasse programmieren, die sich von der Klasse **Thread** ableitet (benutze nicht **javax.swing.Timer**). Übergib der Klasse im Konstruktor einen Verweis auf das Anwendungsfenster. Füge in der **run()**-Methode eine Endlosschleife ein, in der nach einer Wartezeit von 100 Millisekunden immer wieder die **repaint()**-Methode des Anwendungsfensters aufgerufen wird.



Aufgabe 2: Oldtimer

Erstelle wie in der Abbildung ein Anwendungsfenster mit zwei Buttons und einer Zeichenfläche, die von der Klasse **JPanel** abgeleitet wird.



Lade das Bild **car.gif** und zeichne es in der Mitte der Zeichenfläche. Verschiebe die x-Position des Autos in der **paintComponent()**-Methode der Zeichenfläche bei jedem Aufruf ein Stück nach links, so dass das Auto rollt, wenn ein Timer-Thread für ein regelmäßiges Neuzeichnen des Fensters sorgt. Wenn das Auto links aus der Zeichenfläche herausgefahren ist, soll es von rechts wieder in die Zeichenfläche hinein gleiten.

Zu Beginn soll jedoch noch kein Timer-Thread laufen und das Auto steht deshalb still. Der Stopp-Button ist disabled. Wenn man auf den Start-Button drückt, soll ein Timer-Thread gestartet werden. Dieser ruft in einer Schleife wiederholt die **repaint()**-Methode der Anwendung auf. Die Schleife des Threads läuft so lange, bis eine boolesche Variable **anhalten** auf **true** gesetzt wird. Dann beendet der Thread die Schleife und die **run()**-Methode hört damit auf zu existieren. Wenn der Benutzer auf den Stopp-Button drückt, wird die boolesche Variable auf **true** gesetzt, um dem Thread zu signalisieren, dass er seine Arbeit beenden soll.

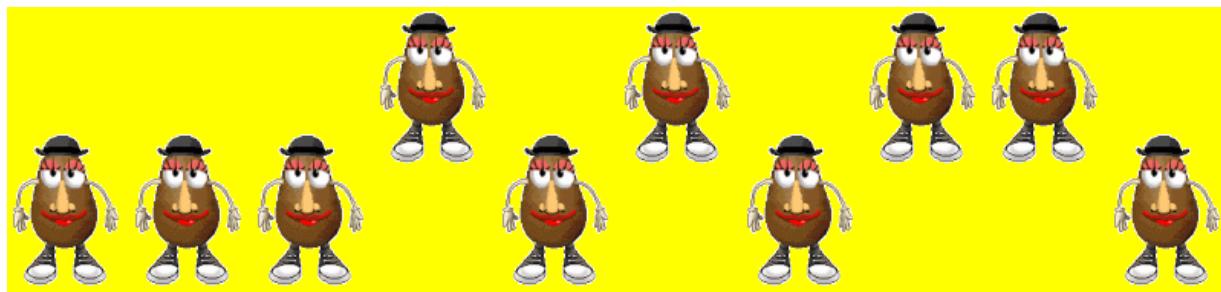
Sorge dafür, dass der Stopp-Button immer dann disabled ist, wenn kein Thread läuft, und der Start-Button disabled ist, wenn ein Thread am Laufen ist.

Aufgabe 3: Jumping Potatoes

Erstelle ein neues Anwendungsfenster mit einer Zeichenfläche mit einer Breite von 900 Pixeln und einer Höhe von 350 Pixeln hat. Lade in dem Anwendungsfenster das Bild potato.gif.

Programmiere außerdem eine Klasse Potato. Die Klasse besitzt als Variablen das Anwendungsfenster und die x- und y-Position der Kartoffel (das Bild muss die Klasse nicht kennen). Alle drei Werte werden im Konstruktor als Parameter übergeben.

Erzeuge im Anwendungsfenster ein Array von zehn Potatoes. Alle Potatoes sollen die y-Position 200 erhalten. Die x-Position des ersten Kartoffel soll bei 40 Pixel liegen. Die anderen Potatoes sollen jeweils im Abstand von 80 Pixel folgen. Zeichne die Potatoes in der `paintComponent()`-Methode der Zeichenfläche, in dem du in einer Schleife zehn mal das Bild zeichnest. Die x- und y-Position holst du dir dabei jeweils von dem Potato-Objekt mit dem entsprechenden Index.



Leite die Klasse Potato nun von der Adapter-Klasse `MouseAdapter` ab. Du musst dann lediglich die Methode `mouseClicked()` sinnvoll überschreiben. Registriere den `MouseListener` auf die Komponente `zeichenflaeche`. Wenn der Benutzer mit der Maus in das Fenster klickt, prüft die Klasse, ob der Benutzer auf das dem Potato-Objekt zugeordnete Bild geklickt hat (ein Bild besitzt eine Breite von 74 Pixel und eine Höhe von 100 Pixel). Wenn die Potato angeklickt wurde, soll sie nach oben springen. Falls der Benutzer die linke Maustaste benutzt (`MouseEvent.BUTTON1`), springt die Kartoffel 80 Pixel hoch. Falls der Benutzer die rechte Maustaste verwendet (`MouseEvent.BUTTON3`), springt die Kartoffel 160 Pixel hoch. Setze dazu jeweils die y-Position geeignet um und veranlasse, dass das Anwendungsfenster neu gezeichnet wird, damit der Benutzer den Sprung auch sieht. Merke dir außerdem die ursprüngliche y-Position in einer globalen Variablen, damit die Potato später wieder dahin zurückkehren kann.

Die Potato soll zwei Sekunden in der erhöhten Position bleiben und anschließend automatisch wieder zurück springen. Erzeuge dazu einen Thread, der das Objekt der angeklickten Kartoffel im Konstruktor als Parameter erhält. Der Thread muss in der `run()`-Methode nur einmal zwei Sekunden warten. Anschließend ruft er eine Methode des Potato-Objekts auf, die die y-Position wieder zurücksetzt und das Anwendungsfenster neu zeichnet.

Wenn du alles richtig programmiert hast, kann man mehrere Potatos nacheinander anklicken, die dann zeitlich versetzt wieder in ihre Ausgangsposition zurückspringen.

Aufgabe 4: Störrischer Esel

- Erzeuge ein Anwendungsfenster mit einer Zeichenfläche mit einer Breite von mindestens 600 Pixeln. Lade im Anwendungsfenster das Bild esel.gif. Das Bild besitzt eine Breite von 140 und eine Höhe von 130 Pixel. Zeichne den Esel zu Beginn ungefähr in der Mitte des Fensters.
- Wenn man mit der Maus vor oder hinter den Esel klickt, soll der Esel weglassen. Überprüfe dazu zunächst, ob sich die y-Position des Mausklicks im Bereich des Eselbildes befindet. Falls dies nicht der Fall ist, reagiert der Esel nicht. Falls sich die y-Position im Eselbereich befindet, weicht der Esel entweder nach links oder nach rechts aus. Wenn die x-Position der Maus größer oder gleich der Bildmitte ist, läuft er so lange nach links, bis sich seine rechte Seite 150 Pixel vor der Mausposition befindet und bleibt dort stehen. Falls die x-Position der Maus kleiner als die Bildmitte ist, läuft er nach rechts, bis sich seine linke Seite 150 Pixel hinter der Mausposition befindet. Falls der Benutzer während einer Laufbewegung erneut in das Fenster klickt, ändert der Esel die Laufbewegung entsprechend der neuen x-Position der Maus.

Am einfachsten kannst du das Weglaufen des Esels programmieren, wenn du gleich zu Beginn einen Timer-Thread startest, der die ganze Zeit durchläuft. Der Timer sorgt dafür, dass das Frame alle 10 Millisekunden neu gezeichnet wird. Merke dir den Zustand des Esels (stehend, nach links oder nach rechts) in einer Zustandsvariable und bewege den Esel bei jedem Aufruf von `paintComponent()` der Komponente `zeichenflaeche` entsprechend seinem Zustand um ein Pixel nach vorne oder nach hinten.

- c) Wenn man mit der Maus direkt auf das Eselbild klickt, springt der Esel vor Schreck nach oben. Setze dazu seine y-Position 50 Pixel nach oben und setzte sie nach 500 Millisekunden automatisch auf den alten Wert zurück. Es empfiehlt sich für diese Aufgabe einen extra Thread zu starten, der ausschließlich die Aufgabe hat 500 Millisekunden zu warten und anschließend die y-Position zurück zu setzen.

Die unter b) programmierte Laufbewegung des Esels soll durch das Springen nicht beeinflusst werden.

26.5 Thread-Synchronisation

Beispiel

Zwei Threads benutzen einen gemeinsamen Zähler. Dadurch, dass das Betriebssystem die Ausführung des Codes an jeder beliebigen Stelle unterbrechen kann, entstehen jedoch Fehler. Die ausgegebenen Zahlen entsprechen nicht der gewünschten Reihenfolge.

Haupt-Klasse

```
public class Synchronisation {
    static int zaehler = 0;

    public static void main(final String[] args) {
        MeinThread thread_1 = new MeinThread(1);
        MeinThread thread_2 = new MeinThread(2);
        thread_1.start();
        thread_2.start();
    }
}
```

Thread-Klasse

```
public class MeinThread extends Thread {
    int threadNummer;

    public MeinThread(int threadNummer) {
        this.threadNummer = threadNummer;
    }

    @Override
    public void run() {
        for (int i=0; i<100; i++) {
            Synchronisation.zaehler++;
            int help = Synchronisation.zaehler;
            // Simulation einer aufwändigen Berechnung:
            for (long x = 0; x<1000000; x++) {
                Math.sin(x);
            }
            System.out.println("Thread" + threadNummer + ":" + help);
        }
    }
}
```

Lösung des Problems

Anweisungen, während deren Ausführung eine Unterbrechung (*Task-Wechsel* durch den *Scheduler* des Betriebssystems) zu unerwünschten Ergebnissen führen, werden als *kritische Anweisungen* bezeichnet.

Anweisungen bzw. Anweisungsfolgen sind dann kritisch, wenn auf eine gemeinsam genutzte Ressource (etwa eine Datei, ein Gerät oder auch ein Textfeld) nicht nur lesend, sondern auch schreibend zugegriffen wird.

Mann kann einen Block mit kritischen Anweisungen vor Unterbrechungen schützen, indem man die entsprechenden Anweisungen in einen *synchronisierten Block* einfasst. Dazu benutzt man das Schlüsselwort **synchronized**. Über ein *Monitor-Objekt* (oft auch einfach *Monitor* genannt) wird der exklusive Zugriff verwaltet.

Als Monitor-Objekt ist grundsätzlich jedes Objekt geeignet, auf das alle Threads zugreifen können. Ein Thread kann den synchronisierten Block nur betreten, wenn kein anderer Thread den entsprechenden Monitor bereits belegt.

Syntax:

```
synchronized (monitorObjekt) {  
    //Anweisungen, die nicht unterbrochen werden dürfen  
}
```

Wichtig ist, dass alle Threads das gleiche Objekt als Monitor benutzen!

Achtung: Damit alle Threads zum Zug kommen, sollten immer so wenig Befehle wie möglich in einem synchronized-Block stehen!

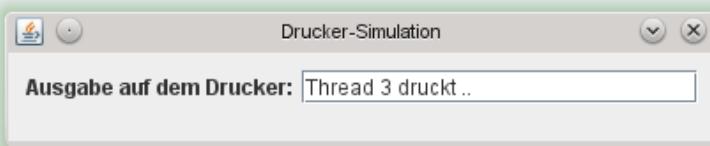
26.6 Thread-Synchronisation – Übungen

Aufgabe 1: Drucker-Simulation

Fünf Threads teilen sich den Zugriff auf einen Drucker und müssen sich untereinander abwechseln. Wir simulieren dies, in dem wir als „Drucker“ ein `JTextField` benutzen, in dem von den Threads ein Text der Form Thread 2 druckt ... ausgegeben werden soll.

Damit das Drucken in das Textfeld – wie beim richtigen Drucken – auch spürbar Zeit verbraucht, kannst du die einzelnen Zeichen des Strings zeitlich verzögert ausgeben lassen:

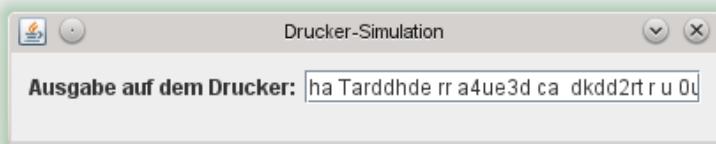
```
for (int i = 0; i < zuDruckenderText.length(); i++) {
    drucker.tfDrucker.setText(drucker.tfDrucker.getText() + zuDruckenderText.charAt(i));
    try {
        Thread.sleep(300);
    } catch (InterruptedException e) {
        // Fehlerfall ignorieren
    }
}
```



Die Threads sollen in einer Endlosschleife immer abwechselnd drucken und anschließend irgendwelche Berechnungen durchführen. Die Berechnung soll durch einen `sleep()`-Befehl, der eine zufällig generierte Zeitspanne lang dauert, simuliert werden. Erzeuge zur Simulation des Druckvorgangs Zufallszahlen zwischen 1 und 10 und multipliziere das Ergebnis mit 150 Millisekunden.

Beachte, dass es in der Anwendung nur einen einzigen Zufallsgenerator geben darf, den sich alle Threads teilen. Wenn jeder Thread seinen eigenen Zufallsgenerator besitzt, erhalten alle Threads mit hoher Wahrscheinlichkeit immer dieselben Zufallszahlen, weil alle Zufallsgeneratoren zur selben Zeit gestartet werden.

Ohne geeignete Synchronisation werden sich die einzelnen Threads beim Drucken ordentlich in die Quere kommen:



Aufgabe 2: Dining Philosophers

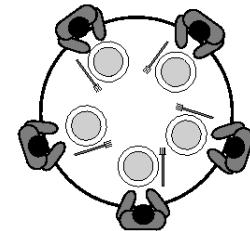
Das berühmte Problem der *Dining Philosophers* wird häufig herangezogen um zu illustrieren, was passieren kann, wenn mehrere Threads um dieselben Ressourcen streiten.

Die Geschichte lautet folgendermaßen:

Fünf Philosophen sitzen um einen runden Tisch. Vor jedem Philosophen steht eine Schüssel mit Reis. Zwischen je zwei Schüsseln liegt ein Stäbchen. Damit ein Philosoph essen kann, benötigt er das Stäbchen links und rechts von seiner Schüssel. Die Philosophen müssen sich so untereinander abstimmen, dass alle abwechselnd essen können.

Überlegungen

- Wie viele Philosophen können maximal gleichzeitig essen?
- Alle Philosophen arbeiten natürlich nach demselben Prinzip, damit man den Code nur einmal programmieren muss. Was kann passieren wenn alle Philosophen zuerst das Stäbchen an ihrer rechten Seite aufnehmen?



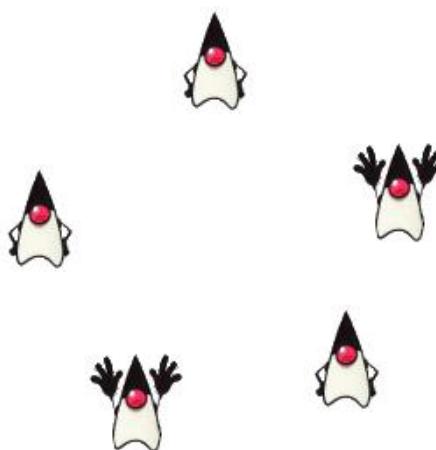
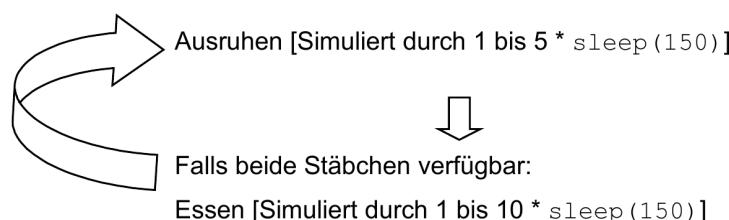
Tipps für die Programmierung

Programmiere eine Klasse **Philosoph**, die sich von der Superklasse **Thread** ableitet. Erzeuge im Frame ein Array mit fünf Philosophen. Das Frame enthält darüber hinaus ein Array mit den fünf Stäbchen (boolesche Werte, die jeweils angeben, ob das betreffende Stäbchen genutzt wird oder nicht) und ein Array mit den Philosophenbildern.

Die Bilder der Philosophen werden in der Frame-Methode **paintComponent()** gezeichnet. Jeder Philosophen-Thread hat eine öffentliche Variable **bildNr**, die die Nummer des aktuellen Philosophen-Bildes angibt. Wenn sich die Nummer ändert, veranlasst der Philosoph ein **repaint()** des Frames.

Arbeitsweise eines Philosophen:

Die Philosophen rasten für eine zufällige Zeitspanne von 1 bis 5 mal 150 Millisekunden. Dann versuchen sie die Stäbchen zu ergreifen. Wenn sie beide Stäbchen bekommen haben, starten sie mit dem Essen, andernfalls rasten sie erneut für eine zufällige Zeitspanne und versuchen anschließend wieder die Stäbchen zu ergattern. Das Essen ist ein längerer Zeitvorgang, der ebenfalls zufällig gesteuert wird. Benutze hierzu **Thread.sleep()** mit zufälligen Wartezeiten von 1 bis 10 mal 150 Millisekunden. Nach dem Essen beginnen die Philosophen wieder von vorne (Zuerst ruhen sie sich von dem anstrengenden Essen aus, dann versuchen sie erneut die Stäbchen zu holen).



27 Dateizugriffe

27.1 Dateien lesen und schreiben

Zum Zugriff auf Dateien bietet Java unterschiedliche Klassen an. Wenn man eine Datei sequentiell auslesen möchte, benutzt man die Klasse `FileInputStream`. Zum sequentiellen Schreiben in eine Datei gibt es die Klasse `FileOutputStream`. Beide Klassen arbeiten nach dem Stream-Konzept. Die Klasse `FileInputStream` liefert sozusagen den Inhalt der Datei als Strom von Charactern, die man der Reihe nach auslesen kann. Man kann dabei weder zurückspringen noch ein Zeichen doppelt auslesen. Die Klasse `FileOutputStream` ermöglicht es analog ein Zeichen nach dem anderen in die Datei zu schreiben.

Wenn man in einer Datei hin und her springen möchte, reicht das Stream-Konzept nicht aus. Dann muss man die Klasse `RandomAccessFile` verwenden.

Damit man mit den oben genannten Klassen arbeiten kann, müssen diese Klassen aus dem Package `java.io` importiert werden. Die Methoden der Dateizugriffsklassen erzeugen bei auftretenden Fehlern (zum Beispiel „Datei existiert nicht“ oder „In die Datei darf nicht geschrieben werden“) eine Exception. Deshalb müssen alle Methoden grundsätzlich mit einem `try-catch`-Block überwacht werden.

27.2 FileInputStream und FileOutputStream InputStreamReader und OutputStreamWriter

Datei öffnen

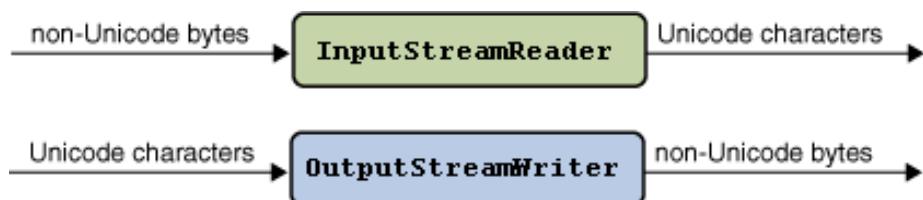
Die Datei öffnet man, in dem man ein Objekt der Klasse `FileInputStream` (lesen) oder `FileOutputStream` (schreiben) erzeugt. Als Parameter gibt man den Namen der Datei an, die geöffnet werden soll. Der Klasse `FileOutputStream` kann man zusätzlich den Parameter `append` übergeben. Wenn `append true` ist, werden die Daten an die Datei angehängt. Andernfalls wird die Datei überschrieben, falls sie vorher schon existiert hat.

```
public FileInputStream(String fileName) throws FileNotFoundException
public FileOutputStream(String fileName) throws IOException
public FileOutputStream(String fileName, boolean append) throws IOException
```

Wenn es um Text-basierte Dateien geht und nicht um Binärdateien (etwa Bilder, oder Musik), dann sollte man die `FileInputStream`- und `FileOutputStream`-Objekte jedoch nicht direkt benutzen. Denn der Inhalt (Text) muss interpretiert werden: Je nach verwendetem Zeichensatz bedeutet eine bestimmte Byte-Folge etwas anderes. Beim Öffnen der Dateien müssen wir also noch dazu sagen, mit welcher Zeichensatzkodierung gearbeitet werden soll.

Zu diesem Zweck wird das `FileInputStream`-Objekt in einem `InputStreamReader` und das `FileOutputStream`-Objekt in einem `OutputStreamWriter` gekapselt:

```
public InputStreamReader(FileInputStream is, String charsetName)
    throws UnsupportedEncodingException
public OutputStreamWriter(FileOutputStream os, String charsetName)
    throws UnsupportedEncodingException
```



Sollen hingegen Binärdaten gelesen und geschrieben werden, dann arbeitet man direkt mit dem `FileInputStream`- beziehungsweise `FileOutputStream`-Objekt (wird bei uns im Unterricht nicht vorkommen).

Einlesen von Daten

Zum Lesen von Daten aus dem Stream besitzt die Klasse `InputStreamReader` unter anderem die folgende Methode:

```
public int read() throws IOException
```

`read()` liest das nächste Zeichen vom Datenstrom ein. Wenn ein Zeichen übertragen wurde, ist das Ergebnis eine Zahl, die mit einem Type-Cast in einen Buchstaben (`char`) umgewandelt werden kann. Falls keine Daten verfügbar sind, blockiert die Methode bis das nächste Zeichen angekommen ist. Wenn die Datenverbindung geschlossen wurde, wird der Wert `-1` zurückgegeben.

Ausgabe von Daten

Zur Ausgabe von Daten besitzt die Klasse `OutputStreamWriter` unter anderem die folgenden Methoden:

```
public void write(int zeichen) throws IOException
public void write(String text) throws IOException
```

Die obere Variante der `write()`-Methode schreibt ein einzelnes Zeichen in den Datenstrom, die untere Variante schreibt einen String. Die mit `write()` geschriebenen Daten werden zunächst (intern) zwischengepuffert und erst dann tatsächlich verschickt, wenn der Puffer gefüllt ist. Mit der Methode `flush()` kann man das tatsächliche Versenden sofort erzwingen:

```
public void flush() throws IOException
```

Datei schließen

Am Ende sollte die Datei mit der Methode `close()` (anzuwenden auf das `FileInputStream`- bzw. auf das `FileOutputStream`-Objekt) geschlossen werden:

```
public void close() throws IOException
```

Das kann man sich allerdings sparen, wenn man das `FileInputStream`- bzw. das `FileOutputStream`-Objekt mit try-with-resource erzeugt (siehe Beispiel-Code weiter unten).

27.3 Dateinamen und -pfade

Um eine Datei mit `FileInputStream` bzw. `FileOutputStream` zu öffnen, muss man ihren Namen angeben. Solange ihr keine Dateien neu erzeugen müsst (sprich: solange die zu öffnende Datei bereits existiert), empfehle ich euch folgende Variante:

```
URL url = getClass().getResource("datei.txt");
InputStream is = new FileInputStream(url.getFile());
InputStreamReader in = new InputStreamReader(is, "UTF-8");
```

bzw.

```
URL url = getClass().getResource("datei.txt");
OutputStream os = new FileOutputStream(url.getFile());
OutputStreamWriter out = new OutputStreamWriter(os, "UTF-8");
```

Das Java-Programm wird dann versuchen die entsprechende Datei im dem Verzeichnis zu öffnen, in dem auch das Java-Programm gestartet wurde. In Eclipse also innerhalb des Packages, in dem das Programm liegt. Wenn es die angegebene Datei nicht gibt, dann wird beim Zugriff über `url.getFile()` eine `NullPointerException` erzeugt.

Alternativ kann man auch zusätzlich zum Dateinamen den Pfad zur Datei mit angeben. Entweder als absoluter Pfad wie etwa C:/Users/hartmut/Documents/Datei.txt¹ oder als relativer Pfad (relativ zum aktuellen Arbeitsverzeichnis), etwa src/dateizugriffe/datei.txt

Beides hat jedoch Nachteile: Absolute Pfade sind plattformabhängig. Was unter Windows ein gültiger Pfad ist macht unter MacOS und Linux/Unix keinen Sinn. Und umgekehrt. Und relative Pfade führen zu unterschiedlichen Ergebnissen, je nachdem, was das aktuelle Arbeitsverzeichnis ist.

Oder man lässt den Benutzer die Datei mit einem Dateiauswahl dialog interaktiv bestimmen (Siehe die Zusatz-aufgabe zu Aufgabe 1 in diesem Kapitel).

27.4 Dateien aus JAR Archiven zum Lesen öffnen

Wenn man Dateien in ein JAR mit einpackt und diese lesen will (braucht ihr im Unterricht nicht), empfiehlt sich folgende Variante:

```
InputStream is = getClass().getResourceAsStream("notizen.txt");
InputStreamReader fileIn = new InputStreamReader(is, "UTF-8");
```

Die im Beispiel benutzte Datei notizen.txt würde gefunden, wenn sie im gleichen Verzeichnis liegt, wie die aufrufende Klasse. Liegt sie in einem Unterverzeichnis (relativ zum Verzeichnis, in dem die Klasse liegt) dann würde man sie entsprechend mit unterverzeichnis/notizen.txt erreichen.

In JAR-Archive kann man jedoch nicht schreiben. Für schreibende Dateizugriffe (im normalen Dateisystem, nicht in JAR-Archiven) benutzt man deswegen die oben genannten Methoden.

27.5 Beispiel: Schreiben in eine Datei

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.net.URL;

public class Schreiben {

    public Schreiben() {
        URL url = getClass().getResource("hallo.txt");

        if (url == null) {
            System.out.println("Fehler beim Schreiben: Datei existiert nicht.");
            return;
        }
        try (OutputStream os = new FileOutputStream(url.getFile());
             OutputStreamWriter out = new OutputStreamWriter(os, "UTF-8")) {
            out.write("Heißer Kaffee!" + System.lineSeparator());
            out.flush();
        } catch (IOException e) {
            System.out.println("Fehler beim Schreiben:" + e.getMessage());
        }
    }

    public static void main(String[] args) {
        new Schreiben();
    }
}
```

¹Der unter Windows übliche Backslash '\' als Pfad-Trenner muss durch einen Slash '/' ersetzt werden.

27.6 Beispiel: Lesen aus einer Datei

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;

public class Lesen {

    public Lesen() {
        URL url = getClass().getResource("hallo.txt");
        int zeichen;

        if (url == null) {
            System.out.println("Fehler beim Lesen: Datei existiert nicht.");
            return;
        }
        try (InputStream is = new FileInputStream(url.getFile());
             InputStreamReader in = new InputStreamReader(is, "UTF-8")) {
            while ((zeichen = in.read()) != -1) {
                System.out.print((char) zeichen);
            }
        } catch (IOException e) {
            System.out.println("Fehler beim Lesen: " + e.getMessage());
        }
    }

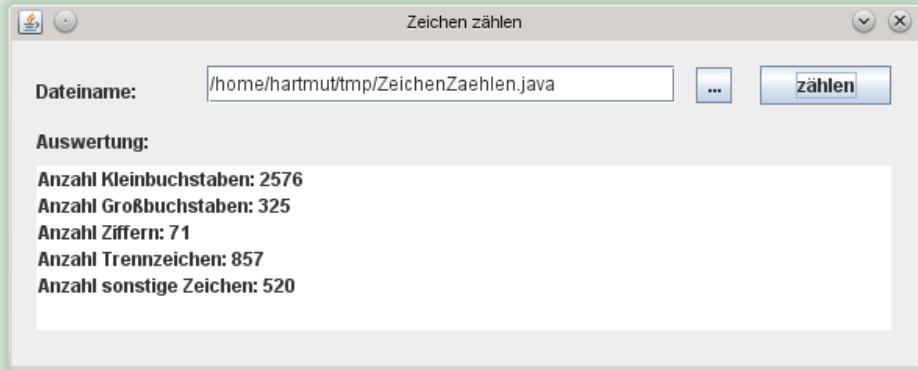
    public static void main(String[] args) {
        new Lesen();
    }
}
```

27.7 Dateizugriffe – Übungen

Aufgabe 1: Zeichen zählen

Schreibe ein Programm, das zählt, wie viele Kleinbuchstaben, Großbuchstaben, Ziffern, Trennzeichen und sonstige Zeichen in einer Datei stehen. Das Ergebnis soll in einer **JList**-Komponente ausgegeben werden.

Beachte, dass die **JList**-Komponente vor der Auswertung einer Datei gelöscht werden muss, um eventuell vorhandene alte Inhalte zu entfernen. Gib in einer Messagebox eine Fehlermeldung aus, falls die angegebene Datei nicht existiert.



Verwende zum Zählen der verschiedenen Arten von Zeichen die folgenden Methoden der Klasse **Character** (siehe Online-Hilfe):

| Methode | Erläuterung |
|---|---|
| <code>static boolean isDigit(char ch)</code> | Prüft, ob der angegebene Buchstabe eine Ziffer (0 bis 9) ist. |
| <code>static boolean isLowerCase(char ch)</code> | Prüft, ob das angegebene Zeichen ein Kleinbuchstabe ist. |
| <code>static boolean isUpperCase(char ch)</code> | Prüft, ob das angegebene Zeichen ein Großbuchstabe ist. |
| <code>static boolean isWhitespace(char ch)</code> | Prüft, ob der angegebene Buchstabe ein Trennzeichen ist (Leerzeichen, Tabulator, Zeilenumbruch, usw.) |

Zusatzaufgabe

Zum Laden einer Datei bietet Java die Klasse **JFileChooser** an. Wenn man ein Objekt dieser Klasse erstellt, erscheint der Standarddialog zum Suchen nach einer Datei. Informiere dich in der Online-Hilfe darüber, wie man die Klasse **JFileChooser** benutzt, und baue sie in dein Programm ein. Rechts neben dem Textfeld für den Dateinamen sollte ein Button mit der Aufschrift „...“ eingefügt werden. Wenn der Benutzer auf den Button klickt, erscheint der Dialog zum Suchen nach einer Datei. Der Dateiname wird nach dem Schließen des Dialogs in das Textfeld eingetragen.

Aufgabe 2: Einfache Verschlüsselung mit dem Caesar-Verfahren

Die sogenannte Caesar-Verschlüsselung ist schon über 2000 Jahre alt wurde von Julius Caesar 50 Jahre vor Christus benutzt. Das Alphabet wird dabei einfach um mehrere Buchstaben verschoben. Zum Beispiel um drei Buchstaben:

ABCDEFGHIJKLMNPQRSTUVWXYZ

DEFGHIJKLMNOPQRSTUVWXYZABC

Damit wird aus dem Klartext hallo der Geheimtext KD00R.

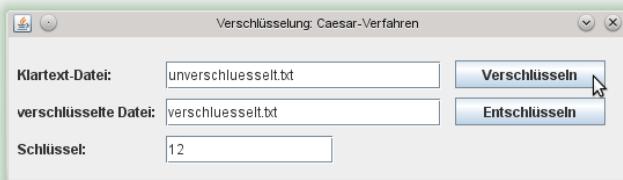
- a) Erstelle ein Programm, das den Inhalt einer Datei einliest und mit dem Caesar-Verfahren verschlüsselt in eine zweite Datei schreibt. Verschlüsselt werden sollen nur Buchstaben von a bis z. Alle anderen Zeichen sollen unverändert bleiben. Wandle zu Beginn alle Buchstaben in der Datei mit der Methode `Character.toLowerCase()` in Kleinbuchstaben um (siehe Online-Hilfe).

Anmerkung:

Mit Variablen vom Typ `char` kann man wie mit Zahlen rechnen.

Beispiel:

```
char c = 'b';
if (c >= 'a' && c <= 'z') {
    c = (char) (c + 3);
}
int differenz = c - 'a';
```



- b) Erweitere das Programm so, dass der Inhalt einer verschlüsselten Datei auch wieder entschlüsseln werden kann. Der Inhalt der Eingabedatei soll wie im ersten Aufgabenteil in eine Ausgabedatei geschrieben werden, nur diesmal nicht verschlüsselt sondern entschlüsselt.

Aufgabe 3: Prüfsumme

Um Sicherzustellen, dass der Inhalt einer Datei nicht manipuliert wurde, berechnet man häufig eine sogenannte **Prüfsumme**. Die Prüfsumme ist eine Zahl, die entsteht, wenn man alle Bytes der Datei miteinander verknüpft. Das einfachste Verfahren zur Berechnung einer Prüfsumme funktioniert folgendermaßen:

Es wird ein Byte für die Prüfsumme angelegt, das zu Beginn den Wert Null enthält. Alle Bytes, die in der Datei enthalten sind, werden der Reihe nach mit dem Prüf-Byte mit einer Exklusiv-ODER Verknüpfung verbunden. Java-Beispiel:

```
checksum = (byte) (checksum ^ b1);
```

Die Exklusiv-ODER Verknüpfung verknüpft die einander entsprechenden Bits der beiden Bytes. Das Ergebnis-Bit ist immer dann wahr (1), wenn beide Bits ungleich sind, und falsch (0), wenn beide Bits gleich sind.

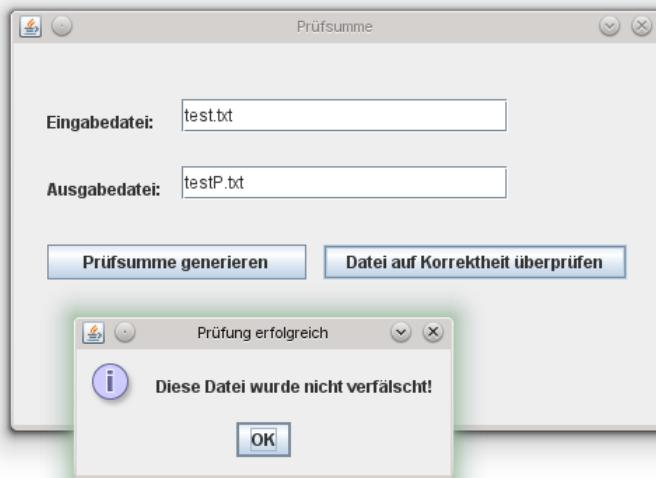
Es soll nach dem oben beschriebenen Verfahren für eine Datei eine Prüfsumme generiert werden, damit überprüft werden kann, ob der Datei-Inhalt manipuliert wurde.

- a) Wenn man auf den linken Button drückt, soll für eine Datei eine Prüfsumme generiert werden. Es wird aus der Eingabedatei eine /home/hartmut/tmpneue Datei generiert, die um die Prüfsumme erweitert ist. Das Ergebnis wird unter dem Namen der Ausgabedatei abgespeichert. Falls die Eingabedatei nicht existiert wird in einer Messagebox eine Fehlermeldung ausgegeben.

In die Ausgabedatei wird als erstes Zeichen ein \$ geschrieben, um zu markieren, dass diese Datei eine Prüfsumme enthält. Dann folgt der eigentliche Inhalt der Datei, und als letztes Byte wird die generierte Prüfsumme angehängt.

- b) Wenn man auf den rechten Button drückt, wird geprüft, ob die Prüfsumme der Eingabedatei korrekt ist oder ob der Inhalt der Datei verändert wurde. Falls die Datei am Anfang kein \$ enthält, besitzt sie keine Prüfsumme, und es wird in einer Messagebox eine Fehlermeldung ausgegeben. Es wird ebenso eine Fehlermeldung ausgegeben, falls die Eingabedatei nicht existiert.

Wenn eine Prüfsumme vorhanden ist, wird die Prüfsumme über den Inhalt der Datei (ohne das voran gehängte \$ und das Prüfbit am Ende) erneut berechnet und mit dem letzten Byte verglichen. Sind beide Werte gleich, wird dem Benutzer in einer Messagebox mitgeteilt, dass der Datei-Inhalt nicht verfälscht wurde. Falls die Werte ungleich sind, erhält der Benutzer die Mitteilung, dass die Datei manipuliert wurde.



Du kannst es mit einer beliebigen Datei selbst austesten: Dieses einfache Prüfsummenverfahren funktioniert! Sobald man in der Datei ein oder mehrere Zeichen verändert, kommt mit hoher Sicherheit eine andere Prüfsumme heraus.

Aufgabe 4: Notizbuch

1. Erstelle die folgende Programmoberfläche:



Die obere Komponente mit der Beschriftung „Inhalt:“ soll eine `JList` sein (keine `JTextArea`).

2. Der Benutzer kann in das `JTextField` neue Einträge für das Notizbuch eingeben. Wenn er auf den „speichern“-Button drückt, liest das Programm den Text aus dem Textfeld aus und hängt ein \$ an das String-Ende an. Anschließend öffnet es eine Datei mit dem Namen `notizen.txt` zum Schreiben und schreibt den neuen Eintrag in die Datei. Falls die Datei bereits existiert, soll der neue Text an den vorhandenen Text angehängt werden. Dies geschieht automatisch, wenn du beim Erzeugen des `FileOutputStream`-Objektes den Wert `true` als zweiten Parameter übergst.

Beispiel:

```
FileOutputStream fileOut = new FileOutputStream("notizen.txt", true);
```

3. Nach dem Abspeichern eines neuen Eintrags und ebenso beim Neustart der Datei, soll der aktuelle Inhalt der Datei notizen.txt in die JList-Komponente geschrieben werden. Lösche dazu zunächst den Inhalt der Listkomponente (für den Fall, dass sich noch alte Daten in der Komponente befinden) und lies dann den Inhalt der Datei ein. Ein \$ markiert jeweils das Ende einer Zeile, das heißt wenn du ein \$-Zeichen liest, solltest du alle seit dem letzten \$-Zeichen eingelesenen Buchstaben mit addElement() in das DefaultListModel-Objekt der JList-Komponente einfügen.
4. Wenn man auf den Button „Notizbuch löschen“ klickt, soll der Inhalt der Datei gelöscht werden. Anschließend wird die JList-Komponente aktualisiert. Die Datei kannst du sehr einfach löschen, in dem du ein neues FileOutputStream-Objekt anlegst, und es wieder schließt ohne etwas hinein zu schreiben. Beachte, dass dieses Mal beim Anlegen des Objektes als zweiter Parameter der Wert false übergeben werden muss (dies bedeutet, dass der neue Inhalt nicht an den vorhandenen Inhalt angehängt wird).
5. Wenn der Benutzer auf den Button „markierte Zeile löschen“ klickt, soll die markierte Zeile aus der JList-Komponente gelöscht werden. Anschließend wird der verbleibende Inhalt der List-Komponente in die Datei notizen.txt geschrieben. Zum Löschen des markierten Eintrags aus der JList-Komponente liest du zunächst mit getSelectedIndex() (eine Methode des DefaultListModel-Objektes) den markierten Eintrag aus. Falls keine Zeile markiert ist, wird mit einer Fehlermeldung abgebrochen. Andernfalls rufst du anschließend die folgende Methode der List-Komponente auf:

```
public void remove(int index)
```

Dies ist ebenfalls eine Methode des DefaultListModel-Objektes. Danach ist die List-Komponente verändert und du musst den Inhalt der JList-Komponente zeilenweise auslesen und in die Datei schreiben. Beachte, dass du beim Anlegen des FileOutputStream-Objektes wie bei Teilaufgabe d) den Wert false als zweiten Parameter übergeben musst, damit die vorhandene Datei überschrieben wird. Die Methode getSize() des DefaultListModel-Objektes liefert die Anzahl der Zeilen zurück. Gehe alle Zeilen in einer Schleife durch. Mit der folgenden Methode kannst du dir den Text in einer Zeile mit einer bestimmten Nummer holen (die Zeilennummerierung beginnt mit 0):

```
public String get(int index)
```

Beachte, dass du vor dem Schreiben einer Zeile in die Datei noch ein \$ an den Text anhängen musst.

27.8 Konfigurations-Dateien (`java.util.Properties`)

Mit der Klasse `Properties` aus dem Package `java.util` kann man in einer Datei Einträge unter einem Stichwort abspeichern. Zum Beispiel könnten die Punktzahl und der Name des Highscore-Inhabers in einer Datei folgendermaßen gespeichert werden:

```
#Sun Jun 06 18:01:53 CEST 2010
Name=Daniela
Punkte=3015
```

Die Klasse `Properties` übernimmt die Verwaltung der Eigenschaften (im Beispiel heißen die Eigenschaften `Name` und `Punkte`).

Zum Abspeichern der Einträge in einer Datei benötigt man außerdem ein Objekt der Klasse `FileOutputStream`, das die Ausgabedatei erzeugt² und das eigentliche Schreiben der Daten in die Datei übernimmt. Falls die Ausgabedatei schon existiert, wird sie überschrieben.

Zum Einlesen der Einträge aus einer Datei, benötigt man zusätzlich ein Objekt der Klasse `FileInputStream`, das die Datei zum Lesen öffnet und die Daten einliest und sie anschließend zur Interpretation an die Klasse `Properties` weiterleitet.

a) Highscore in Datei speichern

Der folgende Code-Auszug liest den Namen und die Punktzahl des Besten aus zwei Textfeldern aus und speichert sie in der Datei `highscore.properties`³ ab:

```
public void laden() {
    URL url = getClass().getResource("highscore.properties");
    if (url == null) {
        System.out.println("Fehler beim Lesen: Die Datei existiert nicht!");
        return;
    }
    try (InputStream is = new FileInputStream(url.getFile());
         InputStreamReader in = new InputStreamReader(is, "UTF-8")) {
        Properties properties = new Properties();
        properties.load(in);
        String name = properties.getProperty("Name");
        String punkte = properties.getProperty("Punkte");
        tfName.setText(name);
        tfPunkte.setText(punkte);
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
}
```

b) Highscore aus Datei einlesen

Der folgende Code-Auszug liest den Namen und die Punktzahl aus der der Datei `highscore.properties` aus und schreibt sie in zwei Textfelder:

```
public void speichern() {
    URL url = getClass().getResource("highscore.properties");
    if (url == null) {
        System.out.println("Fehler beim Schreiben: Die Datei existiert nicht!");
        return;
    }
    try (OutputStream os = new FileOutputStream(url.getFile());
         OutputStreamWriter out = new OutputStreamWriter(os, "UTF-8")) {
        Properties properties = new Properties();
```

²In der von uns verwendeten Form muss die Datei bereits vorab existieren – soll es auch mit noch nicht existierenden Dateien funktionieren, darf man die Datei nicht mit `getClass().getResource()` bestimmen.

³Die Dateiendung `*.properties` ist nicht zwingend, aber üblich. Grundsätzlich darf die Datei beliebig benannt werden.

```
String name = tfName.getText();
String punkte = tfPunkte.getText();
properties.setProperty("Name", name);
properties.setProperty("Punkte", punkte);
properties.store(out, null);
out.flush();
} catch (Exception e) {
    System.out.println(e.getMessage());
}
}
```

28 UML Wiederholung

28.1 UML Zustandsdiagramme

Aufgabe 1: Seehund

In einem Computerspiel soll ein Seehund dargestellt werden. Zeichne dafür ein UML-Zustandsdiagramm, das die verschiedenen Zustände veranschaulicht, in denen sich der Seehund des Computerprogramms befinden kann:

Zu Beginn schwimmt der Seehund ziellos im Wasser herum. Sobald ein Fisch erscheint, verfolgt er diesen. Nachdem er den Fisch gefangen hat, isst er ihn (längerer Vorgang). Falls dies der dritte Fisch war, den er gefangen hat, ist er satt und ruht sich auf einer Sandbank aus. Die Seehund-Simulation ist damit zu Ende. Andernfalls schwimmt er weiter, um erneut einen Fisch zu fangen.

Aufgabe 2: Sprechende Puppe

Ein Spielzeughersteller entwickelt eine neue sprechende Puppe, die elektronisch gesteuert werden soll. Du hast die Aufgabe, das Verhalten der Puppe mit einem UML-Zustandsdiagramm zu beschreiben:

Wenn die Puppe hingelegt wird, sagt sie „Gute Nacht“. Wenn sie aufgerichtet wird, gibt sie den Satz „Spielst du was Schönes mit mir?“ von sich. Solange sie aufgerichtet ist, kann man der Puppe weitere Sätze entlocken, indem man sie an den Händen fasst. Wenn man an ihre rechte Hand fasst, sagt sie „Guten Tag“. Wenn man an die linke Hand fasst, sagt sie „Mama, ich hab dich lieb.“. Außerdem singt die Puppe ein Lied, wenn man auf ihren Bauchnabel drückt. Dies funktioniert immer, egal ob sie sich in aufgerichteter oder liegender Position befindet. Das Lied endet nach einer Zeitspanne von zwei Minuten.

28.2 UML Klassendiagramme

Aufgabe 1: Zelten

Beschreibe mit einem UML-Klassendiagramm die Beziehung zwischen den Klassen „Mensch“, „Zelt“, „Schlafsack“, und „Zeltstange“. Gib den Beziehungen einen geeigneten Beziehungsnamen und trage die Multiplizität ein. Attribute und Methoden der Klassen müssen nicht beschrieben werden.

Aufgabe 2: Gitarren

Beschreibe mit einem UML-Klassendiagramm die Beziehung zwischen den Klassen „Gitarre“, „Mensch“, „Saite“, „Westerngitarre“ und „Klassische Gitarre“. Gib den Beziehungen einen geeigneten Beziehungsnamen und trage die Multiplizität ein. Attribute und Methoden der Klassen müssen nicht beschrieben werden.

Aufgabe 3: Landleben

Es soll ein Computerspiel für Kinder entwickelt werden, dass das Landleben mit mehreren Bauernhöfen simuliert. Schreibe dazu ein UML-Klassendiagramm:

Zu jedem Bauernhof gehören ein Haupthaus, in dem zwei bis sechs Menschen wohnen, und eine oder mehrere Scheunen, in dem die Tiere untergebracht sind. Von jedem Gebäude (egal ob Haupthaus oder Scheune) müssen die Maße (Breite und Länge) abgespeichert werden. Die Menschen unterscheiden sich durch die Eigenschaften Größe, Haarfarbe und Geschlecht.

In einer Scheune leben jeweils fünf bis zehn Tiere. Es gibt Kühe, Pferde und Schweine. Alle Tiere können laufen, stehen oder liegen. Kühe können darüber hinaus wiederkauen. Schweine können grunzen. In einer Scheune stehen in der Regel eine Reihe von Futtertrögen. Von jedem Futtertrog muss man das Fassungsvermögen kennen.

29 Softwaretechnik

Softwaretechnik (englischer Fachbegriff: *Software Engineering*) beschreibt den Prozess der Software-Erstellung. Das beinhaltet neben der Programmierung, der Dokumentation und den Tests auch die Planungs- und Wartungsphase.

Ohne ein ausgewachsenes Projektmanagement ist dies nicht erfolgreich zu leisten.

29.1 Die Entwicklung eines Video-Überwachungssystems oder warum die Firma LogoSoft Pleite ging

Was vorher geschah

Anmerkung: Alle Namen und Daten in der folgenden Geschichte sind frei erfunden. Die Geschichte selbst hat sich so ähnlich aber wirklich zugetragen! Die Autorin hat nach ihren Erlebnissen beschlossen den Job als Softwareentwicklerin aufzugeben, um in der Schule für die Ausbildung fähigerer Generationen zu sorgen.

September 2000: Die Firma LogoSoft ist ein kleines Software-Unternehmen, das seit ca. 10 Jahren vergeblich versucht, Geld durch den Verkauf von Videokonferenzsystemen zu verdienen. Die Videokonferenzsysteme wurden in der Firma ohne festen Auftrag entwickelt, in der Hoffnung, dass sie bei anderen Unternehmen reißenden Absatz finden. Die hergestellte Software ist qualitativ gut, aber der jährliche Umsatz von ca. 30 Systemen zu je 1500 € reicht nicht einmal aus, um die Gehälter der ca. 40 Mitarbeiter in einem einzigen Monat zu bezahlen. Glücklicherweise wird die Firma von einer Muttergesellschaft aus England finanziert, die seit Jahren auf den großen Durchbruch hofft. Allerdings dämmert es den Geschäftsführern in England inzwischen, dass man mit Videokonferenzsystemen wohl doch nicht das große Geld machen kann. Deshalb hat der Geschäftsführer der Firma LogoSoft die Anweisung erhalten, in Zukunft keine eigenen Produkte mehr zu entwickeln. Statt dessen sollen nur noch Produkte hergestellt werden, die von einem Kunden, d.h. einer anderen Firma, in Auftrag gegeben werden. Die Kosten werden in diesem Fall direkt von der auftraggebenden Firma bezahlt.

Der Vertriebsleiter der Firma LogoSoft hat Kontakt zu der Firma BlueEye, die glaubt, dass es eine Marktlücke für preiswerte Video-Überwachungssysteme gibt. Sie wollen einen PC-basierten Videorekorder entwickeln lassen, der in der Lage ist, die Aufnahme von 32 im Haus verteilten Kameras aufzuzeichnen. Die aufgenommenen Bilder sollen auf der Festplatte des PCs abgespeichert werden.

Der Vertrag wird abgeschlossen

Oktober 2000: Der Vertriebsleiter, der Geschäftsführer und der Abteilungsleiter-Technik verhandeln mit der Firma BlueEye über die Erstellung des gewünschten Video-Überwachungssystems. Die Anforderungen an das System werden schriftlich festgelegt. Es wird abgesprochen, dass die 32 Kameras über eine spezielle Hardware-Karte an den PC angeschlossen werden, die von der Schwesterfirma HardSoft entwickelt wird. Die Firma LogoSoft übernimmt die Programmierung der Software auf der Hardware-Karte und der Software, die auf dem PC läuft. Damit das System so preiswert wie möglich wird, werden billige Hardware-Komponenten verwendet, die zum aktuellen Zeitpunkt eigentlich schon veraltet sind. Die Tatsache, dass das System durch die technische Weiterentwicklung in zwei Jahren keine befriedigende Leistung mehr bringen wird, wird ignoriert. Hauptsache Kosten sparen! Daher wird auch beschlossen, dass auf dem PC das Betriebssystem Linux eingesetzt wird, denn dies ist ein preiswertes Betriebssystem, das im Moment „in“ ist. Das Betriebssystem Linux ist allerdings nicht für zeitkritische Anwendungen entwickelt worden. Das Video-Überwachungssystem muss Monate lang ohne Wartung Bilder aufzeichnen können. Dabei darf kein einziges Bild verloren gehen. Es wäre untragbar, wenn gerade das Bild, auf dem vielleicht der Einbrecher zu sehen ist, wegen Systemfehlern nicht abgespeichert werden würde. Ob das Betriebssystem Linux und die verwendeten Hardware-Komponenten in der Lage sind, diese Anforderungen zu erfüllen, wird nicht näher untersucht.

Nachdem die technischen Details geklärt sind, erstellt ein erfahrener Software-Entwickler, ein sogenannter *Senior Developer*, einen Grobplan für die Entwicklung der Software. Er überlegt, aus welchen Komponenten sich das System zusammen setzen muss. Dann schätzt er die dafür benötigte Zeit ab. Er glaubt, das das System von vier gut eingearbeiteten Mitarbeitern in einem halben Jahr erstellt werden kann. Nach dieser Grob-Kalkulation errechnet die Vertriebsabteilung einen Festpreis, zu dem die Software an die Firma BlueEye verkauft wird. Der

Preis, die Anforderungen an das System und der Termin, bis zu dem die Software fertig sein muss, werden in einem Vertrag schriftlich festgehalten. Die Firma LogoSoft verspricht, die Software bis zum Mai 2001 zu liefern. Im Sommer 2001 will die Firma BlueEye bereits die ersten Video-Überwachungssysteme verkaufen.

Das Projekt beginnt

November 2000: Die Arbeit an dem Video-Überwachungssystem in der Firma LogoSoft beginnt. Da die meisten Mitarbeiter in laufenden Projekten eingebunden sind (die alle nur wenig Geld einbringen), hat der Abteilungsleiter-Technik Probleme, die vierköpfige Software-Entwicklungs-Mannschaft zusammen zu stellen. Er ernennt Kai Menzel zum Projektleiter. Kai Menzel ist ein langjähriger, erfahrener Mitarbeiter, der jedoch noch nie ein eigenes Projekt geleitet hat. Als einzigen Vollzeit-Mitarbeiter erhält Kai einen neuen Kollegen der frisch von der Universität kommt, und noch niemals ein Programm entwickelt hat, das länger als fünf DIN A4 Seiten ist. Die weiteren Mitarbeiter des neuen Projektes sind zwei Studenten (mit noch geringerer Programmiererfahrung), die halbtags in der Firma jobben. Alle drei verdienen noch nicht viel und sind für die Firma LogoSoft preiswerte Arbeitskräfte. Keiner der vier Personen hat jemals zuvor Software auf dem Betriebssystem Linux erstellt.

Da man bei großen Projekten nicht einfach so drauf los programmieren kann, erstellt Kai Menzel als erstes eine gründliche Projektplanung. Er vertraut dabei voll auf die zuvor von dem Senior Developer erstellte Grob-Planung. Er definiert genau, welche Aufgaben die einzelnen Software-Komponenten erfüllen müssen. Die Vorgaben werden grafisch und verbal festgehalten. Diese Planung dauert mehrere Wochen. Anschließend entscheidet Kai, welcher Mitarbeiter welche Komponenten erstellt und stellt dafür einen genauen Zeitplan auf.

Ablauf des ersten halben Jahres

Dezember 2000: Die drei Mitarbeiter von Kai Menzel haben ihre Teilaufgaben zugewiesen bekommen und können mit ihrer Arbeit beginnen. Jeder der Mitarbeiter studiert genau die von Kai festgesetzten Anforderungen und macht sich dann einen detaillierten Plan, wie er die ihm zugewiesene Komponente programmieren will. Nach zwei bis drei Wochen Einarbeitung und Planung sind alle Mitarbeiter so weit, dass sie mit der Programmierung loslegen können. Wenn jetzt noch die benötigte Hardware da wäre ...

Februar 2001: Kai Menzel und seine Mitarbeiter haben große Probleme mit den von der Firma HardSoft gelieferten Prototypen für die Hardware-Karte. Es wird mehrfach ein neuer Prototyp geliefert und immer wieder stellen sie fest, dass die Karte fehlerhaft ist. Bei den Hardware-Tests geht eine Menge Zeit verloren. Außerdem stellt sich heraus, dass die Programmierung durch die billigen Hardware-Komponenten zeitaufwendiger ist als angenommen. Der Zeitplan ist nicht mehr einzuhalten.

April 2001: Das Projekt hinkt dem ursprünglichen Zeitplan hoffnungslos hinterher. Um noch zu retten was zu retten ist, beschließt der Abteilungsleiter-Technik, zwei erfahrene Softwareentwickler von ihren laufenden Projekten abzuziehen und sie in das Projekt „Video-Überwachungssystem“ zu stecken. Die beiden neuen Projekt-Mitarbeiter, Marco Zwickel und Eva Kaufmann, beginnen sich einzuarbeiten.

Das Projekt muss verlängert werden

Mai 2001: Die beiden neuen Projekt-Mitarbeiter, Marco und Eva, sind jetzt im Bilde und sind vom Zustand des Projektes entsetzt. Marco soll mit dem Frischling von der Uni zusammen die Software für die Hardware-Karte fertig stellen. Er stellt fest, dass der Neuling aus Unerfahrenheit viele Fehler eingebaut hat. Ein großer Teil seiner Komponenten muss noch einmal überarbeitet werden. Eva hat die Aufgabe, die PC-Software zu schreiben, um die sich bisher überhaupt noch niemand gekümmert hat. Sie stellt fest, dass die veranschlagte Zeitschätzung für die Komponente unrealistisch niedrig ist. Statt wie vorgegeben einen Monat wird sie mindestens drei Monate zur Erstellung der Software benötigen. Außerdem stellen Marco und Eva fest, dass bei der Planung viele wichtige Überlegungen unterlassen wurden. Beide haben große Bedenken, ob das System, so wie es geplant ist, stabil laufen wird.

Juni 2001: Auf dem Betriebssystem Linux treten unerwartete Probleme auf. Hätte man das System vielleicht doch vorher auf Tauglichkeit untersuchen sollen? Ein einzelner Mitarbeiter wird ca. drei Monate brauchen, um diese Probleme zu beseitigen. Der Abteilungsleiter-Technik sieht sich gezwungen, einen weiteren Softwareentwickler

für das Projekt „Video-Überwachung“ abzustellen. Fast die gesamte Softwareentwicklungsabteilung ist jetzt in das Projekt eingebunden.

Der Geldhahn wird zugedreht

Juli 2001: Die Muttergesellschaft in England hat keine Geduld mehr mit der Firma LogoSoft. Einer der Gründe dafür ist sicher auch der katastrophalen Ablauf des Projekts Video-Überwachung. Es wird beschlossen, dass 15 der 40 Mitarbeitern entlassen werden müssen. Da die Firma auf keinen einzigen Software-Entwickler verzichten kann, werden in der Abteilung „Technik“ fünf „einfache“ Mitarbeiter entlassen, die kein Universitätsdiplom besitzen. Die entlassenen Techniker wurden bisher für die Wartung der Geräte, die Kundenbetreuung und den Test von fertigen Software-Komponenten eingesetzt. Für dieses Aufgaben steht jetzt niemand mehr zur Verfügung. Die Mitarbeiter sind von der Firmenpolitik irritiert.

Technische Mängel

August 2001: Das geplante Software-System ist fertig gestellt. Aber es funktioniert nicht. Es stellt sich heraus, dass das System zu langsam ist. Es ist nicht in der Lage die Bilder der 32 Kameras in der erforderlichen Geschwindigkeit aufzuzeichnen. In großen Software-Firmen ist es eine Selbstverständlichkeit, dass zu Beginn der Entwicklung mit einem abgespeckten Prototyp ausgetestet wird, ob alle Vorstellungen technisch umsetzbar sind. Die Firma LogoSoft hat jedoch um Kosten zu sparen auf diesen Schritt verzichtet.

Die Projektleiter und der Senior Developer untersuchen hektisch, wie das System so abgeändert werden kann, dass die vorgegebenen Anforderungen doch noch erfüllt werden können. Schließlich finden sie eine Lösung. Einige Komponenten auf der Hardware-Karte müssen verändert werden, und die Software muss entsprechend umgeschrieben werden. Die Erweiterung wird mehrere Monate Zeit kosten.

September 2001: Die in der Firma verbliebenen Mitarbeiter stellen fest, dass die Gehälter mit Verspätung ausgezahlt werden.

Die Test-Phase

November 2001: Die Software des Video-Überwachungssystems ist fertig gestellt. Nun beginnt eine mehrere Monate lange Testphase. Nachdem die „einfachen“ Techniker von der Firma entlassen wurden, müssen die Tests von den Software-Entwicklern selbst durchgeführt werden. Da die Software-Entwickler ein höheres Gehalt kriegen, wird die Testphase für die Firma ungewöhnlich teuer.

Bei den Langzeit-Tests, die jetzt durchgeführt werden, kommen naturgemäß noch eine Reihe Software-Fehler zutage, die von den Software-Entwicklern behoben werden müssen. Nach einer Fehlerbehebung müssen immer alle Tests noch einmal durchgeführt werden, und häufig kommen dann weitere Fehler zutage. Es gibt mehrere Zyklen mit einer Testphase und einer anschließenden Fehlerbehebung bis die Software endlich „rund“ ist.

Während dessen gehen jede Woche neue Kündigungen von Mitarbeitern ein, die sich wegen der schlechten finanziellen Lage der Firma nach einem neuen Job umgesehen haben. Auch aus der Führungsetage kündigen viele Mitarbeiter.

Das Ende

März 2002: Das Video-Überwachungssystem ist endlich fertig gestellt.

Die Muttergesellschaft in England beschließt, die Firma LogoSoft nicht mehr finanziell zu unterstützen. Alle Mitarbeiter werden zum nächstmöglichen Zeitpunkt entlassen.

Ob die Firma BlueEye es schafft, ihr Video-Überwachungssystem erfolgreich zu verkaufen oder ob sie ebenfalls Pleite geht, bleibt noch abzuwarten.

29.2 Kontrollfragen zum Text

Fragen zum Abschnitt „Was vorher geschah“

- a) Erkläre den Unterschied zwischen eigenen Produkten einer Firma im Gegensatz zu einer Auftragsarbeit.
- b) Überlege welche Vor- und Nachteile die Entwicklung eigener Produkte für Firma besitzt.
- c) Beschreibe mögliche Einsatzgebiete für das beschriebene Video-Überwachungssystem.

Fragen zum Abschnitt „Der Vertrag wird abgeschlossen“

- a) Welche Positionen haben die Mitarbeiter, die an den Vertragsverhandlungen beteiligt sind, in der Firma LogoSoft? Welche Aufgaben kommen ihnen während der Verhandlung wahrscheinlich zu?
- b) Welche Maßnahmen werden getroffen, um die Kosten des Projekts möglichst gering zu halten? Welche Risiken bringen diese Maßnahmen mit sich? Würdest du als Projektleiter oder Geschäftsleiter diese Risiken in Kauf nehmen?
- c) Das noch nicht entwickelte Software-System wird an die Firma BlueEye zu einem festen Preis verkauft. Welches Risiko birgt ein Festpreis für die Firma LogoSoft? Was könnten die Gründe dafür sein, dass sich die Geschäftsleitung trotz des Risikos zu einem Festpreis bereit erklärt?

Fragen zum Abschnitt „Das Projekt beginnt“

- a) Liste die Stärken und Schwächen des Projektteams „Video-Überwachung“ auf. Erfüllt das Team die Bedingungen, von denen der Senior Developer bei seiner Grob-Planung ausgegangen ist (siehe Abschnitt „Der Vertrag wird abgeschlossen“)?
- b) Wer trägt die Verantwortung für die schlechte Zusammensetzung des Teams?
- c) Beschreibe, welche Teilaufgaben in der Planungsphase eines Projekts erledigt werden müssen. Wer führt diese Aufgaben aus?

Fragen zum Abschnitt „Ablauf des ersten halben Jahres“

- a) Welche Vorbereitungen sind nötig, ehe ein Softwareentwickler mit der Programmierung beginnen kann? Wie viel Zeit benötigen die Entwickler im Team „Video-Überwachung“ für die Vorbereitung?
- b) Aus welchen Gründen verzögert sich der Ablauf des Projektes? Fallen dir außer den beiden im Abschnitt genannten Gründen noch weitere Gründe ein, die vermutlich zu der Verzögerung beigetragen haben?
- c) Das Projekt wird einen Monat vor dem geplanten Projektende mit zwei zusätzlichen Softwareentwicklern verstärkt. Was sagt dieser späte Zeitpunkt über die Projektorganisation aus? Welche Maßnahmen hätten helfen können, die Unhaltbarkeit des Liefertermins früher zu entdecken?

Fragen zum Abschnitt „Das Projekt muss verlängert werden“

- a) Dem Neuzugang von der Universität wurde sofort eine sehr anspruchsvolle Aufgabe übertragen, der er nicht gewachsen war. Was sagt dies über die sogenannte „Praxisnähe“ einer Universitätsausbildung aus? Glaubst du, dass die Aufgabe zumutbar war? Hätte ein anderer, durchschnittlicher Universitätsabsolvent die Aufgabe vermutlich besser gelöst?
- b) Eva stellt fest, dass die Zeitschätzung für ihre Komponente um ein Drittel zu niedrig war. Eine solche Fehleinschätzung ist bei kleinen Unternehmen an der Tagesordnung. Welche finanziellen Konsequenzen kann eine solche Fehleinschätzung für die Firmen haben?
Kannst du dir vorstellen, welche psychologischen Gründe es dafür gibt, dass die Projektplaner die benötigte Zeit immer wieder zu niedrig einschätzen?

- c) Im Text steht: „Fast die gesamte Softwareentwicklungsabteilung ist jetzt in das Projekt eingebunden.“ Zähle die Anzahl der Mitarbeiter, die jetzt am Projekt beteiligt sind, zusammen (siehe auch Abschnitte „Das Projekt beginnt“ und „Ablauf des ersten halben Jahres“) und schätze daraus die Größe der Softwareentwicklungsabteilung ab.

Fragen zum Abschnitt „Der Geldhahn wird zugedreht“

In der Abteilung „Technik“ werden sämtliche Mitarbeiter entlassen, die bisher für „die Wartung der Geräte, die Kundenbetreuung und den Test von fertigen Software-Komponenten“ zuständig waren.

- Was hat dies für organisatorische Konsequenzen für die Firma? In der Abteilung Technik sind nur die Softwareentwickler übrig geblieben. Könnten Mitarbeiter aus anderen Abteilungen (Vertrieb, Buchhaltung, usw.) die Aufgaben der Techniker übernehmen?
- Wenn die Aufgaben der entlassenen Mitarbeiter in Zukunft von den Softwareentwicklern übernommen werden, ist dann zu erwarten, dass die Zeit für die Bearbeitung einer Aufgabe länger oder kürzer dauert als bisher? Bedenke, dass die Softwareentwickler zwar eine Universitätsausbildung besitzen aber bisher mit den Aufgaben der „einfachen“ Techniker nicht belastet wurden.
- Welche finanziellen Konsequenzen hat es für die Firma, wenn die Aufgaben der entlassenen Mitarbeiter von den Softwareentwicklern mit übernommen werden müssen? Berücksichtige, dass die Softwareentwickler eine bessere Ausbildung als die entlassenen Mitarbeiter besitzen.

Fragen zum Abschnitt „Technische Mängel“

- Das Video-Überwachungssystem hat technische Mängel, die man nur durch die Erstellung eines Prototyps hätte verhindern können. Erkläre, was in diesem Zusammenhang ein „Prototyp“ ist.
- Zu welchem Zeitpunkt hätte der Prototyp erstellt werden müssen?
- Die Firma LogoSoft zahlt die Gehälter an die Mitarbeiter verspätet aus. Welche Nachteile können einem Angestellten daraus erwachsen, wenn das Gehalt nicht zum erwarteten Termin auf dem Girokonto eingeht? Welche Konsequenzen würdest du als Mitarbeiter der Firma LogoSoft daraus ziehen?

Fragen zum Abschnitt „Die Test-Phase“

- Im Text ist davon die Rede, das „Langzeit-Tests“ durchgeführt werden. Das Video-Überwachungssystem wird also nicht nur für eine halbe Stunde gestartet und dann wieder abgeschaltet, sondern die Software muss einen längeren Zeitraum hindurch ohne Fehler laufen. Was schätzt du, über wie lange Zeit die Software auf einem einzelnen Rechner Bilder aufzeichnen muss, bis man sagen kann, das das System fehlerfrei funktioniert: mehrere Stunden, mehrere Tage oder mehrere Wochen?
- Erkläre den zyklischen Ablauf innerhalb der Testphase mit deinen eigenen Worten.
- Wie lange dauert die Testphase für das Projekt „Video-Überwachung“? Wie lange dauerte die vorhergehende Programmierphase?

Abschließende Auswertung

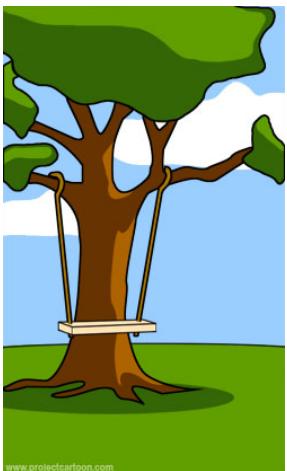
- Welche verschiedenen Arbeitsphasen gibt es bei der Entwicklung eines großen Softwaresystems? Wie lange dauerten die einzelnen Phasen in dem Beispiel „Video-Überwachung“ ungefähr?
- Bei der Entwicklung des Video-Überwachungssystems ging einiges schief. In welcher Arbeitsphase wurden die größten Fehler gemacht?
- Wie groß ist der Zeitunterschied zwischen der geplanten und der tatsächlichen Dauer des Projektes?

- d) Wie groß ist der Unterschied zwischen dem geplanten und dem tatsächlichen Arbeitsaufwand für das Projekt? Schätze dazu ab, wie viele Monate ein einziger Softwareentwickler für die gleiche Arbeit gebraucht hätte (nach Plan und tatsächlich).

29.3 Cartoon: How Projects Really Work



Wie es der Kunde erklärt hat.



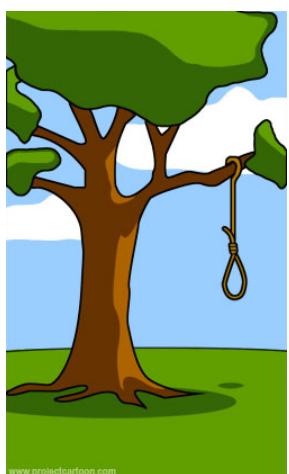
Wie es der Projekt-Manager verstanden hat.



Wie es der Analyst entworfen hat.



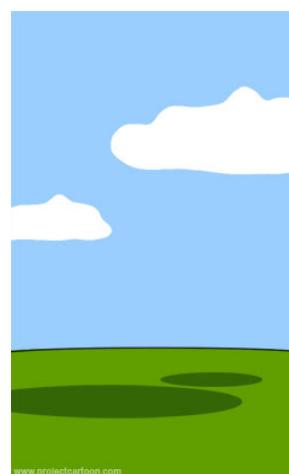
Wie es programmiert wurde.



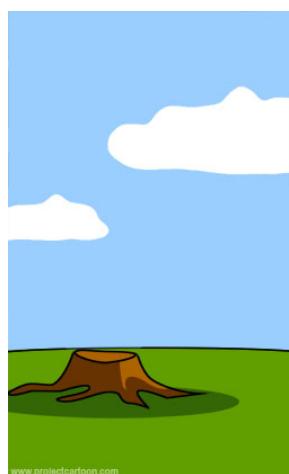
Was die Beta-Tester bekommen haben.



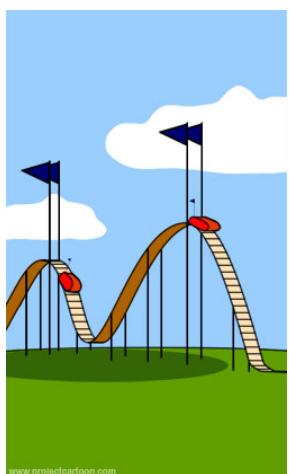
Wie es der Business-Consultant beschrieben hat.



Wie das Projekt dokumentiert wurde.



Wie es supported wurde.



Wie es dem Kunden in Rechnung gestellt wurde.



Wie es vom Marketing beworben wurde.



Wann es ausgeliefert wurde.



Was der Kunde wirklich gebraucht hätte.

30 Javadoc

Um in größeren Programmen nicht den Überblick zu verlieren, und um fremde Klassen effizient nutzen zu können, ist es unerlässlich die Klassen, ihre Attribute und Methoden zu dokumentieren.

Dies könnte klassisch in einem separaten Dokument getan werden. Java bietet für diesen Zweck jedoch mit *Javadoc* eine Möglichkeit, die Kommentare direkt im Quelltext unter zu bringen.

Aus diesen Javadoc-Kommentaren lässt sich anschließend Dokumentation in verschiedenen Formaten (unter anderem HTML, RTF und PDF) automatisiert erstellen.

Für den Programmierer besonders nützlich ist jedoch, dass diese Kommentare in Eclipse (und anderen Entwicklungsumgebungen) dazu genutzt werden, um schon beim Arbeiten mit Java-Quelltexten die entsprechenden Hinweise geben zu können – etwa wenn man mit der Maus über einer entsprechenden Methode schwebt.

30.1 Javadoc als Sonderfall von Kommentaren

Technisch gesehen sind Javadoc-Kommentare ein Sonderfall von Kommentaren in Java-Quelltexten. Zur Erinnerung:

```
// Mit dem doppelten Forward-Slash wird der Rest der Zeile zum Kommentar.

/*
 * Mehrzeilige Kommentare:
 *
 * Entscheidend sind nur die Markierungen der ersten und der letzten Zeile des
 * Kommentars. Die dazwischen liegenden Zeilen müssen nicht mit einem Asterisk
 * (Sternchen) markiert werden.
 */
```

Ein Javadoc-Kommentar ist ein Sonderfall des mehrzeiligen Kommentars: Zur Unterscheidung wird der Javadoc-Kommentar mit `/**` statt mit `/*` eingeleitet:

```
/**
 * Dieser Kommentar wird als Javadoc-Kommentar interpretiert.
 */
```

30.2 Javadoc zum Dokumentieren von Klassen, globalen Variablen und Methoden

Mit Javadoc können nicht beliebige Programmelemente kommentiert werden. So ist es beispielsweise nicht möglich, eine `for`-Schleife mit Javadoc zu kommentieren.

Der Sinn von Javadoc ist die Dokumentation von Schnittstellen einer Klasse. Interna der Programmierung hingegen sind nicht das Ziel von Javadoc.

Konkret lassen sich folgende Elemente eines Java-Programms mit Javadoc-Kommentaren versehen:

- Die Klasse selbst
- globale Variablen der Klasse (Klassen- und Objektvariablen)
- Methoden

Aus dem oben genannten Grund sind lokale Variablen nicht mit Javadoc zu dokumentieren.

30.3 Javadoc-Tags

Je nach Kontext (Klasse, Methode oder Variable) stehen einem verschiedene Tags zur Verfügung, etwa um die Bedeutung von Parametern zu beschreiben.

Die für uns wichtigsten Tags sind:

| Tag & Parameter | Bedeutung | Kontext |
|--|---|-------------------|
| <code>@author name</code> | Name des Autors | Klasse, Interface |
| <code>@param name description</code> | Parameterbeschreibung einer Methode | Methode |
| <code>@return description</code> | Beschreibung des Rückgabewertes einer Methode | Methode |
| <code>@throws classname description</code> | Beschreibung einer Exception, die von einer Methode erzeugt werden kann | Methode |

Siehe

<http://de.wikipedia.org/wiki/Javadoc>

30.4 HTML-Formatierungen

Die Javadoc Kommentare werden als HTML interpretiert und angezeigt. Das bedeutet auch, dass man HTML-Tags zur Formatierung benutzen kann.

So können beispielsweise Absätze mit `<p>` und `</p>` eingefasst werden.

Klassen-, Methoden- und Variablen-Namen (und auch sonstiger Java-Quelltext) kann mit `<code>` und `</code>` eingefasst werden.

Grundsätzlich können alle HTML-Tags benutzt werden. Von den Tags `<h1>` und `<h2>` zur Auszeichnung von Überschriften sollte man allerdings keinen Gebrauch machen, da diese von Javadoc bereits intern verwendet werden.

30.5 Beispiel

Die früher von euch häufiger benutzte Klasse `HJFrame` ist mit Javadoc-Kommentaren versehen:

```
package hilfe;

import java.awt.BorderLayout;

/**
 * Von <code>HJFrame</code> abgeleitete Klassen erzeugen ein Fenster unveränderlicher
 * Größe. Das Fenster wird von einer Zeichenfläche gefüllt, in die mit Hilfe der
 * Methode <code>myPaint()</code> gezeichnet werden kann.
 *
 * @author hartmut meyer
 */
public abstract class HJFrame extends JFrame implements ActionListener {
    // globale Variablen

    /**
     * Die Zeichenfläche, auf der mit <code>myPaint()</code> gezeichnet werden kann.
     */
    protected JPanel zeichenflaeche;

    /**
     * Einziger Konstruktor der Klasse <code>HJFrame</code>
    */
}
```

```
/*
 * @param width Die Breite (in Pixeln) der Zeichenfläche
 * @param height Die Höhe (in Pixeln) der Zeichenfläche.
 * @param background Die Hintergrundfarbe
 * @param foreground Die Vordergrundfarbe
 * @param title Der Titel des Programmfensters
 */
public HJFrame(int width, int height, Color background, Color foreground,
               final String title) {
    super(title);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    zeichenflaeche = new JPanel() {
        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            myPaint(g);
        }
    };
    zeichenflaeche.setPreferredSize(new Dimension(width, height));
    zeichenflaeche.setMaximumSize(new Dimension(width, height));
    zeichenflaeche.setMinimumSize(new Dimension(width, height));
    zeichenflaeche.setOpaque(true);
    zeichenflaeche.setDoubleBuffered(true);
    zeichenflaeche.setBackground(background);
    zeichenflaeche.setForeground(foreground);
    zeichenflaeche.setFont(new Font("Arial", Font.PLAIN, 12));
    contentPane.add(zeichenflaeche);
    setResizable(false);
    pack();
    setLocationRelativeTo(null);
    setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {
    repaint();
}

/**
 * <p>Die Methode <code>myPaint()</code> muss in Klassen, die von
 * <code>HJFrame</code> abgeleitet werden, implementiert werden. Mit dem als
 * Parameter übergebenen <code>Graphics</code>-Objekt kann dann auf der
 * Zeichenfläche gezeichnet werden.</p>
 *
 * <p><code>myPaint()</code> wird immer dann aufgerufen, wenn das Fenster neu
 * gezeichnet werden muss. Dies kann vom Betriebssystem verursacht werden (etwa
 * weil ein zuvor minimiertes Fenster wieder aus der Taskleiste geholt wird) oder
 * weil es durch einen Timer regelmäßig dazu aufgefordert wird.</p>
 *
 * @param g Das <code>Graphics</code>-Objekt der Zeichenfläche.
 */
abstract public void myPaint(Graphics g);
}
```

Und die daraus resultierende Kontext-Hilfe im Eclipse-Editor:



The screenshot shows a portion of Java code in the Eclipse IDE. A tooltip is displayed over the line of code: `myPaint(Graphics g)`. The tooltip contains the following information:

- Overrides:** `myPaint(..)` in `HJFrame`
- Parameters:** `g` Das Graphics-Objekt der Zeichenfläche.

The Java code snippet is as follows:

```
@Override  
public void myPaint(Graphics g) {  
    for (int i = 0; i < 10; i++) {  
        y = 10 * i;  
        x = 10 * i;  
        for (int j = 0; j < 10; j++) {  
            y = 10 * j;  
            x = 10 * j;  
            g.drawLine(x, y, x + 10, y + 10);  
        }  
    }  
}  
  
public static void main(String[] args) {  
    EventQueue.invokeLater(new Runnable() {  
        public void run() {  
            e.printStackTrace();  
        }  
    });  
}
```

31 Projektarbeit

Die Projektarbeit wird als Ersatz für eine der beiden Klausuren des Halbjahres gewertet. Wenn ihr zu zweit oder dritt programmiert, soll jeder einen Teil des Codes alleine schreiben (einige Code-Abschnitte schreibt man sicher am besten zusammen). Im Quellcode muss gekennzeichnet sein, wer welchen Code programmiert hat.

Für die Projektarbeit sollt ihr ein eigens dafür anzulegendes Repository benutzen, auf das die am Projekt beteiligten schreibenden Zugriff und ich Leserechte habe.

Da ihr nun nicht mehr alleine in einem Repository arbeitet, ist es *sehr* zu empfehlen, jeden Commit durch einen knappen, aber aussagekräftigen Kommentar zu versehen (der Commit-Dialog in Eclipse bietet das an). Der Commit-Kommentar sollte für jede geänderte/hinzugefügte Datei entsprechende Hinweise geben. Etwa so:

`Spieler.java:`

- Klasse neu angelegt

`Spielfeld.java:`

- Wert für Breite ist nun abhängig von der gegebenen Höhe (Seitenverhältnis 4:3)
- neue Methode: `resize()`

Auch wenn euch dies zunächst lästig erscheinen mag: Im Laufe der Projektarbeit werdet ihr so viel effizienter zusammen arbeiten können. Denn mit Hilfe dieser Commit-Kommentare lässt sich auch im Nachhinein schnell überblicken, wer wann was geändert hat. Bei der Fehlersuche kann dies ein unschätzbarer Vorteil sein!

31.1 Programm-Planung

Abgabetermin: spätestens ... (wird festgelegt)

Abzugeben sind:

- Ein Text, der beschreibt, was das Programm leisten und wie es aussehen soll. Dies kann eventuell eine Vorversion der Programm-Anleitung sein.
- Eine Skizze von der geplanten Programm-Oberfläche.
- Ein Programmentwurf mit einem UML-Klassendiagramm und eventuell einem oder mehreren Zustandsdiagrammen. Im Klassendiagramm soll gekennzeichnet sein, wer welche Klasse programmiert.

Tipp: Denkt daran, dass das Programm auch fertig werden muss! Plant deshalb lieber erst mal eine ganz einfache Version des Programms. Falls am Ende noch Zeit vorhanden ist, könnt ihr das Programm dann noch weiter ausbauen.

31.2 Programmierung und Test

Abgabetermin: ca. drei bis vier Wochen nach dem Termin für die Abgabe der Programm-Planung (wird festgelegt)

Abzugeben sind:

- Der Quellcode des Programms (im Repository). Der Quellcode muss folgendermaßen mit Javadoc kommentiert sein:
 - In jeder Datei muss ein Javadoc-Kommentar stehen, der angibt, welchem Zweck die Klasse dient.
 - Jede Methode und jede globale Variable einer Klasse müssen mit einem Javadoc-Kommentar versehen werden, der ihren Zweck erklärt (es sei denn, der Name ist selbsterklärend).
 - Hilfsvariablen, die innerhalb einer Methode deklariert sind, brauchen nicht kommentiert werden.
 - Es muss gekennzeichnet sein, wer welchen Code programmiert hat.

- Ein UML-Klassendiagramm, das alle tatsächlich programmierten Klassen und die Beziehungen zwischen den Klassen darstellt. Attribute und Methoden dürfen in diesem Diagramm weggelassen werden (Das Klassendiagramm, das zur Planung dient, soll dagegen auch die wichtigen Attribute und Methoden enthalten).
- Eine Bedienungsanleitung (am besten in Form eines Dialogs als Teil des Programms).

Wichtig: Sobald ihr mich über die Fertigstellung per e-Mail informiert, spätestens aber zum vereinbarten Abgabetermin, dürft ihr solange keine weiteren Änderungen am Repository mehr vornehmen, bis ich euch entsprechend informiert habe.

31.3 Benotung

- a) Programmietechnik (50% der Endnote)
Übersichtlichkeit, Effizienz und Schwierigkeitsgrad der Programmierung. Angemessener Einsatz der im Unterricht besprochenen Konzepte (Klassen, Vererbung, Arrays, Fehlerbehandlung, usw.). Selbständige Arbeitsweise. Am wichtigsten: Das Programm muss fehlerfrei arbeiten.
- b) Spiel-Gestaltung (25% der Endnote)
Programmidee; klar verständliche Bedienoberfläche (Bedienungsfreundlichkeit); optische Wirkung.
- c) Dokumentation (25% der Endnote)
Vollständigkeit, Korrektheit und Verständlichkeit der Dokumentation. Dies bezieht sich sowohl auf die Planung wie auch auf die Endversion.

32 SQL – Einführung

32.1 Relationale Datenbanken – Überblick

Relationale Datenbanken

Daten können auf verschiedene Arten strukturiert abgespeichert werden, z.B. könnte man die Daten nach einem Index ordnen oder in einer Baumstruktur geordnet ablegen. Für große Datenmengen hat sich heutzutage das *relationale Datenmodell* durchgesetzt, bei dem die Daten in Tabellen abgespeichert werden. *Relation* ist eine mathematische Bezeichnung für eine Tabelle. Daher der Name relationale Datenbanken. Ein Programmierer kann nicht nur die Daten einzelner Tabellen abfragen. Relationale Datenbanksysteme besitzen auch die Fähigkeit Tabellen zu verknüpfen und auf diese Weise komplexere Informationen zu generieren. Nahezu alle bekannten Datenbanksysteme arbeiten heute nach dem relationalen Modell.

Datenbanksysteme

Computerprogramme zur Beschreibung, Speicherung und Wiedergewinnung von umfangreichen Datenmengen nennt man Datenbanksysteme. Bekannte Datenbanksysteme sind:

- Oracle
- DB2
- MySQL
- MS SQL Server
- PostgreSQL

Die Software innerhalb eines Datenbanksystems, welche die Eingabe, Verwaltung und Ausgabe von Daten ermöglicht, nennt man *Datenbankmanagementsystem (DBMS)*.

Organisation eines Datenbankmanagementsystems (DBMS)

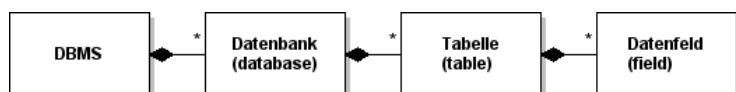


Abbildung 32.1: UML-Klassendiagramm eines DBMS

Ein Eintrag in der Tabelle, der genau zu einer Zeile und einer Spalte gehört, heißt *Datum* (Einzahl von Daten) oder *Datenwert*. Die Datenwerte in einer Zeile der Tabelle bilden zusammen einen *Datensatz* (engl. record).

Terminologie

Wir werden im Unterricht von Datenbanken, Tabellen, Spalten und Zeilen sprechen, parallel dazu aber auch einige andere Begriffe verwenden. Und wer sich anderswo über Datenbanken informiert, wird dann oft noch weitere Begriffe finden. Welche Begriffe als Synonyme benutzbar sind zeigt die folgende Tabelle:

| hier hauptsächlich verwendet | Synonyme |
|-------------------------------------|-----------------------------------|
| Datenbank | Schema |
| Tabelle | Relation, Entitätstyp, Klasse |
| Spalte | Attribut, Feld |
| Zeile | Datensatz, Entität, Tupel, Objekt |

Front End und Back End

Der Benutzer kann die Daten eines DBMS auch von einem entfernten Rechner abfragen ohne zu wissen, wie die Datenbank eigentlich intern aufgebaut ist. Die Software, die der Benutzer zur Abfrage und Veränderung der Datenbank benutzt, nennt man *Front End*. Das Front End besitzt auch die Bedienungsfläche, die der Benutzer sieht. Das DBMS, das sich unter Umständen auf einem ganz anderen Rechner befindet, wird dagegen auch als *Back End* bezeichnet. Um mit einem DBMS arbeiten zu können, muss ein Front End immer eine Verbindung (*Connection*) zu „seinem“ Back End aufbauen.

SQL – Structured Query Language

Alle gängigen Datenbank-Front-Ends benutzen die Sprache SQL für die Abfrage und Manipulation ihres Back Ends. Das Front End sendet an das Back End SQL-Befehle zur Steuerung des DBMS. SQL ist eine sogenannte *deskriptive Sprache*: Mit SQL legt man nur fest, was das DBMS tun soll. Wie es das tut, bleibt dem DBMS überlassen.

Wenn man mit einer Programmiersprache wie z.B. Java oder PHP einen Datenbank-Client programmiert, dann werden die SQL-Befehle zur Steuerung der Datenbank in die Programmiersprache „eingebettet“, d.h. man generiert mit der Programmiersprache SQL-Befehle, die an das DBMS gesendet werden.

32.2 Installationsanleitung für MySQL

Installation

Installation von MySQL

Im Kursverzeichnis findest du im Ordner 32_SQL_Einfuehrung und dort im Ordner Installationsdateien¹ die Datei mysql-installer-web-community-5.7.19.0.msi. Dabei handelt es sich um ein kleines Windows-Programm welches den Download und die Installation der eigentlichen MySQL Programme bequem erledigt.

Im Prinzip kannst du dich einfach durchklicken. Nur dann, wenn nach dem Passwort für den root Benutzer für MySQL gefragt wird, musst du etwas eingeben:

Bei *Current Root Password* gibst du entweder nichts ein (wenn dies die erste Installation von MySQL auf deinem Rechner ist) oder das alte Root-Passwort. Bei *MySQL Root Password* und *Repeat Password* gibst du jeweils root ein.

Installation des Java-Treibers

Um von einem Java-Programm aus auf MySQL zugreifen zu können, muss man einen speziellen Treiber installieren, und in Eclipse einbinden (siehe einleitendes Kapitel zur Konfiguration von Eclipse).

Im selben Ordner wie zuvor den MySQL-Installer findest auch den MySQL JDBC-Connectors:

`mysql-connector-java-5.1.43-bin.jar`

Dieses Java Archiv muss in euer eigenes Java-Projekt in Eclipse eingebunden werden, damit ihr später aus euren Java-Programmen heraus auf den MySQL-Server zugreifen könnt. Siehe Kapitel 1.3.

Konfiguration in Eclipse

Die Konfiguration des SQL Explorer Plugins für Eclipse wird im Kapitel 1 (Eclipse als Java-Entwicklungsumgebung) beschrieben.

¹oder alternativ auch hier: <https://dev.mysql.com/downloads/installer/>

32.3 SQL – Anlegen von Datenbanken

Groß- und Kleinschreibung

SQL-Schlüsselwörter sind *nicht* groß- und kleinschreib-gebunden. Ob Datenbanken und Tabellennamen der Groß- und Kleinschreibung unterliegen hängt bei MySQL vom Betriebssystem ab! Ihr solltet euch von Anfang an daran gewöhnen, Groß- und Kleinschreibung bei Namen von Bezeichnern zu unterscheiden!

Empfehlung: Gewöhnt euch an SQL-Schlüsselwörter immer in GROSSBUCHSTABEN und Bezeichner von Tabellen und Feldern immer in kleinbuchstaben zu schreiben!

Datenbank erstellen

```
CREATE SCHEMA datenbankname;
```

Das Ergebnis überprüft man mit dem Befehl

```
SHOW SCHEMAS;
```

Datenbank auswählen

```
USE datenbankname;
```

Alle nachfolgenden Anweisungen (z.B. zum Anlegen oder Abfragen von Tabellen) beziehen sich auf die ausgewählte Datenbank.

Den verwendeten Zeichensatz festlegen

Beim Anlegen einer neuen Datenbank sollte man festlegen, mit welchem Zeichensatz gearbeitet wird. Wir werden immer den UTF-8 Zeichensatz verwenden:

```
CREATE SCHEMA datenbankname DEFAULT CHARACTER SET utf8;
```

Tabelle erstellen

```
CREATE TABLE [IF NOT EXISTS] tabellename
(
    Spalten-Definition1,
    ...
    Spalten-DefinitionN
);
```

Ausdrücke in eckigen Klammern sind optional. Bei Eingabe der Schlüsselwörter **IF NOT EXISTS** wird die Tabelle nur dann angelegt, wenn sie noch nicht existiert.

Eine *Spalten-Definition* hat folgendes Schema:

```
spaltenname typ [NOT NULL] [DEFAULT wert] [AUTO_INCREMENT]
```

Typ gibt den Datentyp der Spalte an (siehe Tabelle 32.1).

Mit **NOT NULL** kann man festlegen, dass der Datenwert der Spalte nicht leer bleiben darf. „Leere“ Datenwerte werden als **NULL** bezeichnet.

Mit **DEFAULT** kann man einen Standard-Wert für die Spalte festlegen.

Mit **AUTO_INCREMENT** kann man MySQL beauftragen, die Datenwerte automatisch durch zu nummerieren. Die erste Zeile erhält die Nummer 1. Es darf nur eine Spalte pro Tabelle auf **AUTO_INCREMENT** gesetzt werden. Eine Spalte darf nur auf **AUTO_INCREMENT** gesetzt werden, wenn sie einen Index besitzt (eine Art Suchregister für besonders schnelle Zugriffe auf die Einträge). Einen Index kann man manuell anlegen. Wenn man die Spalte zum **PRIMARY KEY** erklärt, wird sie automatisch indiziert:

```
PRIMARY KEY (spaltenname);
```

Mit PRIMARY KEY erklärt man eine Spalte zum Primär-Schlüssel, der jede Zeile eindeutig identifiziert. Der Primär-Schlüssel bestimmt zum Beispiel die Standard-Sortierreihenfolge der Tabelle.

| Typ | Bedeutung |
|-------------------------|--|
| INT | 4-Byte großer Ganzzahlwert |
| FLOAT | Fließkommazahl mit einfacher Genauigkeit |
| DOUBLE | Fließkommazahl mit doppelter Genauigkeit |
| VARCHAR(maximale Länge) | Zeichenkette. In Klammern gibt man die maximale Anzahl der Zeichen an. Höchstmöglicher Wert ist 255. |
| ENUM(Wert1, Wert2, ...) | ermöglicht die Aufzählung einer Werteliste, z.B.: ENUM('rot', 'blau', 'grün') |
| DATE | Datum im Format YYYY-MM-DD |
| TIME | Uhrzeit im Format HH:MM:SS |
| DATETIME | Datum und Uhrzeit im Format YYYY-MM-DD HH:MM:SS |

Tabelle 32.1: Datentypen von MySQL (Auswahl)

Überprüfung der eingegebenen Tabellen

```
SHOW TABLES;          # Zeigt alle Tabellen der Datenbank an.
DESCRIBE tabellenname; # Zeigt den Aufbau einer Tabelle an.
```

Datenbanken und Tabellen löschen

```
DROP SCHEMA [IF EXISTS] datenbankname;
DROP TABLE [IF EXISTS] tabellenname;
```

Tabellenstruktur ändern

Mit der Anweisung ALTER TABLE ... kann man die Struktur einer existierenden Tabelle verändern. Dabei gibt es zahllose Varianten, z.B.:

```
ALTER TABLE tabellenname ADD COLUMN (Spalten-Definition1,Spalten-Definition2,...);
ALTER TABLE tabellenname DROP COLUMN spaltenname;
```

Es können auch mehrere Änderungen in einer ALTER-Anweisung zusammen gefasst werden. Wie in diesem Beispiel:

```
1  DROP SCHEMA IF EXISTS stammbaum;
2  CREATE SCHEMA stammbaum DEFAULT CHARACTER SET utf8;
3  USE stammbaum;
4  CREATE TABLE person (
5      vorname VARCHAR(20) NOT NULL,
6      nachname VARCHAR(20) NOT NULL,
7      geschlecht ENUM('männlich', 'weiblich') DEFAULT 'männlich',
8      anzahl_kinder INT NOT NULL DEFAULT 0
9  );
10 INSERT INTO person VALUES
11 ('Hartmut', 'Meyer', 'männlich', 3);
12 ALTER TABLE person
13     DROP COLUMN anzahl_kinder,
14     ADD COLUMN (
15         person_id INT AUTO_INCREMENT,
16         anzahl_töchter INT NOT NULL DEFAULT 0,
17         anzahl_söhne INT NOT NULL DEFAULT 0
18     ),
19     ADD PRIMARY KEY (person_id)
20 ;
```

```
21 UPDATE person SET anzahl_töchter=1, anzahl_söhne=2
22 WHERE vorname='Hartmut' AND nachname='Meyer';
```

32.4 SQL – Daten eingeben

Zeichenketten und Datum- / Zeit-Angaben

... werden in einfache Anführungsstriche gesetzt, z.B. '2004-08-14'

Daten einfügen

- a) Eingabe von vollständigen Tabellen-Zeilen:

```
INSERT INTO tabellenname VALUES
(Wert1, Wert2, ..., WertN),
...
(WertX1, WertX2, ..., WertXN);
```

- b) Eingabe von unvollständigen Tabellen-Zeilen:

```
INSERT INTO tabellenname (spaltenname1, spaltenname2, ...) VALUES
(Wert1, Wert2, ...),
...
(WertX1, WertX2, ...);
```

Daten der Tabelle ansehen

Sämtliche Einträge einer Tabelle erhält man mit der Anweisung:

```
SELECT * FROM tabellenname;
```

Daten löschen

Mit folgender Anweisung löscht man alle Zeilen einer Tabelle:

```
DELETE FROM tabellenname;
```

Ausgewählte Zeilen löscht man mit der Anweisung:

```
DELETE FROM tabellenname WHERE Bedingung;
```

Beispiel:

```
DELETE FROM adresse WHERE vorname='Michael' AND nachname='Mustermann';
```

Daten aktualisieren

```
UPDATE tabellenname SET spaltenname1=Wert1, spaltenname2=Wert2, ...
WHERE bedingung;
```

Beispiel:

```
UPDATE adresse SET nachname='Meier' WHERE nachname='Mustermann';
```

Vergleichsoperatoren

Ähnlich wie in Java gibt es auch in SQL alle gängigen Vergleichsoperatoren:

| Operator | Bedeutung |
|-------------|---------------------------------|
| = | gleich |
| != oder <> | ungleich |
| < | kleiner |
| > | größer |
| <= | kleiner oder gleich |
| >= | größer oder gleich |
| x IS NULL | testet ob x den Wert NULL hat |
| AND oder && | und |
| OR oder | oder |
| XOR | entweder oder (exklusives oder) |
| NOT oder ! | nicht |

Tabelle 32.2: SQL Vergleichsoperatoren

SQL-Anweisungen aus einer Datei einfügen

Man kann sämtliche SQL-Anweisungen auch in eine Textdatei schreiben. Wenn einzelne Zeilen in der Datei beim Test nicht ausgeführt werden sollen, fügt man an den Anfang der Zeile das „Hash“-Zeichen (#) ein, um aus dem Code einen Kommentar zu machen.

Wenn MySQL noch nicht gestartet ist, wird die Datei folgendermaßen ausgeführt:

```
c:\mysql\bin\mysql -u root < c:/Anna/Datei.txt
```

Wenn MySQL bereits läuft, führt man die Datei mit diesem Kommando aus:

```
source c:/Anna/Datei.txt
```

32.5 SQL – Daten abfragen

SELECT-Anweisung

Das Auslesen von Informationen aus Tabellen geschieht mit der SELECT-Anweisung:

```
SELECT spalte1, spalte2, ...
FROM tabelle1, tabelle2, ...
[WHERE Bedingung]
[GROUP BY gruppe]
[HAVING Gruppen_Bedingung]
[ORDER BY sortierspalten];
```

Hinter dem Schlüsselwort SELECT gibt man einen oder mehrere Spaltennamen an, die ausgelesen werden sollen. Wenn man alle Spalten haben möchte, schreibt man *. Wenn man Spalten aus verschiedenen Tabellen unterscheiden muss, kann man die Spaltenangaben nach folgendem Schema vornehmen:

tabellename.spaltenname

Hinter dem Schlüsselwort FROM gibt man die Tabelle oder die Tabellen an, aus denen man Daten auslesen möchte.

Hinter WHERE gibt man an, welche Bedingung die ausgewählten Zeilen erfüllen sollen.

Zeilen zählen

Mit der Funktion COUNT(Ausdruck) kann man die Anzahl von Zeilen zählen. Die folgende Anweisung zählt z.B. wie viele Zeilen sich in der Tabelle auto befinden:

```
SELECT COUNT(*) FROM auto;
```

Achtung: vor der öffnenden Klammer darf kein Leerzeichen stehen, sonst gibt es einen Fehler!

Duplikate entfernen

Mit dem Schlüsselwort `DISTINCT` kann man aus einem Abfrageergebnis Duplikate herausschmeißen. Beispiel:

```
SELECT job FROM angestellter;
```

ergibt:

```
Buchhalter
Buchhalter
Programmierer
System-Administrator
Programmierer
Programmierer
```

Wenn man die Mehrfachnennungen im Ergebnis nicht wünscht, dann hilft

```
SELECT DISTINCT job FROM angestellter;
```

Dieses ergibt nun:

```
Buchhalter
Programmierer
System-Administrator
```

Gruppen bilden

Mit `GROUP BY` kann man die abgerufenen Zeilen zu Gruppen zusammenfassen. Beispiel:

```
SELECT COUNT(*), job
FROM angestellter
GROUP BY job;
```

Diese Anfrage zählt die Anzahl der Angestellten, die in den einzelnen Jobs tätig sind, und zwar gruppiert nach dem Job. Die Ausgabe könnte z.B. so aussehen:

```
2 Buchhalter
3 Programmierer
1 System-Administrator
```

Gruppen-Bedingungen

Mit `HAVING` kann man für ausgewählte Gruppen zusätzliche Bedingungen einführen. Beispiel:

```
SELECT COUNT(*), job
FROM angestellter
GROUP BY job
HAVING COUNT(*)=1;
```

Mit dieser Abfrage wählt man die Jobs im Unternehmen aus, für die jeweils nur ein Angestellter tätig ist. Das Ergebnis ist nun:

```
1 System-Administrator
```

Suchergebnisse sortieren

Mit `ORDER BY` kann man die Suchergebnisse nach einer oder mehr Spalten sortieren. Die Sortierung kann entweder aufsteigend (`ASC`) oder absteigend (`DESC`) sein. Aufsteigend ist die Standard-Einstellung. Beispiel:

```
SELECT *
FROM angestellter
ORDER BY job ASC, name DESC;
```

Die Zeilen werden zunächst nach dem Job in aufsteigender alphabetischer Reihenfolge sortiert. Falls zwei oder mehr Personen denselben Job haben, werden sie anschließend noch in umgekehrter alphabetischer Reihenfolge nach dem Namen sortiert.

32.6 SQL – Übung 1: Eine einfache Haustierdatenbank

Aufgabe 1: Datenbank erzeugen

Erzeuge eine Haustier-Datenbank. Die Datenbank soll zunächst nur die Tabelle `tier` mit folgenden Datenfeldern enthalten:

- `name` (muss immer angegeben werden)
- `tierart` (muss immer angegeben werden)
- `lebendig` (ja/nein; Standard-Wert: ja)
- `geschlecht` (männlich/weiblich; Standard-Wert: weiblich)
- `geburtstag` (kann unbekannt sein)
- `todestag` (kann unbekannt sein oder kann leer sein, weil das Tier noch lebt)

Bitte halte beim Anlegen der Tabelle die Reihenfolge der Datenfelder exakt ein. Welche Datentypen sollten die einzelnen Datenfelder erhalten?

Aufgabe 2: Daten eingeben

Führe die folgenden Anweisungen bitte exakt aus, damit nachher alle für die weiteren Übungen dieselbe Tabelle haben:

a) Gib mit Hilfe der ersten Form der `INSERT`-Anweisung die folgenden Datensätze in die Tabelle `tier` ein:

- Bello, Hund, ja, männlich, 01.05.2003, null
- Daisy, Kanarienvogel, nein, weiblich, 06.12.1996, 17.08.2004
- Mausi, Katze, ja, weiblich, 17.11.2002, null

b) Gib mit Hilfe der zweiten Form der `INSERT`-Anweisung die folgenden Datensätze ein.

- Daisy, Schildkröte
- Lassie, Hund
- Maja, Hund

c) Gib mit Hilfe der zweiten Form der `INSERT`-Anweisung die folgenden Datensätze ein:

- Hasso, Hund, männlich
- Blacky, Katze, männlich
- Harald, Hamster, männlich

d) Nun sollen nachträglich noch einige Datensätze verändert werden:

- Trage für die Schildkröte Daisy den Geburtstag 06.12.2003 ein.
- Die Hunde Lassie, Maja und Hasso stammen aus einem Wurf. Trage (mit einer einzigen, möglichst kurzen Anweisung) den Geburtstag 23.04.2004 für die drei Hunde ein.
- Der Hamster Harald ist verstorben. Trage für Harald (in einer einzigen Anweisung) den Geburtstag 29.07.2001, den Todestag 15.09.2003 und ein 'nein' für lebendig ein.

Aufgabe 3: Daten abfragen

Nachdem wir die Tabelle tier angelegt haben, wollen wir unsere Mini-Datenbank für Abfragen nutzen. Formuliere geeignete SQL-Anweisungen für die folgenden Abfragen an die Tabelle tier. Speichere die SQL-Anweisungen in einer Textdatei ab, damit du sie später noch weißt.

- a) Zeige die gesamte Tabelle tier an.
- b) Zeige nur die Spalten name und tierart an.
- c) Liste die Spalten name und tierart auf. Sortiere dabei in aufsteigender alphabetischer Reihenfolge nach dem Namen. An zweiter Stelle soll nach der Tierart sortiert werden (auch aufsteigend).
- d) Zeige nur die Spalte geburtstag an. Sorge dafür, dass jeder Wert nur einmal angezeigt wird.
- e) Zeige die Spalten name und tierart für die Tiere an, die noch leben.
- f) Liste die Namen aller Tiere auf, die vor dem Jahr 2004 geboren wurden und noch nicht tot sind. Sortiere die Namen in absteigender alphabetischer Reihenfolge.
- g) Liste die gesamten Spalten aller Tiere auf, die weder Hund noch Katze sind.
- h) Zeige die Namen und die Geburtstage aller Tiere an, für die kein Todestag angegeben wurde.
- i) Zähle wie viele Tiere nach dem 01.01.2003 geboren wurden.
- j) Liste auf, wie viele Tiere es von jeder Tierart gibt. Das Ergebnis soll eine Tabelle mit einer Spalte tierart sein und einer Spalte, die zu jeder Tierart die Anzahl angibt.
- k) Liste alle Tierarten auf, von denen es zwei oder mehr Tiere gibt.
- l) Zähle die Anzahl der unterschiedlichen Tierarten, die es gibt.

32.7 SQL – Übung 2: Die Haustierdatenbank wird erweitert

Aufgabe 1: Datenstruktur erweitern

Die Haustier-Datenbank aus Übung 1 soll erweitert werden. Zu jedem Tier soll jetzt noch sein Besitzer mit eingetragen werden. Über einen Besitzer sollen folgende Daten abgespeichert werden:

- anrede (Herr, Frau, Firma; Standard-Wert ist „Herr“)
- vorname
- nachname (muss eingegeben werden)
- straße (inklusive Hausnummer)
- plz
- ort
- telefonnr

Selbstverständlich kann ein Besitzer mehrere Tiere haben. Achte darauf, dass Änderungen z.B. der Adresse eines Besitzers möglichst wenig Datenbank-Änderungen nach sich ziehen. Wie kann man den Besitzer am geschicktesten in die Datenbank einbauen?

32.8 Tabellen verknüpfen: Primär- und Fremdschlüssel

Primärschlüssel

Um Datensätze einer Tabelle eindeutig identifizieren zu können, braucht man einen sogenannten *Schlüssel*. Als Schlüssel kann jedes einzelne Attribut oder – wenn nötig – auch eine Kombination mehrerer Attribute der Tabelle dienen. Wichtig ist nur, dass sich der Schlüssel *konzeptionell* zur eindeutigen Identifizierung eines jeden Datensatzes der Tabelle eignet. So reicht es etwa nicht, zu sehen, dass in unserer Haustierdatenbank keine Kombination aus Tiername und Tierart doppelt vorkommt. Denn niemand kann garantieren, dass in Zukunft nicht doch noch ein weiteres Tier mit einer zuvor bereits existierenden Kombination aus Tierart und Tiername hinzukommen wird. Tiername und Tierart sind also zusammen kein geeigneter Schlüssel für die Tabelle *tier*!

An einen Schlüssel wird zudem die Anforderung gestellt, dass er *minimal* sein soll: alle Attribute, die man zum Schlüssel dazu zählt müssen auch wirklich *nötig* sein, um die eindeutige Identifizierung aller Datensätze (auch konzeptionell) sicher zu stellen.

Bei einer gegebenen Tabelle können durchaus verschiedene Attribute oder Kombinationen aus Attributen als Schlüssel in Frage kommen. Den Schlüssel, den man aus der Menge der möglichen Schlüsselkandidaten zur weiteren Verwendung auswählt, bezeichnet man als *Primärschlüssel* (engl. *Primary Key*).

Als Primärschlüssel sind besonders gut extra für diesen Zweck erzeugte Attribute vom Typ **INT** geeignet, die von der Datenbank mit Hilfe des Schlüsselwertes **AUTO_INCREMENT** beim Erzeugen eines neuen Datensatzes automatisch mit einem bisher noch nicht vergebenen Wert belegt werden.

Überhaupt ist es empfehlenswert, als Primärschlüssel ein Attribut zu benutzen, welches keine sonstigen Information trägt! Damit umgeht man so manche Falle! Beispielsweise könnte man annehmen, das die ISBN-Nummer ein geeigneter Primärschlüssel für eine *buch*-Tabelle wäre. Oder das die Postleitzahl ein ebenso geeigneter Primärschlüssel für eine *ort*-Tabelle wäre. Beides stellt sich bei näherer Betrachtung als falsch heraus (es gibt Postleitzahlen, aus denen nicht eindeutig auf einen bestimmten Ort geschlossen werden kann und es gibt auch ISBN-Nummern auf die nicht eindeutig auf eine bestimmte Ausgabe eines Buches geschlossen werden kann). Durch die Verwendung eines nicht-informationstragenden Primärschlüssels, umgeht man solche Fallen. Zudem muss man sich keine Sorgen machen, dass es in Zukunft aus inhaltlichen Gründen notwendig werden könnte die Werte des Primärschlüsselattributes zu ändern – so wie es beispielsweise bei den Postleitzahlen mehrfach (zuletzt 1993) geschehen ist.

```
CREATE TABLE besitzer
(
    besitzer_id INT AUTO_INCREMENT,
    anrede ENUM('Herr', 'Frau', 'Firma') DEFAULT 'Herr',
    vorname VARCHAR(20),
    nachname VARCHAR(20) NOT NULL,
    PRIMARY KEY (besitzer_id)
);
```

Fremdschlüssel

Als *Fremdschlüssel* (engl. *Foreign Key*) bezeichnet man ein Attribut einer Tabelle, welches auf die Werte eines Primärschlüssels einer anderen Tabelle verweist. Genau genommen muss der Primärschlüssel noch nicht einmal aus einer anderen Tabelle stammen: auch Verweise auf den Primärschlüssel der eigenen Tabelle sind möglich!

```
CREATE TABLE tier
(
    name VARCHAR(20) NOT NULL,
    tierart VARCHAR(20) NOT NULL,
    lebendig ENUM('ja', 'nein') DEFAULT 'ja',
    geschlecht ENUM('weiblich', 'männlich') DEFAULT 'weiblich',
    geburtstag DATE,
    todestag DATE,
    tier_besitzer_id INT,
    FOREIGN KEY (tier_besitzer_id) REFERENCES besitzer (besitzer_id)
);
```

Hinter **REFERENCES** wird dabei die Tabelle und das entsprechende Attribut benannt, auf welches sich der Fremdschlüssel bezieht. Dadurch ist es auch sehr leicht, in einem gegebenen SQL-Skript die entsprechenden Beziehungen zwischen den Tabellen zu erkennen.

Referentielle Integrität

Indem man der Datenbank mitteilt, dass es sich bei einem Attribut um einen Fremdschlüssel handelt, versetzt man sie auch in die Lage über die sogenannte *referentielle Integrität* der Datenbank zu wachen. Die Datenbank verhindert dann, dass ein neuer Datensatz so angelegt wird, dass der Wert für den Fremdschlüssel nicht durch einen bereits existierenden Wert im Primärschlüsselfeld der referenzierten Tabelle gedeckt ist. Im Beispiel unserer Haustierdatenbank: Nachdem wir das Attribut `tier_besitzer_id` in der `tier`-Tabelle als Fremdschlüssel definiert haben, der auf den Primärschlüssel der `besitzer`-Tabelle verweist, wird es die Datenbank nicht erlauben, dass wir ein neues Tier anlegen, mit einem Wert für `tier_besitzer_id`, den es so noch nicht als Wert für den Primärschlüssel der `besitzer`-Tabelle gibt. Ebenso würde verhindert, dass ein Datensatz aus der `besitzer`-Tabelle gelöscht oder so verändert würde, dass in der Folge ein Datensatz in der `tier`-Tabelle mit einem Verweis auf einen nicht länger existierenden Wert im Primärschlüssel der `besitzer`-Tabelle resultieren würde.

Verknüpfung von Tabellen

Nicht immer, aber sehr oft, werden bei Datenbankabfragen (**SELECT**) Daten aus mehreren Tabellen verknüpft. Diese Verknüpfung erfolgt über die Verbindung von Primär- und Fremdschlüsseln.

In der Ergebnisliste der Abfrage interessieren uns nur solche Ergebnisse, die sich aus zusammengehörigen Datensätzen der beteiligten Tabellen ergeben. Wenn beispielsweise eine Ausgabe aller Tiere und ihrer Besitzer verlangt ist, dann sollen natürlich für jedes Tier auch wirklich nur die passenden Besitzer ausgegeben werden und nicht irgendwelche Datensätze aus der `besitzer`-Tabelle. Und genau dies erreichen wir über eine geeignete Bedingung, die den Fremdschlüssel der einen Tabelle mit dem passenden Primärschlüssel der anderen Tabelle verknüpft:

```
SELECT tier.name, tier.tierart, besitzer.vorname, besitzer.nachname
FROM tier, besitzer
WHERE tier.tier_besitzer_id = besitzer.besitzer_id;
```

Bisher kennen wir nur den Fall mit maximal zwei beteiligten Tabellen. Sobald in einer Abfrage noch weitere Tabellen hinzukommen, ist es nötig, die entsprechenden Primärschlüssel-Fremdschlüssel-Beziehungen aller beteiligter Tabellen in der **WHERE**-Klausel zu benennen. Das kann bei mehr als drei beteiligten Tabellen durchaus unübersichtlich werden. Wenn ihr in der Ergebnisliste zu viele (und dabei auch unsinnige Kombinationen) erhaltet, ist das ein sicherer Hinweis darauf, dass ihr nicht alle Beziehungen in der **WHERE**-Klausel berücksichtigt habt!

32.9 SQL – Übung 3: Erweiterte Haustierdatenbank

Aufgabe 1: Daten eingeben

Gib nacheinander die folgenden Daten ein. Benutze dabei gegebenenfalls die `INSERT`-Variante, bei der man nur einzelne Spalten angibt.

| Besitzer | besitzt die Tiere |
|--|----------------------------|
| Firma, NULL, Zoo Lilliput, Obernstraße 54, 20012, Hamburg, 0721/34 34 12 | Bello, Lassie |
| Frau, Sandra, Sandelmann, Kullerweg 12, 28205, Bremen, NULL | Daisy (Kanarienvogel) |
| Herr, Mirco, Sandelmann, Unterstraße 17, 28232, Bremen, 0421/123456 | Mausi, Blacky, Harald |
| Herr, Tobias, Winkelmann, NULL, NULL, NULL, NULL | Daisy (Schildkröte), Hasso |
| Frau, Sandra, Anderson, NULL, NULL, NULL, NULL | Maja |

Aufgabe 2: Daten ändern

Die Adresse von Frau Anderson wurde nachgereicht: Wilhelminenweg 42, 28315, Bremen. Trage die Änderung in die Tabelle `besitzer` ein.

Aufgabe 3: Daten abfragen

Führe auf der erweiterten Datenbank die folgenden Abfragen durch:

- Zeige ohne Verwendung der `WHERE`-Klausel eine Tabelle mit allen Besitzern (Nachname und Vorname) und Tieren (Name und Tierart) an. Was für eine Tabelle wird erstellt?
- Zeige eine Liste mit allen Besitzern und den zu ihnen gehörenden Tieren an. Es sollen alle Spalten angezeigt werden.
- Zeige eine Tabelle mit allen Besitzern (Nachname und Vorname) und ihren Tieren (Name und Tierart) an. Sortiere die Liste in aufsteigender alphabetischer Reihenfolge zunächst nach dem Nachnamen und dann nach dem Vornamen.
- Zeige eine Tabelle mit allen Besitzern (Vor- und Nachname) und ihren lebendigen Tieren an (Name und Tierart). Sortiere die Liste nach der Tierart.
- Zeige alle Tiere von Mirco Sandelmann an (Name, Tierart und „lebendig“).
- Wähle alle Besitzer von Hunden aus. Zeige Vor- und Nachname der Besitzer sowie Name, Geburtstag und Todestag des Hundes an.
- Zeige eine Liste der Besitzer (Vor- und Nachname) und der Anzahl der Tiere an, die sie besitzen. Sortiere die Liste in umgekehrter alphabetischer Reihenfolge nach Nachnamen.
- Zeige alle Besitzer mit Vor- und Nachnamen an, die zwei oder mehr Tiere besitzen.
- Zähle die Anzahl der Besitzer, die einen Hund oder eine Katze besitzen.

32.10 SQL – Daten abfragen (Fortsetzung)

Kartesisches Produkt

Die Ergebnistabelle, die alle möglichen Zeilen aus der Kombination zweier Tabellen enthält, wird als das *Kartesische Produkt* aus zwei Tabellen bezeichnet.

Das Kartesische Produkt von einer Tabelle a mit einer Tabelle b unter Auswahl sämtlicher Spalten erhält man mit der SQL-Anweisung:

```
SELECT * FROM a, b;
```

Das Ergebnis ist natürlich unsinnig. Statt dessen brauchen wir die Verknüpfung über die Fremdschlüssel-Primärschlüssel-Beziehung zwischen den beteiligten Tabellen:

```
SELECT * FROM a, b
WHERE fremdschlüssel_der_einen_tabelle = primärschlüssel_der_anderen_tabelle;
```

LEFT JOIN und RIGHT JOIN

Gegeben sind die beiden Tabellen:

besitzer:

| besitzer_id | name |
|-------------|---------|
| 1 | Anna |
| 2 | Andrew |
| 3 | Annabel |

fahrzeug:

| fahrzeug | fahrzeug_besitzer_id |
|----------|----------------------|
| Fahrrad | 2 |
| Auto | 4 |
| Auto | 3 |

Die folgenden drei Abfragen demonstrieren die Wirkung eines LEFT JOIN oder eines RIGHT JOIN im Vergleich zu einer normalen Verknüpfung:

```
SELECT * FROM besitzer, fahrzeug
WHERE besitzer_id = fahrzeug_besitzer_id;
```

ergibt:

| besitzer_id | name | fahrzeug | fahrzeug_besitzer_id |
|-------------|---------|----------|----------------------|
| 2 | Andrew | Fahrrad | 2 |
| 3 | Annabel | Auto | 3 |

```
SELECT * FROM besitzer LEFT JOIN fahrzeug
ON besitzer_id = fahrzeug_besitzer_id;
```

ergibt:

| besitzer_id | name | fahrzeug | fahrzeug_besitzer_id |
|-------------|---------|----------|----------------------|
| 1 | Anna | [NULL] | [NULL] |
| 2 | Andrew | Fahrrad | 2 |
| 3 | Annabel | Auto | 3 |

```
SELECT * FROM besitzer RIGHT JOIN fahrzeug
ON besitzer_id = fahrzeug_besitzer_id;
```

ergibt:

| besitzer_id | name | fahrzeug | fahrzeug_besitzer_id |
|-------------|---------|----------|----------------------|
| 2 | Andrew | Fahrrad | 2 |
| [NULL] | [NULL] | Auto | 4 |
| 3 | Annabel | Auto | 3 |

Verknüpfung einer Tabelle mit sich selbst

Gegeben ist die Tabelle angestellter:

| name | job |
|---------------|---------------|
| Jan Bauer | Buchhalter |
| Talina Wendel | Programmierer |
| Noah Richter | Programmierer |
| Andrea Faust | Redakteur |

Um herauszufinden, wer denselben Job hat wie Talina Wendel, kann man die folgende Abfrage stellen:

```
SELECT kollege.name
FROM angestellter talina, angestellter kollege
WHERE talina.name = 'Talina Wendel' AND kollege.job = talina.job;
```

Es wird zweimal die Tabelle `angestellter` verwendet. Um die beiden Tabellen unterscheiden zu können erhalten sie Namen (ein sogenanntes Alias). `talina` bezeichnet die erste Angestellten-Tabelle und `kollege` die zweite Angestellten-Tabelle. Die Alias Bezeichner dürfen frei gewählt werden und es empfiehlt sich hier ganz besonders Namen zu wählen, die erkennen lassen, was gemeint ist.

Aufgabe: Welche Ausgabe erzeugt die Abfrage?

Unterabfragen

Ein sehr nützliches SQL-Konstrukt sind die sogenannten *Unterabfragen* (auch *Sub-Selects* oder *Sub-Queries* genannt). Dabei wird innerhalb eines SQL-Befehls eine weitere SQL-Abfrage benutzt.

Beispiele dafür findest du in den Musterlösungen für die Aufgaben Übung 3, Aufgabe 1 und Übung 4 Aufgabe 1. Etwa:

```
UPDATE tier
SET tier_besitzer_id =
    (SELECT besitzer_id FROM besitzer
     WHERE vorname = 'Mirco'
       AND nachname = 'Sandelmann')
)
WHERE name = 'Harald';
```

32.11 SQL – Übung 4: Die Haustierdatenbank (Fortsetzung)

Aufgabe 1: Datenbank erweitern

Sandra und Mirco Sandelmann haben sich nach einer langen Trennungsphase wieder versöhnt. Sandra zieht wieder zu Mirco in das gemeinsame Haus zurück. Aber wem gehören nun die Tiere? Beiden natürlich. Erweitere die Datenbank entsprechend! Und korrigiere die Adresse von Sandra.

Die Kinder von Sandra Anderson haben sich übrigens beschwert, dass nur ihre Mutter als Besitzerin in der Datenbank eingetragen ist. Sie möchten ebenfalls verzeichnet werden:

Anka und Max Anderson wohnen beide bei der Mutter. Kai Anderson wohnt in der Unsinnstraße 65, 28245 Bremen.

Aufgabe 2: Daten abfragen

Führe auf der erweiterten Datenbank die folgenden Abfragen durch:

- Zeige eine Liste aller Besitzer (Vor- und Nachname) und ihrer Tiere (Name und Tierart) an. Sortiere die Liste absteigend nach dem Namen der Tiere.
- Liste alle Besitzer von Maja auf (Vor- und Nachname).
- Zähle die Anzahl der Besitzer von Blacky.
- Erstelle eine Liste, die angibt, wie viele Tiere jede einzelne Person besitzt (Ausgabe: Anzahl, Vor- und Nachname).
- Liste die vollständigen Daten aller Besitzer auf, deren Telefonnummer nicht bekannt ist.
- Liste Straße, PLZ und Ort aller Besitzer auf. Dabei soll jede Adresse nur einmal ausgegeben werden.

Aufgabe 3: Daten verändern

- Die Katze Mausi ist weggelaufen und hat nun offensichtlich keinen Besitzer mehr. Lösche die Beziehung von Mausi zu ihren beiden Besitzern Sandra und Mirco Sandelmann, aber lösche nicht die Katze selbst aus der Datenbank.
- Johanna Sonntag wünscht sich sehnlich einen Hund und möchte schon vor der Anschaffung eines Tieres in die Datenbank eingetragen werden. Füge in die Besitzer-Tabelle die folgenden Daten ein:
Frau, Johanna, Sonntag, Glücksweg 13, 28333, Bremen

Aufgabe 4: Daten abfragen 2

Führe auf der erweiterten Datenbank die folgenden Abfragen durch:

- Liste die Personen, die momentan kein Tier besitzen, mit Ihrem Vor- und Nachnamen auf.
- Liste die Tiere mit Namen auf, die keinen Besitzer haben.
- Liste alle Personen, die mit Anka Anderson zusammen wohnen (d.h. alle, die dieselbe Straße haben) mit ihren vollständigen Daten auf. Anka selbst darf auch in der Liste erscheinen.
- Liste alle Tiere mit Namen auf, die am selben Tag geboren wurden wie Maja. Maja selbst soll nicht in der Liste erscheinen.
- Liste die Namen der Tiere auf, die am selben Tag geboren wurden wie Maja, und den Vor- und Nachnamen ihrer Besitzer. Maja selbst soll nicht in der Liste erscheinen.
- Liste alle Tiere mit Namen auf, die denselben Besitzer haben wie Hasso. Hasso darf ebenfalls in der Liste erscheinen.

32.12 SQL – Weiterführendes

SQL bietet noch eine ganze Reihe weiterer Sprachelementen, die wir aus Zeitgründen aber nicht alle behandeln können. Hier nur ein kurzer Überblick über das, was wir nicht behandelt haben:

Ähnlichkeitssuche mit LIKE

Bisher weißt du, wie du den Wert eines Datenfeldes auf Gleichheit (oder auch Ungleichheit) mit einem Vergleichswert überprüfen kannst. Etwa in diesem Beispiel:

```
SELECT * FROM schueler WHERE klasse = '7a';
```

Aber was, wenn du nicht nur alle Schüler der 7a, sondern alle Schüler aus allen siebten Klassen suchst?

```
SELECT * FROM schueler WHERE klasse LIKE '7%';
```

Das Prozent-Zeichen ist ein sogenanntes *Wildcard*-Zeichen. % steht für beliebig viele (auch kein) beliebige Zeichen.

Als weiteres Wildcard-Zeichen kann der Unterstrich benutzt werden. _ steht für genau ein beliebiges Zeichen.

MAX(), MIN(), AVG() und SUM()

Mit MAX(ausdruck) lässt sich der größte Wert in Ausdruck ermitteln.

Analoges gilt für MIN(ausdruck) (liefert den kleinsten Wert), AVG(ausdruck) (liefert das arithmetische Mittel) und SUM(ausdruck) (liefert die Summe aller Werte).

Mathematische Funktionen

Man kann in SQL auch rechnen. Wenn in der Tabelle artikel in dem Feld preis die jeweiligen Nettopopreise der Artikel gespeichert sind, so kann man die Bruttoreise direkt in SQL berechnen lassen:

```
SELECT nummer, bezeichnung, preis "Netto-Preis", (preis*1.19) "Brutto-Preis" FROM artikel;
```

Neben den Grundrechenarten kennt SQL aber auch eine große Anzahl von weiteren mathematischen Funktionen, wie etwa SIN(), COS(), TAN(), LN() usw.

Datentyp BLOB

In Datenbanken können nicht nur Text oder Zahlenwerte abgespeichert werden, sondern beliebige Daten. Etwa Grafik-Dateien, Musik oder PDF-Dokumente. Für solche Daten stehen in MySQL spezielle Datentypen zur Verfügung: Von BLOB (maximal 64kB) über MEDIUMBLOB (maximal 16MB), bis hin zu LONGBLOB (bis zu 4GB).

Um die entsprechenden Daten in der Datenbank zu speichern, oder auch von dort auszulesen, muss man allerdings ein Programm schreiben (etwa in Java).

Ein Beispiel findest du im Kurs-Repository in 37_SQL_Datenbankzugriffe im Ordner blob.

Unterabfragen

Man kann die Ergebnisse von Abfragen direkt in anderen Abfragen benutzen.

```
SELECT * FROM konto WHERE kontonummer = (SELECT kontonummer FROM konto WHERE konto_id = 1);
```

Stored Procedures

Ähnlich wie in Java kann man auch in SQL programmieren. Dazu kann man (wie in Java) Variablen, Schleifen und Verzweigungen benutzen.

33 SQL – Komplexes Beispiel

33.1 Firmen-Datenbank

Aufgabe 1: Datenbank Firma untersuchen

Im Kurs-Repository liegt die Datei firma.sql. Erzeuge mit Hilfe dieser Datei die Datenbank firma und untersuche die neue Datenbank.

Erstelle ein UML-Klassendiagramm, das die Tabellen und ihre Beziehungen zueinander beschreibt. Markiere im Klassendiagramm alle Datenfelder, die zum Primärschlüssel (PK) oder Fremdschlüssel (FK) einer Tabelle gehören.

Aufgabe 2: Datenbankabfragen

Führe auf der Firmen-Datenbank die folgenden Abfragen durch:

- a) Erstelle eine Liste aller Abteilungen (Abteilungsname und Abteilungsnummer) und der Namen der zugehörigen Abteilungsleiter (Vor- und Nachname). Sortiere die Liste absteigend nach dem Abteilungsnamen.
- b) Erstelle eine Liste aller Angestellten und ihrer Angehörigen. Es sollen der Vor- und Nachname der Angestellten sowie der Name, das Geschlecht und der Verwandtschaftsgrad der Angehörigen aufgelistet werden. Auch Angestellte, die keine Angehörigen besitzen, sollen in der Tabelle aufgelistet werden. Sortiere die Liste alphabetisch zuerst nach dem Nachnamen und anschließend nach dem Vornamen der Angestellten. Danach soll die Liste nach dem Namen der Angehörigen sortiert werden.
- c) Ermittle den Namen (Vor- und Nachname) des Vorgesetzten von Jennifer Wallace.
- d) Zähle die Anzahl der Projekte, die momentan in der Firma durchgeführt werden.
- e) Erstelle eine Liste mit allen Angestellten (Vor- und Nachname) und den Projekten (Projekt-Name und Projekt-Nummer), an denen sie arbeiten.
- f) Erstelle eine Liste aller Mitarbeiter (Vorname, Nachname, Geburtstag, Geschlecht), die am Projekt „Newbenefits“ arbeiten.
- g) Ermittle den Abteilungsleiter (Vor- und Nachname), der für das Projekt „Reorganization“ zuständig ist.
- h) Zähle die Anzahl der Mitarbeiter im Projekt „Reorganization“.
- i) Liste alle Orte auf, in denen die Firma eine Niederlassung besitzt. Jeder Ort soll nur einmal aufgelistet werden.
- j) Zähle die Anzahl der Projekte, an denen jeder einzelne Mitarbeiter arbeitet. Die Mitarbeiter sollen mit Vor- und Nachnamen aufgelistet werden.
- k) Liste alle Projekte mit Namen auf, die am selben Ort wie das Projekt „Newbenefits“ stattfinden. Das Projekt „Newbenefits“ selbst soll nicht in der Liste erscheinen.
- l) Liste alle Mitarbeiter mit Vornamen, Nachnamen und Angestellten-Nummer auf, die mindestens drei Angehörige besitzen.

34 Entity-Relationship-Modell

Das Entity-Relationship-Modell (kurz: ER-Modell) wird zum Entwurf von Datenbanken eingesetzt.

34.1 Begriffe im ER-Modell

Entität und Entitätstyp

Ein Objekt wird im Entity-Relationship-Modell als *Entität* bezeichnet. Eine Gruppe gleichartiger Objekte, die durch die selbe Tabelle beschrieben werden, bezeichnet man als *Entitätstyp*. Verglichen mit den Begriffen, die du bereits aus der Objektorientierten Programmierung kennst, entspricht die Entität dem Objekt und der Entitätstyp der Klasse. Statt von Entitätstypen spricht man aber üblicherweise einfach von Tabellen.

Attribute

Die Eigenschaften (Attribute) eines Entitätstyps werden analog zu den Attributen einer Klasse im UML-Klassendiagramm dargestellt. Attribute, die Teil des Primärschlüssels sind, werden durch ein PK (für Primary Key) in der ersten Spalte kenntlich gemacht. Mit FK in der ersten Spalte kennzeichnet man einen Fremdschlüssel (Fogeyn Key)). Ein Attribut welches einerseits ein Fremdschlüssel ist und andererseits Teil des Primärschlüssels der eigenen Tabelle ist wird mit PK, FK in der ersten Spalte gekennzeichnet.

| tier | |
|------|------------|
| | |
| PK | tier_id |
| | name |
| | tierart |
| | geschlecht |
| | lebendig |
| | geburtstag |
| | todestag |

Abbildung 34.1: Darstellung einer Tabelle mit einem Primärschlüssel in einem ER-Diagramm

Beziehungen

Die Beziehung zwischen zwei Entitätstypen wird durch eine Verbindungsline dargestellt. Man unterscheidet zwischen drei verschiedenen Beziehungs-Typen:

1:1 Beziehungen Ein Objekt des ersten Entitätstypen steht mit genau einem Objekt des zweiten Entitätstypen in Beziehung. Solche Beziehungen werden durch stilisierte Einsen an den beiden Enden der Verbindung gekennzeichnet.

1:n Beziehungen Ein Objekt des ersten Entitätstypen steht mit mehreren Objekten des zweiten Entitätstypen in Beziehung. In diesem Fall wird das eine Ende der Verbindungsline mit einer stilisierten Eins und das andere Ende mit einer Verzweigung (dem sogenannten Krähenfuß, der auch Namensgeber für die von uns verwendete Art von ER-Diagrammen ist)).

n:m Beziehungen Ein Objekt des ersten Entitätstypen steht mit mehreren Objekten des zweiten Entitätstypen in Beziehung. Ein Objekt des zweiten Entitätstypen steht ebenfalls mit mehreren Objekten des ersten Entitätstypen in Beziehung. Wie wir bereits gesehen haben, sind solche Beziehungen nur über eine Beziehungstabelle abzubilden. In der Krähenfußnotation erkennt man solche Beziehungstabellen daran, dass die Beziehungstabelle selbst von Krähenfüßen berührt wird, während die beiden Tabellen, welche durch die Beziehungstabelle verbunden werden durch die stilisierte Eins (genauer: die stilisierte Doppel-Eins) angebunden sind. Siehe zum Beispiel Abbildung 34.3.

Und es geht sogar noch etwas genauer: Wir können noch unterscheiden zwischen *genau Eins* und *Null oder Eins*. Den ersten Fall stellen wir dar durch zwei stilisierte Einsen, den zweiten Fall durch eine Null und eine Eins am entsprechenden Ende der Verbindungsline.

Ebenso können wir unterscheiden zwischen *beliebig viele (auch Null)* und *beliebig viele, aber mindestens Eins*. Zur Kennzeichnung wird dem Krähenfuß entsprechend eine stilisierte Eins oder Null zur Seite gestellt.

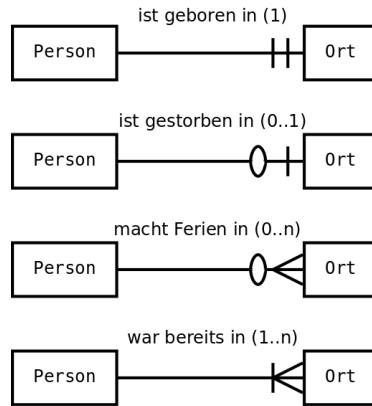


Abbildung 34.2: Darstellung verschiedener Kardinalitäten in ER-Diagrammen in der Krähenfuß-Notation

Anmerkung zur Abbildung: Auf der linken Seite wurden absichtlich keine Kardinalitäten eingetragen, weil es hier um die Beziehung *einer* Person zu den Orten geht. In einem „echten“ ER-Diagramm müsste man natürlich die Beziehung der beiden Entitätstypen *Person* und *Ort* durch Kardinalitäten auf beiden Seiten kennzeichnen. Überlege selbst, wie es dann auf der linken Seite aussehen müsste!

34.2 Einfaches Beispiel eines ER-Diagramms



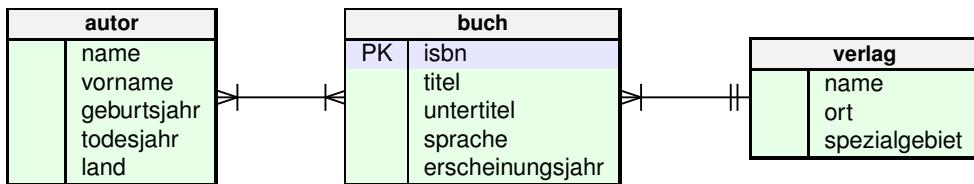
Abbildung 34.3: Das ER-Diagramm zu unserer Haustierdatenbank

Aus diesem ER-Diagramm erkennen wir, dass jeder Besitzer beliebig viele (auch keine!) Einträge in der Beziehungstabelle haben kann. Das gleiche gilt auch für die Tiere. Andererseits sieht man, dass zu jedem Eintrag in der Beziehungstabelle *genau* ein Besitzer und ein Tier zugehörig ist.

34.3 ER-Modell – Übung

Aufgabe 1: Interpretation eines ER-Diagramms

- a) Interpretiere das folgende Entity-Relationship-Diagramm:



Achtung: Das abgebildete ER-Diagramm ist unvollständig. Es fehlen – unter anderem – einige Schlüsselfelder! Welche?

Vervollständige das ER-Diagramm!

- b) Erstelle ein SQL-Skript buecherei.sql, mit dem diese Datenbank erzeugt werden kann.

Aufgabe 2: Kardinalitäten

Welche Art von Beziehung besteht zwischen folgenden Entitätstypen?

Achtung: Entitätstyp (also Tabelle) bedeutet natürlich auch, dass es jeweils mehrere Entitäten (Datensätze) gibt! Von jeweils beiden genannten Entitätstypen ...

- a) Mann, Frau (Beziehung: „verheiratet“)
- b) Lehrer, Schüler
- c) Esstisch, Stuhl
- d) Tutor, Tutand
- e) Schauspieler, Kinofilm (Beziehung: „spielen in“)

Aufgabe 3: Datenbankentwurf

- a) Entwirf ein ER-Diagramm für einen Video-Verleih:

Für die Videos sollen der Titel, der Regisseur, das Erscheinungsjahr und eine Identifikationsnummer abgespeichert werden. Für jeden Kunden wird die vollständige Adresse abgespeichert. Ein Video darf maximal zwei Wochen ausgeliehen werden. Danach ist eine einmalige Verlängerung um weitere zwei Wochen möglich. Es ist wichtig, dass die Informationen über die Ausleihe nach Rückgabe eines Videos nicht gelöscht werden, damit man im Nachhinein überprüfen kann, wer wann welches Video ausgeliehen hatte.

- b) Erstelle ein SQL-Skript videothek.sql, mit dem diese Datenbank erzeugt werden kann.

Aufgabe 4: Vergleich mit UML

Vergleiche das Entity-Relationship-Modell mit dem Modell objektorientierter Programmiersprachen, die man mit UML-Diagrammen entwirft.

- a) Setze die Begriffe aus UML mit dem ER-Modell in Beziehung:

| Begriff in UML | Entsprechender Begriff im ER-Modell |
|------------------------|-------------------------------------|
| Klasse | |
| Objekt | |
| Attribut | |
| Methode | |
| (normale) Beziehung | |
| Aggregations-Beziehung | |
| Vererbungs-Beziehung | |

- b) Was sind die wesentlichsten Unterschiede zwischen dem ER-Modell und dem UML-Modell?

35 Normalisierung

Die Normalisierung ist ein Prozess, der zum Entfernen von Entwurfsfehlern aus einer Datenbank benutzt wird.

Ziel dabei ist es, *Datenredundanzen* zu vermeiden. Mit anderen Worten: Eine Information (etwa die Telefonnummer eines Kunden) sollte nur an einer Stelle in der Datenbank vorliegen. Andernfalls ergeben sich folgende Nachteile:

- Erhöhter Pflegeaufwand: Sollte sich die Telefonnummer des Kunden ändern, dann muss man sie an mehreren Stellen in der Datenbank korrigieren.
- Drohende *Inkonsistenzen* (auch als *Anomalien* bezeichnet): Wenn es bei einer Änderung versäumt wurde bzw. misslang tatsächlich an allen Stellen das Datum (in unserem Beispiel: die Telefonnummer des Kunden) korrekt zu aktualisieren, dann enthält die Datenbank anschließend widersprüchliche Information. Und das ist fast noch schlimmer als Datenverlust! Den Datenverlust bemerkt man üblicherweise (und hat dann hoffentlich ein nicht all zu altes Backup), aber inkonsistente Daten können lange unbemerkt bleiben und in dieser Zeit schwere Folgeschäden verursachen
- Erhöhter Ressourcenbedarf: Speicherbedarf (auf jeden Fall) und Laufzeitverhalten (oft, aber nicht immer) der Datenbank werden durch redundante Datenhaltung negativ beeinflusst.

Der Normalisierungsprozess besteht aus dem Aufteilen der Tabellen in immer kleinere Tabellen, die zusammen einen besseren Entwurf ergeben. Der Datenbank-Entwurf wird nacheinander zuerst in die 1. Normalform, dann in die 2. Normalform und zuletzt in die 3. Normalform gebracht. Jede Normalform legt eine Reihe von Regeln fest, die eine Datenbank erfüllen soll. Die höheren Normalformen schließen dabei die Regeln der darunter liegenden Normalformen mit ein.

35.1 Erster Entwurf einer Tabelle

In den folgenden Tabellen kennzeichnen kursiv gesetzte Attribute die Zugehörigkeit des Attributs zum Primärschlüssel.

| cd_id | album | gründungsjahr | titelliste |
|--------------|--|----------------------|--|
| 4711 | Anastacia – Not That Kind | 1999 | 1. Not That Kind, 2. I'm Outta Love, 3. Cowboys & and Kisses |
| 4712 | Jefferson Airplane – Surrealistic Pillow | 1965 | 1. White Rabbit |
| 4713 | Anastacia – Freak of Nature | 1999 | 1. Freak of Nature, 2. Paid my Dues, 3. Overdue Goodbye |

Welche Datenbank-Probleme bringt diese Form der Tabelle mit sich?

35.2 Erste Normalform

Die Feldinhalte müssen frei von *Wiederholungsgruppen* sein (ein Datenfeld darf keine Liste von gleichartigen Werten beinhalten). In jeder Zeile eines Datenfeldes darf also nur ein Wert stehen.

Außerdem müssen die Feldinhalte *atomar* sein. Damit meint man, dass die Information in den einzelnen Spalten der Tabelle nicht weiter aufteilbar sein dürfen. Etwa wäre ein Feld *adresse*, welches die Komplette Anschrift enthält *nicht* atomar.

Unser Beispiel oben enthält sowohl eine Wiederholungsgruppe (das Feld *titelliste*) als auch zwei Felder, welche nicht atomar sind (*album* und wiederum *titelliste*).

Geänderte Tabelle (erfüllt die Bedingungen der ersten Normalform):

| cd_id | album | interpret | gründungsjahr | track | titel |
|-------|---------------------|--------------------|---------------|-------|------------------|
| 4711 | Not That Kind | Anastacia | 1999 | 1 | Not That Kind |
| 4711 | Not That Kind | Anastacia | 1999 | 2 | I'm Outta Love |
| 4711 | Not That Kind | Anastacia | 1999 | 3 | Cowboys & Kisses |
| 4712 | Surrealistic Pillow | Jefferson Airplane | 1965 | 1 | White Rabbit |
| 4713 | Freak of Nature | Anastacia | 1999 | 1 | Freak of Nature |
| 4713 | Freak of Nature | Anastacia | 1999 | 2 | Paid my Dues |
| 4713 | Freak of Nature | Anastacia | 1999 | 3 | Overdue Goodbye |

(der Primärschlüssel der Tabelle setzt sich zusammen aus den Feldern cd_id und track)

35.3 Zweite Normalform

Zunächst zwei Definitionen:

Ein Attribut einer Tabelle wird als *funktional abhängig* vom Primärschlüssel bezeichnet, wenn sich aus dem Wert des Primärschlüssels eindeutig auf den Wert dieses Attributs schließen lässt (Bei Funktionen gilt: zu jedem x-Wert gehört genau ein y-Wert). Das klingt zwar etwas kompliziert, ist aber eigentlich eine Selbstverständlichkeit: Wäre diese funktionale Abhängigkeit nämlich für irgendein Attribut der Tabelle nicht gegeben, dann wäre der Primärschlüssel kein Primärschlüssel!

Primärschlüssel können sich aber auch aus mehreren Attributen zusammen setzen. In diesem Fall benutzt man den Begriff der *vollfunktionalen Abhängigkeit*, wenn ein Attribut tatsächlich erst durch die Kombination aller am Primärschlüssel beteiligten Attribute eindeutig identifiziert werden kann.

Zweite Normalform: Zusätzlich zu den Bedingungen der ersten Normalform muss jedes nicht dem Schlüssel angehörende Attribut vollfunktional vom Primärschlüssel abhängig sein.

Der Interpret ist z.B. von der cd_id abhängig, d.h. jeder cd_id ist eindeutig ein Interpret zugeordnet. Der Track (der ja Teil des Primärschlüssels der Tabelle ist) wird also gar nicht gebraucht um den Interpreten zu bestimmen! Gleiches gilt übrigens auch für die Attribute album und gründungsjahr. Diese drei Attribute sind nicht *vollfunktional* vom zusammengesetzten Primärschlüssel der Tabelle abhängig.

Geänderte Tabellenstruktur (erfüllt die Bedingungen der zweiten Normalform):

Tabelle cd:

| cd_id | album | interpret | gründungsjahr |
|-------|---------------------|--------------------|---------------|
| 4711 | Not That Kind | Anastacia | 1999 |
| 4712 | Surrealistic Pillow | Jefferson Airplane | 1965 |
| 4713 | Freak of Nature | Anastacia | 1999 |

Tabelle stück (der Primärschlüssel der Tabelle setzt sich zusammen aus den Feldern cd_id und track):

| cd_id | track | titel |
|-------|-------|------------------|
| 4711 | 1 | Not That Kind |
| 4711 | 2 | I'm Outta Love |
| 4711 | 3 | Cowboys & Kisses |
| 4712 | 1 | White Rabbit |
| 4713 | 1 | Freak of Nature |
| 4713 | 2 | Paid my Dues |
| 4713 | 3 | Overdue Goodbyes |

Welche Schwachstellen bestehen in diesem Datenbankschema noch?

35.4 Dritte Normalform

Zusätzlich zu den Bedingungen der ersten und zweiten Normalform, dürfen die Attribute, die nicht Teil des Primärschlüssels sind, nicht funktional voneinander abhängig sein.

Das Gründungsjahr ist vom Interpret, der bisher kein Schlüsselattribut ist, funktional abhängig.

Das Problem ist hierbei wieder Datenredundanz. Wird zum Beispiel eine neue CD mit einem existierenden Interpreten eingeführt, so wird das Jahr der Gründung redundant gespeichert.

Die Tabellen werden also noch einmal aufgespalten:

Tabelle cd:

| cd_id | album | cd_interpret_id |
|-------|---------------------|-----------------|
| 4711 | Not That Kind | 311 |
| 4712 | Surrealistic Pillow | 312 |
| 4713 | Freak of Nature | 311 |

Tabelle interpret:

| interpret_id | name | gründungsjahr |
|--------------|--------------------|---------------|
| 311 | Anastacia | 1999 |
| 312 | Jefferson Airplane | 1965 |

Tabelle stück:

| cd_id | track | titel |
|-------|-------|------------------|
| 4711 | 1 | Not That Kind |
| 4711 | 2 | I'm Outta Love |
| 4711 | 3 | Cowboys & Kisses |
| 4712 | 1 | White Rabbit |
| 4713 | 1 | Freak of Nature |
| 4713 | 2 | Paid my Dues |
| 4713 | 3 | Overdue Goodbyes |

Zwar lässt sich auch jetzt wieder das Gründungsjahr des Interpreten aus dem Namen des Interpreten ableiten (Tabelle interpret), aber es droht keine Redundanz mehr: Sollte ein weiteres Album eines Interpreten der Datenbank hinzugefügt werden, muss dafür kein neuer Eintrag in der Tabelle interpret erzeugt werden: Das Feld cd_interpret_id in der Tabelle cd ist jetzt ein Fremdschlüssel, der auf den Primärschlüssel der Tabelle interpret verweist.

35.5 Höhere Normalformen

Es existieren noch höhere Normalformen (vierte, fünfte, usw.), die jedoch hauptsächlich akademisch interessant sind. Sie werden in der Praxis nicht eingesetzt, da eine weitere Normalisierung einen zu hohen Aufwand bedeuten würde. Die dritte Normalform genügt zur Beseitigung typischer Datenredundanzen.

35.6 Lesetipp

[http://de.wikipedia.org/wiki/Normalisierung_\(Datenbank\)](http://de.wikipedia.org/wiki/Normalisierung_(Datenbank))

35.7 Normalisierung – Übung

Aufgabe 1

Die folgende Tabelle beschreibt in welcher Abteilung die Mitarbeiter einer Firma arbeiten und an welchem Projekt sie wie viele Stunden gearbeitet haben:

| personal_id | name | vorname | abt_id | abteilung | proj_id | proj-beschreibung | zeit |
|--------------------|-------------|----------------|---------------|------------------|----------------|---|--------------------|
| 1 | Lorenz | Sophia | 1 | Personal | 2 | Verkaufspromotion | 83 |
| 2 | Hohl | Tatjana | 2 | Einkauf | 3 | Konkurrenzanalyse | 29 |
| 3 | Willschrein | Theodor | 1 | Personal | 1, 2, 3 | Kundenumfrage, Verkaufspromotion, Konkurrenzanalyse | 140, 92, 110 |
| 4 | Richter | Hans-Otto | 3 | Verkauf | 2 | Verkaufspromotion | 67 |
| 5 | Wiesenland | Brunhilde | 2 | Einkauf | 1 | Kundenumfrage | 160 |

Wandle das Datenbankschema nacheinander in die erste, zweite und dritte Normalform um.

Aufgabe 2

Gegeben ist die Tabelle bestellung mit den folgenden Spalten:

kunde_id, kunde_name, kunde_adresse, bestellung_id, bestellung_datum, artikel_id, artikel_name, artikel_anzahl.

In einer Bestellung können mehrere Artikel gleichzeitig bestellt werden. Die Artikel-Anzahl gibt an, wie viele Artikel einer Sorte in einer Bestellung angefordert werden.

Überföhre das Datenbankschema in die dritte Normalform.

36 SQL – Wiederholung

36.1 Aufgaben zur Wiederholung

Aufgabe 1: Datenbankentwurf

Für das Warenlager eines großen Möbelhauses soll eine Datenbank entworfen werden. Ein Mitarbeiter des Möbelhauses erklärt dir, welche Informationen in der Datenbank abgespeichert werden sollen:

Jeder Artikel, den wir führen, besitzt eine Artikelnummer. In der Datenbank soll jedoch auch die Bezeichnung des Artikels abgespeichert werden, da nicht alle Mitarbeiter mit der Nummer etwas anfangen können. Außerdem muss abgespeichert werden, wie viele Stücke eines Artikels wir zur Zeit auf Lager haben. Für den Fall, dass ein Artikel nachbestellt werden muss, muss aus der Datenbank ersichtlich sein, von welchem Lieferanten wir den Artikel beziehen. Jeder Artikel wird von genau einem Lieferanten bezogen.

Die meisten Lieferanten liefern uns selbstverständlich mehrere Artikel. Von dem Lieferanten müssen wir den Firmennamen, die Telefonnummer und die Adresse wissen.

Die Datenbank soll auch eine Kundenverwaltung enthalten. Alle Kunden sollen mit Vorname, Nachname und Adresse gespeichert werden. Falls ein Kunde eine Bestellung vornimmt, soll aus der Datenbank ersichtlich sein, wie viele Stücke eines Artikels er bestellt hat, wann die Bestellung aufgegeben wurde und welchen Status die Bestellung hat (erfolgreich geliefert, noch offen, konnte nicht geliefert werden).

- a) Entwirf ein ER-Diagramm für die Datenbank. Die Adresse eines Kunden und eines Lieferanten braucht nicht detailliert in einzelne Attribute unterteilt werden.
- b) Berücksichtige, dass verschiedene Kunden mit der gleichen Adresse existieren können. Musst du deinen Datenbankentwurf korrigieren, um die Bedingungen der dritten Normalform zu erfüllen?

Aufgabe 2: Normalformen

Erkläre die Bedeutung der drei Normalformen, in denen sich eine Datenbank befinden kann.

Aufgabe 3: Urlaubs-Datenbank

In einer großen Firma beschließen einige Mitarbeiter, ihren nächsten Sommerurlaub gemeinsam zu verbringen. Um alle Wünsche unter einen Hut bringen zu können, legt Helga Schneider eine Datenbank an.

In der Datenbank befinden sich vier verschiedene Vorschläge für Reiseziele. Jeder Mitarbeiter konnte angeben, welche der vier Reisen er mitmachen würde. Außerdem sollte jeder Mitarbeiter die Zeiten angeben, zu denen er an der Reise teilnehmen kann.

- a) Im Kurs-Repository findest du die Datei `urlaubsdatenbank.sql`. Erzeuge mit Hilfe dieser Datei eine neue Datenbank auf deinem Computer. Analysiere die Datenbank und zeichne ein ER-Diagramm, das die Struktur der Datenbank abbildet.
- b) Die Datenbank befindet sich in der ersten Normalform. Begründe, wieso die erste Normalform erfüllt wird, und wieso die zweite Normalform nicht erfüllt wird.
- c) Nimm die nötigen Änderungen vor, um die Bedingungen der dritten Normalform zu erfüllen und speichere das geänderte SQL-Skript ab.
- d) Erzeuge SQL-Kommandos, die die nachfolgenden Aufgaben lösen (basierend auf dem von dir geänderten SQL-Skript der Urlaubs-Datenbank), und speichere sie in einer Datei.
 1. Ermittle, wie viele Mitarbeiter in der Datenbank eingetragen sind.
 2. Erstelle eine Liste, die angibt welche Mitarbeiter (Vorname, Nachname) sich für welches Reiseziel (Land) interessieren. Sortiere die Liste aufsteigend nach dem Land.

3. Zähle die Anzahl der Mitarbeiter, die sich für die einzelnen Reisen interessieren. Das Ergebnis soll eine Tabelle sein mit der Reise (Land und Beschreibung) und einer Spalte, die die Anzahl der Interessenten angibt.
4. Ermittle diejenigen Mitarbeiter mit Vorname und Nachname, die noch keine Urlaubszeiten angegeben haben.
5. Ermittle alle Mitarbeiter (Vorname und Nachname), die mit Helga Schneider in der selben Abteilung arbeiten. Helga Schneider darf nicht in der Liste erscheinen.
6. Liste die Namen (Vorname, Nachname) aller Mitarbeiter auf, die gerne einen Badeurlaub machen würden („Badeurlaub“ steht in der Spalte Beschreibung der Tabelle Reiseziel). In der Liste darf kein Name doppelt aufgeführt werden.
7. Zähle die Anzahl der Mitarbeiter, die gerne nach Spanien fahren würden. Das Ergebnis ist eine einzige Zahl.
8. Liste alle Mitarbeiter mit Vor- und Nachname auf, die sich für mehr als zwei Reiseziele interessieren.
9. Ermittle alle Mitarbeiter, die mit Kai Schneider ein Reiseziel gemeinsam haben, mit Vornamen und Nachnamen. Kai Schneider darf auch in der Liste erscheinen. Es dürfen jedoch keine Namen doppelt aufgelistet werden.

37 Datenbankzugriffe mit Java

Um mit einer Datenbank zu arbeiten, braucht man neben dem Datenbank-Server auch einen Client. Bisher hast du immer das SQL Explorer Plugin in Eclipse benutzt um mit der Datenbank zu arbeiten. Aber es gibt Alternativen. Etwa den einfachen MySQL-Kommandozeilen-Client oder auch das Tool MySQL Workbench.

Neben diesen und anderen verfügbaren Clients, kann man aber auch einen eigenen Client programmieren. Wenn du etwa im Reisebüro bist, läuft dort im Hintergrund sicher eine Datenbank. Auf diese wird über einen speziell für diese Aufgabe entwickelten Client zugegriffen. Solche Clients können Web-basiert sein (also im Browser laufen), oder auch ein normales Anwendungsprogramm. So wie man es etwa mit Java entwickeln könnte.

37.1 SQL-Package

Um Datenbankzugriffe mit Java ausführen zu können muss man das Package `java.sql` importieren:

```
import java.sql.*;
```

Alle nachfolgend beschriebenen Java-Befehle werden grundsätzlich mit einem try-catch-Block überwacht, weil beim Auftreten von Fehlern Exceptions geworfen werden.

37.2 Verbindung zur Datenbank aufbauen

Zunächst muss die Verbindung zur der gewünschten Datenbank (im Beispiel MeineDatenbank) hergestellt werden. Dabei muss auch der Name des Benutzers angegeben werden, der die Datenbank verwenden möchte. Wenn die Verbindung erfolgreich aufgebaut wurde, wird ein Objekt vom Typ `Connection` zurückgegeben.

```
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://localhost/MeineDatenbank", "root", "root");
```

Um Kommandos an die Datenbank schicken zu können, muss man sich anschließend vom `Connection`-Objekt ein Objekt vom Typ `Statement` besorgen.

```
Statement stmt = conn.createStatement();
```

Sowohl `Connection`- als auch `Statement`-Objekte sind Ressourcen, die man nach der Benutzung wieder frei geben sollte. Zu diesem Zweck bieten beide Klassen jeweils eine Methode `close()`.

Seit Java SDK7 ist es allerdings nicht mehr nötig (aber möglich), diese Ressourcen manuell frei zu geben. Statt dessen bietet sich das mit JDK7 neu eingeführte try-with-ressource an (welches sich im Übrigen nicht nur für Datenbank-Objekte, sondern auch für Datei-Objekte anbietet):

```
try (Initialisierung der Ressourcen) {
    Code, in dem es zu Exceptions kommen kann
} catch (Exceptions) {
    Fehlerbehandlung
}
```

Wenn nicht nur eine, sondern wie in unserem Beispiel mehrere Ressource initialisiert werden sollen, dann werden die einzelnen Initialisierungsanweisungen durch Semikolon von einander getrennt.

Die in den Runden Klammern nach dem Schlüsselwort `try` initialisierten Ressourcen werden automatisch mit dem Verlassen des `try-catch`-Blocks geschlossen.

37.3 Datenbankabfragen

Datenbankabfragen kann man mit der Methode `executeQuery()` des `Statement`-Objekts stellen:

```
public ResultSet executeQuery(String sql) throws SQLException
```

Als Parameter übergibt man einen String mit einer SQL-Select-Abfrage. Zurückgegeben wird ein Objekt vom Typ `ResultSet`, das die Ergebnis-Datensätze enthält. Beispiel:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM tier WHERE geschlecht='w'");
```

Das `ResultSet`-Objekt besitzt eine Methode `next()`, mit der alle Datensätze der Ergebnismenge schrittweise durchlaufen werden können:

```
boolean next()
```

Nach dem Aufruf von `executeQuery()` steht der Satzzeiger noch vor dem ersten Datensatz. Durch einen Aufruf der Methode `next()` wird der Zeiger auf den ersten Datensatz gesetzt, sofern die Ergebnismenge nicht leer ist. Falls ein (weiterer) Datensatz gefunden wurde, gibt `next()` `true` zurück. Wenn kein weiterer Datensatz existiert, erhält man den Rückgabewert `false`.

Nachdem man sich mit `next()` einen Datensatz besorgt hat, kann man mit `getXXX()`-Methoden, auf die Spalten des Datensatzes zugreifen. Es gibt eine `getXXX()`-Methode für alle verschiedenen Datentypen, z.B.: `getBoolean()`, `getInt()`, `getString()`, usw.

Von jeder `getXXX()`-Methode gibt es zwei Varianten. Man kann entweder die Nummer der gewünschten Spalte als Parameter angeben (die erste Spalte hat die Nummer 1) oder man kann als Parameter den Spaltennamen angeben. Beispiel:

```
while (rs.next()) { // alle Datensätze der Reihe nach durchgehen
    String name = rs.getString(2); // der Name steht in der zweiten Spalte
    int tiernr = rs.getInt("tier_id"); // holt den Inhalt der Spalte tier_id als int
}
```

37.4 Datenbankänderungen

Für Datenbankänderungen mit den SQL-Anweisungen `INSERT INTO`, `UPDATE` oder `DELETE FROM` besitzt das `Statement`-Objekt die Methode `executeUpdate()`:

```
public int executeUpdate(String sql) throws SQLException
```

`executeUpdate()` erhält genau wie `executeQuery()` die SQL-Anweisung als Parameter vom Typ `String`. Im Erfolgsfall wird die Anzahl der geänderten Datensätze zurückgegeben. Beispiel:

```
int anzahl = stmt.executeUpdate("DELETE FROM tier WHERE name='Mausi'");
```

37.5 Beispiel

```
1 import java.awt.*;
2 import java.sql.*;
3 import javax.swing.*;
4 import javax.swing.border.EmptyBorder;
5
6 public class Tierdatenbank extends JFrame {
7     private static final int WIDTH = 250;
8     private static final int HEIGHT = 200;
9     private DefaultListModel<String> tierliste = new DefaultListModel<String>();
10
11    public Tierdatenbank(final String title) {
```

```
12     super(title);
13     createGUI();
14     datenbankAbfrage();
15 }
16
17 public void datenbankAbfrage() {
18     try {
19         Connection conn = DriverManager.getConnection(
20             "jdbc:mysql://localhost/haustier", "root", "root");
21         Statement stmt = conn.createStatement()
22     ) {
23         ResultSet rs = stmt.executeQuery("SELECT name, tierart FROM tier");
24         while (rs.next()) {
25             tierliste.addElement(rs.getString("name") + ", " + rs.getString("tierart"));
26         }
27     } catch (SQLException e) {
28         JOptionPane.showMessageDialog(this, e.getMessage(),
29             "Fehlermeldung", JOptionPane.ERROR_MESSAGE);
30         e.printStackTrace();
31     }
32 }
33
34 private void createGUI() {
35     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
36     JPanel contentPane = new JPanel();
37     contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
38     contentPane.setLayout(new BorderLayout(0, 0));
39     contentPane.setPreferredSize(new Dimension(WIDTH, HEIGHT));
40     setContentPane(contentPane);
41     JLabel lblTierListe = new JLabel();
42     lblTierListe.setText("Liste aller Tiere:");
43     contentPane.add("North", lblTierListe);
44     JList<String> listTiere = new JList<String>(tierliste);
45     contentPane.add("Center", listTiere);
46     pack();
47     setLocationRelativeTo(null);
48     setResizable(true);
49     setVisible(true);
50 }
51
52 public static void main(final String[] args) {
53     EventQueue.invokeLater(new Runnable() {
54         public void run() {
55             try {
56                 new Tierdatenbank("Haustier-Datenbank");
57             } catch (Exception e) {
58                 e.printStackTrace();
59             }
60         }
61     });
62 }
63 }
```

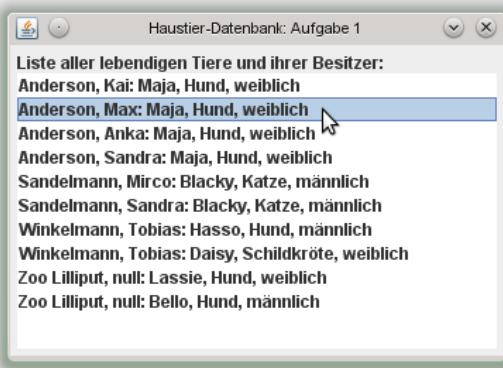
37.6 Programmierübung zur Haustierdatenbank

Aufgabe 1: Ausgabe mit einer JList-Komponente

Zeige in einer JList-Komponente alle Besitzer mit ihren Tieren an, sofern die Tiere noch lebendig sind. Die Ausgabe soll nach dem folgenden Schema erfolgen:

Besitzer-Nachname, Besitzer-Vorname: Tiername, Tierart, Tier-Geschlecht

Sortiere die Ausgabe nach dem Nachnamen des Besitzers.



Aufgabe 2: Datensätze durchblättern

Im Fenster werden jeweils die Daten eines Tieres angezeigt. Zu Beginn werden die Daten des Tieres mit der Tiernummer 1 geladen. Über einen „Vorwärts“-Button gelangt der Benutzer zu den Daten des Tieres mit der nachfolgenden Tiernummer. Über einen „Rückwärts“-Button gelangt der Benutzer zu den Daten des Tieres mit der vorhergehenden Nummer. Wenn es zu einer id keinen Datensatz gibt, dann soll die id mit anderweitig leeren Textfeldern angezeigt werden.

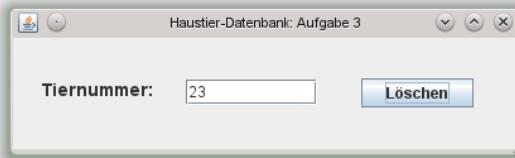
Die Button Vorwärts und Zurück werden deaktiviert, wenn es keinen vorherigen bzw. nachfolgenden Datensatz gibt. Dazu muss zu Beginn des Programms die Anzahl der vorhandenen Tiere abgefragt werden.

| | | | |
|-------------|---------------|---------------|------------|
| id: | 2 | Name: | Daisy |
| Tierart: | Kanarienvogel | Geschlecht: | weiblich |
| Geburtstag: | 1996-12-06 | Todestag: | 2004-08-17 |
| vor | | zurück | |

Alle Textfelder sollen nicht-editierbar sein.

Aufgabe 3: Tiere löschen

Der Benutzer kann die Tiernummer eines Tieres eingeben, das gelöscht werden soll. Wenn der Benutzer den „Löschen“-Button drückt, erscheint zunächst eine Sicherheitsabfrage (z.B. „Wollen Sie das Tier mit der Nummer 2 wirklich löschen?“). Nach einer positiven Bestätigung werden der Tiereintrag und alle Verweise auf das Tier in der Beziehungstabelle gelöscht.



Achtung: Man kann ein Tier nicht löschen, wenn es in der Beziehungstabelle referenziert ist (durch das Löschen würde in der Beziehungstabelle ansonsten ein Toter Verweis entstehen. Man sagt: *Die referentielle Integrität würde verletzt.*)

Es gibt nun zwei Möglichkeiten: entweder müssen zunächst alle Einträge in der Tabelle `beziehung`, die sich auf das zu löschende Tier beziehen entfernt (gelöscht) werden, oder man erweitert die Definition des Fremdschlüssels wie folgt:

```
FOREIGN KEY (beziehung_tier_id) REFERENCES tier (tier_id) ON DELETE CASCADE
```

Durch den Zusatz `ON DELETE CASCADE` wird MySQL angewiesen auch die betroffenen Einträge in der Beziehungstabelle zu löschen, wenn ein Tier gelöscht wird. Dieser Automatismus ist zum einen bequem, zum anderen aber auch höchst gefährlich, weshalb die Verwendung dieser Technik in vielen Unternehmen auch verboten ist.

Aufgabe 4: Neue Tiere hinzufügen

Im Fenster können die Daten eines neuen Tieres angelegt werden. Es gibt Eingabefelder für jeden Datumswert und einen „Hinzufügen“-Button. Wenn der Benutzer auf den Button drückt, wird ein neues Tier mit den von ihm eingegebenen Daten angelegt. Falls der Benutzer keinen Todestag angegeben hat, soll für den Todestag der Wert `NONE` eingefügt werden.



37.7 Programmierung eines Terminplaners

Es soll ein einfacher Terminplaner erstellt werden.

Aufgabe 1: Erzeugen einer geeigneten Datenbank

Erzeuge mit Hilfe von SQL-Befehlen manuell eine geeignete Datenbank für den Terminplaner. Die Datenbank soll nur eine einzige Tabelle enthalten mit den folgenden Spalten:

- Nummer des Eintrags (Primärschlüssel; wird automatisch vom System generiert)
- Datum
- Zeit
- Text (in dieser Spalte beschreibt der Benutzer seinen Termin)

Aufgabe 2: Programmierung des Terminplaners

Programmiere ein Java-Programm mit dem der Benutzer neue Termine eingeben und löschen kann (siehe Abbildung 37.1).

Im oberen Teil des Anwendungsfensters werden in einer `JList`-Komponente alle vorhandenen Termine in folgendem Format angezeigt:

Nummer) Datum, Zeit, Text

Wenn neue Einträge hinzugefügt werden oder alte Einträge gelöscht werden, wird die Liste automatisch aktualisiert.

Wenn der Benutzer auf den „Einfügen“-Button drückt, wird ein neuer Eintrag mit den vom Benutzer angegebenen Werten eingefügt. Die Korrektheit der Benutzereingaben muss nicht überprüft werden. Nachdem der Datensatz erfolgreich eingefügt wurde, gibt das Programm eine Erfolgsmeldung mit einem `showMessageDialog()` aus.

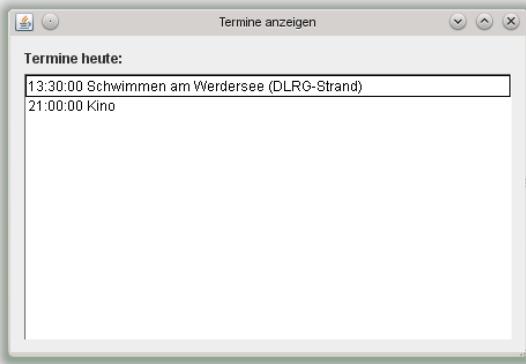
Wenn der Benutzer auf den „Löschen“-Button drückt, wird der Eintrag mit der vom Benutzer eingegebenen Nummer gelöscht. Die Korrektheit der Benutzereingabe muss auch hier nicht überprüft werden. Das Programm gibt mit einem `showMessageDialog()` eine Erfolgsmeldung zurück, wenn der Datensatz erfolgreich gelöscht wurde, oder eine Fehlermeldung, falls der Datensatz nicht existiert hat.



Abbildung 37.1: Aufgabe 2: So soll der Terminplaner aussehen

Aufgabe 3: Ein Programm zum Anzeigen der aktuellen Termine

Programmiere ein Java-Programm, das die aktuellen Termine des Benutzers anzeigt. Dieses Programm könnte später zu den Autostart-Programmen gepackt werden und so bei jedem Start des Computers automatisch aufgerufen werden. Die Oberfläche des Termin-Anzeigers soll so aussehen:



Der Termin-Anzeiger zeigt die Termine des aktuellen Tages in einer `JList`-Komponente an. Die Termine werden aufsteigend nach der Zeit sortiert.

Außerdem löscht er unbemerkt vom Benutzer alle Termine aus der Datenbank, die vor dem aktuellen Datum liegen.

Gib im ersten Schritt das heutige Datum ('2018-01-10') fest als String in die benötigten SQL-Kommandos ein. Hole dir im zweiten Schritt die aktuelle System-Zeit von der Java-Klasse `System` mit der statischen Methode `currentTimeMillis()` (also durch Aufruf von `System.currentTimeMillis()`). Übergib diese Zeit an ein Objekt der Klasse `Date` aus dem Package `java.sql` (Achtung: es gibt auch eine Klasse `Date` im Package `java.util!`) und lass dir von der Klasse `Date` ein `String`-Objekt mit dem aktuellen Datum generieren.

37.8 SQL-Injection

Unter *SQL-Injection* versteht man das Einschleusen von SQL-Befehlen in eine Datenbankanwendung durch einen Angreifer. Dabei handelt es sich um eine der populärsten Angriffsmethoden (nicht nur) im Internet.

Der Angreifer nutzt dabei aus, dass bei Benutzereingaben Metazeichen wie etwa das Hochkomma oder auch Kommentarzeichen nicht überprüft bzw. maskiert werden.

Auf diese Weise kann er

- direkt oder indirekt Informationen über die Struktur der Datenbank erhalten.
- Zugang zu einem System erlangen.
- Daten in der Datenbank verändern.
- Daten, Tabellen oder Datenbanken löschen.
- Dateien im Dateisystem des Datenbankservers erzeugen oder überschreiben.

Ein hoffentlich anschauliches Beispiel findest du im Kursrepository, im Ordner 37_SQL_Datenbankzugriffe findet ihr einen weiteren Ordner: `sqlinjection`. Wenn ihr dessen Inhalt in euer eigenes Repository kopiert, könnt ihr mit dem Programm `OnlineBanking` ein wenig „spielen“. Zuvor müsst ihr allerdings das SQL-Skript `online-banking.sql` einmal ausführen.

In der kleinen Datenbank ist genau ein Datensatz vorhanden. Das Konto mit der Kontonummer 1234567 ist mit dem Passwort 1234 versehen. Kontoinhaber ist Frau Monika Mustermann. Der aktuelle Kontostand beträgt -703,28 €.

Mit dem kleinen Programm könnt ihr nur zwei Sachen tun: Ihr könnt euch anmelden und ihr könnt das Passwort ändern. Wenn ihr angemeldet seid, wird euch der aktuelle Kontostand angezeigt.

In der Datei `Angriffsmöglichkeiten.txt` öffnet, findet ihr einige Beispiel dafür, wie ihr mit Hilfe von SQL-Injections das Programm missbrauchen könnt!

Wenn du es dir genauer anschaust, wirst du erkennen, warum die beschriebenen Angriffe erfolgreich sind.

Glücklicherweise, kannst du als Java-Programmierer solche Angriffe leicht abwehren. Dazu musst du nur die Klasse `PreparedStatement`, statt der Klasse `Statement` verwenden:

Aus

```
Statement stmt = conn.createStatement();
String kontonummer = tfKontonummer.getText();
String passwort = tfPasswort.getText();
String sql = "SELECT * FROM konto WHERE kontonummer = "
            + kontonummer + " AND passwort = '" + passwort + "'";
System.out.println(sql);
ResultSet rs = stmt.executeQuery(sql);
```

wird

```
String sql = "SELECT * FROM konto WHERE kontonummer = ? AND passwort = ?";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setString(1, tfKontonummer.getText());
ps.setString(2, tfPasswort.getText());
System.out.println(ps.toString());
ResultSet rs = ps.executeQuery();
```

Dem vorbereiteten SQL-Befehl mussten vor der Ausführung nur noch die passenden Argumente an Stelle der Fragezeichen verpasst werden.

Die Arbeit mit `executeUpdate()` funktioniert analog zu diesem Beispiel. Du kannst dir die entsprechend korrigierten Java-Klassen im Unterordner `preparedstatement` anschauen. Damit laufen die SQL-Injection Angriffe alle ins Leere.

38 Java mit SQL – Wiederholung

Klausurvorbereitung

Aufgabe 1: Umwandlung von Strings in Zahlen

- Schreibe den Text aus dem JTextField `eingabe` in eine String-Variablen und konvertiere ihn anschließend in eine Integer-Zahl.
- Was passiert, wenn in dem Textfeld gar keine Zahl stand? Erweitere den Code-Auszug so, dass gegebenenfalls eine Fehlermeldung auf der Konsole ausgegeben wird.

Aufgabe 2: String-Vergleich

- Gegeben sind die String-Variablen `text1` und `text2`. Frage ab, ob in beiden Variablen der selbe Text steht und gib auf der Konsole eine Erfolgsmeldung oder eine Fehlermeldung aus.
- Wie muss man den Code-Auszug verändern, wenn die Groß- und Kleinschreibung bei der Abfrage auf Gleichheit keine Rolle spielen soll?

Aufgabe 3: Programmierübung zur Haustier-Datenbank

Programmiere zur Übung noch einmal eine kleine Anwendung, die sich auf die Haustier-Datenbank bezieht. Verwende bei der Erstellung des Programms ausschließlich die in der Klausur erlaubten Merkblätter!



Die Textfelder zur Ausgabe des Vornamens und Nachnamens des Besitzers sollen nicht editierbar sein. In der JList-Komponente sollen nur die lebendigen Tiere des jeweiligen Besitzers aufgelistet werden. Falls ein Besitzer mit der angegebenen BesitzerNr nicht existiert, soll eine Fehlermeldung ausgegeben werden.

Nach dem Löschen eines Besitzers soll eine Erfolgs- oder Fehlermeldung ausgegeben werden. Der Inhalt der Textfelder und der JList-Komponente soll beim erfolgreichen Löschen eines Besitzers ebenfalls gelöscht werden.

Der Rest sollte selbsterklärend sein.

Aufgabe 4: KFZ-Werkstatt

Es soll eine Anwendung für die KFZ-Werkstätten erstellt werden, in der die zur Reparatur oder Wartung gegebenen Fahrzeuge verwaltet werden. In dieser Datenbank sollen sich nur Fahrzeuge und Aufträge befinden, die am aktuellen Tag noch bearbeitet werden müssen. Alle bearbeiteten Aufträge werden sofort aus der Datenbank gelöscht.

Ein Kollege von dir hat bereits die SQL-Anweisungen zur Erstellung einer geeigneten Datenbank (inklusive Testdaten) geschrieben. Du findest die SQL-Anweisungen in der Datei `werkstatt.sql` auf. Erzeuge mit Hilfe dieser Datei eine neue Datenbank auf deinem Computer.

- a) Analysiere die Datenbank und zeichne ein ER-Diagramm, das die Struktur der Datenbank darstellt.
- b) Gib an, in welcher Normalform sich die Datenbank `werkstatt` befindet. Begründe deine Antwort.
- c) Später soll ein Computerprogramm erstellt werden, das den Mitarbeitern der KFZ-Werkstatt eine einfache Bedienoberfläche für die Datenbank liefert. Formuliere als Vorbereitung dafür geeignete Datenbank-Abfragen, mit denen die folgenden Informationen aus der Datenbank `werkstatt` gewonnen werden können. Speichere die SQL-Abfragen in einer Datei auf dem USB-Stick ab.
 1. Liste Vor- und Nachname aller Mechaniker auf, die das Fahrzeug mit der Nummer 2 bearbeiten.
 2. Liste alle Fahrzeuge mit ihrer Fahrzeug-Nummer auf, die der Mechaniker Karl Sonntag zu bearbeiten hat.
 3. Liste Fahrzeug-Nummer und Typ derjenigen Fahrzeuge auf, die noch vor 16 Uhr fertig werden müssen.
 4. Ein Auftrag hat länger gedauert als erwartet und die versprochenen Zeiten können nicht eingehalten werden. Die Kunden müssen darüber informiert werden, dass sie ihre Fahrzeuge erst später abholen können. Liste alle Daten der Kunden auf, deren Fahrzeuge bis einschließlich 17 Uhr fertig werden sollten. Es sollen keine Kunden doppelt in der Liste erscheinen. Sortiere die Liste aufsteigend, zuerst nach dem Nachnamen und dann nach dem Vornamen der Kunden.
 5. Liste alle Fahrzeuge mit Fahrzeug-Nummer auf, bei denen zwei oder mehr Tätigkeiten durchgeführt werden müssen.
 6. Liste alle Fahrzeuge mit ihrer Fahrzeug-Nummer auf, die zur selben Zeit fertig werden müssen wie das Fahrzeug mit der Nummer 4. Das Fahrzeug mit der Nummer 4 soll nicht in der Liste erscheinen.

Aufgabe 5: Programmierung einer Software für ein Reiseunternehmen

Ein Reiseunternehmen möchte exklusive Studienreisen anbieten für Kleingruppen von maximal zwölf Personen. Du hast die Aufgabe, den Prototyp für die Software zur Buchung der Reisen zu programmieren.

- a) Die Datenbank zur Speicherung der angebotenen Reisen und der Buchungen wurde bereits von einem Kollegen erstellt. Im Kurs-Repository findest du die Datei `exklusiv_reisen.sql`. Erzeuge mit Hilfe dieser Datei eine neue Datenbank auf deinem Computer und analysiere sie. Erstelle ein ER-Diagramm, das die Struktur der Datenbank abbildet.
- b) Erläutere die Bedeutung der drei Normalformen und gib an, in welcher Normalform sich die Datenbank befindet.
- c) Erstelle ein Java-Programm, mit dem Reisen gebucht und storniert werden können. Abbildung 38.1 zeigt, wie die Oberfläche des Programms aussehen soll.

In der oberen JList-Komponente werden alle verfügbaren Reisen aufgelistet. Schema:

ReiseNr) Land, Anfangsdatum bis Enddatum

Falls in der Spalte ausgebucht der Wert ja gesetzt ist, wird zusätzlich die Zeichenkette ", ausgebucht" an den Eintrag angehängt.

Wenn man auf den Button *Reise-Liste* klickt, werden für die mit Vornamen und Nachnamen angegebene Person alle gebuchten Reisen in der unteren JList-Komponente aufgelistet. Es wird dabei vereinfacht davon

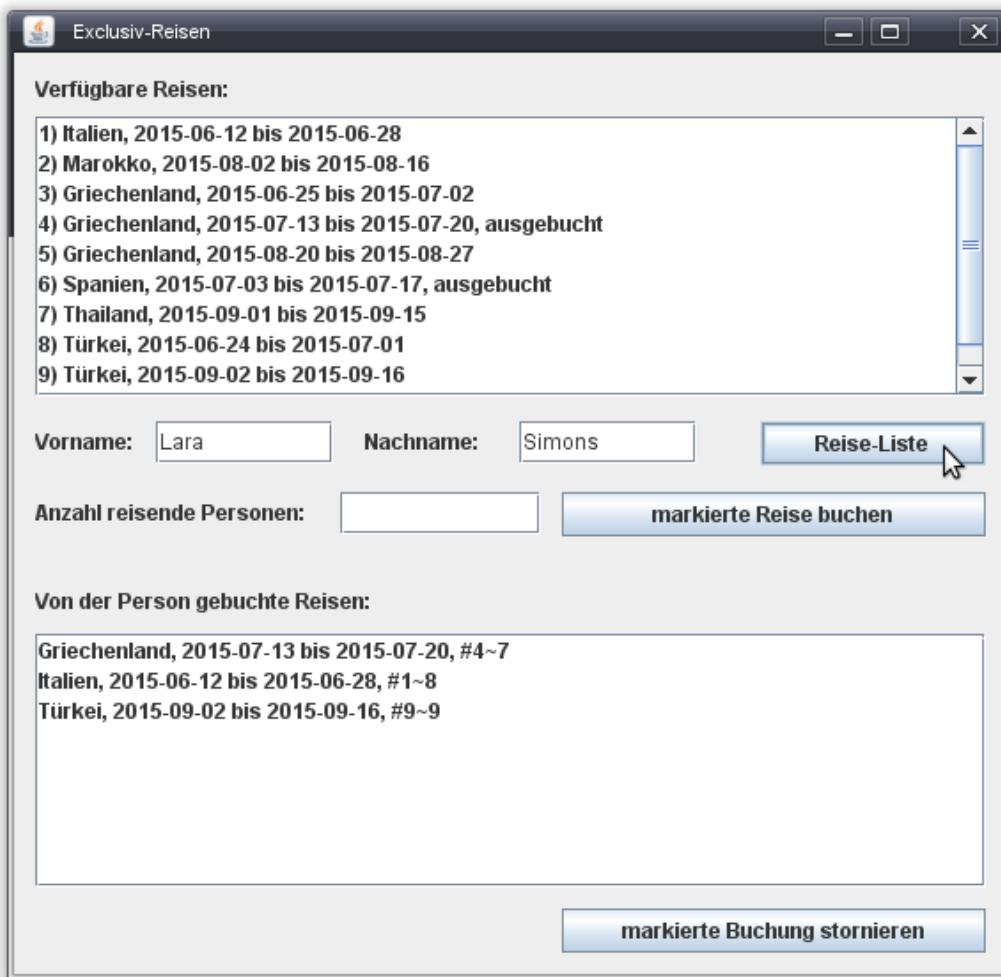


Abbildung 38.1: Die Programmoberfläche von „Exklusiv-Reisen“

ausgegangen, dass es keine zwei Personen mit dem gleichen Vor- und Nachnamen geben kann. Die Ausgabe der gebuchten Reisen erfolgt nach dem Schema:

Land, Anfangsdatum bis Enddatum, #ReiseNr~BuchungsNr

Falls keine gebuchten Reisen gefunden werden, bleibt die JList-Komponente leer. Es braucht keine Fehlermeldung ausgegeben werden.

Wenn man auf den Button *markierte Buchung stornieren* drückt, wird die in der unteren JList- Komponente selektierte Reise storniert. Falls keine Zeile in der unteren JList-Komponente markiert ist, wird eine entsprechende Fehlermeldung ausgegeben (z.B. "Bitte wählen Sie die Buchung aus, die storniert werden soll."). Andernfalls wird die Buchung mit der angegebenen BuchungsNr gelöscht. Außerdem wird in der Reise mit der angegebenen ReiseNr die Spalte ausgebucht auf nein gesetzt, weil ja jetzt auf jeden Fall wieder ein Reiseplatz verfügbar ist. Es muss nicht überprüft werden, ob die Reise zuvor tatsächlich ausgebucht war oder nicht. Der Inhalt beider JList-Komponenten soll nach dem Löschen automatisch aktualisiert werden. Eine Erfolgsmeldung muss nicht ausgegeben werden.

Wenn man auf den Button *markierte Reise buchen* drückt, wird für die mit Vornamen und Nachnamen angegebene Person die in der oberen JList-Komponente ausgewählte Reise für die angegebene Anzahl von Personen gebucht, sofern eine Buchung möglich ist. Dazu wird zunächst geprüft, ob der Benutzer alle Angaben korrekt vorgenommen hat. Falls eines der Textfelder leer ist oder im Anzahl-Textfeld keine gültige Zahl steht (z.B. ein Buchstabe oder ein Wert der kleiner als 1 ist), wird die Buchung mit einer entsprechenden Fehlermeldung abgebrochen. Die Buchung wird ebenfalls mit einer Fehlermeldung abgebrochen, wenn in der oberen JList-Komponente kein Eintrag selektiert wurde. Wenn die Eingaben korrekt sind, muss überprüft werden, ob für die Reise noch die gewünschte Anzahl von Plätzen verfügbar ist. Dies ist der Fall, wenn die Gesamtzahl der Personen in allen Buchungen, die sich auf die Reise beziehen, plus die Anzahl der gewünschten

Plätze kleiner oder gleich zwölf ist. Falls die Anzahl der verfügbaren Plätze nicht ausreicht, wird die Buchung mit einer Fehlermeldung abgebrochen; andernfalls wird ein neuer Eintrag in die Tabelle Buchung eingefügt.

Beachte dabei, dass die BuchungsNr von der Datenbank automatisch generiert werden soll. Nachdem der neue Eintrag in die Datenbank eingefügt wurde, wird die untere **JList**-Komponente aktualisiert, damit der Benutzer den Erfolg der Buchung sieht. Falls die Reise nach Durchführung der Buchung ausgebucht ist (also insgesamt zwölf Personen für die Reise gebucht sind), wird für die Reise die Spalte ausgebucht in der Reisetabelle auf ja gesetzt. In diesem Fall muss anschließend die obere **JList**-Komponente aktualisiert werden.

Aufgabe 6: Programmierung einer Software für einen Getränkehandel

Du hast die Aufgabe, den Prototyp für die Software eines Getränkehandels zu programmieren. Die Datenbank zur Speicherung der vorhandenen Getränke und der Getränke-Lieferanten wurde bereits von einem Kollegen erstellt. Du findest die SQL-Anweisungen in der Datei `getraenkehandel.sql` im Kurs-Repository. Erzeuge mit Hilfe dieser Datei eine neue Datenbank auf deinem Computer.

a) Analysiere die Datenbank. Erstelle ein ER-Diagramm, das die Struktur der Datenbank abbildet.

In der Tabelle `getränk` gibt es drei Spalten, die einer Erläuterung bedürfen:

- Die Spalte `anzahl` gibt die Anzahl der Getränkekisten an, die das Geschäft momentan vorrätig hat.
- Die Spalte `min_anzahl` gibt an, wie viele Kisten des Getränks sich mindestens im Lager befinden sollten. Sobald die Anzahl der Getränkekisten unter den Wert `min_anzahl` sinkt, ist eine Nachbestellung erforderlich.
- Die Spalte `max_anzahl` gibt an, wie viele Kisten des Getränks sich maximal im Lager befinden sollten. Wenn eine automatische Nachbestellung durchgeführt wird, werden so viele Kisten nachbestellt, dass die Summe der noch vorhandenen und der nachbestellten Kisten zusammen `max_anzahl` ergibt.

b) Erläutere die Bedeutung der drei Normalformen und gib an, in welcher Normalform sich die Datenbank befindet.

c) Erstelle ein Java-Programm für den Getränkehandel. Abbildung 38.2 zeigt, wie die Oberfläche des Programms aussehen soll.

In der Listkomponente werden alle Getränke aufgelistet, die das Geschäft führt. Die Ausgabe der Getränke-Daten erfolgt nach folgendem Schema:

`name: anzahl [min_anzahl; max_anzahl] L:getränk_lieferant_id`

Vor- und nach dem Teil in den eckigen Klammern sollten mehrere Leerzeichen eingefügt werden, damit die Ausgabe besser lesbar wird. Die Anzeige aller Getränke in der Listkomponente soll automatisch beim Start des Programms erfolgen. Außerdem soll die Anzeige nach dem Hinzufügen oder Abziehen von Getränkekisten aktualisiert werden.

Wenn neue Getränkekisten geliefert wurden, gibt der Benutzer den Namen des gelieferten Getränkes und die Anzahl der gelieferten Kisten in die Textfelder ein und drückt auf den Button *Hinzufügen*. Das Programm erhöht in der Datenbank die Anzahl der vorhandenen Kisten um den neuen Wert und aktualisiert die Ausgabe in der Listkomponente. Es muss nicht überprüft werden, ob die Benutzereingaben korrekt sind.

Wenn Getränkekisten verkauft wurden, gibt der Benutzer den Namen des verkauften Getränkes und die Anzahl der verkauften Kisten in die Textfelder ein und drückt auf den Button *Abziehen*. Das Programm erniedrigt in der Datenbank die Anzahl der vorhandenen Kisten um den angegebenen Wert und aktualisiert die Ausgabe in der Listkomponente. Es muss nicht überprüft werden, ob die Benutzereingaben korrekt sind.

Das Programm soll später einmal automatisch Briefe generieren können, in denen Getränke nachbestellt werden. Da dies nur ein Prototyp ist, werden die „Briefe“ in vereinfachter Form auf der Konsole ausgegeben. Eine Textzeile kannst du mit dem Kommando `System.out.println(...);` auf der Konsole ausgeben.

Wenn der Benutzer auf den Button *Nachbestellung* drückt, soll das im linken Textfeld angegebene Getränk in der im rechten Textfeld stehenden Anzahl bestellt werden. Die Korrektheit der Benutzereingaben braucht auch hier nicht überprüft zu werden. Die Ausgabe soll nach folgendem Schema erfolgen:

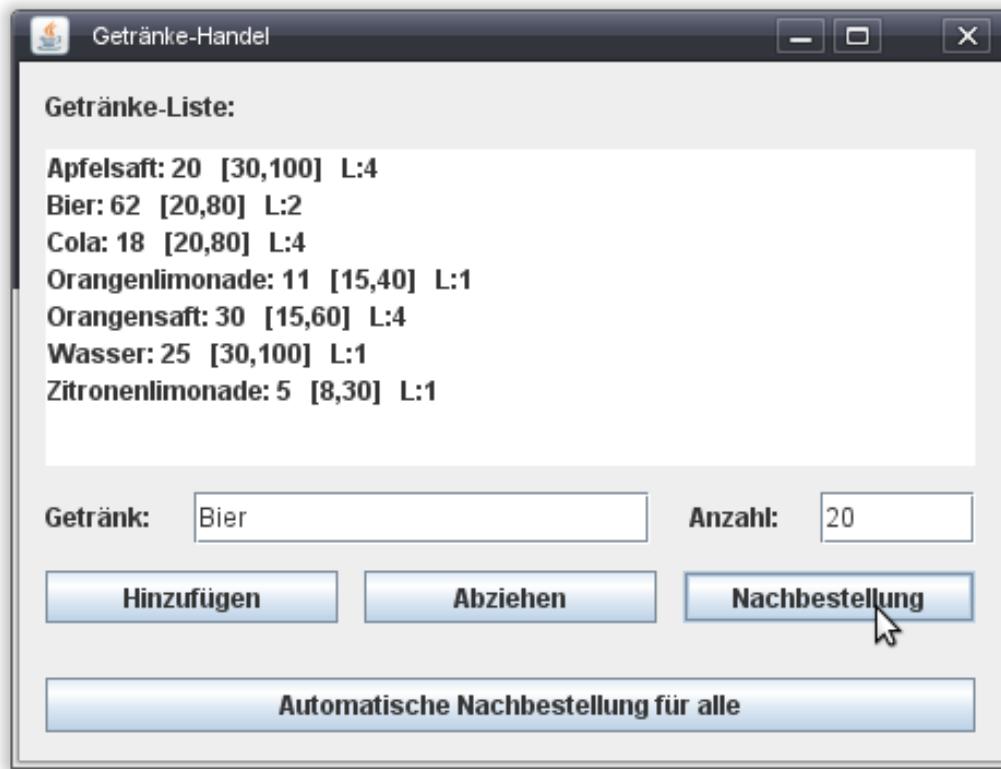


Abbildung 38.2: Die Programmoberfläche von „Getränke-Handel“

```
firma
straße
ort
Wir benötigen anzahl Kisten getränk.
```

Im abgebildeten Beispiel würde zum Beispiel folgender Text auf der Konsole ausgegeben werden:

```
Brauerei Hansen
Buchenweg 244
10352 Lummerstadt
Wir benötigen 78 Kisten Bier.
```

Wenn der Benutzer auf den Button *Automatische Nachbestellung für alle* klickt, soll eine automatische Nachbestellung für alle Getränke generiert werden, deren Anzahl unter den Wert `min_anzahl` gesunken ist. Die Ausgabe soll nach dem gleichen Schema wie bei einer manuellen Nachbestellung erfolgen. Alle Nachbestellungen, die an denselben Lieferanten gehen, sollen in einem „Brief“ zusammen gefasst werden. Leite die Automatische Nachbestellung durch einen geeigneten Anfangstext (z.B. „AUTOMATISCHE BESTELLUNG – ANFANG“) gefolgt von einer Leerzeile ein. Hinter der letzten Bestellung soll zuerst eine Leerzeile und dann ein End-Text (z.B. „AUTOMATISCHE BESTELLUNG – ENDE“) stehen. Zwischen zwei Nachbestellungen soll jeweils eine Leerzeile liegen (nicht mehrere!). Im abgebildeten Beispiel würde die Automatische Nachbestellung folgendermaßen aussehen:

AUTOMATISCHE BESTELLUNG – BEGIN

Limonaden-Mix GmbH
Karl-Hauser-Str. 33-35
32011 Quakenbrück
Wir benötigen 75 Kisten Wasser.
Wir benötigen 29 Kisten Orangenlimonade.
Wir benötigen 25 Kisten Zitronenlimonade.

Brauerei Hansen
Buchenweg 244
10352 Lümmersdorf
Wir benötigen 78 Kisten Bier.

Cola und Co. KG
Breitenweg 214-215
88123 Heiligenrode
Wir benötigen 80 Kisten Apfelsaft.
Wir benötigen 62 Kisten Cola.

AUTOMATISCHE BESTELLUNG – ENDE

Um die Nachbestellungen eines Lieferanten zusammenfassen zu können, kannst du dir der Reihe nach die Daten der Lieferanten mit den Nummern 1 bis 10 holen (Achtung: nicht alle Nummern sind vergeben!). Wenn ein Lieferant mit der entsprechenden Nummer existiert, holst du dir die Daten aller Getränke, die von diesem Lieferanten geliefert werden und überprüfst, ob Nachbestellungen nötig sind.

Beachte, dass die Anschrift eines Lieferanten nur ausgegeben werden darf, wenn es auch tatsächlich eine Nachbestellung für den Lieferanten gibt.

Anmerkung: Man kann auch auf das Hochzählen der Lieferanten-Nummer verzichten und direkt die Liste der vorhandenen Lieferanten durchgehen. Für diese Lösung muss man jedoch zwei verschiedene **SELECT**-Abfragen ineinander verschachteln. Beachte, dass eine Verschachtelung von zwei verschiedene Datenbankabfragen nur möglich ist, wenn man für beide Zugriffe ein eigenes **Statement**-Objekt und ein eigenes **ResultSet**-Objekt erzeugt.

39 Wiederholung

39.1 Java

Aufgabe 1: String-Manipulationen



- Baue die abgebildete Oberfläche nach. Das untere Textfeld soll nicht editierbar sein. Dieses Textfeld dient zur Ausgabe der Ergebnisse.
- Wenn der Benutzer auf den Button „Anzahl Zeichen“ klickt, wird die Anzahl der Zeichen im Eingabetextfeld ermittelt. Das Ergebnis wird im unteren Textfeld ausgegeben.
- Wenn der Benutzer auf den Button „'n' vorhanden?“ klickt, wird untersucht, ob in der Zeichenkette im oberen Textfeld ein 'n' vorkommt. Das Ergebnis („ja“ oder „nein“) wird im unteren Textfeld ausgegeben.
- Wenn der Benutzer auf den Button „Anzahl 'E's“ klickt, wird gezählt, wie viele kleine oder große 'E's in der Zeichenkette im Eingabetextfeld vorkommen. Das Ergebnis wird im unteren Textfeld ausgegeben.
- Wenn der Benutzer auf den Button „Anzahl der Ziffern?“ klickt, wird gezählt, wie viele Ziffern (Zahlen zwischen 0 und 9) in der Zeichenkette im Eingabetextfeld vorkommen. Das Ergebnis wird im unteren Textfeld ausgegeben.
- Wenn der Benutzer auf den Button „in Kleinbuchstaben“ klickt, werden alle Buchstaben des Eingabetextes in Kleinbuchstaben umgewandelt. Das Ergebnis wird im unteren Textfeld ausgegeben.
- Wenn der Benutzer auf den Button „erste 5 Zeichen“ klickt, werden die ersten fünf Zeichen aus dem String im Eingabetextfeld heraus geholt und in das untere Textfeld geschrieben. Falls dabei ein Fehler auftritt wird in das untere Textfeld geschrieben: „Der Text ist zu kurz.“.
- Wenn der Benutzer auf den Button „Hallo?“ klickt, wird untersucht, ob im Eingabetextfeld der String „Hallo“ steht. Das Ergebnis („ja“ oder „nein“) wird im unteren Textfeld ausgegeben.

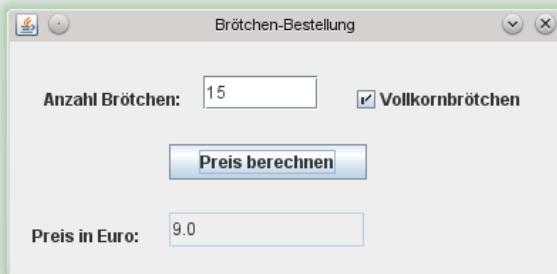
Aufgabe 2: Count Down (Thread-Übung)

Erstelle ein Anwendungsfenster mit einem einzigen, nicht editierbaren Textfeld, in dem zu Beginn die Zahl 10 steht. In einem Thread wird die Zahl im Sekundentakt um eins herunter gezählt. Wenn der Wert 0 erreicht ist, wird das Anwendungsfenster rot eingefärbt (mit der JFrame-Methode `setBackground()`) und der Thread beendet sich.



Brötchenbestellung

Erstelle die abgebildete Programmoberfläche:



Wenn man auf den Button „Preis berechnen“ drückt, wird der Preis für die gewünschte Anzahl Brötchen bestimmt. Dazu wird zuerst überprüft, ob der Benutzer eine korrekte Zahl eingegeben hat (einen Integer-Wert, der größer als Null ist). Falls nicht wird in einer Messagebox eine geeignete Fehlermeldung ausgegeben. Falls die Eingabe korrekt ist, wird abgefragt, ob das Häkchen bei „Vollkornbrötchen“ gesetzt ist. Für Vollkornbrötchen beträgt der Preis 0,60 € pro Stück. Für normale Brötchen zahlt man nur 0,30 € pro Stück.

Tipp: Nutze aus, dass die Methode `Integer.parseInt()` eine `NumberFormatException` erzeugt, wenn der eingegebene String nicht in eine ganze Zahl umgewandelt werden kann.

Aufgabe 4: Lauftext (Thread-Übung)

Erstelle ein Anwendungsfenster mit einem nicht editierbaren Textfeld, das zu Beginn leer ist. Im Sekundentakt sollen die Buchstaben des Wortes „Abrakadabra“ einfliegen („A“, „Ab“, „Abr“, „Abra“, und so weiter).

Schreibe das Wort „Abrakadabra“ dazu in eine String-Variable und programmiere den Code so, dass er auch funktioniert, wenn ein anderes Wort in der Variable steht.

Aufgabe 5: Steinen ausweichen

Im Kursrepository findest du die Klasse `Mann` mit folgendem Code:

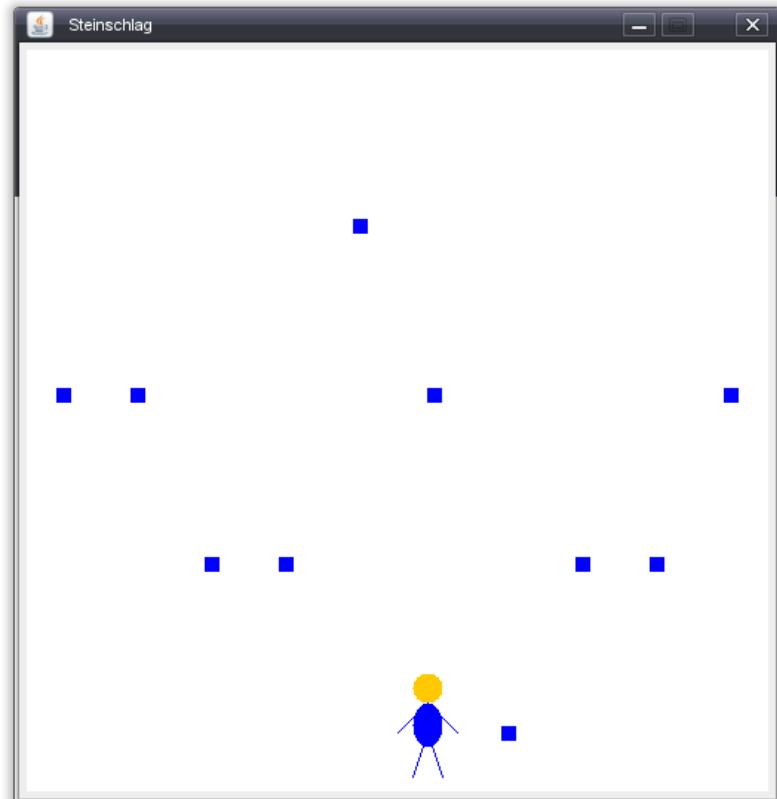
```
import java.awt.*;

public class Mann {
    int x;
    int y;

    public Mann(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void zeichnen(Graphics g) {
        // Kopf
        g.setColor(Color.ORANGE);
        g.fillOval(x + 10, y, 20, 20);
        // Bauch
        g.setColor(Color.BLUE);
        g.fillOval(x + 10, y + 20, 20, 30);
    }
}
```

```
// Arme
g.drawLine(x, y + 40, x + 20, y + 20);
g.drawLine(x + 20, y + 20, x + 40, y + 40);
// Beine
g.drawLine(x + 20, y + 40, x + 10, y + 70);
g.drawLine(x + 20, y + 40, x + 30, y + 70);
}
}
```



- Erzeuge ein neues Anwendungsfenster mit einer Zeichenfläche, die eine Breite und Höhe von je 500 Pixeln hat. Erzeuge und zeichne im Anwendungsfenster ein Objekt der Klasse **Mann** an der Position (250, 420).
- Programmiere einen Timer-Thread, der die `repaint()`-Methode des Anwendungsfensters alle 20 Millisekunden aufruft. Der Thread soll gleich zu Beginn des Programms gestartet werden und die ganze Zeit durchlaufen.
- Der Mann soll sich nun auf Tastendruck bewegen.
Wenn der Benutzer den Buchstaben 'r' (für rechts) drückt, beginnt der Mann eine Bewegung nach rechts. Seine x-Position soll dabei wiederholt um zwei Pixel nach rechts verschoben werden, bis der Benutzer einen anderen Buchstaben drückt. Wenn der Benutzer den Buchstaben 'l' (für links) drückt, bewegt sich der Mann wiederholt um zwei Pixel nach links. Wenn der Benutzer ein 's' (für stopp) drückt, steht der Mann still.
- Der Mann besitzt eine Breite von 40 Pixeln. Der Benutzer muss den Mann so steuern, dass er immer im Fenster bleibt. Wenn sich einer der Arme des Mannes außerhalb des Fensters befindet, soll die Anwendung beendet werden. Dazu musst du den Code

```
System.exit(0);
```

aufrufen.

- Vom Himmel sollen Steine fallen, denen der Mann ausweichen muss.
Programmiere dazu eine neue Klasse **Stein**.

Dem Stein werden im Konstruktor seine x-Position, seine Geschwindigkeit und ein Verweis auf den Mann als Parameter übergeben. Die y-Position des Steins wird zu Beginn fest auf den Wert 0 gesetzt.

Zeichne den Stein als ausgefülltes schwarzes Quadrat mit einer Breite und Höhe von zehn Pixeln. Lass den Stein außerdem bei jedem Aufruf der Methode `zeichnen()` entsprechend seiner Geschwindigkeit ein Stück nach unten fallen.

Wenn der Stein aus dem Fenster heraus gefallen ist, wird er automatisch wieder nach oben positioniert, so dass er erneut herunterfallen kann.

Erzeuge im Anwendungsfenster ein Array von zehn Steinen. Der erste Stein soll die x-Position 20 haben. Die anderen Steine sollen jeweils im Abstand von 50 Pixeln folgen. Übergib den Steinen für die Geschwindigkeit Zufallswerte zwischen eins und vier.

- f) Wenn einer der Steine den Mann trifft, ist das Spiel vorbei und die Anwendung soll mit `System.exit(0)` beendet werden. Dazu vergleicht jeder Stein in der `zeichnen()`-Methode seine eigene Position mit der Position des Mannes. Der Stein trifft den Mann, wenn die y-Position seiner unteren Kante größer als 420 ist, und wenn sich entweder seine linke oder seine rechte Seite innerhalb der 40 Pixel großen Ausdehnung des Mannes befindet.

39.2 UML-Zustandsdiagramme

Aufgabe 1: Hausbau

Als Teil eines größeren Computerspiels wird eine Gruppe von Handwerkern modelliert, die ein Haus aufbauen. Beschreibe die Zustände eines einzelnen Handwerkers in einem UML-Zustandsdiagramm:

Ein Handwerker hackt zuerst im Wald einen Stapel Holz, dann läuft er mit dem Holz zur Baustelle und baut die Balken ein. Wenn das Haus noch nicht fertig ist, läuft er anschließend wieder zurück zum Wald um neues Holz zu holen und beginnt den Arbeitszyklus von vorne.

Falls das Haus fertig ist, bleibt der Handwerker untätig neben dem Haus stehen. Wird das Haus von Feinden zerstört, so läuft er wieder zum Wald und beginnen erneut mit Holzhacken, um das Haus wieder aufzubauen.

Aufgabe 2: Igelspiel

Ziel des Spieles:

Der Igel steht am linken Rand des Fensters und muss bis zum rechten Rand des Fensters laufen ohne von einem Fuchs gebissen zu werden. Der Igel darf dabei maximal vier mal stehen bleiben.

Von rechts kommen ab und zu Füchse an. Jeder Fuchs kann nur einmal zubeißen. Wenn der Igel in diesem Moment die Stacheln oben hat, ist er gerettet.

Bedienungsanleitung:

Wenn der Igel steht, musst du 'l' (für Laufen) drücken, damit der Igel los läuft. Wenn du auf 'a' (für Anhalten) drückst, bleibt der Igel stehen. Du kannst den Igel mehrfach laufen und wieder anhalten lassen. Wenn du aber zum fünften Mal auf 'a' drückst, hast du das Spiel verloren.

Wenn ein Fuchs kommt, muss du den Igel zum stehen bringen und anschließend 's' (für Stacheln) drücken, damit der Igel seine Stacheln zum Schutz aufstellt (ein 's' hat keine Wirkung, wenn der Igel am Laufen ist). Nur wenn der Igel seine Stacheln oben hat, kann er den Biss eines Fuchses überleben. Wenn ein Fuchs zubeißt ohne dass der Igel seine Stacheln oben hat, hast du verloren.

Der Igel stellt seine Stacheln für eine Sekunde hoch. Anschließend muss er sich drei Sekunden von der Anstrengung erholen ehe er wieder zum Stehen kommt und du ihn weiter laufen lassen kannst.

40 Datensicherheit

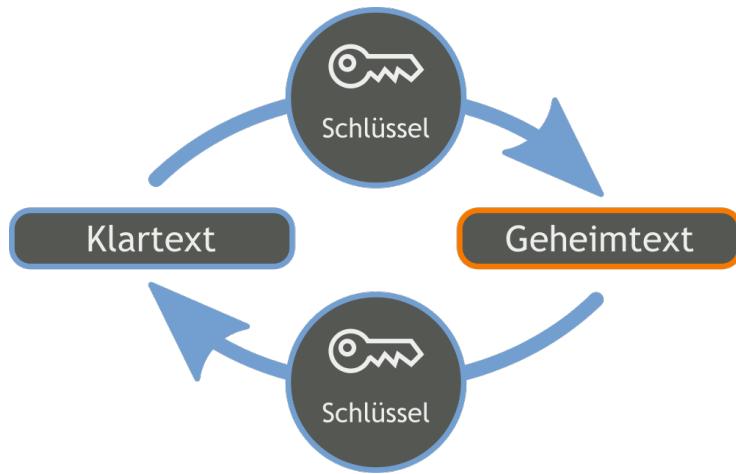
41 Kryptologie

Die Kryptologie ist die Wissenschaft, die sich mit Verfahren zur Ver- und Entschlüsselung von Daten beschäftigt. Sie lässt sich unterteilen in die beiden Bereiche *Kryptografie* und *Kryptoanalyse*.

Während die Kryptografie sich damit beschäftigt, Verfahren zu entwickeln, die benutzt werden können, um Nachrichten für andere unlesbar zu machen und es dem legitimen Empfänger erlauben, die verschlüsselte Nachricht wieder zu entschlüsseln, beschäftigt sich die Kryptoanalyse mit der Sicherheit (oder anders herum gesagt: der Angreifbarkeit) von kryptografischen Verfahren.

Ein zentraler Begriff der Kryptologie ist der des *Schlüssels*. Mit Hilfe des Schlüssels kann die Nachricht auf der Senderseite verschlüsselt und auf der Empfängerseite auch wieder entschlüsselt werden.

Ohne Kenntnis des Schlüssels ist die verschlüsselte Nachricht nicht oder nur mit hohem Aufwand (Kryptoanalyse) zu entschlüsseln.



Der Klartext wird mittels eines Schlüssels verschlüsselt. Mit Hilfe des selben Schlüssels kann der Geheimtext wieder entschlüsselt werden

41.1 Monoalphabetische Verfahren

Bei monoalphabetischen Verfahren wird aus einem Klartextbuchstaben immer derselbe Geheimtextbuchstabe.

Das Caesar-Verfahren

Dieses Verfahren wurde von Julius Caesar 50 Jahre vor Christus benutzt. Das Alphabet wird dabei einfach um mehrere Buchstaben verschoben. Zum Beispiel um drei Buchstaben:

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |

Damit wird aus dem Klartext hallo der Geheimtext KDOOR.

Kryptoanalyse des Caesar-Verfahrens

Das Caesar-Verfahren ist schon über 2000 Jahre alt. Es ist deshalb nicht erstaunlich, dass es sehr leicht zu brechen ist. Die Analysemethode nennt sich *Brute-Force*, was nichts anderes heißt als „rohe Gewalt“. Der Trick besteht darin, dass einfach alle möglichen Schlüsse ausprobiert werden, und man schaut, was dabei jeweils als Klartext herauskommt.

Substitutionsverfahren

Irgendwann haben Caesars Nachfahren herausgefunden, dass dieses Verfahren nicht sehr sicher ist. Wenn jedoch die Anzahl der möglichen Schlüssel erhöht werden könnte, dann würde die Kryptoanalyse massiv erschwert werden. Dies gelingt mit dem Substitutionsverfahren. Dabei wird jeder Buchstabe des Klartextes durch einen beliebigen Buchstaben aus dem Geheimtext ersetzt („substituieren“ = „ersetzen“). Es werden nicht mehr alle Buchstaben um gleich viele Stellen verschoben wie beim Caesar-Verfahren. Damit man sich dazu den Schlüssel gut merken kann, geht man folgendermaßen vor:

Die Buchstaben des Klartextalphabets werden der Reihe nach hingeschrieben. Darunter wird zuerst das Schlüsselwort (im Beispiel: KRYPTO) geschrieben, und dann kommen der Reihe nach alle im Schlüsselwort nicht benutzten Buchstaben des Alphabets.

Beispiel mit Schlüsselwort KRYPTO:

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| K | R | Y | P | T | O | A | B | C | D | E | F | G | H | I | J | L | M | N | Q | S | U | V | W | X | Z |

Damit wird aus dem Klartext hallo der Geheimtext BKFFI.

41.2 Polyalphabetische Verschlüsselungsverfahren

Nachdem die Kryptoanalytiker auch das Substitutionsverfahren geknackt hatten, waren die Verschlüsselungsspezialisten wieder gefragt. Solange aus einem Klartextbuchstaben immer der gleiche Geheimtextbuchstabe entsteht, kann man immer die Häufigkeitsanalyse anwenden.

So entstand die polyalphabetische Verschlüsselung, bei der aus einem Klartextbuchstaben nicht immer der gleiche Geheimtextbuchstabe wird („poly“ = „viel“).

Vigenère-Verfahren

Das bekannteste polyalphabetische Verschlüsselungsverfahren heißt Vigenère und funktioniert folgendermaßen:

$$\begin{array}{r}
 d \ i \ e \ s \ i \ s \ t \ d \ e \ r \ k \ l \ a \ r \ t \ e \ x \ t \\
 + \ k \ e \ y \ k \ e \ y \ k \ e \ y \ k \ e \ y \ k \ e \ y \ k \ e \ y \\
 \hline
 O \ N \ D \ D \ N \ R \ E \ I \ D \ C \ P \ K \ L \ W \ S \ P \ C \ S
 \end{array}$$

Der Schlüssel (hier key) wird endlos wiederholt unter den Klartext geschrieben. Danach werden zu den Buchstaben des Klartextes die Buchstaben des Schlüssels hinzu addiert. So wird aus d (4. Buchstabe) plus k (11. Buchstabe) ein O ($4 + 11 = 15$. Buchstabe). Eine verschlüsselte Nachricht wird entschlüsselt, indem der Schlüssel vom Geheimtext subtrahiert wird.

Kryptoanalyse des Vigenère-Verfahren

Wieso ist das Knacken der Nachricht auch bei einer polyalphabetischen Verschlüsselung noch möglich? Der Schlüssel hat eine bestimmte Länge, zum Beispiel 3 wie im oben gewählten Beispiel.

Das hat zur Folge, dass jeder dritte Buchstabe um die gleiche Anzahl Buchstaben verschoben wird (der erste, der vierte, etc., alle werden um $k = 11$ Stellen verschoben).

Wenn nun wie im Beispiel an der zweiten und fünften Stelle der gleiche Buchstabe steht (i) so wird aus diesem auch der gleiche Geheimtextbuchstabe (N):

$$\begin{array}{r}
 d \ i \ e \ s \ i \ s \ t \ d \ e \ r \ k \ l \ a \ r \ t \ e \ x \ t \\
 + \ k \ e \ y \ k \ e \ y \ k \ e \ y \ k \ e \ y \ k \ e \ y \ k \ e \ y \\
 \hline
 O \ N \ D \ D \ N \ R \ E \ I \ D \ C \ P \ K \ L \ W \ S \ P \ C \ S
 \end{array}$$

Durch solche auftretende Muster lässt sich per Computer relativ einfach die Schlüssellänge L bestimmen, indem man gleiche Buchstabenfolgen im Geheimtext sucht und deren Abstand bestimmt. Hat man erst einmal die Schlüssellänge L nimmt man zur Bestimmung des 1. Buchstabens des Schlüssels den 1., den $1+L$, den $1+2L$ usw.

Buchstaben und macht auf diesen eine einfache Häufigkeitsanalyse. Dies funktioniert, da alle diese Buchstaben um die gleiche Anzahl Stellen verschoben wurden (wie beim Caesar-Verfahren). Man muss nur den häufigsten Buchstaben finden, und schon ist die Verschiebung bekannt (durch Vergleich mit 'e').

One-Time-Pad

Das Knacken von Vigenère gelingt, da es wegen der fixen Schlüssellänge zu Wiederholungen kommt. Nimmt man einen Schlüssel der mindestens genau so lang ist wie der zu verschlüsselnde Text gibt es keine Wiederholungen. Dieses Verfahren heißt *One-Time-Pad*.

Das One-Time-Pad ist ein 100% sicheres Verfahren, denn jeder Schlüsselbuchstabe wird nur ein mal (one time) verwendet. Es hat nur einen – im wahrsten Sinne des Wortes großen – Nachteil: Im vornherein muss ein riesengroßer geheimer Schlüssel vereinbart werden. Dieses Verfahren wurde während des Kalten Krieges zwischen Moskau und Washington eingesetzt („heißer Draht“). Dafür mussten regelmäßig Diplomaten mit Koffern voller Zufallszahlen hin und her reisen.

Denn auch beim One-Time-Pad gilt:

- Der Schlüssel muss über einen sicheren separaten Kanal übermittelt werden.
- Ist der Schlüssel dem Gegner bekannt ist die Sicherheit dahin.

DES

Das heute im kommerziellen Gebrauch am häufigsten eingesetzte Verfahren heißt *DES* („Data Encryption Standard“). Es funktioniert im Prinzip wie ein mehrfach hintereinander angewandtes Substitutionsverfahren.

Der DES erlaubt mit 56 Bits Schlüssellänge = $72\ 057\ 594\ 037\ 927\ 936 = 72$ Billiarden mögliche Schlüssel. Dennoch ist dies heute nicht mehr ausreichend: DES kann in wenigen Stunden mittels der Brute-Force-Methode geknackt werden!

IDEA

Der *IDEA* („International Data Encryption Algorithm“) funktioniert ähnlich wie DES und arbeitet mit 128 Bits Schlüssellänge. Damit besitzt er $2^{128} = 3.43669 \cdot 10^{38}$ mögliche Schlüssel. Dies zu knacken benötigt derzeit viele Jahre. Deshalb gilt der IDEA heute als sicher.

41.3 Asymmetrische Verschlüsselung

Bei symmetrischen Verschlüsselungsverfahren, bei denen zum Kodieren und Dekodieren derselbe Schlüssel verwendet wird, besteht das Problem, dass der Empfänger der Nachricht irgendwie auf einem geheimen Kanal an den Schlüssel kommen muss. Wenn die Übertragung des Schlüssels abgefangen wird, kann die Nachricht auch von Fremden entschlüsselt werden. Dieses Problem wird mit der asymmetrischen Verschlüsselung gelöst.

Grundprinzip

In asymmetrischen Verschlüsselungsverfahren existieren zwei verschiedene Schlüssel, von denen einer zur Verschlüsselung und der andere zur Entschlüsselung verwendet wird. Beide Schlüssel sind mathematisch miteinander verbunden. Das zusammengehörende Schlüsselpaar besteht aus:

Private Key Dieser verbleibt grundsätzlich beim Eigentümer des Schlüssels.

Public Key Dieser ist öffentlich bekannt und kann von jedermann benutzt werden.

Anwendungen

Verschlüsselung: Eine Nachricht wird mit dem Public Key des Empfängers verschlüsselt. Sie kann dann nur mit Hilfe des zugehörigen Private Keys entschlüsselt werden. Deshalb kann niemand anders als der Empfänger die Nachricht dekodieren.

Digitale Unterschriften: Für den öffentlich lesbaren Text wird eine Prüfsumme berechnet. Die Prüfsumme wird mit dem Private Key des Senders kodiert. Mit dem bekannten Public Key des Senders kann nun jedermann die Prüfsumme dekodieren und so überprüfen, ob die Nachricht tatsächlich von der angegebenen Person stammt. Diese digitalen Unterschriften gelten als unfälschbar und sind seit Januar 2000 im deutschen Signaturgesetz rechtlich der handschriftlichen Unterschrift gleichgestellt.

Mathematische Grundlage

Zum Verschlüsseln werden mathematische Einweg-Operationen angewandt, deren Umkehrung erheblich komplizierter ist, als die Operation selbst. Das Produkt zweier großer Primzahlen zu berechnen ist auch von Hand nicht besonders schwer. Ein Algorithmus, der eine gegebene (sehr große) Zahl in ihre Primfaktoren zerlegt, ist dagegen viel langwieriger. Es ist bisher kein effektives Verfahren dafür bekannt. Die Grundideen zur Verwendung mathematischer Einweg-Funktionen stammen möglicherweise aus den Sechziger Jahren (Communications Electronics Security Group), wurden aber durch die Veröffentlichungen von Whitefield Diffie und Martin Hellman im Jahre 1976 bekannt. Auf dieser Basis wurde 1978 ein auf Potenzierung und ganzzahliger Division mit Rest (Modulo-Operation) beruhendes Verfahren vorgestellt. Die Entwickler Ron Rivest, Adi Shamir und Leonard Adleman vermarkten dieses Verfahren weltweit unter dem Namen RSA. Für Privatpersonen ist die Nutzung des RSA-Algorithmus mit dem Programm *Pretty Good Privacy* (PGP) kostenlos. Das Verfahren ermittelt für jeden Teilnehmer einen privaten und einen öffentlichen Schlüssel. Dazu werden zunächst per Zufall zwei üblicherweise 1024 Bit lange Primzahlen p und q ausgewählt.

RSA-Algorithmus

Zuerst werden per Zufall zwei große Primzahlen p und q erzeugt, die später vernichtet werden. Aus den Primzahlen berechnet man die Zahlen

$$\begin{aligned} n &= p \cdot q \\ s &= (q - 1) \cdot (p - 1) \end{aligned}$$

Öffentlicher Schlüssel

Für den Public Key wird eine Zahl e so gewählt, dass e und s keinen anderen gemeinsamen Teiler besitzen als die Zahl 1. e muss dabei kleiner sein als s .

Privater Schlüssel

Für den Private Key wird eine Zahl d so gewählt, dass $(d \cdot e) \bmod s$ den Wert 1 ergibt. (\bmod ist der Modulo-Operator, der in Java als `%` geschrieben wird)

Verschlüsselung einer Botschaft

Die Botschaft wird zunächst in Zahlen umgewandelt. Die Zahlen werden blockweise kodiert, wobei jeder Block kleiner als n sein muss. Ein in Zahlen kodierter Klartext b wird folgendermaßen in einen Geheimtext c kodiert:

$$c = b^e \bmod n$$

Entschlüsseln einer Botschaft

Ein Geheimtext c wird folgendermaßen in den Klartext b zurück gewandelt:

$$b = c^d \bmod n$$

Schwachstelle des Public Key Verfahrens

Ein Angriffspunkt ist die Verbreitung des öffentlichen Schlüssels. Man kann seinen öffentlichen Schlüssel in einer Datei speichern, die man dann an seine Freunde weitergeben oder im Internet veröffentlichen kann. Was aber, wenn ein Fremder einen öffentlichen Schlüssel unter deinem Namen veröffentlicht? Eine mit diesem Schlüssel chiffrierte Nachricht kann dann nur von dem Fremden gelesen werden und nicht von dir selbst.¹

41.4 Anwendungsgebiete der Kryptographie

Kryptographische Methoden wurden und werden an vielen Stellen und in vielen Anwendungsszenarien eingesetzt.

- Signierung von e-Mails (Nachweis der Authentizität der Nachricht).
<http://de.wikipedia.org/wiki/E-Mail-Verschlüsselung>
- Verschlüsselung von Datenverkehr über unsichere Verbindungen (etwa Online-Banking). Dazu gibt es eine ganze Reihe von Protokollen. Beispiele: HTTPS statt HTTP, IMAPS statt IMAP oder auch die komplette Kapselung von Netzwerkverkehr über VPNs (Virtual Private Networks), womit beispielsweise verschiedene Standorte eines Unternehmens über das Internet (unsicher) so verbunden werden können, dass die vertraulichen Inhalte des internen Firmennetzwerkes durch „Mithörer“ nicht entschlüsselt werden können.
http://de.wikipedia.org/wiki/Transport_Layer_Security
http://de.wikipedia.org/wiki/Virtual_Private_Network
- Verschlüsselung von Funkverkehr (etwa Mobilfunknetze).
http://de.wikipedia.org/wiki/Global_System_for_Mobile_Communications#Sicherheitsfunktionen
- Rechtssicherer elektronischer Geschäftsverkehr. Etwa bei Vertragsabschlüssen.
[http://de.wikipedia.org/wiki/Signaturgesetz_\(Deutschland\)](http://de.wikipedia.org/wiki/Signaturgesetz_(Deutschland))
- Geldautomaten übertragen die PIN nie unverschlüsselt:
http://de.wikipedia.org/wiki/Encrypting_PIN_Pad
- Der Netzwerkverkehr in WLANs kann (und wird meistens auch) verschlüsselt um den naturgemäß offenen Zugang zu beschränken:
<http://de.wikipedia.org/wiki/Wlan#Datensicherheit>
- PayTV: Hier wird der „Content“ (also das Fernsehsignal) so verschlüsselt, dass nur der zahlende Kunde das Programm richtig sehen kann. Für alle anderen gibt es nur Bildrauschen.
<http://de.wikipedia.org/wiki/Zugangsberechtigungssystem>
- Kryptowährungen wie etwa Bitcoins als virtuelle Zahlungsmittel existieren nicht als Münzen oder Geldscheine, sondern lediglich als Dateien. Diese müssen entsprechend stark kryptographisch gesichert sein.
<http://de.wikipedia.org/wiki/Kryptowährung>

41.5 Kryptoanalyse

Die Kryptoanalyse beschäftigt sich mit der Sicherheit aber auch den Schwachstellen von kryptographischen Verfahren. Es folgt eine Auflistung der gängigsten Angriffsmethoden gegen kryptographische Verfahren, mit dem

¹Quelle: Sicherheit in Netzen – Netzmafia <http://www.netzmafia.de/skripten/sicherheit/sicher5.html>

Ziel die Verschlüsselung zu knacken:

Brute Force Typischerweise mit Hilfe von Computern werden „einfach“ alle möglichen Schlüssel durchprobiert.

Das wird umso aufwändiger (im zu wünschenden Extremfall aussichtslos), je größer die Anzahl der möglichen Schlüssel ist. Eng damit verknüpft ist die sogenannte Schlüssellänge: Je länger ein Schlüssel, desto größer ist der sogenannte „Schlüsselraum“ der durchsucht werden muss.

http://de.wikipedia.org/wiki/Brute_force#Kryptologie

Häufigkeitsanalyse Im speziellen (aber nicht ausschließlich) bei monoalphabetischen Verfahren kann man über die statistische Analyse des verschlüsselten Textes Rückschlüsse auf die Verschlüsselung ziehen. So ist etwa in der deutschen Sprache das 'E' der häufigste Buchstabe. Ein hinreichend langer deutscher Text wird deshalb auch in seiner verschlüsselten Form schnell zeigen, welches Zeichen dem 'E' entspricht.

<http://de.wikipedia.org/wiki/Häufigkeitsanalyse>

Social Engineering Über den direkten Kontakt mit der „Zielperson“ (oder Personen aus deren Umfeld) können oft Informationen gewonnen werden, die entweder direkt („Hallo, ich arbeite bei der Bank, bei der Sie Online-Banking betreiben. Jemand hat versucht sich in Ihr Konto zu holen. Ich muss Sie deshalb bitten, mir Ihr Passwort und die nächsten TANs zu nennen.“), oder indirekt (Informationen, die anschließend zum Erraten von Schlüsseln bzw. Passwörtern genutzt werden können).

[http://de.wikipedia.org/wiki/Social_Engineering_\(Sicherheit\)](http://de.wikipedia.org/wiki/Social_Engineering_(Sicherheit))

Erraten Schlechte Passwörter lassen sich erraten (Name eines Verwandten oder Haustieres, Geburtstage oder ähnliches. Oft in Verbindung mit „Social Engineering“).

Wörterbuchangriffe (Dictionary Attacks) Eine übliche rechnergestützte Methode um schlechte/schwache Passwörter zu knacken. Dabei werden von einem Rechner alle möglichen bekannten Wörter (auch Namen und Geburtstage) aus einer Wörterbuch-Datei, oft auch kombiniert mit zusätzlichen Ziffern oder der inzwischen üblichen Ersetzungen wie etwa '3' für 'E' etc.

http://de.wikipedia.org/wiki/Dictionary_Attack

41.6 Kryptologie – Übungen

Aufgabe 1: Caesar-Verfahren I

Wie viele Schlüssel-Möglichkeiten gibt es beim Caesar-Verfahren?

Aufgabe 2: Caesar-Verfahren II

Verschlüssele von Hand ein Wort mit dem Caesar-Verfahren. Das Wort darf keine Umlaute enthalten. Tausche anschließend das Wort und den benutzten Schlüssel (das heißt die Anzahl der Buchstaben, um die das Alphabet verschoben wurde) mit einem deiner Sitznachbarn aus und versuche das Wort deines Nachbarn zu dekodieren.

Aufgabe 3: Caesar-Verfahren III

Der folgende Text wurde mit dem Caesar-Verfahren verschlüsselt. Versuche herauszufinden um wie viele Buchstaben das Alphabet verschoben wurde und dekodiere den Anfang des Textes.

Ajwxhmqzjxxjqzslxyjhmsnp nxy jns xjmw xufssjsijx Ymjrf, ifx snhmy jwxy xjny Gjlnss ijk Htruzy-jwejnyfqyjwx wjcjafsy nxy. Xhmts Ozqnzx Hfjxfwx mfy fs xjnsj Ajwgzsijyjs ljmjnry Gtyxhmfkyjs ljhmnhpj. Inj Yjcyj bzuwj jnsjr Gtyjs ns ajwxhmqzjxxjqyjw Ktwr fsajwywfzy. Kfqqx ijk Gtyj ats Kjnsijs zjgjwkfqqs bzuwj, ptssyjs xnj fzx ijs fgljkfsljsjs Sfhmwnhmyjs snhmy xhmqfz bjwijs. Ozqnzx Hfjxfw bfw jgjs enjrqnhm ljsnfq.

Aufgabe 4: Substitutionsverfahren I

- Nenne für das Substitutionsverfahren einen Schlüssel, bei dem es keine echte Verschlüsselung gibt und alle Wörter unverändert bleiben.
- Wie viele verschiedene Verschlüsselungsmöglichkeiten gibt es mit dem Substitutionsverfahren?

Aufgabe 5: Substitutionsverfahren II

Verschlüssele von Hand ein Wort mit dem Substitutionsverfahren. Das Wort darf keine Umlaute enthalten. Tausche anschließend das Wort und den benutzten Schlüssel mit einem deiner Sitznachbarn aus und versuche das Wort deines Nachbarn zu dekodieren.

Aufgabe 6: Substitutionsverfahren III

Wie könnte man das Substitutionsverfahren entschlüsseln?

Aufgabe 7: Substitutionsverfahren IV

Im Kursverzeichnis findest du das Tool `textanalyse.exe`. Kopiere zunächst einige unverschlüsselte Texte in das Eingabefeld und untersuche die Häufigkeitsverteilung der Buchstaben. Am schnellsten geht dies, wenn du irgendwelche Texte markierst (zum Beispiel im Internet-Browser) und in das Textfeld einfügst. Untersuche anschließend den nach der Substitutionsmethode kodierten Text in der Datei `Substitution.txt`. Versuche die Geheimtext-Buchstaben für diejenigen Klartext-Buchstaben herauszubekommen, die besonders häufig oder besonders selten auftreten.

Aufgabe 8: Vigenère-Verfahren I

Verschlüssele von Hand ein Wort mit dem Vigenère-Verfahren. Das Wort darf keine Umlaute enthalten. Tausche anschließend das Wort und den benutzen Schlüssel mit einem deiner Sitznachbarn aus und versuche das Wort deines Nachbarn zu dekodieren.

Aufgabe 9: Vigenère-Verfahren II

Im Kursverzeichnis findest du den Text `Vigenere.txt`. Untersuche mit Hilfe des Textanalyse-Tools, ob auch bei einem mit dem Vigenère-Verfahren verschlüsselten Text Buchstaben durch ihre Häufigkeitsverteilung hervortreten. Überlege dir anschließend, wie man einen mit Vigenère verschlüsselten Text dekodieren kann.

Aufgabe 10: Programmierübung

Programmiere eines der drei klassischen Verschlüsselungsverfahren, die wir kennen gelernt haben (Caesar, Substitution oder Vigenère). Schreibe dazu ein Programm, dass einen Text aus einer Datei einliest und ihn verschlüsselt in einer anderen Datei abspeichert. Programmiere anschließend die Dekodierung der Nachricht, in dem du den verschlüsselten Code aus einer Datei ausliest und ihn entschlüsselt in eine zweite Datei schreibst.

Aufgabe 11: Asymmetrische Verfahren

Stelle die asymmetrische Kodierung einer Nachricht in einem Schaubild dar. Das Schaubild soll einen Kreislauf zeigen, der bei der ursprünglichen Nachricht beginnt und dort auch wieder endet.

Aufgabe 22: Digitale Signatur

Erkläre was eine digitale Unterschrift ist. Nach welchem Prinzip werden digitale Unterschriften erstellt?

Aufgabe 13: RSA

Das Wort `HALLO` soll nach dem RSA Verfahren verschlüsselt und wieder entschlüsselt werden.

- Wandle die Buchstaben in ihren ASCII-Code um.
- Berechne den Public Key und den Private Key für die Kodierung. Als Primzahlen werden die Zahlen $p = 13$ und $q = 19$ gewählt. Setze e auf 5 und d auf 173. Überprüfe, ob e und d die notwendigen Bedingungen erfüllen.
- Verschlüssele jeden Buchstaben einzeln mit dem unter b) berechneten Public Key.
- Entschlüssle die unter c) erzeugten Code-Zahlen mit dem unter b) berechneten Private Key. Die dabei zu berechnenden Zahlen werden so hoch, dass sie weder von einem normalen Taschenrechner noch von den in der Programmiersprache Java verfügbaren Datentypen gespeichert werden können. Benutze deshalb zur Berechnung den CAS-Modus (CAS steht für Computer-Algebra-System) von GeoGebra. Diesen erreichst du in GeoGebra über Ansicht → CAS. Die Rechnungen werden dann in die Eingabezeile eingetippt.
Für x^y schreibt man im CAS-Fenster von GeoGebra x^y .
Für $x \bmod y$ schreibt man im CAS-Fenster von GeoGebra `mod(x,y)`.
- Ist es möglich, diese Berechnungen auch in einem Java-Programm zu implementieren, obwohl die Java-Datentypen zu klein sind, um die Zahlen zu fassen?

Aufgabe 14: Vergleich von symmetrischer und asymmetrischer Verschlüsselung

Liste die Vor- und Nachteile der asymmetrischen Verschlüsselung gegenüber der symmetrischen Verschlüsselung auf.

Aufgabe 15: Web of Trust

Die Schwachstelle der asymmetrischen Verschlüsselung ist, dass man nicht immer sicher sein kann, ob ein veröffentlichter Public Key wirklich dem angegebenen Anwender gehört. Informiere dich im Internet auf den deutschen PGP-Seiten darüber, wie die Vertrauenswürdigkeit eines Schlüssels sichergestellt wird.

42 Netzwerke

42.1 Netzwerk-Kommunikation

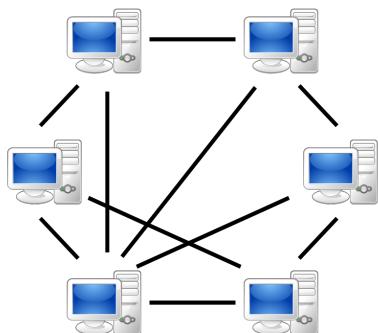
Heute ist es eine Selbstverständlichkeit, dass Computer miteinander vernetzt sind. Das macht natürlich nur Sinn, wenn Programme diese Netzwerkverbindungen nutzen. Etwa für e-Mail-Kommunikation, Online-Banking oder Zugriff auf das WWW.

Bei der Netzwerk-Kommunikation von Programmen wird zwischen zwei Modellen unterschieden: Dem sogenannten *Client-Server-Modell* und dem *Peer-to-Peer-Modell*.

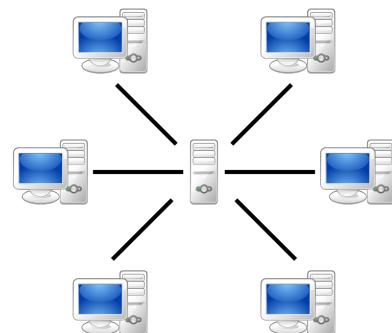
Peer to Peer (P2P)

Im Peer-to-Peer-Modell sind alle Kommunikationspartner (Programme) gleichberechtigt. Es gibt keine Aufgabentrennung und jeder Teilnehmer kann – zumindest potentiell – direkt mit jedem anderen Teilnehmer kommunizieren.

Man spricht deshalb auch vom *symmetrischer Kommunikation*.



Peer-to-Peer-Modell



Client-Server-Modell

Client-Server

Der Gegensatz zum Peer-to-Peer-Modell ist das Client-Server-Modell. Bei diesem bietet ein *Server* einen Dienst an und ein *Client* nutzt diesen Dienst.

Ein Web-Server zum Beispiel ist ein Programm, das HTML-Seiten zur Verfügung stellt. Der Web-Server nimmt Anfragen von Browsern (den Web-Clients) entgegen und schickt ihnen anschließend das gewünschte Dokument.

Die Bezeichnungen Client und Server werden häufig auch für Computer verwendet. Ein Computer wird als Server bezeichnet, wenn auf ihm ein Server-Programm läuft. Ein Computer wird als Client bezeichnet, wenn auf ihm Client-Software ausgeführt wird. Da auf einem Computer gleichzeitig sowohl Server- als auch Client-Programme laufen können ist diese begriffliche Zuordnung nicht immer eindeutig. So läuft beispielsweise auf jedem Rechnern in unserem Informatik-Raum ein Datenbank-Server aber auch verschiedene Clients.

42.2 Protokoll

Wenn Computer über das Netzwerk Daten austauschen sollen, muss vorher genau definiert werden, welche Daten in welcher Reihenfolge zu senden sind, damit die Computer sich auch richtig verstehen. Die festgelegten Regeln, die notwendig sind, um einen kontrollierten und eindeutigen Verbindungsauflaufbau, Datenaustausch und Verbindungsabbau zwischen zwei Rechnern zu gewährleisten bezeichnet man als *Protokoll*. Das Protokoll ist für

Computer genauso wie für Diplomaten die Summe der Umgangsformen. Wer sich an die vereinbarten Regeln hält, wird verstanden.

42.3 OSI-Schichtenmodell

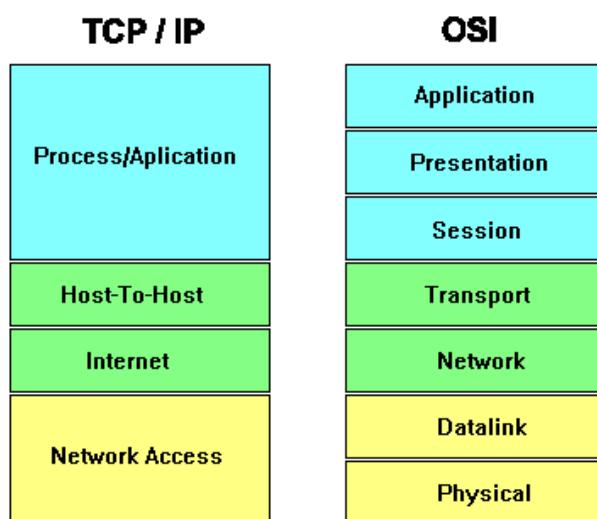
Wenn ein einziges Programm alle Aufgaben übernehmen sollte, die zur Übertragung von Daten über das Netz notwendig sind, wäre dieses Programm so komplex, dass die zuständigen Programmierer im wahrsten Sinne des Wortes verrückt werden würden. Um die Komplexität zu bewältigen hat man deshalb die Aufgaben in verschiedene Teilbereiche unterteilt, die einzeln programmiert werden und aufeinander aufsetzen – eine Art „Baukastensystem“. 1983 wurde von der Internationalen Standard Organisation (ISO) das berühmte *OSI-Schichtenmodell* festgelegt, dass sieben verschiedene „Bauteile“ (Schichten) definiert:

| | |
|------------------|---|
| Schicht 7 | Anwendungsschicht – Application Layer Anwendungsprogramme: e-Mail, Browser, Dateitransfer |
| Schicht 6 | Darstellungsschicht – Presentation-Layer Konvertierung von Darstellungsformaten, zum Beispiel ASCII, EBCDIC, UTF |
| Schicht 5 | Sitzungsschicht – Session Layer Kommunikation von Sitzungen in Multi-User-Umgebungen |
| Schicht 4 | Transportschicht – Transport Layer Aufteilung bzw. Zusammensetzung von Daten in Pakete |
| Schicht 3 | Vermittlungsschicht – Network Layer Auswahl der Übertragungswege für Datenpakete |
| Schicht 2 | Sicherungsschicht – Data Link Layer Ver- und Entpacken der Datenpakete für das genutzte Medium |
| Schicht 1 | Bitübertragungsschicht – Physical Layer Festlegung der elektrischen und mechanischen Parameter einer Datenkommunikation |

Siehe <http://de.wikipedia.org/wiki/OSI-Modell>

42.4 TCP/IP im vereinfachten Schichtenmodell

Für die Internet-Protokolle (TCP/IP) wird häufig ein vereinfachtes Schichtenmodell mit vier Ebenen benutzt:



Grafik entnommen von <http://www.netzmafia.de/skripten/netze/netz8.html>

Für die *physische Schicht* kommen je nach verwendeter Hardware sehr unterschiedliche Protokolle zum Einsatz (zum Beispiel Ethernet, ATM, usw.). Die Protokolle auf dieser Ebene wissen wie die spezielle Hardware angesprochen werden muss und können Datenpakete zwischen zwei Rechnern austauschen.

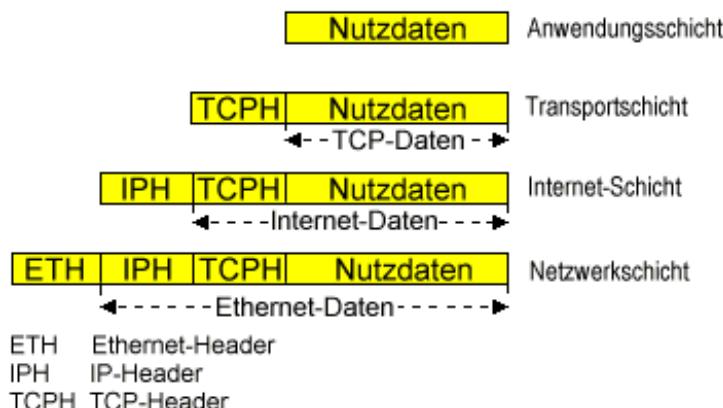
Auf der *Netzwerkschicht* wird in Internet-basierten Netzen immer das *IP-Protokoll* verwendet. Das IP-Protokoll verwaltet unter anderem die Adressen der Rechner und ermöglicht so die Übertragung eines Datenpakets durch das Netz zu einem bestimmten Zielrechner.

Die *Transportschicht* übernimmt die Adressierung des Prozesses auf einem Rechner und vergibt dazu sogenannte *Port-Nummern*. Damit gelangen übertragene Datenpakete nicht nur an den richtigen Rechner, sondern auch zu der Anwendung, für die sie gedacht sind. Dafür stehen zwei verschiedene Protokolle zur Verfügung:

- Das *TCP-Protokoll* teilt die Daten eines Anwendungsprogramms in Datenpakete auf und übergibt sie zur Übertragung an das IP-Protokoll. Auf der Empfänger-Seite sorgt TCP dafür, dass die Daten aus den einzelnen Paketen in der richtigen Reihenfolge wieder zusammen gesetzt werden und verloren gegangene Pakete erneut gesendet werden.
- Alternativ kann man das *UDP-Protokoll* verwenden, das jedoch weit weniger Funktionalität besitzt als TCP. Bei Verwendung des UDP-Protokolls muss das Anwendungsprogramm seine Daten selber in Datenpakete aufteilen. Das UDP-Protokoll stellt auch nicht sicher, dass die Datenpakete bei Fehlern erneut gesendet werden. Um verloren gegangene Datenpakete muss sich die Anwendung selber kümmern.

Auf oberster Ebene stehen die Protokolle für die verschiedenen Anwendungen (zum Beispiel FTP zum Filetransfer, SMTP zum Mail-Versand oder HTTP zur Übertragung von Web-Seiten). Für Spezial-Anwendungen kann jeder Programmierer ein eigenes Protokoll entwickeln.

Wenn das Anwendungsprogramm Daten übertragen möchte, werden nacheinander die einzelnen Protokoll-Schichten aufgerufen und jede Schicht packt zu dem Datenpaket eigene Daten hinzu, die sie für ihre Verwaltungsarbeiten benötigt:



Grafik entnommen von <http://www.netzmafia.de/skripten/netze/netz8.html>

Das einfache UDP-Protokoll erzeugt zum Beispiel für jedes Datenpaket einen zusätzlichen *Header* mit folgendem Format:



Grafik entnommen von <http://www.netzmafia.de/skripten/netze/netz8.html>

Source Port: Identifiziert den sendenden Prozess.

Destination Port: Identifiziert den Prozess des Zielknotens.

Length: Länge des UDP-Datagramms in Bytes (mindestens 8 = Headerlänge)

UDP-Checksum: Optionale Angabe einer Prüfsumme (falls nicht verwendet auf Null gesetzt).

Die Verwaltungsinformationen, die das TCP-Protokoll und das IP-Protokoll in ihren Headern generieren, sind noch weit komplizierter.

42.5 IP-Adressen

Die vom IP-Protokoll vergebene IP-Nummer (oder IP-Adresse) ist die Adresse eines Computers im Internet. Die IP-Adresse besteht aus einer weltweit einmaligen 32-Bit-Zahl. Die ersten Bits kennzeichnen die Klasse des Netzes, in dem sich der Computer befindet: A (großes Netz), B (mittelgroßes Netz) oder C (kleines Netz). Anschließend folgen Bits zur Identifikation des Netzes. Die restlichen Bits identifizieren den jeweiligen Computer.

Zur leichteren Lesbarkeit werden die 32-Bit-Adressen in vier Bytes zerlegt und durch Punkte getrennt dezimal dargestellt.

Bsp.: 193.246.253.248

Anmerkung: Hier wurde die heute noch gebräuchliche Version 4 (IPv4) beschrieben. Der Adressraum ist bei IPv4 jedoch inzwischen deutlich zu klein, so dass in den nächsten Jahren von IPv4 auf IPv6 umgestellt werden muss (die Umstellung läuft bereits – aber so lange noch nicht alle Teilnehmer im Internet mit IPv6 umgehen können wird parallel weiterhin auch mit IPv4 gearbeitet). Der Adressraum von IPv6 ist so riesig ($3.4 \cdot 10^{38}$ Adressen statt $4.3 \cdot 10^9$ Adressen bei IPv4), dass man theoretisch jedem Molekül im Universum seine eigene IP-Adresse geben könnte. IPv6 Adressen sehen etwas anders aus:

```
> host heise.de
heise.de has address 193.99.144.80
heise.de has IPv6 address 2a02:2e0:3fe:100::8
```

42.6 Host- und Domain-Namen

Wir Menschen können uns IP-Nummern äußerst schlecht merken. Darum dürfen Computer auch Namen besitzen.

Als *Host* wird der Name des Rechners bezeichnet, als *Domainname* der Rest der Adresse. Ein Beispiel: pop3.web.de — pop3 ist der Hostname und web.de der Domainname. Das Ganze (pop3.web.de) wird auch als *Fully Qualified Domain Name* (FQDN) bezeichnet.

Um zwischen dem für uns Menschen und den für die Adressierung tatsächlich verwendeten numerischen IPs hin und her übersetzen zu können braucht es einen Dienst: den *Domain Name Service* (DNS). Ein Rechner, der diesen Dienst anbietet wird *DNS-Server* genannt.

Die Zuordnung IP ↔ FQDN ist nicht eindeutig. In keine Richtung! Zu einer IP gehören oft mehrere FQDNs, da auf einem Server oft mehrere Dienste angeboten werden (z.B. www.example.com für den HTTP-Server, smtp.example.com für den SMTP-Server). Außerdem werden von großen Hostern aber auch innerhalb von Firmen oft mehrere Webserver virtuell auf dem gleichen physikalischen Server angeboten.

```
> host www.humboldtgymnasium-bremen.de
www.humboldtgymnasium-bremen.de has address 89.31.143.7
> host www.gabys-modewelt.de
www.gabys-modewelt.de has address 89.31.143.7
> host www.waldorfschule-hassfurt.de
www.waldorfschule-hassfurt.de has address 89.31.143.7
> host www.sommerbadminden.de
www.sommerbadminden.de has address 89.31.143.7
```

(Es gibt noch viele weitere Webseiten, die alle auf dem selben Webserver gehostet sind wie unsere Schul-Homepage und deshalb auch alle auf die gleiche IP-Nummer zeigen)

In diesem Fall lässt sich der Webserver übrigens auch nicht über die IP adressieren: Zwar kommt die Verbindung zustande, aber der Server weiß ja nicht für welche der Domains, für die er zuständig ist, er Daten ausliefern soll.

Umgekehrt „verstecken“ sich hinter einem FQDN oft auch mehrere physikalische Server. Das kann zum einen zur Steigerung der Ausfallsicherheit wünschenswert sein, aber oft auch allein schon um den möglichen Durchsatz zu steigern. Oder könnt ihr euch vorstellen, dass hinter `www.google.com` nur ein einziger physikalischer Server hängt?

```
> host www.google.com
www.google.com has address 173.194.112.16
www.google.com has address 173.194.112.17
www.google.com has address 173.194.112.18
www.google.com has address 173.194.112.19
www.google.com has address 173.194.112.20
www.google.com has IPv6 address 2a00:1450:4001:800::1014
```

Das geht sogar noch weiter: Stichwort *Round Robin DNS*

Loopback-Adresse

Von besonderer Bedeutung für die Programmentwicklung ist die *Loopback-Adresse* `127.0.0.1`, die mit dem Namen `localhost` bezeichnet wird.

Datenpakete mit dieser Klasse-A-Adresse treffen, ohne jemals auf das Netzwerk zu gelangen, unmittelbar wieder beim sendenden Computer ein und können deshalb zum Funktionstest bei der Programmierung von Internet-Anwendungen verwendet werden. Oder um Programme auf dem lokalen Rechner über das TCP/IP-Protokoll zu erreichen (etwa den bei uns lokal laufenden Datenbankserver).

42.7 IP-Adressen in Java

Zur Arbeit mit IP-Adressen wird die Klasse `InetAddress` des Pakets `java.net` verwendet. Um ein Objekt der Klasse `InetAddress` zu generieren, ruft man die statische Methode `InetAddress.getByName()` auf und gibt als Parameter entweder eine IP-Adresse oder den Domain-Namen an (beides als String):

```
public static InetAddress getByName(String host) throws UnknownHostException
```

Wenn die Adresse vom DNS-Server nicht aufgelöst werden kann, wird eine `UnknownHostException` geworfen. Diese Exception sollte man abfangen und eine sinnvolle Fehlermeldung erzeugen.

Nachdem ein `InetAddress`-Objekt generiert wurde, kann man mit den beiden folgenden Methoden den Domain-Namen und die IP-Adresse des Rechners ermitteln:

```
String getHostName()      // gibt den Domain-Namen zurück
String getHostAddress()  // gibt die IP-Adresse zurück (als String)
```

Beispiel-Code (Auszug):

```
import java.net.*;
...
String eingabe = "127.0.0.1";
InetAddress inet = InetAddress.getByName(eingabe);
String host = inet.getHostName();           // Rückgabe: "localhost"
String ip = inet.getHostAddress();          // Rückgabe: "127.0.0.1"
```

42.8 Port-Nummern

Da häufig mehrere Server auf einem Rechner laufen, reicht für die Kommunikation die IP-Adresse des Rechners nicht aus. Das TCP-Protokoll übernimmt die Adressierung der Anwendungen auf einem Rechner und vergibt dazu sogenannte Port-Nummern, die zwischen 0 und 65535 liegen. Portnummern im Bereich von 0 bis 1023 sind für Anwendungen mit Administratorrechten reserviert. Für bekannte Internet-Dienste gibt es festgelegte Portnummern (*Well Known Ports*). Beispiel:

| Internet-Dienst | FTP (Filetransfer) | HTTP (Web-Server) | SMTP (Mail-Versand) |
|-----------------|--------------------|-------------------|---------------------|
| Portnummer | 21 | 80 | 25 |

42.9 Übungsaufgaben: Protokolle

Aufgabe 1: Vier Gewinnt

Das Spiel „Vier Gewinnt“ soll als Client-Server-Anwendung programmiert werden.

Du hast die Aufgabe auf Papier ein geeignetes Protokoll zu entwerfen, mit dem der Client und der Server untereinander Daten austauschen können. Ein Programmierer erklärt dir seinen Plan für die Spielfläche:

Jeder Spieler sieht auf dem Bildschirm ein Spielbrett mit einer Breite von sieben und einer Höhe von sechs Feldern vor sich. Beide Spieler dürfen abwechselnd immer einen runden Stein in eine Spalte des Spielbretts fallen lassen (entweder durch Mausklick oder in dem sie in einem Eingabefeld die Spaltennummer eingeben). Der Stein fällt auf das unterste freie Feld der Spalte. Wenn ein Spieler es schafft vier seiner Steine entweder waagerecht, senkrecht oder diagonal in eine Reihe zu bekommen, hat er gewonnen. Der Spieler, der anfängt, hat rote Steine. Der andere Spieler hat gelbe Steine. Neben dem Spielbrett befinden sich zwei Buttons. Mit dem ersten Button kann das Spiel vorzeitig beendet werden, wenn ein Spieler aufgeben möchte. Mit dem zweiten Button kann ein neues Spiel begonnen werden. Der Computer entscheidet per Zufallsgenerator darüber, welcher Spieler anfangen kann und die roten Steine erhält. Außerdem gibt es für die Spieler eine Chat-Möglichkeit. Es gibt ein Eingabefeld, in das der Spieler einen Text eintragen kann, den er seinem Mitspieler schicken möchte. Daneben befindet sich ein Button zum Abschicken der Nachricht. In einer TextArea werden die vom Mitspieler empfangenen Nachrichten angezeigt.

- Skizziere die Oberfläche des Spiels „Vier Gewinnt“.
- Entwirf ein geeignetes Protokoll.

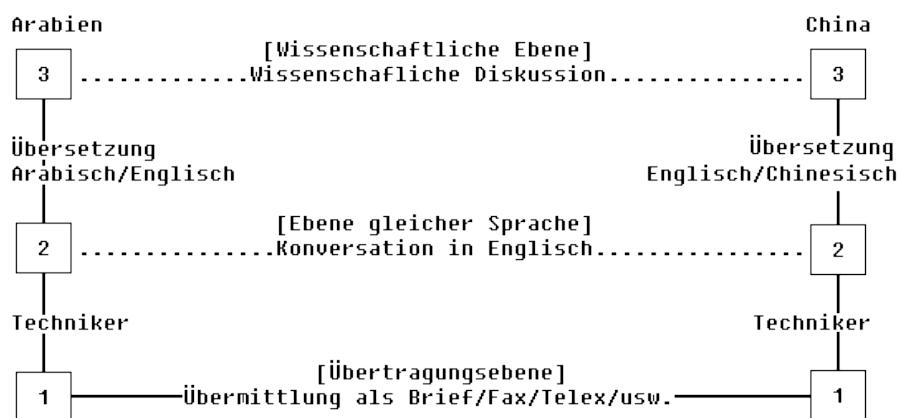
Aufgabe 2: Schiffe Versenken

Das Spiel „Schiffe versenken“ soll als Client-Server-Anwendung programmiert werden.

- Entwirf eine geeignete Programmoberfläche.
- Entwirf ein Protokoll, mit dem der Client und der Server untereinander Daten austauschen können.

Aufgabe 3: Veranschaulichung des Schichtenmodells

Das Beispiel arbeitet nur mit drei Schichten. Die Ausgangssituation besteht in zwei Wissenschaftlern in Arabien und China, die ein Problem diskutieren wollen. Beide sprechen nur Ihre Landessprache und auch Dolmetscher, die Arabisch und Chinesisch können, sind nicht aufzutreiben. Beide suchen sich nun Dolmetscher, die Englisch können. Der Weg der Nachrichten:



Grafik entnommen von <http://www.netzmafia.de/skripten/netze/netz0.html>

Aufgaben

- Der arabische Wissenschaftler möchte an den chinesischen Wissenschaftler einen Brief schicken. Beschreibe den Ablauf des Vorgangs.
- Mit welchen der beteiligten Personen tritt der arabische Wissenschaftler direkt in Kontakt?
- Welche Kenntnisse benötigt der arabische Wissenschaftler über den arabischen Techniker und den chinesischen Techniker?

Aufgabe 4: TCP und UDP

- Der Header des UDP-Protokolls ist recht übersichtlich. Welche zusätzlichen Informationen muss das TCP-Protokoll in seinen Header packen, damit es die Inhalte der Datenpakete in der richtigen Reihenfolge zusammenbauen kann und mitbekommt, ob ein Datenpaket verloren gegangen ist? Da die Datenpakete verschiedene Wege im Netz nehmen können, kann nicht vorher gesagt werden, in welcher Reihenfolge die Datenpakete das Ziel erreichen.
- Warum steht im Header des UDP-Protokolls keine Information über die Adresse des Rechners, zu dem die Daten gesendet werden sollen?

42.10 Übungsaufgaben: IP-Adressen und Ports

Aufgabe 5: IP-Adresse des eigenen Rechners

Ermittle die IP-Nummer deines Rechners.

Aufgabe 6: Ports

- Erkläre was man in der Informatik unter einem Port versteht.
- Der folgende Text beschreibt, wie Hacker einen PC unter Verwendung der Ports angreifen können. Beschreibe den Sachverhalt mit deinen eigenen Worten.

Ports – ein offenes Tor

Jedes Programm, das eine Schnittstelle zum Internet bereit stellt, muss einen Port (eine Tür) öffnen, damit ihr PC Daten senden und empfangen kann. Ist so ein Port einmal geöffnet, kann er theoretisch von jedermann (auch missbräuchlich) benutzt werden. Allerdings kann ein Eindringling, um in diesem Bild zu bleiben, von außen dieses Tor nicht durchschreiten. Er benötigt einen „Helfer im Haus“, der für ihn Aufgaben wie Datendiebstahl oder Sabotage erledigt. Die Anweisungen an den „Helfer“ oder der Transport der gestohlenen Daten werden nur durch das geöffnete Tor ermöglicht. Ein geöffneter Port stellt also immer eine potentielle Gefahr dar.

Weshalb sind Ports geöffnet und wie kommen die internen „Helfer“ in den PC?

Wie oben beschrieben, benötigt das Betriebssystem Ports für die externe Kommunikation und erledigt bei Anfragen über diese Ports bestimmte Aufgaben. Einige „Hacker“ benutzen Insiderkenntnisse und missbrauchen (i.d.R. bei MS-Windows) das Betriebssystem für kriminelle Zugriffe. Wenn dies möglich ist, darf man das getrost als schwere Sicherheitslücke bezeichnen.

Solche Hacker verbreiten nun Programme, die sich als nette, kleine Spielerei, als Schutzsoftware oder als kostenfreier Internetzugang usw. tarnen. AOL4FREE.EXE ist so ein Programm. Diese Programme haben oft nur die eine Aufgabe, sich fest in ihrem System zu installieren und einen Port (als Hintertür) zu öffnen. Ein Geschenk vom Gegner, das sich später als Falle herausstellt, kennt man aus der Geschichte als *Trojanisches Pferd*. Odysseus besiegte mit dieser List die belagerte Stadt Troja. Programme, die nach dem gleichen Muster arbeiten, bezeichnet man ebenfalls als Trojaner oder als „Backdoor“, da sie dem Angreifer eine Hintertür öffnen.

Entsprechend ihrer Programmierung können solche Trojaner Daten und Programme zerstören, vertrauliche Daten wie Geschäftsgeheimnisse, Kontonummern, Passwörter und PINs ausspionieren und damit erheblichen Schaden anrichten.

Insgesamt stehen 65535 verschiedene Ports zur Verfügung.

Da es jedem Programmierer letztlich selbst überlassen ist, welchen Port er nutzt, könnte sich beispielsweise ein Trojaner hinter dem Port 80 verstecken, solange auf ihrem PC kein Webserver läuft. Port 80 ist ein *well known Port* und sollte eigentlich nur für die HTTP-Übertragung genutzt werden.

Aufgabe 7: Programmierübung

Programmiere ein Java-Frame, das eine vom Benutzer eingegebene Rechner-Adresse (IP-Nummer oder Domain-Name) vom DNS-Server überprüfen lässt und den vom DNS-Server ermittelten Namen und die IP-Adresse in zwei Textfeldern ausgibt. Falls der DNS-Server die Adresse nicht auflösen kann, wird eine Fehlermeldung ausgegeben.



43 Client/Server

43.1 Client

Sockets

Die Programmierschnittstelle zum Zugriff auf ein TCP/IP-Netz bezeichnet man als Socket. Ein Socket ist stream-basiert, das heißt die Daten werden in einem „Strom“ von Bytes (ein Byte nach dem anderen) über das Netz geschickt und in derselben Reihenfolge empfangen.

Ein Client-Programm nimmt Verbindung zu einem Server auf, in dem es ein Objekt der Klasse `Socket` aus dem Package `java.net` erzeugt. Im Konstruktor wird die Rechner-Adresse (IP-Name, Domain-Name oder `InetAddress`-Objekt) und die Port-Nummer des Servers angegeben:

```
public Socket(String host, int port) throws UnknownHostException, IOException
```

oder

```
public Socket(InetAddress address, int port) throws IOException
```

Wenn die Rechner-Adresse nicht aufgelöst werden konnte, wird eine `UnknownHostException` erzeugt. Wenn der Socket nicht geöffnet werden konnte, wird eine `IOException` erzeugt.

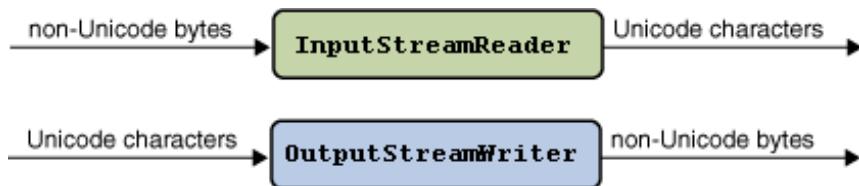
Streams

Nachdem die Socket-Verbindung erfolgreich aufgebaut wurde, kann mit den beiden Methoden `getInputStream()` und `getOutputStream()` je ein Stream zum Empfangen und zum Senden von Daten beschafft werden. Um Streams verwenden zu können, muss das Package `java.io` importiert werden.

```
public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException
```

Man könnte direkt mit diesen beiden Streams arbeiten, um Daten über das Netzwerk zu lesen bzw. zu schreiben. Aber das ist nicht zu empfehlen (zumindest dann nicht, wenn nicht nur Binärdaten in Form von einzelnen Bytes, sondern Zeichen bzw. Texte verschickt werden sollen). Besser ist es das `OutputStream`-Objekt in einen `OutputStreamWriter` und das `InputStream`-Objekt in einen `InputStreamReader` zu verpacken. Das bietet nämlich den Vorteil, dass man die Kodierung des Zeichensatzes festlegen kann.

```
public InputStreamReader(InputStream is, String charsetName)
    throws UnsupportedEncodingException
public OutputStreamWriter(OutputStream os, String charsetName)
    throws UnsupportedEncodingException
```



Sollen hingegen Binärdaten gelesen und geschrieben werden, dann arbeitet man direkt mit dem `InputStream`- bzw. `OutputStream`-Objekten (wird bei uns im Unterricht nicht vorkommen).

Einlesen von Daten

Zum Lesen von Daten aus dem Stream besitzt die Klasse `InputStreamReader` unter anderem die folgende Methode:

```
public int read() throws IOException
```

`read()` liest das nächste Zeichen vom Datenstrom ein. Wenn ein Zeichen übertragen wurde, ist das Ergebnis eine Zahl, die mit einem Type-Cast in einen Buchstaben (`char`) umgewandelt werden kann. Falls keine Daten verfügbar sind, blockiert die Methode bis das nächste Zeichen angekommen ist. Wenn die Datenverbindung geschlossen wurde, wird die Zahl `-1` zurückgegeben.

Beispiel: Einlesen von Daten aus einem Socket (Code-Auszug)

```
int zeichen;
String text = "";
try (Socket socket = new Socket ("localhost", 12345)) {
    InputStreamReader in = new InputStreamReader(socket.getInputStream(), "UTF-8");
    while ((zeichen = in.read()) != -1) {
        text += (char) zeichen;
        System.out.println(text);
    }
} catch (Exception e) {
    System.out.println("Fehler: " + e.getMessage());
}
```

Ausgabe von Daten

Zur Ausgabe von Daten besitzt die Klasse `OutputStreamWriter` unter anderem die folgenden Methoden:

```
public void write(int zeichen) throws IOException
public void write(String text) throws IOException
```

Die erste Variante der `write()`-Methode schreibt ein einzelnes Zeichen in den Datenstrom, die untere Variante schreibt einen String. Die mit `write()` geschriebenen Daten werden zunächst (intern) zwischengepuffert und erst dann tatsächlich verschickt, wenn der Puffer gefüllt ist. Mit der Methode `flush()` kann man das tatsächliche Versenden sofort erzwingen:

```
public void flush() throws IOException
```

Beispiel für die Verwendung der Methoden:

```
String text = "Hallo";
try (Socket socket = new Socket ("localhost", 12345)) {
    OutputStreamWriter out = new OutputStreamWriter(socket.getOutputStream(), "UTF-8");
    out.write(text);
    out.write(System.lineSeparator());           // Zeilenumbruch senden
    out.flush();
} catch (Exception e) {
    System.out.println("Fehler: " + e.getMessage());
}
```

Verbindung beenden

Nach Ende der Kommunikation sollte der Socket selbst mit der Methode `close()` geschlossen werden:

```
public void close() throws IOException
```

Durch das Schließen des Sockets werden auch der In- und OutputStream mit geschlossen.

43.2 Server

Die Programmierung eines Servers funktioniert ganz ähnlich wie die eines Clients. Lediglich der Verbindungsauflauf ist etwas komplizierter:

Ein Server muss in der Lage sein mit beliebig vielen Clients gleichzeitig zu kommunizieren. Für jede Client-Verbindung benötigt er einen eigenen Socket. Wenn der Server gestartet wird, öffnet er einen Haupt-Socket, in dem er auf eingehende Verbindungen wartet. Für jede aufgebaute Verbindung generiert das System einen neuen Socket, damit der Haupt-Socket für weitere „Anrufe“ frei bleibt.

Den Haupt-Socket eines Servers generiert man mit der Klasse `ServerSocket` aus dem Package `java.net`. Wichtig sind der Konstruktor der Klasse und die Methode `accept()`:

```
public ServerSocket(int port)    // erzeugt Haupt-Socket mit angegebener Port-Nummer
public Socket accept()          // wartet auf eingehende Verbindungen
```

Die Methode `accept()` blockiert so lange, bis sich ein Client bei der Server-Anwendung meldet. Ist der Verbindungsauflauf erfolgt, liefert `accept()` ein `Socket`-Objekt zurück, das zur Kommunikation mit dem Client verwendet werden kann.

Beispiel-Code (Auszug)

```
try (ServerSocket hauptSocket = new ServerSocket(12345)) {
    while (true) {
        Socket clientSocket = hauptSocket.accept();
        OutputStreamWriter out =
            new OutputStreamWriter(clientSocket.getOutputStream(), "UTF-8");
        String text = "Hallo!";
        out.write(text);
        out.flush();
        clientSocket.close();
    }
} catch(Exception e) {
    System.out.println("Fehler: " + e.getMessage());
}
```

43.3 Allgemeiner Aufbau von Client- und Server-Programmen

Sowohl Clients als auch Server haben fast immer den gleichen Aufbau.

- Clients brauchen neben der Klasse für das Anwendungsfenster eine Thread-Klasse, in der auf eingehende Daten vom Server reagiert werden kann.
- Server haben im Vergleich zum Client üblicherweise ein deutlich weniger komplexes User-Interface (Anwendungsfenster) oder verzichten gar ganz auf ein solches. Dafür brauchen sie zwei Thread-Klassen. Im Haupt-Thread wartet der Server auf eingehende Verbindungswünsche von Clients. Dazu erstellt er zunächst einen Server-Socket und „horcht“ auf diesem auf eingehende Verbindungen. Sobald solch eine Verbindung zustande kommt erzeugt der Haupt-Thread einen normalen Socket und übergibt diesen für die weitere Kommunikation der zweiten Thread-Klasse: dem Client-Thread. Der Haupt-Thread ist jetzt wieder frei für weitere Clients. Auf diese Weise kann ein Server mit quasi beliebig vielen Clients gleichzeitig kommunizieren. Für jede Client-Verbindung wird dann jeweils ein eigener Client-Thread erzeugt.

Im Folgenden ist ein Gerüst für Client und Server gegeben. Es ist noch ohne echte Funktionalität. Zwar wird eine Verbindung aufgebaut (Sockets, sowie `InputStreams` und `OutputStreams` auf Server- und Client-Seite erstellt), aber es ist noch kein Protokoll implementiert.

Genau darin: In der Implementierung des jeweiligen Protokolls liegt eure eigentliche Arbeit!

Neben der Implementierung des Protokolls muss oft noch anderes geleistet werden: Bei schreibenden Dateizugriffen sowie schreibenden Zugriffen auf Textfelder und andere Elemente des Anwendungsfensters muss sichergestellt

werden, dass nicht mehrere Threads gleichzeitig die selben Daten verändern wollen. Dies muss über einen geeignet gewählten *Monitor* in einem **synchronized**-Block gewährleistet werden.

Tipps

- Nehmt euch ausreichend Zeit um das Protokoll zu verstehen! Da die Umsetzung des Protokolls der schwierigste Teil der Client/Server-Programmierung ist, lohnt es sich zunächst mit Stift und Papier die verschiedenen Nachrichten des Protokolls aufzuschreiben und zu strukturieren. Danach wird euch die Implementierung im Programm erheblich leichter fallen!
- Testet immer gegen einen bestehenden (und verlässlich funktionierenden) „Gegenspieler“. Bei Aufgaben, in denen sowohl Client- als auch Server entwickelt werden sollen, ist es extrem ratsam, zunächst gegen einen bereits vorhandenen (vom Lehrer zur Verfügung gestellten) Server oder Client zu testen. Wenn ihr zuerst den Client entwickeln wollt, dann testet ihr gegen den Server den ihr vom Lehrer bekommen habt. Und umgekehrt. Ansonsten werdet ihr mit Sicherheit viel Zeit und Nerven lassen. Mit einem halbfertigen Client gegen einen halbfertigen Server zu testen macht keinen Spaß!
- In **catch**-Blöcken unbedingt immer mit **e.printStackTrace()** die komplette Fehlermeldung ausgeben lassen. Dort wird euch neben dem Fehlercode sogar ein Hinweis auf die konkrete Stelle in eurer Java-Datei gegeben, in der die Ausnahme (Exception) erzeugt wurde. Dies nicht zu tun ist fahrlässig und kostet euch im Zweifelsfall viel Zeit und Punkte!
- „Sprechende“ Bezeichner verwenden. **btnSenden** ist erheblich aussagekräftiger als **button3!** Genau so wie **serverName** instruktiver ist als **text**!

Server-Gerüst

Datei Server.java:

```
import java.awt.*;
import java.awt.EventQueue;
import javax.swing.*;
import javax.swing.border.EmptyBorder;

public class Server extends JFrame {
    // globale Variablen
    private static final int WIDTH = 300;
    private static final int HEIGHT = 300;
    private HauptThread thread;

    public Server(final String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new BorderLayout(0, 0));
        contentPane.setPreferredSize(new Dimension(WIDTH, HEIGHT));
        setContentPane(contentPane);
        pack();
        setLocationRelativeTo(null);
        setResizable(false);
        setVisible(true);

        thread = new HauptThread(this);
        thread.start();
    }
}
```

```

public static void main(final String[] args) {
    EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            try {
                new Server("Server");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
}

```

Datei HauptThread.java

```

import java.net.*;

public class HauptThread extends Thread {
    // Die Aufgabe des Hauptthreads ist es, auf eingehende Verbindungswünsche
    // von Clients zu warten und diese dann mit einem Client-Socket zu
    // "versorgen".
    // Sobald dies geschehen ist wird der Client-Thread gestartet (dieser
    // bekommt Client-Socket als Parameter übergeben).
    Server main;

    public HauptThread(Server main) {
        this.main = main;
        // Der Zugriff auf die Klasse des Anwendungsfensters über "main"
        // ist wichtig, wenn im Hauptthread oder im Clientthread auf Elemente
        // Des Benutzer-Interfaces (Textfelder etc.) des Anwendungsfensters
        // zugegriffen werden soll.
    }

    @Override
    public void run() {
        try (ServerSocket serverSocket = new ServerSocket(22222)) {
            while (true) {
                Socket socket = serverSocket.accept();
                ClientThread client = new ClientThread(main, socket);
                client.start();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Anmerkung

Die HaupThread-Klasse kann so in den meisten Fällen fast oder völlig unverändert benutzt werden!

```

Datei ClientThread.java

import java.net.*;
import java.io.*;

public class ClientThread extends Thread {
    Server main;
    Socket socket;

    public ClientThread(Server main, Socket sock) {
        this.main = main;
        this.socket = sock;
    }

    @Override
    public void run() {
        try {
            int zeichen;
            InputStreamReader in =
                new InputStreamReader(socket.getInputStream(), "UTF-8");
            OutputStreamWriter out =
                new OutputStreamWriter(socket.getOutputStream(), "UTF-8");
            while ((zeichen = in.read()) != -1) {
                // Hier werden die Daten vom Client gelesen und verarbeitet
                // (InputStream)
                // und Daten zum Client geschickt (OutputStream)
                // Was hier tatsächlich zu tun ist wird durch das Protokoll
                // definiert

                // Das ist eure eigentliche Arbeit!

            }
            // Die Verbindung (InputStream des Servers bzw. OutputStream des
            // Clients wurde vom Client getrennt --> in.read() == -1
            // und damit die while-Schleife oben verlassen.
            // Jetzt muss nur noch aufgeräumt werden (der Socket - und indirekt
            // damit auch die Streams - werden geschlossen).
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Anmerkung

Die Übungsaufgabe zum Rechentrainer stellt einen Sonderfall dar, in so fern, als dort der Server nicht darauf wartet, dass der Client die Verbindung beendet

```
while ( (zeichen = in.read()) != -1 )
```

sondern vielmehr seinerseits mit zählt und nach fünf erfolgreich gelösten Aufgaben die Verbindung beendet:

```
for (int i = 0; i < 5; i++) {
    aufgabeSenden();
    antwortLesen();
}
```

Client-Gerüst

Datei Client.java

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.Socket;
import javax.swing.*;
import javax.swing.border.EmptyBorder;

public class Client extends JFrame {
    // globale Variablen
    private static final int WIDTH = 500;
    private static final int HEIGHT = 400;
    private JTextField tfServer, tfStatus, tfEingabe;
    private JButton btnVerbinden, btnTrennen, btnSenden;
    JTextArea textAreaAusgabe;
    private boolean verbunden = false;
    Socket socket;
    InputStreamReader in;
    private OutputStreamWriter out;
    private LeseThread thread;

    public Client(final String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(10, 10, 10, 10));
        contentPane.setPreferredSize(new Dimension(WIDTH, HEIGHT));
        setContentPane(contentPane);
        GridBagLayout gbl_contentPane = new GridBagLayout();
        gbl_contentPane.columnWidths = new int[] { 26, 322, 0, 0 };
        gbl_contentPane.rowHeights = new int[] { 0, 0, 0, 0, 0, 0 };
        gbl_contentPane.columnWeights = new double[] { 1.0, 1.0, 0.0,
            Double.MIN_VALUE };
        gbl_contentPane.rowWeights = new double[] { 0.0, 0.0, 0.0, 0.0, 1.0,
            Double.MIN_VALUE };
        contentPane.setLayout(gbl_contentPane);

        JLabel lblServer = new JLabel("Server:");
        GridBagConstraints gbc_lblServer = new GridBagConstraints();
        gbc_lblServer.anchor = GridBagConstraints.EAST;
        gbc_lblServer.insets = new Insets(0, 0, 5, 5);
        gbc_lblServer.gridx = 0;
        gbc_lblServer.gridy = 0;
        contentPane.add(lblServer, gbc_lblServer);

        tfServer = new JTextField();
        GridBagConstraints gbc_tfServer = new GridBagConstraints();
        gbc_tfServer.insets = new Insets(0, 0, 5, 5);
        gbc_tfServer.fill = GridBagConstraints.HORIZONTAL;
        gbc_tfServer.gridx = 1;
        gbc_tfServer.gridy = 0;
        contentPane.add(tfServer, gbc_tfServer);
        tfServer.setColumns(10);

        btnVerbinden = new JButton("verbinden");
        btnVerbinden.addActionListener(new ActionListener() {
```

```
public void actionPerformed(ActionEvent e) {
    verbinden();
}
});

GridBagConstraints gbc_btnVerbinden = new GridBagConstraints();
gbc_btnVerbinden.insets = new Insets(0, 0, 5, 0);
gbc_btnVerbinden.gridx = 2;
gbc_btnVerbinden.gridy = 0;
contentPane.add(btnVerbinden, gbc_btnVerbinden);

JLabel lblStatus = new JLabel("Status:");
GridBagConstraints gbc_lblStatus = new GridBagConstraints();
gbc_lblStatus.anchor = GridBagConstraints.EAST;
gbc_lblStatus.insets = new Insets(0, 0, 5, 5);
gbc_lblStatus.gridx = 0;
gbc_lblStatus.gridy = 1;
contentPane.add(lblStatus, gbc_lblStatus);

tfStatus = new JTextField();
tfStatus.setEditable(false);
GridBagConstraints gbc_tfStatus = new GridBagConstraints();
gbc_tfStatus.insets = new Insets(0, 0, 5, 5);
gbc_tfStatus.fill = GridBagConstraints.HORIZONTAL;
gbc_tfStatus.gridx = 1;
gbc_tfStatus.gridy = 1;
contentPane.add(tfStatus, gbc_tfStatus);
tfStatus.setColumns(10);

btnTrennen = new JButton("trennen");
btnTrennen.setEnabled(false);
btnTrennen.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        trennen();
    }
});
GridBagConstraints gbc_btnTrennen = new GridBagConstraints();
gbc_btnTrennen.fill = GridBagConstraints.HORIZONTAL;
gbc_btnTrennen.insets = new Insets(0, 0, 5, 0);
gbc_btnTrennen.gridx = 2;
gbc_btnTrennen.gridy = 1;
contentPane.add(btnTrennen, gbc_btnTrennen);

JLabel lblEingabe = new JLabel("Eingabe:");
GridBagConstraints gbc_lblEingabe = new GridBagConstraints();
gbc_lblEingabe.anchor = GridBagConstraints.EAST;
gbc_lblEingabe.insets = new Insets(0, 0, 5, 5);
gbc_lblEingabe.gridx = 0;
gbc_lblEingabe.gridy = 2;
contentPane.add(lblEingabe, gbc_lblEingabe);

tfEingabe = new JTextField();
GridBagConstraints gbc_tfEingabe = new GridBagConstraints();
gbc_tfEingabe.insets = new Insets(0, 0, 5, 5);
gbc_tfEingabe.fill = GridBagConstraints.HORIZONTAL;
gbc_tfEingabe.gridx = 1;
gbc_tfEingabe.gridy = 2;
contentPane.add(tfEingabe, gbc_tfEingabe);
tfEingabe.setColumns(10);
```

```
btnSenden = new JButton("senden");
btnSenden.setEnabled(false);
btnSenden.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        senden();
    }
});
GridBagConstraints gbc_btnSenden = new GridBagConstraints();
gbc_btnSenden.fill = GridBagConstraints.HORIZONTAL;
gbc_btnSenden.insets = new Insets(0, 0, 5, 0);
gbc_btnSenden.gridx = 2;
gbc_btnSenden.gridy = 2;
contentPane.add(btnSenden, gbc_btnSenden);

JLabel lblAusgabe = new JLabel("Ausgabe:");
GridBagConstraints gbc_lblAusgabe = new GridBagConstraints();
gbc_lblAusgabe.anchor = GridBagConstraints.EAST;
gbc_lblAusgabe.insets = new Insets(0, 0, 5, 5);
gbc_lblAusgabe.gridx = 0;
gbc_lblAusgabe.gridy = 3;
contentPane.add(lblAusgabe, gbc_lblAusgabe);

JScrollPane scrollPane = new JScrollPane();
GridBagConstraints gbc_scrollPane = new GridBagConstraints();
gbc_scrollPane.gridwidth = 3;
gbc_scrollPane.insets = new Insets(0, 0, 0, 5);
gbc_scrollPane.fill = GridBagConstraints.BOTH;
gbc_scrollPane.gridx = 0;
gbc_scrollPane.gridy = 4;
contentPane.add(scrollPane, gbc_scrollPane);

textAreaAusgabe = new JTextArea();
scrollPane.setViewportView(textAreaAusgabe);

pack();
setLocationRelativeTo(null);
setResizable(false);
setVisible(true);
}

public void verbinden() {
    // Die Verbindung zum Server wird aufgebaut.
    // InputStreamReader und OutputStreamWriter des Sockets werden
    // geholt.
    // Anschließend wird der InputStreamReader an den Lesethread
    // übergeben und dieser gestartet.

    try {
        if (!verbunden) {
            verbunden = true;
            String serverName = tfServer.getText();
            socket = new Socket(serverName, 22222);
            System.out.println("Verbindung hergestellt!");
            in = new InputStreamReader(socket.getInputStream(), "UTF-8");
            out = new OutputStreamWriter(socket.getOutputStream(), "UTF-8");
            thread = new LeseThread(this, in);
            thread.start();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
        tfServer.setEnabled(false);
        btnVerbinden.setEnabled(false);
        btnTrennen.setEnabled(true);
        btnSenden.setEnabled(true);
        tfStatus.setText("verbunden");
    }
} catch (Exception exc) {
    tfStatus.setText("Fehler: " + exc.getMessage());
    tfServer.setEnabled(true);
    btnVerbinden.setEnabled(true);
    btnTrennen.setEnabled(false);
    btnSenden.setEnabled(false);
    tfStatus.setText("getrennt");
    verbunden = false;
}
}

public void senden() {
    if (verbunden) {
        try {
            // Was hier gesendet wird (und auch wodurch das Senden
            // ausgelöst wird) wird durch das Protokoll definiert!
            //
            // ACHTUNG: Je nach Definition des Protokolls kann das
            // Senden auch unabhängig von Benutzereingaben geschehen.
            // Etwa als automatisierte Reaktion auf eine Anfrage vom
            // Server. In diesem Fall würde dies vermutlich im Lesethread
            // geschehen. Dieser müsste dann neben dem InputStreamReader-
            // auch das OutputStreamWriter-Objekt als Parameter
            // übergeben bekommen.
        } catch (Exception exc) {
            textAreaAusgabe.append("Fehler: " + exc.getMessage());
        }
    }
}

public void trennen() {
    System.out.println("trennen");
    try {
        socket.close();
        tfStatus.setText("getrennt");
        tfServer.setEnabled(true);
        btnVerbinden.setEnabled(true);
        btnTrennen.setEnabled(false);
        btnSenden.setEnabled(false);
        verbunden = false;
    } catch (Exception exc) {
        tfStatus.setText("Fehler: " + exc.getMessage());
    }
}

public static void main(final String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                new Client("Client");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

```
        }
    }
});  
}  
}
```

Anmerkungen

Die Datei Client.java ist zwar mit Abstand die Größte, aber für euch vermutlich nicht die Schwierigste! Sie ist nur deshalb so groß, weil hier das User-Interface implementiert wird.

Die anspruchsvollere Arbeit steckt in der `while`-Schleife im Lese-Thread. Dort wird ein Großteil des Protokolls implementiert!

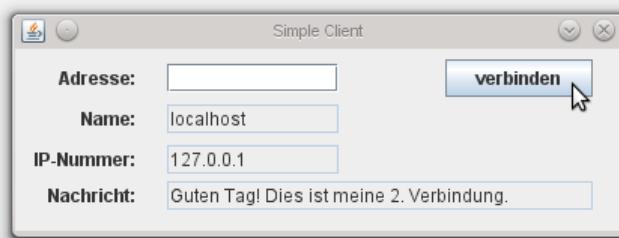
Datei LeseThread.java

```
import java.io.InputStreamReader;  
  
public class LeseThread extends Thread {  
  
    Client main;  
    InputStreamReader in;  
  
    public LeseThread(Client main, InputStreamReader in) {  
        this.main = main;  
        this.in = in;  
    }  
  
    @Override  
    public void run() {  
        int zeichen;  
        try {  
            while ((zeichen = in.read()) != -1) {  
                // Hier werden die Daten vom Server gelesen und verarbeitet  
                // (InputStream)  
                // Was hier tatsächlich zu tun ist wird durch das Protokoll  
                // definiert  
  
                // Das ist eure eigentliche Arbeit!  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
        try {  
            main.socket.close();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

43.4 Übungsaufgaben: Client/Server

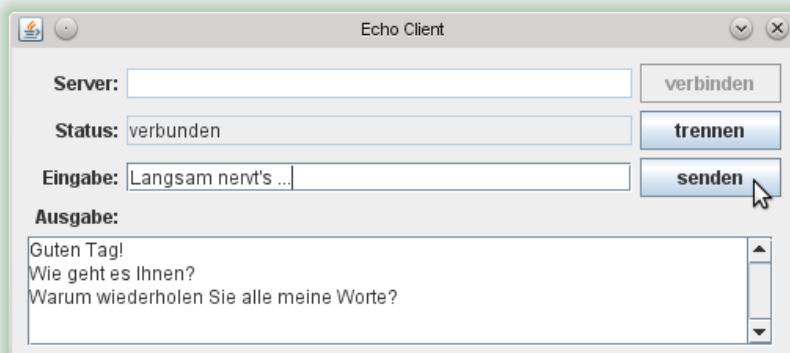
Aufgabe 1: Einfacher Client

Im Kurs-Repository findest du die ausführbare JAR-Datei SimpleServer.jar. Wenn du diese startest, läuft der Server auf Port 11111. Schreibe einen Client, der eine Verbindung zu diesem Server herstellt. Der Benutzer sollte die Bezeichnung des Servers in einem Textfeld eingeben können. Die Portnummer kann fest in das Programm eingebaut werden. Der Server sendet an den Client einen kurzen Text und beendet dann die Verbindung. Das Client-Programm soll den Text in einem Textfeld ausgeben.



Aufgabe 2: Echo-Client

Im Kurs-Repository findest du die ausführbare JAR-Datei EchoServer.jar. Wenn du diese startest, läuft der Server auf Port 22222. Der Echo-Server sendet alle Daten, die an ihn geschickt werden, wieder an den Absender zurück. Schreibe einen Client, der eine Verbindung zum Echo-Server aufbaut. Der Benutzer kann den Namen des Servers eingeben. Nachdem die Verbindung erfolgreich aufgebaut wurde, kann der Benutzer in einem Textfeld Sätze eintippen. Sobald er auf den daneben stehenden Button drückt, wird die eingegebene Zeile an den Server gesendet. Wenn eine Verbindung aufgebaut wird, wird für das Einlesen der Daten, die der Server zurücksendet, ein eigener Thread erzeugt. Dieser Lese-Thread schreibt die empfangenen Daten in eine JTextArea (eingebettet in eine JScrollPane) des Anwendungsfensters. Die Verbindung zum Echo-Server kann über einen Button getrennt werden. Wenn der Client die Verbindung beendet, schließt auch der Server die Verbindung. Der Lese-Thread erhält nach dem Schließen der Verbindung durch den Server eine IOException. Wenn er eine IOException erhält, soll sich der Lese-Thread selbstständig beenden. Dies überprüft man am besten durch eine Ausgabe auf der Konsole.



Tipp: Zeilenumbrüche:

Wenn man in der `JTextArea` eine neue Zeile anfangen möchte, kann man dies kodieren, indem man nacheinander die Zeichen *Carriage Return* (Wagenrücklauf, ASCII-Zeichen Nr. 13) und *Line Feed* (Zeilenvorschub, ASCII-Zeichen Nr. 10) sendet. Man kann entweder die ASCII-Zeichen als Integer-Werte senden oder sie mit Hilfe einer Escape-Sequenz in einem String kodieren. Beispiel für die Kodierung eines Zeilenumbruchs in einem String:

```
String text = "Erste Zeile\r\nZweite Zeile.;"
```

`\r` steht für Carriage Return

`\n` steht für Line Feed

Sinnvoll ist, dass der EchoClient beim Senden eines Strings an den Server automatisch einen Zeilenumbruch an den Text des Benutzer anfügt.

Allerdings werden Zeilenumbrüche je nach verwendetem Betriebssystem unterschiedlich kodiert (`\r\n` ist richtig für Windows). Sinnvoller ist es deshalb die statische Methode `System.lineSeparator()` zu benutzen. Sie erzeugt je nach verwendetem Betriebssystem die richtige Zeichenfolge:

```
String text = "Erste Zeile" + System.lineSeparator() + "Zweite Zeile.;"
```

Aufgabe 3: HTTP-Client

Das Herunterladen einer Web-Seite mit Hilfe des HTTP-Protokolls ist relativ einfach. Web-Server benutzen laut Konvention immer den TCP-Port 80. Nachdem ein Browser eine Verbindung zu einem HTTP-Server aufgebaut hat, kann er mit folgendem Befehl eine Seite herunterladen:

```
GET <Seite> <Protokoll>
Host: <Server-Adresse>
```

Wenn man zum Beispiel die URL

```
http://www.heise.de/developer/
```

laden möchte, baut man zuerst eine Verbindung zum Server mit dem Namen `www.heise.de` auf. Anschließend sendet man an den Server den Befehl:

```
GET /developer/ HTTP/1.1\r\n
Host: www.heise.de\r\n\r\n
```

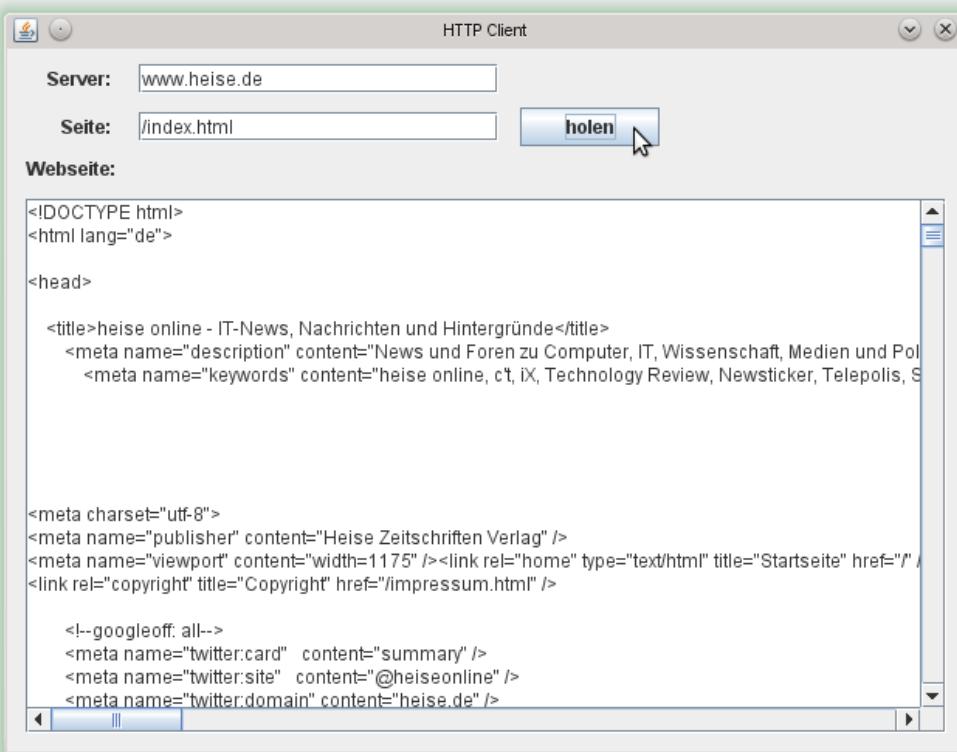
Am Ende des Befehls müssen zwei Carriage Return/Line Feed Anweisungen gesendet werden. Die dadurch gesendete Leerzeile am Ende bedeutet, dass keine weiteren Informationen gesendet werden.

Der Server überträgt zuerst einen „Header“ mit Zusatzinformationen wie beispielsweise den Servertyp und das Datum der letzten Änderung und anschließend den Inhalt der angeforderten Datei. Nach der Übertragung beendet der Server die Verbindung. Der Browser muss den übertragenen HTML-Code interpretieren und dem Benutzer geeignet anzeigen. Befinden sich in der Seite Verweise auf Images, Applets oder Frames, so fordert der Browser die entsprechenden Inhalte in weiteren GET-Transaktionen vom Server an.

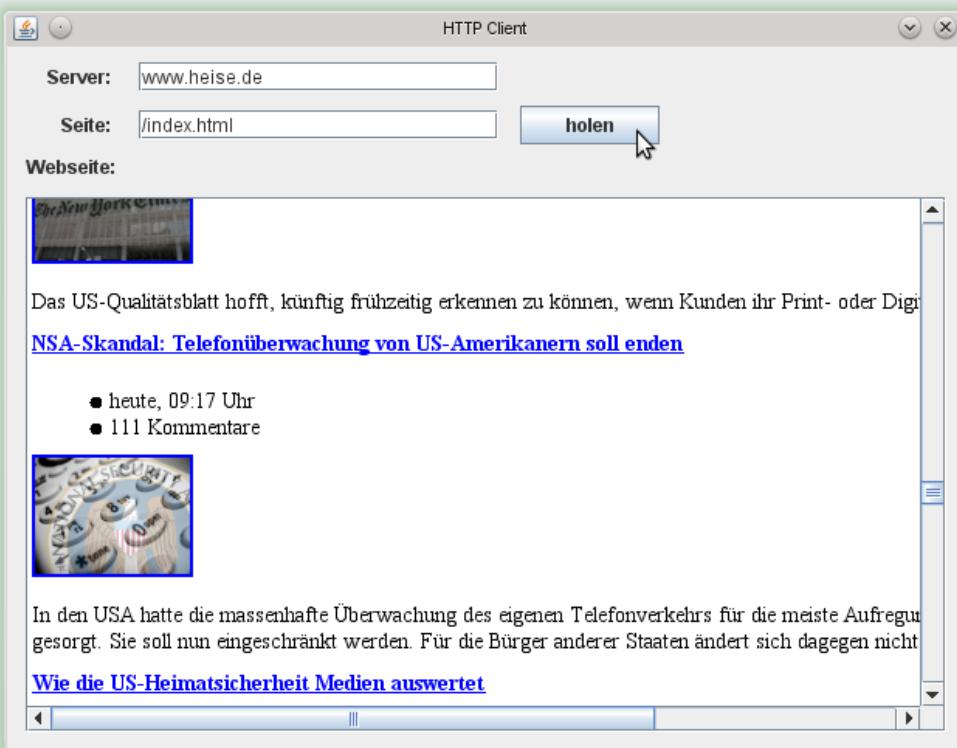
Aufgabe

Programmiere einen einfachen Mini-Browser, der eine Verbindung zu einem HTTP-Server im Internet aufbauen kann und eine vom Benutzer angegebene Seite anfordert. Die vom Server übertragenen Daten sollen in eine `JTextArea` geschrieben werden.

Damit die Verbindung während der Übertragung der Daten gegebenenfalls beendet werden kann, sollte das Lesen der Daten in einem eigenen Thread erfolgen, weil das Frame sonst während des Einlesens nicht auf Tastendrücke reagieren kann. In einer ersten, einfachen Test-Version kann man jedoch auf einen Hintergrund-Thread verzichten und die Daten direkt in der Ereignis-Behandlungs-Methode des Buttons (`actionPerformed()`) einlesen.



Tipp: Alternativ lässt sich mit wenig Zusatzaufwand die HTML-Seite auch richtig darstellen:



Dazu muss man lediglich die `JTextArea` durch eine `JTextPane` ersetzen. Dann kann man wie folgt vorgehen:

```
textPaneAusgabe = new JTextPane();
textPaneAusgabe.setEditorKit(new javax.swing.text.html.HTMLEditorKit());
```

Im Lese-Thread sammelt man dann zunächst alles, was vom HTTP-Server zurück geliefert wird, ein und speichert es in einer String Variablen ab.

Zum Schluss wird das Ergebnis dann in der `JTextPane`-Komponente dargestellt:

```
int zeichen;
String html = "";
while ((zeichen = in.read()) != -1) {
    html += (char) zeichen;
}
main.textPaneAusgabe.setText(html);
```

Aufgabe 4: POP3-Protokoll

Startet das Programm `GenericClient.jar` aus dem Kurs-Repository.

Arbeitsaufträge:

1. Überprüft zunächst ob dieser Client sich gegenüber dem SimpleServer (Port 11111) und dem EchoServer (Port 22222) genau so verhält wie die zuvor von euch selbst programmierten Clients.
2. Mit dem im Standard RFC 1939 beschriebene Post Office Protocol 3 (POP3) kann man e-Mails aus einem Postfach beim Provider abholen. POP3 benutzt den TCP-Port 110.
 - a) Sucht im Internet nach Informationen zum POP3-Protokoll. Ihr braucht für die weitere Arbeit eine Übersicht über die gängigsten POP3-Befehle.
 - b) Überlegt und diskutiert untereinander, ob und wie man unseren Client dazu verwenden könnte um e-Mails von einem POP3-Server abzuholen.
3. Zum Testen könnt ihr den Server `pop.1und1.de` benutzen. Versucht die Mails des Benutzers `AvHG@online.de` abzuholen. Als Passwort gebt ihr Humboldt an.
4. Ihr könnt auch versuchen vorhandene e-Mails zu löschen. Wenn dies erfolgreich war, solltet ihr an `AvHG@online.de` eine e-Mail schicken, damit das Postfach wieder gefüllt ist.
5. Überlegt und diskutiert, ob mit diesem Client beliebige Netzwerkserver (erfolgreich) kontaktiert und bedient werden können. Welche Protokolle könnten wir noch testen?

Aufgabe 5: Einfacher Server

Schreibe einen einfachen Server als Gegenstück zu dem einfachen Client, den wir als erstes programmiert haben. Der Server besitzt ein Textfeld mit einer Statuszeile, in der der aktuelle Zustand des Servers und die bisherige Anzahl der Verbindungen angezeigt wird (siehe Abbildung). Um die korrekte Arbeitsweise des Servers zu kontrollieren empfiehlt es sich, zusätzlich Debug-Ausgaben in die Konsole zu schreiben (zum Beispiel Verbindung aufgebaut, Verbindung beendet).



Da der Haupt-Thread des Frames immer auf verschiedene Ereignisse (wie zum Beispiel das Neuzeichnen des Fensters) reagieren muss, darf der Server nicht im Thread des Frames auf ankommende Verbindungen warten, denn damit würde er die Ereignisbehandlung blockieren. Es muss also ein zusätzlicher Thread generiert werden, in dem der Server ankommende Verbindungen entgegen nimmt. Wenn eine Verbindung aufgebaut ist, schickt der Server einen kurzen Text an den Client. Damit jedes Mal ein anderer Text generiert wird, wird in dem Text die aktuelle Verbindungsnummer eingebaut (zum Beispiel Dies ist meine 10. Verbindung.). Nach dem Senden des Textes beendet der Server die Verbindung zum Client und wartet mit `accept()` auf die nächste Verbindung. Dieser einfache Server braucht noch nicht mehrere Clients parallel bedienen zu können, da die Dauer einer Verbindung äußerst kurz ist.

Aufgabe 6: Echo-Server

Programmiere einen Echo-Server, der alle Eingaben, die er von einem Client erhält, wieder an den Client zurück sendet. Der Echo-Server soll mehrere Clients parallel bedienen können. Wenn eine Client-Verbindung aufgebaut wurde, generiert er für die Kommunikation mit dem Client einen eigenen Thread. Im Konstruktor des Threads wird der Socket, der für diesen Client erzeugt wurde, als Parameter übergeben. Der Thread beendet sich, wenn die Verbindung von Seiten des Clients geschlossen wurde. Dies kann entweder durch eine Exception angezeigt werden oder dadurch, dass der Server in der `read()`-Methode ein Byte mit dem Wert -1 erhält.

Zusatzaufgabe: Chat-Client und -Server

Client

Der Benutzer des Chat-Clients kann die Adresse des Servers und einen selbst gewählten Namen eingeben. Nachdem der Benutzer einen Button gedrückt hat, baut der Chat-Client eine Verbindung zum Chat-Server auf.

Der Benutzer kann über ein Textfeld zeilenweise Text eingeben, der an den Server gesendet wird und vom Server an alle anderen angemeldeten Clients weiter geleitet wird. Der Client sendet zuerst den selbstgewählten Namen des Benutzers (damit die anderen Benutzer wissen, wer den Text gesendet hat), dann folgen ein oder mehrere Trennzeichen und dahinter der vom Benutzer geschriebene Text, der mit einem Zeilenumbruch (`System.lineSeparator()`) endet:

Name: Text

Im Frame gibt es neben dem Eingabefeld eine `JTextArea`, in der alle gesendeten Texte angezeigt werden. Das Empfangen der Texte vom Server geschieht in einem Hintergrund-Thread. Die Texte können unverändert in die `JTextArea` eingetragen werden. Die durch den Aufruf von `System.lineSeparator()` erzeugten Steuerzeichen sorgen dafür, dass nach jedem Text ein Zeilenumbruch erfolgt.

Server

Der Server ermöglicht es einer beliebigen Anzahl Clients miteinander zu kommunizieren. In einem Textfeld zeigt er die Anzahl der Clients an, die momentan mit dem Server verbunden sind. Zur Verwaltung der Clients besitzt der Server außerdem eine Liste aller `OutputStreamWriter` der angeschlossenen Clients, die mit der Klasse `ArrayList` verwaltet wird.

Der Server erzeugt in einem speziellen Thread seinen Haupt-Socket (mit der vereinbarten Port-Nummer), in dem er auf eingehende Verbindungen wartet. Wenn ein neuer Client „anruft“ holt er sich den `OutputStreamWriter` für die neue Client-Verbindung und trägt ihn in die Liste der `OutputStreamWriter` ein. Dann generiert er einen eigenen Thread für die neue Client-Verbindung. Im Konstruktor des Threads werden das Anwendungsfenster sowie der `InputStreamReader` und `OutputStreamWriter` dieses speziellen Clients übergeben.

Jeder Client-Thread wartet auf eingehende Texte von seinem Client. Ein Text wird bis zum Zeilenumbruch eingelesen und anschließend in einer Schleife an alle Clients geschrieben (auch an den, der den Text gesendet hat, damit auch dieser den Text in seine `JTextArea` einträgt). Damit die Texte von zwei verschiedenen Benutzern sich nicht vermischen, muss sollte Senden der Texte an die Clients mit einem Monitor geschützt werden.

Wenn ein Client die Verbindung zum Server schließt, muss der für diesen Client zuständige Thread den `OutputStreamWriter` des Clients aus der Liste der `OutputStreamWriter`-Objekte löschen und die Anzeige der Clientanzahl im Anwendungsfenster aktualisieren. Anschließend beendet sich der Client-Thread.

Klasse ArrayList

Mit der Klasse `ArrayList` kann man dynamische Listen verwalten. Der Vorteil gegenüber der Verwendung von Arrays ist, dass die Klasse bei Bedarf automatisch Speicherplatz für neue Objekte anlegt, und dass man Objekte aus der Liste heraus löschen kann, ohne dass Lücken entstehen.

Benötigte Import-Anweisung

```
import java.util.ArrayList;
import java.util.List;
```

Anlegen einer `ArrayList`

Schema:

```
List<Datentyp> variable = new ArrayList<Datentyp>();
```

Beispiel:

```
List<OutputStreamWriter> listOSW = new ArrayList<OutputStreamWriter>();
```

Methoden (Auswahl):

| Methode | Beschreibung |
|--|--|
| <code>public boolean add(Datentyp o)</code> | Hängt ein neues Objekt an die Liste an. |
| <code>public boolean remove(Datentyp o)</code> | Löscht das angegebene Objekt aus der Liste. |
| <code>public void clear()</code> | Löscht alle Objekte aus der Liste. |
| <code>public int size()</code> | Gibt an, wie viele Objekte sich in der Liste befinden. |
| <code>public Datentyp get(int index)</code> | Gibt das Element mit dem angegebenen Index zurück. |

44 Abi-Training

44.1 Allgemeines

Aufgabe 1: Programmietechnik

1. Wie ruft man Methoden auf, vor denen das Schlüsselwort **static** steht?
2. Überprüfe mithilfe einer geeigneten Methode, ob in der **char**-Variable **c** eine Ziffer (Zahl zwischen 0 und 9) steht.
3. Überprüfe, ob in einer String-Variablen **s** die Zeichenkette "ja" steht.
4. Schneide die ersten vier Buchstaben aus dem String **s** aus.
5. Konvertiere den String **s** in eine Integer-Zahl.
Was passiert, wenn **s** keine Ziffern enthält?
6. Gegeben ist die Variable:

```
int i;
```

Überprüfe, ob die Zahl **i** durch fünf teilbar ist.
7. Konvertiere die Ganzzahl **i** in einen String.

Aufgabe 2: Netzwerktechnik

Beantworte folgende Fragen:

1. Was versteht man unter einem Protokoll?
2. Was ist das OSI-Schichten-Modell?
3. Benenne die vier Schichten des vereinfachten Schichten-Modells und erkläre ihre Bedeutung.
4. Auf welcher Schicht des OSI-Modells (bzw. des Vier-Schichten-Modells) liegt das IP-Protokoll? Welche Aufgabe hat es?
5. Auf welcher Schicht des OSI-Modells (bzw. des Vier-Schichten-Modells) liegt das TCP-Protokoll? Welche Aufgabe hat es?
6. Auf welcher Schicht des OSI-Modells (bzw. des Vier-Schichten-Modells) liegt das UDP-Protokoll? Welche Aufgabe hat es?
7. Was ist ein DNS-Server?

Aufgabe 3: Kryptologie

Erkläre die folgenden Verfahren und Fachbegriffe:

1. Caesar-Verschlüsselung
2. Substitutionsverfahren
3. Vigenère-Verfahren
4. Monoalphabetische Verfahren im Vergleich zu polyalphabetischen Verfahren
5. One-Time-Pad
6. DES
7. 3DES
8. AES

9. Symmetrische Verschlüsselung im Vergleich zu asymmetrischer Verschlüsselung
10. Digitale Signatur
11. RSA
12. Steganographie

44.2 Einfache Programmieraufgaben

Aufgabe 1: Satz in Wörter zerlegen

Schreibe ein Programm, das einen Satz in einzelne Worte zerlegt.

Das Frame enthält ein Textfeld zur Eingabe des Satzes und einen Button, mit dem die Berechnung gestartet wird. Wenn der Benutzer auf den Button drückt, wird der Satz in einzelne Worte zerlegt, die zeilenweise auf der Konsole ausgegeben werden. Beispiel:

Eingabe:

In 8 Monaten ist Weihnachten.

Ausgabe:

```
In
8
Monaten
ist
Weihnachten.
```

Aufgabe 2: Zeichen extrahieren

Schreibe ein Programm, das einen beliebigen Zeichen aus einem String herausholt und in einer Messagebox ausgibt.

Das Frame enthält ein Textfeld zur Eingabe einer Zeichenkette und einen Button, mit dem die Berechnung gestartet wird. Die eingegebene Zeichenkette soll folgendermaßen aufgebaut sein:

Zahl\$Text

Beispiele:

| | |
|------------------------------|------------|
| Eingabe: 1\$Abitur | Ausgabe: A |
| Eingabe: 12\$Abiturvorschlag | Ausgabe: h |

Wenn der Benutzer eine falsche Eingabe macht, soll in einer Messagebox eine Fehlermeldung ausgegeben werden.
Beispiele:

| | |
|----------------------------|---|
| Eingabe: 1Abitur | Ausgabe: Das Dollarzeichen fehlt. |
| Eingabe: \$Abiturvorschlag | Ausgabe: Die Zeichenkette muss mit einer Zahl beginnen. |
| Eingabe: 10\$Abitur | Ausgabe: Das angegebene Zeichen existiert nicht. |

Teste gründlich aus, ob dein Programm bei allen denkbaren Fehleingaben korrekt reagiert.

44.3 Q2.2: Client/Server

Aufgabe 1: Dateiinhalte Senden

Es soll ein Server programmiert werden, der auf Anfrage den Inhalt verschiedener Dateien an einen oder mehrere Clients sendet. Die Dateien enthalten eine versteckte Botschaft, die der Server jedoch nur preisgibt, wenn man zuvor ein geheimes Passwort eingegeben hat.

Zum Testen findest du im Kursverzeichnis ein Client-Programm, mit dem du beliebige Kommandos an den Server senden kannst. Außerdem gibt es vier Testdateien, die du in das Verzeichnis deines Servers kopieren solltest.

Der Server soll so programmiert werden, dass er auch mehrere Clients gleichzeitig bedienen kann. Er wartet auf Verbindungen von Clients auf Port 6666. Die Oberfläche des Servers kann einfach leer bleiben. Es wird lediglich die Standard-Funktionalität zum Schließen des Programmfensters benötigt.

Die Namen der vorhandenen Dateien bestehen aus einem Kleinbuchstaben gefolgt von der Endung *.txt. Wenn ein Client eine Datei angezeigt bekommen möchte, sendet er den Kleinbuchstaben, mit dem der Dateiname beginnt, an den Server. Zum Testen sind nur die Dateien a.txt, b.txt und c.txt vorhanden. Wenn die gewünschte Datei existiert, schickt der Server den Inhalt der Datei an den Client und hängt am Ende noch ein Zeilenumbruch-Zeichen an ('\n' bzw. das ASCII-Zeichen mit dem Code 10). Falls die Datei nicht existiert, sendet der Server den Text "Die Datei existiert nicht.\n" an den Client. Falls ein unerlaubtes Kommando vom Client kommt (zum Beispiel ein Großbuchstabe), sendet der Server den Text "Falsche Eingabe\n".

Neben dem Senden eines Kleinbuchstabens, gibt es noch drei weitere Kommandos, die der Client an den Server schicken kann:

- \$Passwort\$ sendet ein Passwort an den Server, mit dem man in einen Geheimmodus wechseln kann
- % schaltet den Geheimmodus wieder aus
- #Passwort# ändert das Passwort. Dies geht nur, wenn der Geheimmodus aktiviert ist.

Wenn der Server das Zeichen '\$' empfängt, weiß er, dass alle weiteren Buchstaben bis zum nächsten '\$' zu einem Passwort gehören. Er liest sie ein und vergleicht sie mit dem Text, der in der Datei passwort.txt steht. Zu Beginn steht in dieser Datei das Passwort "1234". Wenn der vom Client empfangene Text mit dem Passwort identisch ist, schaltet der Server in den Geheimmodus um. An den Client sendet er die Erfolgsmeldung "Geheimmodus aktiviert\n". Falls das Passwort falsch ist, sendet er "Falsches Passwort\n" zurück.

Wenn der Server das Zeichen '%' empfängt, schaltet er den Geheimmodus wieder aus und sendet den Text "Geheimmodus ausgeschaltet\n" an den Client.

Wenn der Geheimmodus aktiviert ist, wird der Inhalt einer angeforderten Datei für den Client entschlüsselt und statt des eigentlichen Dateiinhalts das Geheimwort gesendet, das in dem Text versteckt ist, gefolgt von einem Zeilenumbruch-Zeichen. Das Geheimwort erhält man, wenn man von jedem Wort den zweiten Buchstabe nimmt. Alle anderen Buchstaben und Zeichen (Leerzeichen, Satzzeichen oder Steuerzeichen) werden ignoriert. Wenn das Geheimwort korrekt entschlüsselt wird, wird aus allen drei Testdateien ein sinnvolles Wort generiert. Die Datei b.txt enthält zum Beispiel das Geheimwort abitur.

Wenn der Server das Zeichen '#' empfängt, weiß er, dass alle weiteren Buchstaben bis zum nächsten '#' zu einem neuen Passwort gehören. Er liest das neue Passwort ein. Falls der Geheimmodus ausgeschaltet ist, gibt er die Fehlermeldung "Geheimmodus ist nicht aktiviert\n" zurück. Andernfalls schreibt er das neue Passwort in die Datei passwort.txt und sendet die Erfolgsmeldung "Passwort geändert\n" an den Client.

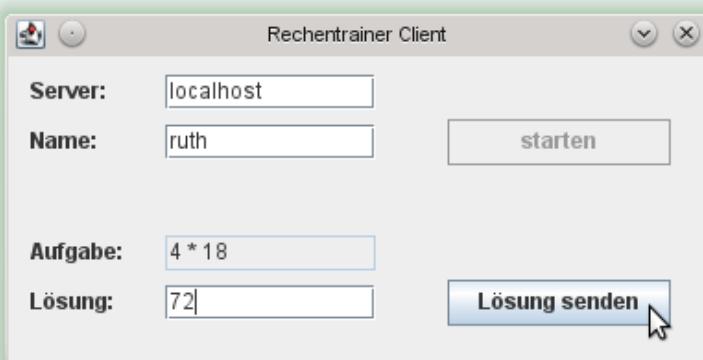
Sorge auf geeignete Weise dafür, dass nur ein Client zur Zeit zum Lesen oder Schreiben auf die Datei passwort.txt zugreifen kann.

Aufgabe 2: Rechentrainer

Es soll ein Rechentrainer als Client-Server-Anwendung programmiert werden.

Client

Erstelle die abgebildete Programmoberfläche. Der untere Button mit der Aufschrift Lösung senden ist zu Beginn deaktiviert. Der obere Button mit der Aufschrift starten ist zu Beginn aktiviert. Das Textfeld mit der Beschriftung Aufgabe ist nicht editierbar.



Wenn der Benutzer auf den starten-Button drückt, prüft der Client, ob der Benutzer seinen Namen eingegeben hat. Falls das Namen-Textfeld leer ist, gibt er eine Fehlermeldung aus. Andernfalls stellt er eine Verbindung zum Server her, der auf Port 33333 wartet. Anschließend sendet er den Namen des Benutzers an den Server gefolgt von einem \$-Zeichen:

Schema: Name\$

Beispiel: Johanna\$

Außerdem deaktiviert er den oberen Button und aktiviert den unteren Button. Falls die Verbindung zum Server nicht hergestellt werden konnte, wird dem Benutzer mit `JOptionPane.showMessageDialog()` eine Fehlermeldung ausgegeben.

Wenn der Benutzer auf den unteren Button klickt, wird der Text, den er in das Lösung-Textfeld geschrieben hat, an den Server gesendet gefolgt von einem \$-Zeichen:

Schema: Lösung\$

Beispiel: 686\$

Nachdem die Verbindung aufgebaut ist, sendet der Server eine Reihe von Rechenaufgaben an den Client, die der Client im Aufgaben-Textfeld anzeigt. Eine Rechenaufgabe kommt in folgendem Format:

Schema: ?Aufgabe\$

Beispiel: ?7 * 98\$

Außerdem können vom Server Erfolgs- und Fehlermeldungen kommen. Diese soll der Client mit Hilfe von `JOptionPane.showMessageDialog()` anzeigen:

Schema: %Meldung\$

Beispiel: %Falsche Antwort. Probier es noch einmal.\$

Der Server stellt dem Benutzer nacheinander fünf Aufgaben (der Client muss dies nicht mitzählen). Wenn der Benutzer alle Aufgaben erfolgreich gelöst hat, sendet der Server eine Erfolgsmeldung und schließt dann die Verbindung. Der Client liest dann entweder eine -1 ein oder erhält beim Lesen eine Exception. In beiden Situationen soll er als Zeichen für das Ende der Verbindung den unteren Button deaktivieren und den oberen Button wieder aktivieren. Der Benutzer kann erneut eine Verbindung zum Server aufbauen, wenn er möchte.

Server

Der Server wartet auf Port 33333 auf eingehende Verbindungen. Er kann beliebig viele Clients parallel bedienen. Für die Bearbeitung eines Clients erzeugt er jeweils einen eigenen Thread.

Nachdem ein Client eine Verbindung zu ihm aufgenommen hat, liest der Server als erstes den Namen des Benutzers ein, der mit einem \$-Zeichen beendet wird.

Anschließend stellt er fünf mal hintereinander eine Rechenaufgabe. Dazu generiert er zwei Zufallszahlen. Die erste Zahl liegt zwischen 2 und 9. Die zweite Zahl liegt zwischen 10 und 99. Er merkt sich die Lösung der Rechenaufgabe in einer Variablen und sendet die Aufgabe in folgendem Format an den Client:

Schema: ?zahl1 * zahl2\$

Beispiel: ?7 * 98\$

Dann wartet er auf die Antwort des Clients, die mit einem \$-Zeichen endet. Er liest die Antwort in eine String-Variablen ein und wandelt sie dann in einen Integer-Wert um. Beachte, dass dabei unter Umständen eine Exception geworfen wird, falls der Benutzer keine korrekte Zahl angegeben hat. Diese Exception musst du abfangen, damit in diesem Fall nicht gleich der ganze Thread beendet wird. Falls das vom Benutzer eingegebene Ergebnis falsch ist, sendet der Server an den Client zurück:

%Falsche Antwort. Probier es noch einmal.\$

Wenn der Benutzer keine korrekte Antwort gegeben hat, wartet der Server erneut auf die Antwort des Benutzers und gibt gegebenenfalls wieder die gleiche Fehlermeldung aus, bis die Antwort richtig ist. Wenn die Antwort korrekt war, wird sofort die nächste Aufgabe gestellt bis der Benutzer fünf Aufgaben bearbeitet hat.

Der Server misst die Zeit, die der Benutzer für die Bearbeitung der fünf Rechenaufgaben benötigt und speichert den aktuellen Bestwert und den Namen des Benutzers ab, der den Bestwert erzielt hat. Dazu benutzt er folgende Methode aus der Klasse System:

```
public static long currentTimeMillis()
```

Anwendungsbeispiel:

```
long anfangszeit = System.currentTimeMillis();
...
...
long endzeit = System.currentTimeMillis();
double zeit = endzeit - anfangszeit;
zeit = zeit / 1000; // Umrechnung Millisekunden in Sekunden
```

Die Methode gibt die aktuelle Uhrzeit in Millisekunden seit 1970 Millisekunden zurück (1970 ist sozusagen die Geburtsstunde der Computer). Wenn man die Methode einmal vor und einmal nach der Bearbeitung der Aufgaben aufruft, kann man errechnen wie viele Millisekunden der Benutzer zur Bearbeitung der Aufgaben gebraucht hat. Schreibe das Ergebnis in eine Fließkomma-Variablen und rechne den Wert in Sekunden um. Wenn der Benutzer die Bestzeit erzielt hat, wird an den Client zurückgegeben:

%Gratuliere Name! Du hast nur Zeit Sekunden gebraucht.\nDas ist die neue Bestzeit!\$

Andernfalls wird zurückgegeben:

%Du hast die Aufgaben in Zeit Sekunden gelöst.\nDie aktuelle Bestzeit von Name beträgt Zeit2 Sekunden.\$

Sorge auf geeignete Weise dafür, dass keine Fehler bei der Speicherung des Bestwertes und des Benutzernamens entstehen können, weil zwei Clients parallel zu dem Schluss kommen, dass sie die Besten sind.

Nachdem der Server die Erfolgsmeldung gesendet hat, beendet er die Verbindung zum Client, indem er den Socket schließt, und beendet auch den Thread, der für den Client erzeugt wurde. Falls die Verbindung zwischendurch abbricht, zum Beispiel weil der Benutzer das Client-Programm geschlossen hat, soll der Thread ebenfalls beendet werden.

Wenn der Client-Thread gestartet wird, liest er die Bestzeit und den Namen des Benutzers wieder aus einer Datei aus. Speichere die Werte in folgendem Format in der Datei ab:

Schema: Zeit\$Name

Beispiel: 49.86\$Johanna

Bevor der Client-Thread beendet wird, werden eine möglicherweise verbesserte Bestzeit und der Name des Benutzers, der die Bestzeit erzielt hat, in der Datei abgespeichert.

Betrachte den Code des Servers und überlege, an welchen Stellen mehrere Threads zeitgleich auf dieselben Variablen zugreifen könnten. Schütze diese Bereiche auf geeignete Weise mit einem Monitor.

Zusatzaufgabe: UML-Zustandsdiagramm des Servers

Überlege dir, in welchen Zuständen der Server (es reicht den Client-Thread zu betrachten) sich befinden kann und bilde die Zustände in einem UML-Zustandsdiagramm ab.

Versuche anschließend den Client-Thread noch einmal zu programmieren. Und zwar so, dass der Ablauf über eine Zustandsvariable gesteuert wird. Dabei solltest du die Zustände benutzen, die du zuvor im UML-Zustandsdiagramm abgebildet hast.

44.4 Q2.1: Datenbanken

Aufgabe 1: Osterhasen GmbH

Die Osterhasen GmbH möchte ihre Verwaltung modernisieren und hat dir einen Auftrag zur Erstellung einer Datenbank gegeben. Die erfahrene, altgediente Osterhäsin Emma erklärt dir die Struktur der GmbH:

Es gibt verschiedene Lieferbezirke, die jeweils einen bestimmten Stadtteil in einem Ort umfassen. Wichtig ist dabei die Speicherung der Postleitzahl und des Stadtteilnamens. In jedem Stadtteil gibt es eine ganze Reihe Familien, die beliefert werden müssen. Einige Familien haben süße Ostereier bestellt, andere gekochte Eier und viele möchten auch beides haben. Wichtig ist, dass wir von jeder Familie den Familiennamen wissen sowie die Anzahl der zu beliefernden Personen und ihre Anschrift (Straße und Hausnummer).

Je nach Größe des Lieferbezirks gibt es einen bis maximal fünf Mitarbeiter, die sich um die Auslieferung unserer Produkte kümmern. Bei besonders kleinen Bezirken kann es auch mal vorkommen, dass ein Osterhase nicht ausgelastet ist und gleich mehrere Lieferbezirke bedient. Im Zuge der Emanzipation hat es sich durchgesetzt, dass nicht nur männliche sondern auch weibliche Osterhasen bei uns angestellt sind. Wir müssen von jedem Hasen seinen (bzw. ihren) Namen und das Dienstalter wissen.

Erstelle zum Entwurf der Datenbank ein geeignetes ER-Diagramm.

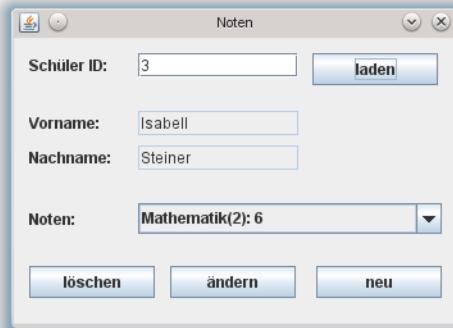
Aufgabe 2: Schuldatenbank I

Im Kursverzeichnis findest du die Datei schule.sql, die SQL-Befehle zur Generierung einer Schuldatenbank erhält. Öffne die Datei als Skript im SQL Explorer und führe ihre Befehle aus.

- Analysiere die Datenbank und zeichne ein ER-Diagramm, das die Struktur der Datenbank darstellt.
- In welcher Normalform befindet sich die Datenbank schule? Begründe deine Antwort, in dem du Schritt für Schritte die Kriterien der einzelnen Normalformen benennst und überprüfst.
- Formuliere geeignete Datenbank-Abfragen, mit denen die folgenden Informationen aus der Datenbank schule gewonnen werden können:
 - Liste alle Lehrer mit ihren vollständigen Attributen auf. Sortiere die Lehrer dabei aufsteigend zunächst nach dem Vornamen und dann nach dem Nachnamen.
 - Liste alle Kurse (mit KursId und Fach) auf, die der Lehrer Donald Dusel unterrichtet.
 - Erstelle eine Liste mit allen Lehrern (Vor- und Nachname) und den Fächern, die sie unterrichten. Achte darauf, dass keine Zeilen mehrfach erscheinen.

4. Erstelle für den Schüler Daniel Weber eine Tabelle mit allen seinen Fächern und der Note, die er jeweils in dem Fach hat. Sortiere die Liste absteigend nach dem Wert der Note.
 5. Liste alle Schüler auf (Vor- und Nachname), die bei der Lehrerin Pia Bachmann Unterricht haben.
 6. Liste alle Schüler auf, die im Fach Mathematik einen Unterkurs haben (Vorname, Nachname und Note).
 7. Ermittle die Anzahl der weiblichen und die Anzahl der männlichen Lehrer.
 8. Liste alle Schüler auf (SchülerId, Vorname und Nachname), für die noch keine Noten eingetragen wurden.
 9. Liste alle Fächer auf, für die mehr als ein Kurs angeboten wird (Fach und Kursanzahl).
 10. Liste alle Lehrer auf (Vor- und Nachname), die mit der Lehrerin Pia Bachmann ein Unterrichtsfach gemeinsam haben. Frau Bachmann selbst darf auch in der Liste erscheinen. Es soll jedoch kein Lehrer doppelt aufgelistet werden.
- d) Formuliere geeignete Datenbankanweisungen für folgende Datenänderungen:
1. Susi Sonntag hat ihren Kollegen Donald Dusel geheiratet und seinen Nachnamen angenommen. Ändere ihren Nachnamen entsprechend um.
 2. Kurs 12 wird von Susi Dusel übernommen. Ändere die LehrerId in der Kurs-Tabelle entsprechend.
 3. Isabell Steiner verlässt die Schule. Lösche ihren Eintrag in der Schüler-Tabelle und alle zugehörigen Einträge in der Noten-Tabelle.
 4. Die Schule hat einen neuen Lehrer bekommen. Er heißt Arne Klee. Füge für Herrn Klee einen neuen Eintrag in die Lehrer-Tabelle ein.

Aufgabe 3: Schuldatenbank II



Es soll eine grafische Oberfläche für die Schul-Datenbank programmiert werden. Gehe in folgenden Schritten vor:

- a) Zunächst gibt es im Anwendungsfenster nur drei Textfelder und einen Button. In das erste Textfeld, das editierbar ist, kann der Benutzer die Id eines Schülers eingeben. Wenn er auf den Button klickt, schreibt das Programm den Vor- und Nachnamen des entsprechenden Schülers in die beiden anderen Textfelder, die nicht editierbar sind. Falls kein Schüler mit der angegebenen Id existiert, wird in einer Messagebox eine Fehlermeldung ausgegeben.
- b) In das Anwendungsfenster wird zusätzlich eine JComboBox-Komponente eingefügt, in der alle Noten des Schülers nach folgendem Schema aufgelistet werden:
Fach(Kursnummer): Note
Beispiel:
Mathematik(2): 13

- c) In das Anwendungsfenster wird zusätzlich ein Button mit der Aufschrift Löschen eingefügt. Wenn der Benutzer den Button anklickt, wird der vom Benutzer selektierte Eintrag in der JComboBox-Komponente aus der Tabelle note gelöscht (nach einer Sicherheitsabfrage) und die JComboBox-Komponente wird aktualisiert.
- d) In das Anwendungsfenster wird zusätzlich ein Button mit der Aufschrift Note ändern eingefügt. Wenn der Benutzer den Button anklickt, wird das selektierte Fach aus der JComboBox-Komponente ausgelesen, und der Benutzer wird nach einer neuen Note für den aktuellen Schüler in dem selektierten Fach gefragt. Die Note wird in die Datenbank eingetragen und die JComboBox-Komponente wird aktualisiert.
- e) In das Anwendungsfenster wird zusätzlich ein Button mit der Aufschrift Neu eingefügt, mit dem ein neuer Eintrag in die Tabelle note eingefügt werden kann. Wenn der Benutzer den Button anklickt, wird er in zwei aufeinander folgenden Message-Boxen zunächst nach der KursId und anschließend nach der Note gefragt. Die angegebenen Daten werden in die Datenbank eingefügt und die JComboBox-Komponente wird aktualisiert.

Aufgabe 4: Kartenverkauf

Es soll eine Datenbank für den Kartenverkauf in einem kleinen Theater angelegt werden. Um in Erfahrung zu bringen, welche Daten abgespeichert werden müssen, wird die Theaterleitung interviewt, die folgendes berichtet:

Das Theater besitzt acht Sitzreihen (durchnummert von A bis H) mit je 20 Sitzplätzen (durchnummert von 1 bis 20). Pro Tag findet maximal eine Aufführung statt. Für jede Aufführung müssen das Datum, die Anfangszeit, der Name des aufgeführten Stücks und der Autor abgespeichert werden. Für jeden Kunden werden sein Vor- und Nachname, seine Adresse (bitte alle nötigen Details auflisten) und seine Telefonnummer erfasst. Aus der Datenbank soll ersichtlich sein, welcher Kunde in welcher Veranstaltung welchen Sitzplatz gebucht hat. Stammkunden können auch ein Abonnement erwerben. Die Datenbank muss abspeichern, ob ein Kunde ein Abonnement besitzt oder nicht. Die weitere Bearbeitung eines Abonnements übernimmt ein Computerprogramm.

Entwirf auf Papier ein Entity-Relationship-Diagramm für eine Datenbank, die alle angegebenen Informationen über die Kunden, die Aufführungen und vorgenommene Buchungen abspeichern kann. Achte darauf, dass es in der Datenbank keine Redundanzen gibt.

Aufgabe 5: Fahrzeug-Datenbank

Im Kurs-Repository findest du die Datei fahrzeuge.sql, die SQL-Befehle zur Generierung einer Fahrzeug-Datenbank enthält. Öffne die Datei als Skript in im SQL Explorer und führe ihre Befehle aus. Alle weiteren Aufgaben beziehen sich auf diese Datenbank.

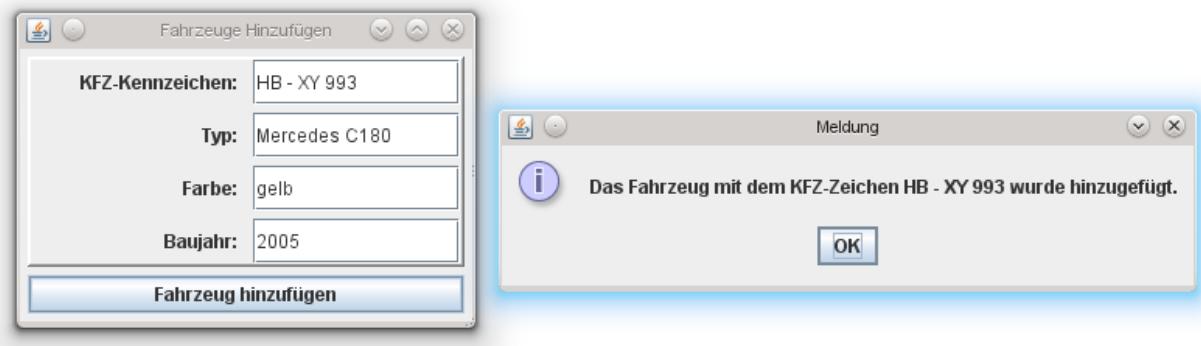
- a) Untersuche die Datenbank und erstelle ein Entity-Relationship-Diagramm, das die Struktur der Datenbank beschreibt.
- b) In welcher Normalform befindet sich die Datenbank? Begründe deine Antwort.
- c) Erstelle geeignete SQL-Abfragen, mit denen die folgenden Informationen gewonnen werden können.
 1. Liste alle Fahrzeuge mit ihren (aktuellen und ehemaligen) Fahrzeughaltern auf. Für die Fahrzeuge sollen das Kennzeichen und der Typ angegeben werden. Für die Fahrzeughalter werden der Vor- und Nachnamen sowie das Anmelde- und Abmeldedatum angezeigt. Sortiere die Liste aufsteigend zuerst nach dem Fahrzeug-Kennzeichen und dann nach dem Anmeldedatum.
 2. Erstelle dieselbe Abfrage wie unter 1., aber liste diesmal nur die Fahrzeughalter auf, die das Fahrzeug aktuell besitzen (d.h. die das Fahrzeug noch nicht abgemeldet haben).
 3. Erstelle eine Liste aller Personen aus der Fahrzeughalter-Tabelle mit Vor- und Nachnamen. In der Liste sollen keine Namen doppelt aufgelistet werden. Sortiere die Liste absteigend nach dem Nachnamen.
 4. Liste alle aktuellen oder ehemaligen Fahrzeuge von Emil Mühlthal mit ihren vollständigen Fahrzeug-Daten auf.

5. Liste alle Fahrzeuge mit Kennzeichen und Typ auf, die im selben Jahr gebaut wurden, wie das Fahrzeug HB – AZ 123. Das Fahrzeug HB – AZ 123 soll nicht in der Liste erscheinen.
6. Erstelle eine Liste aller Fahrzeuge (mit vollständigen Fahrzeug-Daten), die schon mindestens drei Halter hatten.

Aufgabe 6: Fahrzeuge hinzufügen

Schreibe ein Java-Programm, das ein neues Fahrzeug in die Fahrzeug-Datenbank einträgt. Bevor der neue Datensatz in die Datenbank eingefügt wird, wird überprüft, ob bereits ein aktuell angemeldetes Fahrzeug mit dem angegebenen Kennzeichen existiert. Falls dies der Fall ist wird der Datensatz nicht eingefügt und stattdessen mit einem `showMessageDialog()` eine Fehlermeldung ausgegeben. Wenn der Datensatz erfolgreich eingefügt wurde, wird mit einem `showMessageDialog()` eine Erfolgsmeldung ausgegeben.

Die Oberfläche des Programms könnte zum Beispiel so aussehen:



Aufgabe 7: Fahrzeuge abmelden (mit JList)

Erstelle ein Programm mit der abgebildeten Programmoberfläche.



In der List-Komponente sollen alle Fahrzeuge mit ihren aktuellen Fahrzeughaltern nach folgendem Schema aufgelistet werden:

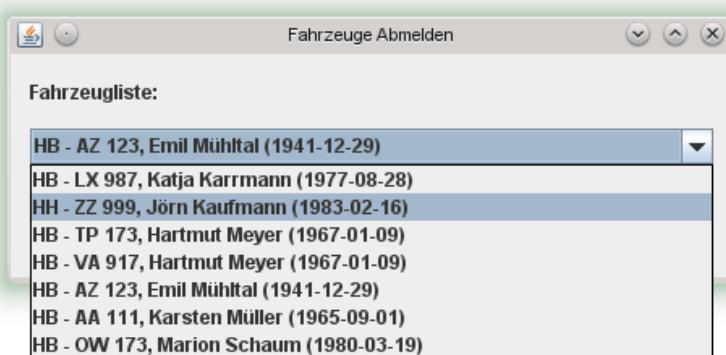
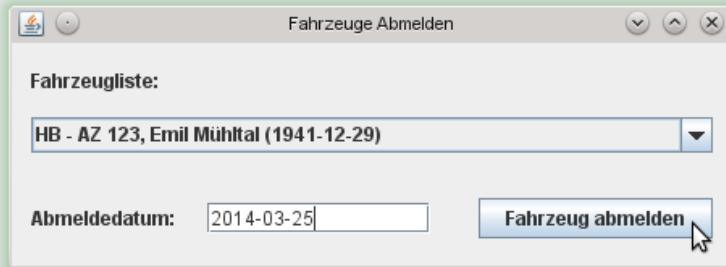
KFZ-Zeichen, Vorname Nachname (Geburtstag)

Als aktuelle Fahrzeughalter werden diejenigen Personen angesehen, bei denen das Fahrzeug noch nicht abgemeldet ist.

Wenn man auf den Button abmelden klickt, soll das selektierte Fahrzeug abgemeldet werden. Dazu trägt das Programm das angegebene Abmeldedatum in den Datensatz des aktuellen Fahrzeughalters ein. Falls kein Fahrzeug selektiert ist oder der Benutzer kein Abmeldedatum angegeben hat, werden entsprechende Fehlermeldungen ausgegeben. Beachte, dass die List-Komponente nach der Abmeldung eines Fahrzeugs aktualisiert werden muss.

Aufgabe 8: Fahrzeuge abmelden (mit JComboBox)

Ändere das Programm aus Aufgabe 6 dahingehend ab, dass statt der JList-Komponente eine JComboBox-Komponente benutzt wird:



An der Programmierung ändert sich dadurch nicht viel: JComboBox-Komponenten verhalten sich aus Sicht des Programmierers ganz ähnlich wie JList-Komponenten!

44.5 Gemischte Aufgaben (Datenbank plus Client/Server)

Aufgabe 1: Buchhandel

Es soll der Prototyp für eine Client-Server-Software zum Verkauf von Büchern erstellt werden. Die Verwaltung der Benutzer wird dabei vernachlässigt. Es wird für jeden Benutzer einfach eine Benutzernummer eingegeben, ohne den Namen und die Adresse des Benutzers genauer zu spezifizieren. Auch eine Zugriffsberechtigung wird nicht abgefragt.

Erzeuge mit Hilfe der Datei buchladen.sql aus dem Kurs-Repository die Datenbank Buchladen auf deinen Computer.

ER-Diagramm

Untersuche die Datenbank und erstelle ein Entity-Relationship-Diagramm, das die Struktur der Datenbank beschreibt.

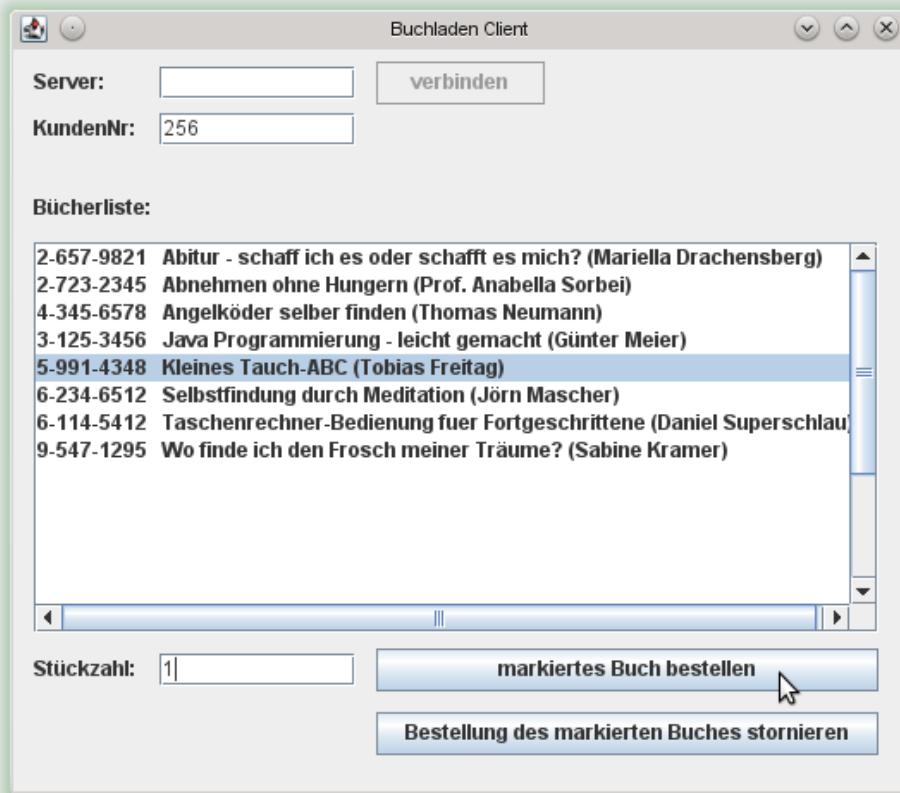
Normalform

In welcher Normalform befindet sich die Datenbank? Falls sich die Datenbank noch nicht in der dritten Normalform befindet, gib an, welche Änderungen man vornehmen müsste, um sie in die dritten Normalform zu bringen.

Programmierung des Clients

Programmiere das Client-Programm für den Buchverkauf.

Im Kurs-Repository findest du einen Server, mit dessen Hilfe du dein Programm austesten kannst. Erstelle zunächst die Oberfläche des Clients:



Zu Beginn soll der verbinden-Button aktiviert sein, und die beiden unteren Buttons sollen deaktiviert sein. Die Bücherliste soll in einer JList-Komponente angezeigt werden (nicht in einer JTextArea).

Wenn der Benutzer auf den verbinden-Button drückt, wird versucht eine Verbindung zum Server aufzubauen, der auf Port 44444 wartet. Falls der Verbindungsauftbau erfolgreich ist, wird der verbinden-Button deaktiviert und die beiden unteren Buttons werden aktiviert.

Der Server schickt dem Client sofort eine Liste der vorhandenen Bücher, die der Client in seiner List-Komponente anzeigen soll. Beachte, dass der Client zunächst den Inhalt der List-Komponente löschen muss, für den Fall dass sich dort bereits alte Ausgaben befinden. Die Nachricht des Servers beginnt mit einem L (für Liste). Danach

wird zeilenweise der fertig aufbereitete Text für die List-Komponente gesendet. Eine Zeile endet mit einem **§**, falls noch weitere Zeilen folgen. Die letzte Zeile endet mit einem **\$**. Schema:

LZeile§Zeile§....§Zeile\$

Wenn der Benutzer auf den Button markiertes Buch bestellen drückt, prüft der Client, ob der Benutzer in den Textfeldern für die Kundennummer und Stückzahl einen Inhalt eingegeben hat. Er überprüft auch, ob sich der Eintrag im Textfeld Stückzahl in eine positive ganze Zahl umwandeln lässt, und ob der Benutzer in der List-Komponente ein Buch ausgewählt hat. Im Fehlerfall gibt der Client jeweils mit `showMessageDialog()` eine geeignete Fehlermeldung aus. Wenn alle Eingaben korrekt sind, sendet der Client nach folgendem Schema eine Bestellung an den Server:

BKundennr\$Isbn\$Stückzahl%

Beispiel:

B777\$5-991-4348§30%

Der Server schickt zur Antwort den Buchstaben **J** (für Ja), wenn die Bestellung erfolgreich durchgeführt werden konnte. Falls nicht mehr ausreichend Bücher vorhanden sind, schickt der Server ein **N** (für Nein) gefolgt von der Anzahl der noch verfügbaren Bücher und einem **\$**-Zeichen als Endmarkierung. Der Client muss dem Benutzer in beiden Fällen mit `showMessageDialog()` einen geeigneten Antworttext ausgeben. Beispiel:

Empfangen: **J** Ausgabe: Ihre Bücher werden in Kürze versandt.

Empfangen: **N30\$** Ausgabe: Ihre Bestellung kann leider nicht durchgeführt werden.\n
Es sind nur noch 30 Bücher vorhanden.

Wenn der Benutzer auf den Button Bestellung des markieren Buches stornieren drückt, prüft der Client, ob in der List-Komponente ein Buch markiert ist, und ob die Benutzernummer korrekt eingegeben ist. Falls dies nicht der Fall ist, gibt er in einem Dialog eine Fehlermeldung aus. Wenn die Eingaben korrekt sind, fordert er auf folgende Weise vom Server die Stornierung der Bestellung an:

SKundennr\$Isbn\$ Beispiel: **S777\$5-991-4348\$**

Der Server schickt als Antwort den Buchstaben **S**, falls die Stornierung erfolgreich durchgeführt wurde. Falls der Benutzer das markierte Buch gar nicht bestellt hatte, sendet der Server als Antwort den Buchstaben **F**. Der Client muss in beiden Fällen mit `showMessageDialog()` eine geeignete Meldung an den Benutzer ausgeben. Beispiel:

Empfangen: **S** Ausgabe: Stornierung erfolgreich durchgeführt.

Empfangen: **F** Ausgabe: Es ist keine Bestellung des markierten Buches vorhanden.

Wenn der Client beim Lesen vom Server eine **-1** empfängt oder wenn eine Exception auftritt, ist die Verbindung zum Server verloren gegangen, weil der Server gestoppt wurde. In diesem Fall soll der Client seinen Lese-Thread beenden und die Zustände der Buttons wieder auf den Anfangszustand zurücksetzen, so dass der Benutzer erneut eine Verbindung zum Server aufbauen kann, nachdem der Server wieder gestartet wurde. Der verbinden-Button wird dazu wieder aktiviert und die beiden unteren Buttons werden deaktiviert.

Programmierung des Servers

Der Server soll auf Port **44444** auf eingehende Verbindungen von Clients warten. Er soll beliebig viele Clients parallel bedienen können.

Sobald sich ein Client angemeldet hat, liest der Server aus der Datenbank die Liste aller Bücher aus und schickt sie an den Client (Schema: **LZeile§Zeile§....§Zeile\$**). Die Buch-Liste soll alphabetisch nach dem Titel sortiert sein. Eine einzelne Zeile, wird nach folgendem Schema aufgebaut:

ISBN Titel (Autor)

Anschließend wartet der Server auf Kommandos, die vom Client gesendet werden. Wenn eine Buchbestellung vom Client kommt (Schema: **BKundennr\$Isbn\$Stückzahl%**), prüft der Server zunächst durch eine Datenbankabfrage, ob noch ausreichend Bücher vorhanden sind. Falls nein, sendet er an den Client die noch vorhandene Anzahl der Bücher zurück (Schema: **NAnzahl\$**). Falls ja, speichert er die neue Bestellung in der Tabelle bestellung

ab. Außerdem reduziert er in der Tabelle `buch` die Anzahl der vorhandenen Bücher um die bestellte Menge. Anschließend sendet er an den Client eine Erfolgsmeldung (J)

Wenn eine Stornierung vom Client kommt (Schema: `SKundennr$Isbn$`), bucht der Server alle Bestellungen des angegebenen Buches für den angegebenen Kunden wieder zurück (Beachte, dass ein Kunde ein Buch auch mehrere Male bestellen kann!). Dabei muss der Server nicht nur die entsprechenden Einträge aus der Tabelle `bestellung` löschen, sondern er muss auch die Anzahl der vorhandenen Bücher in der Tabelle `buch` um die stornierte Menge erhöhen.

Da der Server mehrere Clients parallel bedienen kann, könnte es passieren, dass zwei Clients gleichzeitig eine Bestellung oder Stornierung für dasselbe Buch anfordern. Sorge auf geeignete Weise dafür, dass keine Fehler in der Datenbank durch parallele Zugriffe auf die Spalte vorhanden in der Tabelle `buch` entstehen können.

45 Typische Fehler ... und wie man sie vermeidet

Programmieren ist ein Handwerk. Und da kann man wie überall Fehler machen. Einige Fehler passieren aus Unerfahrenheit oder wegen mangelnder Übung. Dazu kommen noch Flüchtigkeitsfehler.

Außerdem erkennen „Anfänger“ üblicherweise noch nicht den Wert von Namenskonventionen und einheitlicher Formatierung (vor Allem der Einrückung) von Quelltexten, was im Resultat dazu führt, dass die geschriebenen Programme unübersichtlicher und schwerer verständlich werden. Eben weil das eigene Verständnis für die Programme beim Anfänger zwangsläufig noch nicht weit entwickelt ist, fallen ihm die so selbst verursachten zusätzlichen Probleme zunächst gar nicht auf. Der Aufwand, der beispielsweise mit sauberem Einrücken und vernünftiger und Standard konformer Wahl geeigneter Namen für Klassen, Variablen und Methoden einhergeht, erscheint so üblicherweise übertrieben.

Schließlich gibt es noch Mängel in der Struktur von Programmen.

In diesem Kapitel will ich auf die typischen Fehler eingehen. Einige davon lassen sich leicht vermeiden. Andere zumindest leicht aufspüren. Auch dazu - zum Aufspüren von Fehlern (*Debugging*) will ich hier einige Tipps geben.

45.1 Debugging: Fehler finden

Fehler passieren. Anfängern, aber auch Fortgeschrittenen und auch Profis. Deshalb ist es wichtig zu lernen, wie man Fehler möglichst schnell findet.

Debugging mit einfachen Mitteln: Textausgabe auf der Konsole

Ein geeignetes und leicht einzusetzendes Mittel ist die Ausgabe von Text auf der sogenannten *Konsole* (Falls es den *Console-View* bei dir in Eclipse noch nicht gibt, kannst du ihn mit *Window → Show View → Console* hinzufügen).

Wenn du nun in einem Programm die Anweisung

```
System.out.println("Irgend ein Text");
```

schreibst, so wird im *Console-View* genau dieser Text ausgegeben. Das ist nützlich, wenn dein Programm sich nicht so verhält, wie du es erwarte. Dann kannst du über diese `System.out.println()` Anweisungen an geeigneter Stelle in deinem Programm dafür sorgen, dass du das Verhalten des Programms während dessen Ausführung besser nachvollziehen kannst.

Im einfachsten Fall möchtest du einfach überprüfen, ob ein bestimmter Teil deines Programms überhaupt durchlaufe wird. Etwa, ob eine von dir geschriebene Methode überhaupt ausgeführt wird:

```
private int kleinsteZahl(int zahl1, int zahl2) {
    System.out.println("kleinsteZahl(): wurde aufgerufen");
    if (zahl1 < zahl2) {
        return zahl1;
    } else {
        return zahl2;
    }
}
```

Oft ist es aber auch hilfreich bei dieser Gelegenheit auch gleich noch den Inhalt einer oder auch mehrerer Variablen zu überprüfen:

```
private int kleinsteZahl(int zahl1, int zahl2) {
    System.out.println("kleinsteZahl(): zahl1 = " + zahl1 + ",    zahl2 = " + zahl2);
    if (zahl1 < zahl2) {
        return zahl1;
    } else {
```

```

    return zahl2;
}
}
```

45.2 Fehlende geschweifte Klammern

Mehrere Anweisungen können durch geschweifte Klammern zu einem Anweisungsblock zusammengefasst werden. So werden Programme strukturiert: Welche Anweisungen gehören zur Methodendefinition? Welche Anweisungen sollen durch die Kontrollstruktur gesteuert werden?

Beispiel für einen typischen Anfängerfehler:

```

int i = 0;
int j = 10;
if (i > 0)
    i = i + 1;
    j = i;
System.out.println("j hat den Wert " + j);
```

Welche Ausgabe ist zu erwarten? Die Einrückung des Quelltextes in diesem Beispiel lässt vermuten, dass zumindest der Autor dieses Programms folgende Ausgabe erwartet:

```
j hat den Wert 1
```

Tatsächlich wird die Ausgabe aber lauten

```
j hat den Wert 0
```

Grund sind die fehlenden geschweiften Klammern. So hat es der Autor des fehlerhaften Beispiels oben vermutlich gemeint:

```

int i = 0;
int j = 10;
if (i > 0) {
    i = i + 1;
    j = i;
}
System.out.println("j hat den Wert " + j);
```

Erst durch die geschweiften Klammern werden die beiden eingerückten Zeilen zu einem Anweisungsblock, der als ganzer in Abhängigkeit vom `if` ausgeführt wird – oder eben nicht. Ohne die geschweiften Klammern wirkt das `if` nur auf die nächste Anweisung (`i = i + 1;`). Die zweite Anweisung (`j = i;`) wird unabhängig vom `if` auf jeden Fall ausgeführt!

45.3 Das tödliche Semikolon

Gerade in der E-Phase sehe ich diesen Fehler relativ oft: Eine Kontrollstruktur (`if`, `while` oder `for`) wird direkt durch ein Semikolon „kurzgeschlossen“. Beispiel:

```

int i = 0;
while (i < 10); {
    i = i + 1;
}
```

Welchen Wert wird `i` anschließend haben? Würde ich diese Frage in der E-Phase stellen, würden mir viele Schüler antworten „Neun“. Andere vielleicht mit „Zehn“.

Beides ist falsch! Tatsächlich gibt es gar kein *anschließend*, denn die `while`-Schleife wird endlos ausgeführt.

Schuld ist das Semikolon direkt hinter dem Schleifenkopf: Alle Kontrollstrukturen funktionieren so, dass die nächste Anweisung bzw. der folgende Anweisungsblock gesteuert wird.

Das Semikolon direkt hinter einer Kontrollstruktur ist aber eine *leere Anweisung*. Im Beispiel oben kontrolliert der Schleifenkopf auch nur diese eine leere Anweisung, nicht aber – wie sicher intendiert – den folgenden Anweisungsblock.

... und wie man es vermeiden kann

Der Eclipse Editor kann den Quelltext nach (konfigurierbaren) Regeln formatieren. Dazu benutzt man entweder die Tastenkombination <Strg>-<Umschalten>-F oder ruft diese Funktion über das Menü auf: *Source → Format*.

Dann hätte man schon am resultierenden Quellcode sehen können, wo das Problem liegt:

```
int i = 0;
while (i < 10)
    ;
{
    i = i + 1;
}
```

Auch die fehlenden geschweiftem Klammern aus dem vorangegangenen Abschnitt wären so sofort augenfällig geworden:

```
int i = 0;
int j = 10;
if (i > 0)
    i = i + 1;
j = i;
System.out.println("j hat den Wert " + j);
```