PROGRAMMIEREN LERNEN MIT JAVA

ÜBUNGSAUFGABEN

MALTE LUTTERMANN

Inhaltsverzeichnis

Übungsaufgaben

Hier finden Sie eine Auswahl an Übungsaufgaben zum Üben der Programmiersprache Java. Es empfiehlt sich, die Reihenfolge zu beachten, da diese an die Reihenfolge der Kapitel im Buch angepasst ist.

Variablen, Datentypen und Operatoren

1. Definieren Sie eine Variable vom Datentyp `float` und geben Sie den Inhalt dieser Variablen auf der Konsole aus.

2. Teilen Sie dem Benutzer Ihres Programms mit, dass er seinen Namen eingeben soll. Sobald er dies getan hat, soll Ihr Programm den vom Benutzer eingegebenen Namen anzeigen.

3. Entwerfen Sie ein Programm, welches zwei Zahlen (egal, ob ganzzahlig oder nicht) vom Benutzer erwartet, diese dann multipliziert und das Ergebnis ausgibt.

Tipp

Mit der Methode `nextInt()` bzw. `nextFloat()` oder `nextDouble()` können Zahlen eingelesen werden.

4. Entwerfen Sie ein Programm, welches zwei Zahlen (egal, ob ganzzahlig oder nicht) vom Benutzer erwartet, welche dann auf Gleichheit geprüft werden sollen. Geben Sie das Ergebnis auf der Konsole aus.

Typumwandlung, Bedingungen und Schleifen

1. Speichern Sie den Wert 5,5 in einer Variablen ab. Führen Sie nun eine Typumwandlung in den Datentyp `int` durch und geben Sie den konvertierten Wert auf der Konsole aus.

2. Entwerfen Sie ein Programm, welches den Namen vom Benutzer erwartet. Geben Sie den Namen anschließend aus und fragen Sie den Benutzer, ob die Eingabe korrekt ist. Der Benutzer soll dann mit der Taste »Y« (Yes) bestätigen bzw. mit der Taste »N« (No) verneinen können (denken Sie auch daran, dass die Groß- und Kleinschreibung bei Vergleichen relevant ist). Geben Sie danach aus, ob der Name korrekt war.

Tipp

Um Zeichenketten zu vergleichen, nutzen wir `“a“.equals(“b“)` statt (“a“ == “b“), da wir sonst falsche Ergebnisse erhalten können. Warum das so ist, wird im Kapitel zu Zeichenketten erklärt.

3. Erweitern Sie das Programm aus Aufgabe 2. Dazu soll es nun im Falle einer fehlerhaften Eingabe möglich sein, die Eingabe solange zu korrigieren, bis sie korrekt ist.

Tipp

Verwenden Sie eine `while`-Schleife.

Arrays

1. Initialisieren Sie ein zweidimensionales Array so, dass es als Schachbrett fungieren kann. Dazu benötigen Sie ein Array, welches acht Arrays der Länge acht als Einträge besitzt. Sie können sich aussuchen, ob das äußere Array die Reihen oder die Spalten darstellen soll. Der Einfachheit halber sollen die inneren Arrays Einträge vom Datentyp `String` enthalten, die so gewählt sein sollen, dass sie die Bezeichnung der Felder (A1-H8) speichern. Geben Sie zum Schluss alle Einträge der inneren Arrays aus, sodass als Ausgabe eine Aufzählung aller Felder erscheint (A1, A2, . . . , H8).

Methoden, Klassen und Datenkapselung

1. Entwerfen Sie eine Methode, welche die Einheit Stunden in Sekunden umwandelt. Dazu bekommt die Methode eine ganze Zahl übergeben, welche die Anzahl der umzurechnenden Stunden angibt. Die Methode liefert ebenfalls ein ganzzahliges Ergebnis zurück (die Stundenanzahl in Sekunden umgerechnet). Rufen Sie die Methode mit einigen Beispielwerten auf und lassen Sie sich die Ergebnisse in der Konsole anzeigen.

2. Erstellen Sie eine neue Klasse, die eine private Variable (inklusive Getter und Setter für diese Variable; Datentyp beliebig wählbar) enthält. Beim Erstellen eines neuen Objektes dieser Klasse soll die Variable initialisiert werden. Instanziieren Sie dann zwei Objekte der Klasse und weisen Sie der Variablen jedes Mal

einen anderen Wert zu. Lassen Sie den Wert der Variablen beider Objekte in der Konsole ausgeben.

3. Ändern Sie die eben erstellte Variable nun in eine statische Variable um. Nehmen Sie sonst keine weiteren Änderungen vor und achten Sie darauf, inwiefern sich die Ausgabe nun ändert.

4. Implementieren Sie eine statische Methode, die drei Parameter erwartet: Einen Geldbetrag in Euro, einen Zinssatz in Prozent und eine Dauer in Jahren. Die Methode soll dann berechnen, wie sich der Geldbetrag zu einem angegebenen Zinssatz nach einem bestimmten Zeitraum verändert hat. Rufen Sie die Methode auf und geben Sie das Ergebnis auf der Konsole aus.

Tipp

Es gilt die folgende Formel:

$$\text{Endkapital} = \text{Startkapital} \cdot \left(1 + \frac{\text{Zinssatz in Prozent}}{100}\right)^{\text{Dauer}}$$

Einen Exponenten kann man in Java folgendermaßen berechnen:

```
Math.pow(Basis, Exponent)
```

Es ist sicherlich hilfreich, den Rückgabetyp `double` zu verwenden, da Ergebnisse hierbei meist nicht ganzzahlig sind.

Vererbung und Polymorphismus

1. Schreiben Sie ein Programm, welches eine Schnittstelle enthält, die von einer weiteren Klasse implementiert wird. Diese Schnittstelle soll zwei verschiedene Methoden enthalten, deren Signatur und Inhalt beliebig gewählt sein dürfen.

2. Ändern Sie die Schnittstelle nun in eine abstrakte Klasse um und passen Sie den restlichen Code an. Dabei sollen die beiden Methoden weiterhin ohne Implementierung bleiben und erst in der abgeleiteten Klasse definiert werden.

3. Entwerfen Sie nun eine Klasse, welche eine Methode beinhaltet, die mehrfach überladen ist (Signatur und Methodeninhalt beliebig wählbar). Führen Sie dann alle Überladungen der Methode aus und geben Sie gegebenenfalls die Ergebnisse auf der Konsole aus.

4. Erstellen Sie zwei Klassen. Die eine Klasse erbt von der anderen Klasse. Implementieren Sie nun eine polymorphe Methode in beiden Klassen und führen Sie

die Methoden aus (die Methoden haben die gleiche Signatur, aber einen anderen Inhalt; wobei Signatur und Inhalt beliebig gewählt werden können).

Ausnahmebehandlungen

1. Schreiben Sie eine Methode, die bei einer fehlerhaften Eingabe eine Ausnahme wirft. Führen Sie die Methode mit einer korrekten Eingabe und mit einer fehlerhaften Eingabe aus. Sorgen Sie bei der fehlerhaften Eingabe dafür, dass das Programm weiterläuft (sich nicht beendet) und dafür, dass die Informationen der abgefangenen Ausnahme ausgegeben werden.

Dateien

1. Entwerfen Sie ein Programm, welches eine Methode enthält, die beliebige Texte (`String`s) in eine Textdatei schreibt. Die Methode bekommt dabei als Parameter den Dateipfad und den zu schreibenden Text übergeben. Sollte die Datei nicht existieren, so soll die Methode eine neue Datei erstellen und dann darin schreiben. Existiert die Datei bereits, so soll die Methode bereits vorhandene Inhalte nicht überschreiben.

2. Fügen Sie nun eine Methode hinzu, die den Inhalt einer Textdatei liest. Führen Sie dann beide Methoden aus und geben Sie den Inhalt der Datei auf der Konsole aus.

Grafische Benutzeroberflächen

1. Erstellen Sie ein Fenster, das einen farbigen Punkt in der Mitte des Fensters enthält.

Tipp

Als Punkt verwenden wir ein sehr kleines Rechteck.

2. Fügen Sie nun Ereignisse hinzu, sodass sich der Punkt bewegt, wenn eine Pfeiltaste gedrückt wird (die Pfeiltaste gibt die Bewegungsrichtung an). Der Punkt muss nicht mehr in der Mitte des Fensters starten (da dies sehr komplex werden kann).

Tipp

Sie benötigen einen `KeyListener`. Wenn Sie diesen zu einem `JPanel`-Objekt statt zu einem `JFrame`-Objekt hinzufügen, dann sollten die beiden Methoden `setFocusable(true)` und `requestFocusInWindow()` in dieser Reihenfolge auf das `JPanel`-Objekt aufgerufen werden, damit der Listener korrekt arbeitet.

3. Ändern Sie das Programm nun dahingehend, dass eine durchgängige Bewegung des Rechtecks möglich ist, wenn die Pfeiltasten gedrückt gehalten werden (falls Sie dies nicht bereits getan haben). Es soll ebenfalls möglich sein, zwei Pfeiltasten gleichzeitig zu drücken, sodass die resultierende Richtung beide Einflüsse berücksichtigt, d.h. wenn »oben« und »rechts« gedrückt gehalten werden, soll das Rechteck nach Nordosten bewegt werden.

4. Fügen Sie nun ein Mausereignis hinzu, das dafür sorgt, dass das Rechteck seine Farbe jedes Mal ändert, wenn eine Maustaste betätigt wird.

Auflistungsklassen und generische Datentypen

1. Implementieren Sie eine `ArrayList` und fügen Sie fünf verschiedene Zahlenwerte ein.

2. Addieren Sie nun alle Inhalte der `ArrayList` und geben Sie dann das Ergebnis auf der Konsole aus.

3. Implementieren Sie eine (beliebige) generische Methode.

Lösungsvorschläge

Hier finden Sie Lösungsvorschläge zu den gestellten Aufgaben. Bitte bedenken Sie, dass es immer mehrere Lösungsmöglichkeiten gibt. Sollten Sie also einen anderen Lösungsweg als die hier vorgeschlagene Lösung gewählt haben, ist das überhaupt kein Problem.

Variablen, Datentypen und Konvertierung

1. Wir dürfen das Suffix »f« nicht vergessen, wenn wir den Datentyp `float` verwenden möchten.

```
public class Exercise {
    public static void main(String[] args) {
        float PI = 3.14159f;
        System.out.println(PI);
    }
}
```

2. Diese Aufgabe entspricht genau dem Beispiel zu Benutzereingaben aus dem Buch.

```
import java.util.Scanner;

public class Exercise {
    public static void main(String args []) {
        System.out.println("Wie lautet Ihr Name?");
        Scanner scanner = new Scanner(System.in);
        String name = scanner.next();
        scanner.close();
        System.out.println("Ihr Name lautet also " + name);
    }
}
```

3. Wir lesen zunächst die Werte vom Benutzer ein und multiplizieren diese dann mit dem dafür vorgesehenen Operator. Dann geben wir das Ergebnis aus.

```java
import java.util.Scanner;

public class Exercise {
    public static void main(String args []) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Geben Sie die erste Zahl ein.");
        int zahl1 = scanner.nextInt();

        System.out.println("Geben Sie die zweite Zahl ein.");
        int zahl2 = scanner.nextInt();

        scanner.close();
        System.out.printf("%d*%d = %d", zahl1, zahl2, zahl1 * zahl2);
    }
}
```

4. Wir lesen zunächst die Werte vom Benutzer ein und prüfen diese dann mit dem dafür vorgesehenen Operator auf Gleichheit. Dann geben wir das Ergebnis aus.

```java
import java.util.Scanner;

public class Exercise {
    public static void main(String args []) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Geben Sie die erste Zahl ein.");
        int zahl1 = scanner.nextInt();

        System.out.println("Geben Sie die zweite Zahl ein.");
        int zahl2 = scanner.nextInt();

        scanner.close();
        if (zahl1 == zahl2) {
            System.out.printf("%d=%d", zahl1, zahl2);
        }
        else {
            System.out.printf("%d!=%d", zahl1, zahl2);
        }
    }
}
```

Operatoren, Bedingungen und Schleifen

1. Um den Wert umzuwandeln, schreiben wir einfach `(int)` vor den Wert.

```java
public class Exercise {
    public static void main(String args []) {
        float zahl = 5.5f;
        System.out.println((int) zahl);
    }
}
```

2. Wir lesen den eingegebenen Wert ein, nachdem wir den Benutzer dazu aufgefordert haben, seinen Namen einzugeben.

   Dann fragen wir den Benutzer, ob der eingegebene Wert korrekt ist, oder ob er sich möglicherweise vertippt hat. Dazu kann mit der Taste »Y« bestätigt und mit der Taste »N« verneint werden.

   Nun können wir zum Beispiel eine `if`-Abfrage nutzen, um herauszufinden, ob der Benutzer seine Eingabe bestätigt hat, verneint hat, oder eine ungültige Eingabe getätigt hat.

   Dabei ist zu beachten, dass die Groß- und Kleinschreibung der Eingabe eine Rolle spielt.

```java
import java.util.Scanner;

public class Exercise {
    public static void main(String args []) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Geben Sie Ihren Namen ein.");
        String name = scanner.next();

        System.out.printf("Ihr Name lautet also %s.%n", name);
        System.out.println("Ist das korrekt? (Y/N)");

        String antwort = scanner.next();
        scanner.close();

        if (antwort.equals("y") || antwort.equals("Y")) {
            System.out.println("Sie haben Ihre Eingabe bestaetigt.");
        }
        else if (antwort.equals("n") || antwort.equals("N")) {
            System.out.println("Ihre Eingabe war nicht korrekt.");
        }
        else {
            System.out.println("Ungueltige Eingabe.");
        }
    }
}
```

3. Dieses Mal findet die Eingabe in einer `while`-Schleife statt. Dabei wird die Eingabe solange wiederholt, bis der Benutzer sie bestätigt. Zusätzlich befindet sich

in dieser while-Schleife eine weitere while-Schleife, welche überprüft, ob ein ungültiger Wert eingegeben worden ist.

```
import java.util.Scanner;

public class Exercise {
    public static void main(String args []) {
        Scanner scanner = new Scanner(System.in);
        String antwort = "";
        while (!antwort.equals("y") && !antwort.equals("Y")) {
            System.out.println("Geben Sie Ihren Namen ein.");
            String name = scanner.next();
            System.out.printf("Ihr Name lautet also %s.%n", name);
            System.out.println("Ist das korrekt? (Y/N)");

            antwort = scanner.next();
            while (!antwort.equals("y") && !antwort.equals("Y") &&
                   !antwort.equals("n") && !antwort.equals("N")) {
                System.out.println("Bestaetigen Sie mit 'Y' oder
                   kehren Sie mit 'N' zur Eingabe zurueck");
                antwort = scanner.next();
            }
        }
        System.out.println("Eingabe wurde bestaetigt.");
        scanner.close();
    }
}
```

Genau genommen tut die zweite while-Schleife hier nicht Not, man kann auch mit einer Schleife auskommen. Die zweite Schleife sorgt lediglich dafür, dass die Ausgabe etwas schicker aussieht.

Mit einer einzigen while-Schleife könnte man die Aufgabe folgendermaßen lösen:

```
import java.util.Scanner;

public class Exercise {
    public static void main(String args []) {
        Scanner scanner = new Scanner(System.in);
        String antwort = "";
        while (!antwort.equals("y") && !antwort.equals("Y")) {
            System.out.println("Geben Sie Ihren Namen ein.");
            String name = scanner.next();
            System.out.printf("Ihr Name lautet also %s.%n", name);
            System.out.println("Ist das korrekt? (Y/N)");

            antwort = scanner.next();
        }
        System.out.println("Eingabe wurde bestaetigt.");
        scanner.close();
    }
}
```

Arrays

1. Hier sind die Einträge optisch so angeordnet, wie sie es auch bei einem echten Schachbrett sind. Es sind jedoch auch durchaus andere Anordnungen möglich. Dann müssten lediglich die Schleifen zur Ausgabe angepasst werden. Wichtig ist nur, dass man weiß, wo sich welches Element befindet. Die Ausgabe hier mag auf den ersten Blick vielleicht etwas schwer nachvollziehbar sein, aber das ist normal. Wenn man sich das Programm nochmal genau ansieht, versteht man auch, was genau passiert (verschachtelte Schleifen können anfangs immer etwas verwirrend wirken).

   Diese Beispiellösung zeigt zudem, dass man `for`-Schleifen auch rückwärts laufen lassen kann. Natürlich ist das zur Lösung der Aufgabe nicht erforderlich, aber es ist möglich.

```
public class Exercise {
    public static void main(String args []) {
        String[][] schachbrett = {
            new String[]{ "A8","B8","C8","D8","E8","F8","G8","H8" },
            new String[]{ "A7","B7","C7","D7","E7","F7","G7","H7" },
            new String[]{ "A6","B6","C6","D6","E6","F6","G6","H6" },
            new String[]{ "A5","B5","C5","D5","E5","F5","G5","H5" },
            new String[]{ "A4","B4","C4","D4","E4","F4","G4","H4" },
            new String[]{ "A3","B3","C3","D3","E3","F3","G3","H3" },
            new String[]{ "A2","B2","C2","D2","E2","F2","G2","H2" },
            new String[]{ "A1","B1","C1","D1","E1","F1","G1","H1" }
        };

        for (int i = 0; i < schachbrett.length; i++) {
            for (int j = schachbrett[i].length - 1; j >= 0; j--) {
                System.out.println(schachbrett[j][i]);
            }
        }
    }
}
```

Methoden, Klassen und Datenkapselung

1. Zu beachten ist hier vor allem, dass die Methode nur über ein Objekt der Klasse, welche die Methode beinhaltet, aufgerufen werden kann. In diesem Fall ist das die Klasse `Exercise`.

```
public class Exercise {
    public static void main(String args []) {
        Exercise obj = new Exercise();

        System.out.println(obj.ToSeconds(1));
        System.out.println(obj.ToSeconds(12));
        System.out.println(obj.ToSeconds(24));
    }
    int ToSeconds(int hours) {
        int minutes = hours * 60;
        int seconds = minutes * 60;
        return seconds;
    }
}
```

Wir können die Methode auch in einer Zeile zusammenfassen:

```
public class Exercise {
    public static void main(String args []) {
        Exercise obj = new Exercise();

        System.out.println(obj.ToSeconds(1));
        System.out.println(obj.ToSeconds(12));
        System.out.println(obj.ToSeconds(24));
    }
    int ToSeconds(int hours) {
        return hours * 3600;
    }
}
```

2. Wir haben eine neue Klasse erstellt, welche eine private Integer-Variable enthält. Der Konstruktor setzt den Wert dieser Variablen. Wollen außerhalb dieser Klasse auf die Variable zugreifen, so benutzen wir die Getter- bzw. Setter-Methode. Dieses Beispiel sollte vor allem verdeutlichen, dass Instanzmethoden zu genau einem Objekt gehören.

```
public class Exercise {
    public static void main(String args []) {
        NeueKlasse obj1 = new NeueKlasse(5);
        NeueKlasse obj2 = new NeueKlasse(10);

        System.out.println(obj1.getPrivateVar());
        System.out.println(obj2.getPrivateVar());
    }
}
```

```
public class NeueKlasse {
    private int privateVar;

    public NeueKlasse(int wert) {
        privateVar = wert;
    }

    public int getPrivateVar() {
        return privateVar;
    }
    public void setPrivateVar(int wert) {
        privateVar = wert;
    }
}
```

3. Was Ihnen hier auffallen soll, ist, dass die Variable nun den gleichen Wert bei beiden Objekten hat. Das liegt daran, dass statische Variablen für alle Objekte gleich sind und nicht jedes Objekt seine eigene zugehörige Variable besitzt.

```
public class Exercise {
    public static void main(String args []) {
        NeueKlasse obj1 = new NeueKlasse(5);
        NeueKlasse obj2 = new NeueKlasse(10);

        System.out.println(obj1.getPrivateVar());
        System.out.println(obj2.getPrivateVar());
    }
}
```

```
public class NeueKlasse {
    private static int privateVar;

    public NeueKlasse(int wert) {
        privateVar = wert;
    }

    public int getPrivateVar() {
        return privateVar;
    }
    public void setPrivateVar(int wert) {
        privateVar = wert;
    }
}
```

4. Durch den gegebenen Tipp dürfte das Abtippen der Formel kein Problem gewesen sein. Bei dieser Aufgabe war zu beachten, dass die Methode über die Klasse `Exercise` aufgerufen werden muss, da sie statisch ist.

```java
public class Exercise {
    public static void main(String args []) {
        System.out.println(Exercise.zinsenBerechnen(10000, 3, 5));
    }
    static double zinsenBerechnen(double betrag, double prozent, int
       jahre) {
        return betrag * (Math.pow((1 + (prozent / 100)), jahre));
    }
}
```

Vererbung und Polymorphismus

1. Hierbei ist zu bedenken, dass alle Methoden einer Schnittstelle in allen Klassen, welche die Schnittstelle implementieren, definiert werden müssen.

```java
public interface GeometrischesObjekt {
    double berechneFlaecheninhalt();
    double berechneUmfang();
}
```

```java
public class Rechteck implements GeometrischesObjekt {
    private double laenge;
    private double breite;

    public Rechteck(double laenge, double breite) {
        this.laenge = laenge;
        this.breite = breite;
    }
    @Override
    public double berechneFlaecheninhalt() {
        return laenge * breite;
    }
    @Override
    public double berechneUmfang() {
        return 2 * (laenge + breite);
    }
}
```

```java
public class Exercise {
    public static void main(String[] args) {
        Rechteck r = new Rechteck(10, 5);
        System.out.println("A = " + r.berechneFlaecheninhalt());
        System.out.println("U = " + r.berechneUmfang());
    }
}
```

2. Nun müssen die Methoden ohne Implementierung auch als `abstract` gekennzeichnet werden. Darüber hinaus müssen wir nun das Schlüsselwort `extends` statt `implements` verwenden.

```java
public abstract class GeometrischesObjekt {
    abstract double berechneFlaecheninhalt();
    abstract double berechneUmfang();
}
```

```java
public class Rechteck extends GeometrischesObjekt {
    private double laenge;
    private double breite;
    public Rechteck(double laenge, double breite) {
        this.laenge = laenge;
        this.breite = breite;
    }
    @Override
    public double berechneFlaecheninhalt() {
        return laenge * breite;
    }
    @Override
    public double berechneUmfang() {
        return 2 * (laenge + breite);
    }
}
```

```java
public class Exercise {
    public static void main(String[] args) {
        Rechteck r = new Rechteck(10, 5);
        System.out.println("A = " + r.berechneFlaecheninhalt());
        System.out.println("U = " + r.berechneUmfang());
    }
}
```

3. Beim Überladen von Methoden ist es wichtig, dass sich die Methoden in der Parameterliste unterscheiden. Die restliche Signatur der Methoden stimmt überein.

```java
public class Exercise {
    public static void main(String[] args) {
        Exercise obj = new Exercise();
        System.out.println(obj.addieren(10, 10));
        System.out.println(obj.addieren(10, 10, 10));
        System.out.println(obj.addieren(10, 10, 10, 10));
    }
    int addieren(int a, int b) {
        return a + b;
    }
    int addieren(int a, int b, int c) {
        return a + b + c;
    }
    int addieren(int a, int b, int c, int d) {
        return a + b + c + d;
    }
}
```

4. Wir haben hier eine polymorphe Methode `bellen()` vorliegen. Dabei haben wir die von der Klasse `Hund` vererbte Methode `bellen()`) überschrieben und unsere eigene Definition für die Klasse `Dackel` verwendet.

   Wäre die Methode `bellen()` nicht in der Klasse `Dackel` definiert, so würde `dackel.bellen();` einfach die Methode aus der Klasse `Hund` aufrufen, da diese an die Klasse `Dackel` vererbt worden ist.

```java
public class Hund {
    void bellen() {
        System.out.println("bellen() in Hund");
    }
}
```

```java
public class Dackel extends Hund {
    @Override
    void bellen() {
        System.out.println("bellen() in Dackel");
    }
}
```

```java
public class Exercise {
    public static void main(String[] args) {
        Hund hund = new Hund();
        hund.bellen();

        Dackel dackel = new Dackel();
        dackel.bellen();

        Hund nochEinDackel = new Dackel();
        nochEinDackel.bellen();
    }
}
```

Ausnahmebehandlungen

1. Die Ausnahme wird durch das Schlüsselwort `throw` geworfen und durch einen `try{...}` `catch{...}`-Block abgefangen, sodass sich das Programm nicht beendet, sobald die Ausnahme auftritt.

```
import java.lang.ArithmeticException;

public class Exercise {
    public static void main(String[] args) {
        try {
            // Gueltige Eingabe
            System.out.println(Exercise.dividieren(10, 2));

            // Fehlerhafte Eingabe
            System.out.println(Exercise.dividieren(10, 0));
        }
        catch (ArithmeticException e) {
            e.printStackTrace();
        }
        System.out.println("Programm laeuft weiter.");
    }
    static double dividieren(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException();
        }
        else {
            return a / b;
        }
    }
}
```

Bei diesem Beispiel würde man die `if{...} else{...}`-Bedingung nicht unbedingt benötigen, da die Ausnahme auch geworfen wird, wenn man einfach durch Null teilt. Das bedeutet, dass das obige Beispiel äquivalent zu diesem Code ist:

```
import java.lang.ArithmeticException;

public class Exercise {
    public static void main(String[] args) {
        try {
            // Gueltige Eingabe
            System.out.println(Exercise.dividieren(10, 2));

            // Fehlerhafte Eingabe
            System.out.println(Exercise.dividieren(10, 0));
        }
        catch (ArithmeticException e) {
            e.printStackTrace();
        }
        System.out.println("Programm laeuft weiter.");
    }
    static double dividieren(int a, int b) {
        return a / b;
    }
}
```

Dateien

1. Hier wird der Text einfach mit einem Objekt der Klasse `FileWriter` in die Textdatei geschrieben. Dieses erstellt automatisch eine neue Textdatei, falls unter dem angegebenen Pfad keine Textdatei existiert. Beim Erzeugen des Objektes gibt der zweite Parameter an, dass der Text angehängt werden soll und bereits vorhandener Inhalt nicht überschrieben werden soll (dazu könnte man alternativ natürlich auch einfach die Methode `append()` anstelle von `write()` nutzen).

   Durch das Schlüsselwort `finally` sorgen wir dafür, dass die geöffnete Datei auch wieder geschlossen wird, egal ob eine Ausnahme auftritt oder nicht.

```
import java.io.*;

public class Exercise {
    public static void main(String[] args) {
        String pfad = "test.txt";
        String text = "Dieser Text wird in die Datei geschrieben.";

        Exercise.schreibeInTextdatei(pfad, text);
    }
    static void schreibeInTextdatei(String pfad, String text) {
        Writer writer = null;
        try {
            writer = new FileWriter(pfad, true);
            writer.write(text);
            writer.write(System.getProperty("line.separator"));
        }
        catch (IOException e) {
            System.err.println("Fehler beim Schreiben!");
            e.printStackTrace();
        }
        finally {
            if (writer != null) {
                try {
                    writer.close();
                }
                catch (IOException e) {
                    System.err.println("Fehler beim Schliessen!");
                    e.printStackTrace();
                }
            }
        }
    }
}
```

2. Nun wurde eine Methode hinzugefügt, die den gelesenen Inhalt als `String` zurückgibt. Man kann den Inhalt der Textdatei auch direkt innerhalb der Methode in die Konsole schreiben, wenn man den Rückgabetyp `void` verwendet.

```java
import java.io.*;

public class Exercise {
    public static void main(String[] args) {
        String pfad = "test.txt";
        String text = "Dieser Text wird in die Datei geschrieben.";
        Exercise.schreibeInTextdatei(pfad, text);
        System.out.println(Exercise.leseAusTextdatei(pfad));
    }
    static void schreibeInTextdatei(String pfad, String text) {
        Writer writer = null;
        try {
            writer = new FileWriter(pfad, true);
            writer.write(text);
            writer.write(System.getProperty("line.separator"));
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            if (writer != null) {
                try {
                    writer.close();
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
    static String leseAusTextdatei(String pfad) {
        Reader reader = null;
        String text = "";
        try {
            reader = new FileReader(pfad);
            for (int c; (c = reader.read()) != -1;) {
                text += (char) c;
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            if (reader != null) {
                try {
                    reader.close();
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return text;
    }
}
```

Grafische Benutzeroberflächen

1. Für einen kleinen Punkt setzen wir sowohl die Pixelbreite als auch die Pixelhöhe des Rechtecks auf 1 (um den Punkt besser sehen zu können, werden wir den Wert in der nächsten Aufgabe etwas erhöhen). Um den Mittelpunkt des Fensters zu erhalten, teilen wir einfach Breite und Höhe durch zwei. Dabei ist zu bedenken, dass lediglich die linke obere Ecke des gezeichneten Objektes genau in der Mitte des Fensters liegt (bei einem Rechteck mit einem Pixel Breite und Höhe spielt das keine Rolle).

   Hat man hingegen ein größeres Objekt, so muss man den Punkt anpassen, wenn man beabsichtigt, das Objekt genau in der Mitte des Fensters zu zeichnen.

Achtung!

Um die Mitte des Fensters zu berechnen, gehen wir von den Maßen des `JPanel` aus. Würden wir von den Maßen des `JFrame` ausgehen, so würde es nicht ganz passen, da bei der Höhe die Höhe der Titelleiste mit inbegriffen ist.

```
import javax.swing.JPanel;
import java.awt.Graphics;
import java.awt.Color;

public class Grafikcontainer extends JPanel {
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.RED);
        g.fillRect(this.getWidth()/2, this.getHeight()/2, 1, 1);
    }
}
```

```
import javax.swing.*;

public class Exercise {
    public static final int BREITE = 200;
    public static final int HOEHE = 200;

    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(BREITE, HOEHE);
        f.setLocationRelativeTo(null);
        f.add(new Grafikcontainer());
        f.setVisible(true);
        f.repaint();
    }
}
```

2. Damit sich der Punkt auf Tastendruck bewegt, passen wir einfach seine Position an, sobald eine Pfeiltaste gedrückt wird (der »Punkt« ist genau genommen nun kein Punkt mehr, sondern ein Rechteck, da die Maße zur besseren Sichtbarkeit erhöht wurden).

   Es ist gar nicht so leicht, das Rechteck in der Mitte des Fensters zu positionieren, da die Methoden `getWidth()` bzw. `getHeight()` im Konstruktor des `JPanel`-Objektes 0 zurückliefern, wenn wir nicht manuell eine Größe für das `JPanel` gesetzt haben. In der letzten Aufgabe haben die Methoden korrekt funktioniert, weil sie erst aufgerufen wurden, **nachdem** das `JPanel`-Objekt zum `JFrame` hinzugefügt worden ist.

   Eine mögliche Lösung für dieses Problem: Wir rufen einmal die Methode `pack()` auf unser `JFrame`-Objekt auf, damit wir danach die Ränder korrekt bestimmen können (ohne diesen Aufruf würde die Methode `getInsets()` immer 0 zurückliefern). Da wir nun die innere Größe des Fensters ermitteln können, wissen wir auch, wie groß das `JPanel`-Objekt sein muss. Somit können wir die Koordinaten für die Startposition setzen.

   Da dieses Vorgehen sehr kompliziert und möglicherweise schwer nachvollziehbar ist, war es auch nicht mehr Teil der Aufgabe. Es sollte nur für Interessierte gezeigt werden, wie man das Rechteck mittig platzieren könnte, wenn man es denn unbedingt möchte.

```
import javax.swing.*;

public class Exercise {
    public static final int BREITE = 200;
    public static final int HOEHE = 200;

    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLocationRelativeTo(null);
        f.pack();

        Grafikcontainer container = new Grafikcontainer();
        int startX = (BREITE - f.getInsets().left - f.getInsets().
            right) / 2;
        int startY = (HOEHE - f.getInsets().top - f.getInsets().
            bottom) / 2;
        container.setStartPosition(startX, startY);

        f.add(container);
        f.setSize(BREITE, HOEHE);
        f.setVisible(true);
        f.repaint();
    }
}
```

```
import javax.swing.JPanel;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.event.*;

public class Grafikcontainer extends JPanel {
    private int xPos;
    private int yPos;

    public Grafikcontainer() {
        this.addKeyListener(new KeyListener() {
            @Override
            public void keyReleased(KeyEvent e) {}
            @Override
            public void keyTyped(KeyEvent e) {}
            @Override
            public void keyPressed(KeyEvent e) {
                if (e.getKeyCode() == KeyEvent.VK_UP) {
                    yPos--;
                }
                else if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
                    xPos++;
                }
                else if (e.getKeyCode() == KeyEvent.VK_LEFT) {
                    xPos--;
                }
                else if (e.getKeyCode() == KeyEvent.VK_DOWN) {
                    yPos++;
                }
                repaint();
            }
        });

        this.setFocusable(true);
        this.requestFocusInWindow();
    }

    public void setStartPosition(int xPos, int yPos) {
        this.xPos = xPos;
        this.yPos = yPos;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.RED);
        g.fillRect(xPos, yPos, 4, 4);
    }
}
```

3. Auch diese Aufgabe ist nicht ganz so einfach. Beim Drücken einer Taste wird ein Wahrheitswert auf wahr gesetzt, beim Loslassen einer Taste wird er auf falsch

gesetzt. Immer, wenn eine Taste gedrückt oder losgelassen wird, wird eine Methode zum Aktualisieren der Position aufgerufen. In dieser Methode wird überprüft, welche Wahrheitswerte wahr sind und dementsprechend die Position des Rechtecks angepasst und das Fenster neu gezeichnet. Dies erlaubt auch flüssige Bewegungen, wenn mehrere Tasten gleichzeitig gedrückt werden.

```
import javax.swing.*;

public class Exercise {
    public static final int BREITE = 200;
    public static final int HOEHE = 200;

    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLocationRelativeTo(null);
        f.pack();
        Grafikcontainer container = new Grafikcontainer();
        int startX = (BREITE - f.getInsets().left - f.getInsets().
            right) / 2;
        int startY = (HOEHE - f.getInsets().top - f.getInsets().
            bottom) / 2;
        container.setStartPosition(startX, startY);
        f.add(container);
        f.setSize(BREITE, HOEHE);
        f.setVisible(true);
        f.repaint();
    }
}
```

```
import javax.swing.JPanel;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.event.*;

public class Grafikcontainer extends JPanel {
    private int xPos;
    private int yPos;
    private boolean[] r = new boolean[4];

    public Grafikcontainer() {
        this.addKeyListener(new KeyListener() {
            @Override
            public void keyReleased(KeyEvent e) {
                if (e.getKeyCode() == KeyEvent.VK_UP)   r[0] = false;
                if (e.getKeyCode() == KeyEvent.VK_RIGHT)r[1] = false;
                if (e.getKeyCode() == KeyEvent.VK_LEFT) r[2] = false;
                if (e.getKeyCode() == KeyEvent.VK_DOWN) r[3] = false;
                updatePosition();
            }
            @Override
            public void keyTyped(KeyEvent e) {}
            @Override
            public void keyPressed(KeyEvent e) {
                if (e.getKeyCode() == KeyEvent.VK_UP)    r[0] = true;
                if (e.getKeyCode() == KeyEvent.VK_RIGHT) r[1] = true;
                if (e.getKeyCode() == KeyEvent.VK_LEFT)  r[2] = true;
                if (e.getKeyCode() == KeyEvent.VK_DOWN)  r[3] = true;
                updatePosition();
            }
        });
        this.setFocusable(true);
        this.requestFocusInWindow();
    }
    public void setStartPosition(int xPos, int yPos) {
        this.xPos = xPos;
        this.yPos = yPos;
    }
    private void updatePosition() {
        if (r[0]) { yPos--; }
        if (r[1]) { xPos++; }
        if (r[2]) { xPos--; }
        if (r[3]) { yPos++; }
        repaint();
    }
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.RED);
        g.fillRect(xPos, yPos, 4, 4);
    }
}
```

4. In dieser Lösung wechselt die Farbe bei einem Mausklick zu einer zufällig berechneten Farbe. Dazu erstellen wir bei einem Mausklick ein neues `Color`-Objekt, wobei die RGB-Werte zufällig sind. Zufällige Werte erhalten wir über die Methode `nextInt(256)`, welche einen ganzzahligen Wert zwischen 0 und 255 zurückliefert.

   Natürlich kann man das Ganze auch einfacher halten und zum Beispiel zwischen zwei fest gewählten Farben wechseln. Dazu könnte man die Methode `mouseClicked()` zum Beispiel wie folgt implementieren:

```
@Override
public void mouseClicked(MouseEvent e) {
    if (color == Color.RED) {
        color = Color.BLUE;
    }
    else {
        color = Color.RED;
    }
    repaint();
}
```

   Leider ist der Quelltext der Datei »Grafikcontainer.java« zu lang, um auf eine Seite zu passen. Daher beinhaltet dieser Code einen Seitenumbruch.

```
import javax.swing.*;

public class Exercise {
    public static final int BREITE = 200;
    public static final int HOEHE = 200;

    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLocationRelativeTo(null);
        f.pack();
        Grafikcontainer container = new Grafikcontainer();
        int startX = (BREITE - f.getInsets().left - f.getInsets().
            right) / 2;
        int startY = (HOEHE - f.getInsets().top - f.getInsets().
            bottom) / 2;
        container.setStartPosition(startX, startY);
        f.add(container);
        f.setSize(BREITE, HOEHE);
        f.setVisible(true);
        f.repaint();
    }
}
```

```
import javax.swing.JPanel;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.event.*;
import java.util.Random;

public class Grafikcontainer extends JPanel {
    private int xPos;
    private int yPos;
    private boolean[] r = new boolean[4];
    private Color color = Color.RED;

    public Grafikcontainer() {
        this.addKeyListener(new KeyListener() {
            @Override
            public void keyReleased(KeyEvent e) {
                if (e.getKeyCode() == KeyEvent.VK_UP)   r[0] = false;
                if (e.getKeyCode() == KeyEvent.VK_RIGHT)r[1] = false;
                if (e.getKeyCode() == KeyEvent.VK_LEFT) r[2] = false;
                if (e.getKeyCode() == KeyEvent.VK_DOWN) r[3] = false;
                updatePosition();
            }
            @Override
            public void keyTyped(KeyEvent e) {}
            @Override
            public void keyPressed(KeyEvent e) {
                if (e.getKeyCode() == KeyEvent.VK_UP)    r[0] = true;
                if (e.getKeyCode() == KeyEvent.VK_RIGHT) r[1] = true;
                if (e.getKeyCode() == KeyEvent.VK_LEFT)  r[2] = true;
                if (e.getKeyCode() == KeyEvent.VK_DOWN)  r[3] = true;
                updatePosition();
            }
        });
        this.addMouseListener(new MouseListener() {
            @Override
            public void mouseExited(MouseEvent e) {}
            @Override
            public void mouseEntered(MouseEvent e) {}
            @Override
            public void mouseReleased(MouseEvent e) {}
            @Override
            public void mousePressed(MouseEvent e) {}
            @Override
            public void mouseClicked(MouseEvent e) {
                Random rand = new Random();
                color = new Color(rand.nextInt(256), rand.nextInt
                   (256), rand.nextInt(256));
                repaint();
            }
        });
        this.setFocusable(true);
        this.requestFocusInWindow();
    }
```

```
    public void setStartPosition(int xPos, int yPos) {
        this.xPos = xPos;
        this.yPos = yPos;
    }

    private void updatePosition() {
        if (r[0]) { yPos--; }
        if (r[1]) { xPos++; }
        if (r[2]) { xPos--; }
        if (r[3]) { yPos++; }
        repaint();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(color);
        g.fillRect(xPos, yPos, 4, 4);
    }
}
```

Auflistungsklassen und generische Datentypen

1. Nachdem die letzten Aufgaben etwas schwieriger waren, hier nun wieder eine leichte Aufgabe. Die Lösung dürfte selbsterklärend sein.

```
import java.util.*;

public class Exercise {
    public static void main(String[] args) {
        List<Integer> liste = new ArrayList<Integer>();

        liste.add(1);
        liste.add(2);
        liste.add(3);
        liste.add(4);
        liste.add(5);

        System.out.println(liste);
    }
}
```

2. Auch diese Aufgabe dürfte nicht sonderlich schwer gewesen sein. Gerade wenn man bereits zuvor angegeben hat, dass die `ArrayList` nur Objekte vom Datentyp `int` enthalten darf, so konnte man jetzt einfach alle Elemente aufaddieren. Ansonsten wäre eine Typumwandlung nach `int` notwendig gewesen, bevor eine Addition stattfinden kann.

```java
import java.util.*;

public class Exercise {
    public static void main(String[] args) {
        List<Integer> liste = new ArrayList<Integer>();

        liste.add(1);
        liste.add(2);
        liste.add(3);
        liste.add(4);
        liste.add(5);

        int summe = 0;
        for (int i : liste) {
            summe += i;
        }
        System.out.println(summe);
    }
}
```

Zum Vergleich: Geben wir nicht an, dass die Liste vom Typ `int` sein soll, so ist eine Typumwandlung nötig:

```java
import java.util.*;

public class Exercise {
    public static void main(String[] args) {
        List liste = new ArrayList();

        liste.add(1);
        liste.add(2);
        liste.add(3);
        liste.add(4);
        liste.add(5);

        int summe = 0;
        for (Object o : liste) {
            summe += (int) o;
        }
        System.out.println(summe);
    }
}
```

3. Hier haben sie freie Auswahl gehabt, was für eine Methode Sie implementieren möchten. Für die Beispiellösung habe auch ich einige Spielereien betrieben:

```
public class Exercise {
    public static void main(String[] args) {
        Exercise.test(5);
        Exercise.test(5.0);
        Exercise.test(true);
        Exercise.test(new Exercise());
        Exercise.test(new Character('a'));
    }
    static <E> void test(E input) {
        System.out.print("Der Datentyp ist ");
        if (input instanceof Integer) {
            System.out.println("int.");
        }
        else if (input instanceof Float) {
            System.out.println("float.");
        }
        else if (input instanceof Double) {
            System.out.println("double.");
        }
        else if (input instanceof Boolean) {
            System.out.println("boolean.");
        }
        else if (input instanceof Character) {
            System.out.println("char.");
        }
        else if (input instanceof String) {
            System.out.println("String.");
        }
        else {
            System.out.println("unbekannt.");
        }
    }
}
```

Die Liste an Datentypen ist natürlich nicht vollständig.