

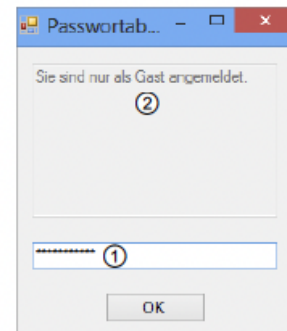
## Aufgabensammlung

### Übung 2: Passwortabfrage

Übungsdatei: --

Ergebnisdatei: *Passwortabfrage.sln*

1. Erstellen Sie eine Windows-Anwendung zur Passwort-Abfrage. Verwenden Sie einen Button- und zwei TextBox-Komponenten.
2. Aktivieren Sie bei der Ausgabe-Textbox über das SmartTask-Menü der Textbox die Eigenschaft `Multiline`.  
Setzen Sie die Eigenschaft `Enabled` der Ausgabe-Textbox auf `False`.  
Tragen Sie bei der Eigenschaft `PasswordChar` der Textbox für die Eingabe ① das Zeichen 8 ein, damit die Passwörter bei der Eingabe verdeckt erscheinen.  
Legen Sie zwei Passwörter (z. B. *ad2013min* für den Administrator und *einUser* für User) fest.  
Nachdem der Benutzer die Schaltfläche *OK* betätigt hat, soll die eingegebene Zeichenkette geprüft werden: Je nachdem, welches der Passwörter der Anwender eingibt, soll in der Textbox ② folgender Text ausgegeben werden: *Sie sind als Administrator angemeldet.* Oder: *Sie haben sich als User angemeldet.* Bei jedem anderen eingegebenen Passwort soll *Sie sind nur als Gast angemeldet.* angezeigt werden.  
Verwenden Sie eine entsprechende Kontrollstruktur, mit der Sie noch weitere Passwörter prüfen könnten.



### Übung 2: Vererbung

Übungsdatei: *Kap10\SomeCars\SomeCars.sln*

Ergebnisdatei: *SomeCars2.sln*

1. Erstellen Sie eine neue Konsolenanwendung.
2. Erweitern Sie die erste Übung des Kapitels 9 um eine abgeleitete Klasse `PKW`. Implementieren Sie darin ein zusätzliches Feld zum Speichern der Anzahl der Sitzplätze und eine Eigenschaft zum Schreib-/Lesezugriff auf dieses Feld.
3. Legen Sie einen Konstruktor an, der den Konstruktor der Basisklasse zur Initialisierung der Felder für Farbe und Baujahr aufruft und selbst nur das zusätzliche Feld (Sitze) initialisiert.
4. Passen Sie das Programm so an, dass es auch mit Instanzen der abgeleiteten Klasse arbeitet und die Werte aller drei Felder (Farbe, Baujahr, Sitzplätze) auf dem Bildschirm ausgibt. Demonstrieren Sie die Funktionalität mit je einer Instanz der Basisklasse und der abgeleiteten Klasse.
5. Verwenden Sie für die drei Klassen des Programms drei separate Dateien.
6. Erzeugen Sie ein Klassendiagramm und blenden Sie darin alle Member und jeweils auch die Datentypen ein.

### Übung 1: Vererbung

Übungsdatei: --

Ergebnisdatei: *Datum.sln*

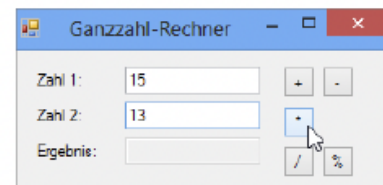
1. Erstellen Sie eine neue Konsolenanwendung.
2. Entwickeln Sie analog zum Beispiel *Zahlen.sln* aus diesem Kapitel eine Klasse, die ein Datum als `Date` speichern und mit einer Methode auf dem Bildschirm ausgeben kann. Erzeugen Sie eine davon abgeleitete Klasse, die eine zusätzliche Methode zum Speichern eines Datumswertes besitzt. Testen Sie die Methoden mithilfe einer Instanz der abgeleiteten Klasse.

## Übung 4: Ganzzahl-Taschenrechner objektorientiert

**Übungsdatei:** *Kap09\IntegerCalc\IntegerCalcOOP.sln*

**Ergebnisdatei:** *IntegerCalcProp.sln*

1. Verändern Sie den Taschenrechner aus der Übung des Kapitels 9. Kapseln Sie die beiden Felder für die Speicherung der eingegebenen Zahlen und implementieren Sie zwei Schreib-/ Leseigenschaften, um auf die beiden Felder zuzugreifen.



## Übung 3: Statische Member

**Übungsdatei:** --

**Ergebnisdatei:** *Steuern.sln*

1. Erstellen Sie eine neue Konsolenanwendung.
2. Schreiben Sie in der Datei *Program.cs* eine Klasse *Steuern*, die ein statisches Feld *Mehrwertsteuer* vom Typ *double* und eine statische Eigenschaft *MWST* für den Schreib-/Lesezugriff auf diese Variable enthält.
3. Implementieren Sie zusätzlich in der Klasse eine öffentliche Methode *InclMWST*, die zu einem übergebenen Betrag vom Typ *double* die Mehrwertsteuer hinzurechnet und das Gesamtergebnis zurückgibt.
4. Legen Sie in der Klasse *Program* mit der Methode *Main()* die Mehrwertsteuer auf 19 % (=0,19) fest. Speichern Sie in einer Variablen einen beliebigen Betrag (*netto*) und berechnen Sie anschließend den Gesamtbetrag inklusive Mehrwertsteuer (*brutto*). Geben Sie den Mehrwertsteuersatz sowie den Netto- und den Bruttobetrag aus.

## Übung 1: Klassen und Klasseninstanzen

Übungsdatei: --

Ergebnisdatei: *SomeCars.sln*

1. Erstellen Sie eine Konsolenanwendung.
2. Deklarieren Sie eine Klasse `Auto` und in dieser Klasse je ein Feld für das Baujahr und die Farbe sowie zwei Eigenschaften für den Schreib-/Lesezugriff auf die Felder. Deklarieren Sie außerdem einen Konstruktor, mit dem Sie die beiden Felder initialisieren können.
3. Erzeugen Sie in der Hauptmethode der Anwendung zwei Instanzen der Klasse `Auto` und zeigen Sie die Werte der Felder nach der Initialisierung mithilfe der Eigenschaften auf dem Bildschirm an.
4. Ändern Sie dann die Farbwerte der beiden Objekte. Geben Sie zur Kontrolle der Änderung die Werte der Felder noch einmal aus.

## Übung 2: Kapselung – Zugriff auf Felder über Eigenschaften

Übungsdatei: --

Ergebnisdatei: *Kontofuehrung.sln*

1. Erstellen Sie eine Konsolenanwendung, um ein Programm für eine einfache Kontoführung zu schreiben.
2. Entwickeln Sie eine in einer separaten Datei gespeicherte Klasse `Konto`. Als einziges Feld soll diese Klasse eine als `private` deklarierte Variable für den Kontostand besitzen.
3. Der Zugriff auf den Kontostand erfolgt ausschließlich über eine als `public` deklarierte Eigenschaft für den Schreib-/Lesezugriff.
4. Implementieren Sie einen Konstruktor, über den Sie den Kontostand mit einem bestimmten Wert (hier 5000) initialisieren können.
5. Die Benutzerschnittstelle sollte so aufgebaut sein, dass sowohl zu Beginn des Programms als auch nach jeder Ein- bzw. Auszahlung der aktuelle Kontostand angezeigt wird. Mit einer `switch`-Anweisung ermöglichen Sie dem Benutzer, Ein- bzw. Auszahlungen vorzunehmen sowie das Programm zu beenden.

```
Kontostand: 5000
Einzahlen(1), Auszahlen(2), Beenden(0) :
1
Betrag: 900
Kontostand: 5900
Einzahlen(1), Auszahlen(2), Beenden(0) :
2
Betrag: 750
Kontostand: 5150
Einzahlen(1), Auszahlen(2), Beenden(0) :
```

*Die Ausgabe des Programms*



## Übung 1: Werte ein- und ausgeben

Übungsdatei: --


Ergebnisdatei: *InputConsole.sln*

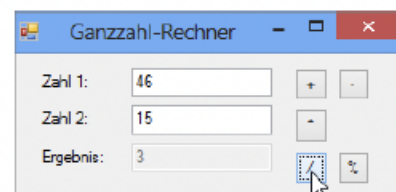
1. Fordern Sie in der Anwendung den Anwender dazu auf, eine ganze Zahl einzugeben, und speichern Sie den Wert in einer Variablen vom Typ `int`.
2. Lassen Sie auch einen `double`- und einen `string`-Wert mithilfe der Tastatur eingeben und speichern Sie die Werte ebenfalls in geeigneten Variablen.
3. Geben Sie die Werte der drei Variablen abschließend auf dem Bildschirm aus.

## Übung 2: Ein einfacher Rechner für ganze Zahlen

Übungsdatei: --

Ergebnisdatei: *IntegerCalc.sln*

1. Erstellen Sie ein neues Projekt als Windows-Anwendung.
2. Platzieren Sie auf dem Formular:
  - ✓ drei `TextBox`-Steuerelemente (zwei zur Eingabe und eines zur Ausgabe der Werte),
  - ✓ drei `Label`-Steuerelemente zur Beschriftung,
  - ✓ fünf `Button`-Steuerelemente.
3. Setzen Sie die Eigenschaften `Enabled` und `ReadOnly` des Ergebnis-Ausgabefelds (`TextBox`-Steuerelement) auf den Wert `false`.
4. Erzeugen Sie durch Doppelklick auf die Schaltfläche  das Gerüst für eine entsprechende Ereignisbehandlungsmethode. Mit der Ereignisbehandlungsmethode sollen zwei eingegebene Zahlen (`Zahl1` und `Zahl2`) addiert werden.
5. Lesen Sie die Eigenschaft `Text` des ersten Felds aus und speichern Sie die Zeichenkette in einer `string`-Variablen.



Zum Auslesen der Eigenschaft `Text` schreiben Sie hinter den Namen des `TextBox`-Steuerelements (hier `txtZahl1`) durch einen Punkt getrennt den Eigenschaftsnamen `Text` und weisen diesen Ausdruck einer Variablen vom Typ `string` zu.

```
string txt;  
txt = txtZahl1.Text;  
...
```

6. Wandeln Sie die Eingabe in den gewünschten Datentyp (`int`) um.
7. Speichern Sie auch die zweite eingegebene Zahl in einer geeigneten Variablen.
8. Berechnen Sie das Ergebnis in einer Variablen vom Typ `int`.
9. Wandeln Sie das Ergebnis wieder in einen `String` um und weisen Sie diesen der Eigenschaft `Text` des Ausgabefelds zu.
10. Programmieren Sie auf die gleiche Weise die Ereignisbehandlungsmethoden für die anderen Schaltflächen. Neben der bereits programmierten Addition sollen Subtraktion, Multiplikation, Division und Modulo-Berechnung (Divisionsrest) möglich sein.

```
...  
txtErgebnis.Text = txt;
```

## Übung 3: Überladen einer Methode

Übungsdatei: --

Ergebnisdatei: *Ueberladen.sln*

1. Erstellen Sie in einem Konsolenprogramm eine Klasse `Wert`. Deklarieren Sie in dieser Klasse überladene Methoden, die einen `int`- bzw. einen `double`-Wert oder einen `string` als Parameter entgegennehmen und eine entsprechende Ausgabe durchführen:
  - ✓ Die Ganzzahl lautet: *übergebene Zahl*
  - ✓ Die Dezimalzahl lautet: *übergebener Wert*
  - ✓ Der Text lautet: *übergebener Text*
2. Testen Sie die Methoden, indem Sie sie je einmal mit einem Wert der drei verschiedenen Datentypen aufrufen.

## Übung 2: Methode für Primzahltest

Übungsdatei: --

Ergebnisdatei: *Primzahlen.sln*

1. Schreiben Sie eine Klasse `Zahl`, die nur die eine Methode `IsPrime()` enthalten soll. Diese Methode prüft einen übergebenen `int`-Wert daraufhin, ob es sich um eine Primzahl handelt. Primzahlen sind nur durch sich selbst und durch 1 teilbar. Die Methode soll den Rückgabebetyp `bool` besitzen. Sie gibt `true` zurück, wenn es sich um eine Primzahl handelt, ansonsten `false`.
2. Testen Sie die Methode in einer Konsolenanwendung. Nach der Eingabe eines Zahlenbereichs sollen alle Zahlen dieses Bereichs in einer Schleife durchlaufen werden. Darin werden sie mithilfe der Primzahl-Methode getestet. Im Falle des Rückgabewertes `true` wird die Primzahl auf dem Bildschirm ausgegeben.

**Lösungshinweis:** Für einen simplen Primzahltest reicht es aus, die betreffende Zahl durch alle Zahlen von 2 bis Zahl geteilt durch 2 zu teilen und zu prüfen ob der Rest 0 ist. In diesem Fall würde es sich nicht um eine Primzahl handeln.

```
for (int i = 2; i <= (zahl / 2) + 1; i++)  
    if (zahl % i == 0)  
        ...
```

*Beispiel für einen Primzahltest*

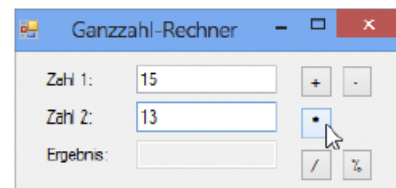
**Hinweis:** Beachten Sie bei der Eingabe des Zahlenbereichs, dass sehr große Zahlen möglicherweise nicht dem Wertebereich der verwendeten Variablen entsprechen und dadurch zu einem Fehler führen. Testen Sie die Anwendung beispielsweise mit dem Zahlenbereich 10...500.

## Übung 4: Ganzzahl-Taschenrechner objektorientiert

Übungsdatei: *Kap07\IntegerCalc\IntegerCalc.sln*

Ergebnisdatei: *IntegerCalcOOP.sln*

1. Verändern Sie den Taschenrechner aus der Übung des Kapitels 7.
2. Erstellen Sie in einer separaten Datei eine Klasse `Rechner`. Die Klasse erhält zwei als `public` deklarierte Felder zur Speicherung der beiden Operanden (`int`).
3. Ergänzen Sie in der Klasse `Rechner` fünf Methoden für die verschiedenen mathematischen Operationen, die jeweils das Ergebnis zurückgeben.
4. Erzeugen Sie in der Formular-Klasse eine Instanz `rechenoperationen` der Klasse `Rechner`. Schreiben Sie eine Initialisierungsmethode `init()`, die die Textfelder ausliest, in Zahlenwerte konvertiert und in den Feldern des Objekts `rechenoperationen` speichert. Erstellen Sie außerdem eine Methode `anzeigen()`, die ein als Parameter übergebenes Ergebnis (`int`) in dem Textfeld `Ergebnis` anzeigt.
5. Schreiben Sie die Ereignisbehandlungsmethoden für das Click-Ereignis der Schaltflächen, in denen Sie die Initialisierung durchführen, die entsprechende Methode zur Berechnung aufrufen und das Ergebnis anzeigen lassen.



**Hinweis:** Achten Sie bei der Eingabe von Werten in die Textfelder *Zahl 1* und *Zahl 2* darauf, dass der Wertebereich eingehalten wird. Sehr große Zahlen sowie Texteingabe in die Felder führen zu einem Programmabbruch.

### 4.6 Übungsaufgabe: Rechnen mit Variablen

Schreiben Sie ein Programm, das vom Benutzer nacheinander zwei ganze Zahlen einliest, die beiden Zahlen dann addiert und das Ergebnis am Bildschirm ausgibt.

#### Musterlösung:

```
1 using System;
2
3 namespace Variablen
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Geben Sie eine ganze Zahl ein:");
10            var a = Console.ReadLine();
11            Console.WriteLine("Geben Sie noch eine ganze Zahl
12            ein:");
13            var b = Console.ReadLine();
14            var ausgabe = int.Parse(a) + int.Parse(b);
15            Console.WriteLine($"Die Summe von {a} und {b} ist
16            {ausgabe}.");
17        }
18    }
19 }
```



Abb. 4.6.1 Ausgabe der Musterlösung

## 5.6 Übungsaufgabe: Programmieren mit Verzweigungen

In dieser Übung werden Sie dreimal das gleiche Programm schreiben, aber mit drei verschiedenen Techniken.

Schreiben Sie ein Programm, das den Benutzer nach dem Namen eines Wochentags fragt und dann „Hurra Wochenende!“ ausgibt, falls der Benutzer „Samstag“ oder „Sonntag“ eingibt und „Oje, Sie müssen arbeiten!“, falls der Benutzer einen der Wochentage

von „Montag“ bis „Freitag“ eingibt. Falls der Benutzer keinen Wochentag, sondern irgendetwas anderes eingibt, soll das Programm „Das ist kein Wochentag!“ ausgeben.

Schreiben Sie das Programm zuerst so, dass die Entscheidungsfindung nur durch if-else-

Verzweigungen realisiert wird. Dann schreiben Sie es ein zweites Mal und verwenden zur Entscheidungsfindung eine Switch-Case-Mehrfachverzweigung. Schreiben Sie das Programm ein drittes Mal und benutzen Sie dabei nur bedingte Zuweisungen zur Entscheidungsfindung.

### Musterlösung mit If-else-Verzweigungen:

```
1 using System;
2
3 namespace Musterloesung_5_6_1
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Geben Sie einen Wochentag ein:");
10            var wochentag = Console.ReadLine();
11
12            if (wochentag == "Samstag" || wochentag == "Sonntag")
13            {
14                Console.WriteLine("Hurra Wochenende!");
15            }
16            else
17            {
18                if (wochentag == "Montag" ||
19                    wochentag == "Dienstag" ||
20                    wochentag == "Mittwoch" ||
21                    wochentag == "Donnerstag" ||
22                    wochentag == "Freitag")
23                {
24                    Console.WriteLine("Oje, Sie müssen
25                        arbeiten!");
26                }
27                else
28                {
29                    Console.WriteLine("Das ist kein Wochentag!");
30                }
31            }
32        }
33    }
34 }
```



### Musterlösung mit bedingten Zuweisungen:

```
1 using System;
2
3 namespace Musterlösung_5_6_3
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Geben Sie einen Wochentag ein:");
10            var wochentag = Console.ReadLine();
11
12            var ausgabe = (wochentag == "Samstag" || wochentag
13 == "Sonntag")
14                ? "Hurra Wochenende!"
15                : (
16                    wochentag == "Montag" ||
17                    wochentag == "Dienstag" ||
18                    wochentag == "Mittwoch" ||
19                    wochentag == "Donnerstag" ||
20                    wochentag == "Freitag"
21                )
22                ? "Oje, Sie müssen arbeiten!"
23                : "Das ist kein Wochentag!";
24
25            Console.WriteLine(ausgabe);
26        }
27    }
```

### Musterlösung mit einer Switch-Case-Mehrfachverzweigung:

```
1 using System;
2
3 namespace Musterloesung_5_6_2
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Geben Sie einen Wochentag ein:");
10            var wochentag = Console.ReadLine();
11
12            switch(wochentag)
13            {
14                case "Samstag":
15                case "Sonntag":
16                    Console.WriteLine("Hurra Wochenende!");
17                    break;
18                case "Montag":
19                case "Dienstag":
20                case "Mittwoch":
21                case "Donnerstag":
22                case "Freitag":
23                    Console.WriteLine("Oje, Sie müssen
24 arbeiten!");
25                    break;
26                default:
27                    Console.WriteLine("Das ist kein Wochentag!");
28                    break;
29            }
30        }
31    }
32 }
```



## 6.6 Übungsaufgabe: Programmsteuerung mit Schleifen

Schleifen gehören in der Programmierung mit zu den wichtigsten Techniken. Daher ist es für Programmierer essenziell, dieses Thema zu beherrschen. Deswegen machen wir hier auch zum Thema „Schleifen“ eine längere Übung.

### Übungsaufgabe 1:

Erstellen Sie in Visual Studio eine neue Konsolen-App und mit folgendem Programmcode für die Datei „Program.cs“.

```
1 using System;
2
3 namespace Uebung_6_6_Aufgabe1
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             int zahl = 0;
10            Console.WriteLine("Geben Sie eine ganze Zahl ein:");
11            do
12            {
13                Console.WriteLine($"{5 geteilt durch {zahl} = {(5 / zahl)} Rest {5 % zahl}");
14
15                Console.WriteLine("Geben Sie eine ganze Zahl ein:");
16            }
17            while ( (zahl = int.Parse(Console.ReadLine())) != 0);
18            Console.WriteLine("Durch 0 kann nicht geteilt werden!");
19        }
20    }
21 }
```

Wenn Sie das Programm starten, erhalten Sie die Fehlermeldung „System.DivideByZeroException“.

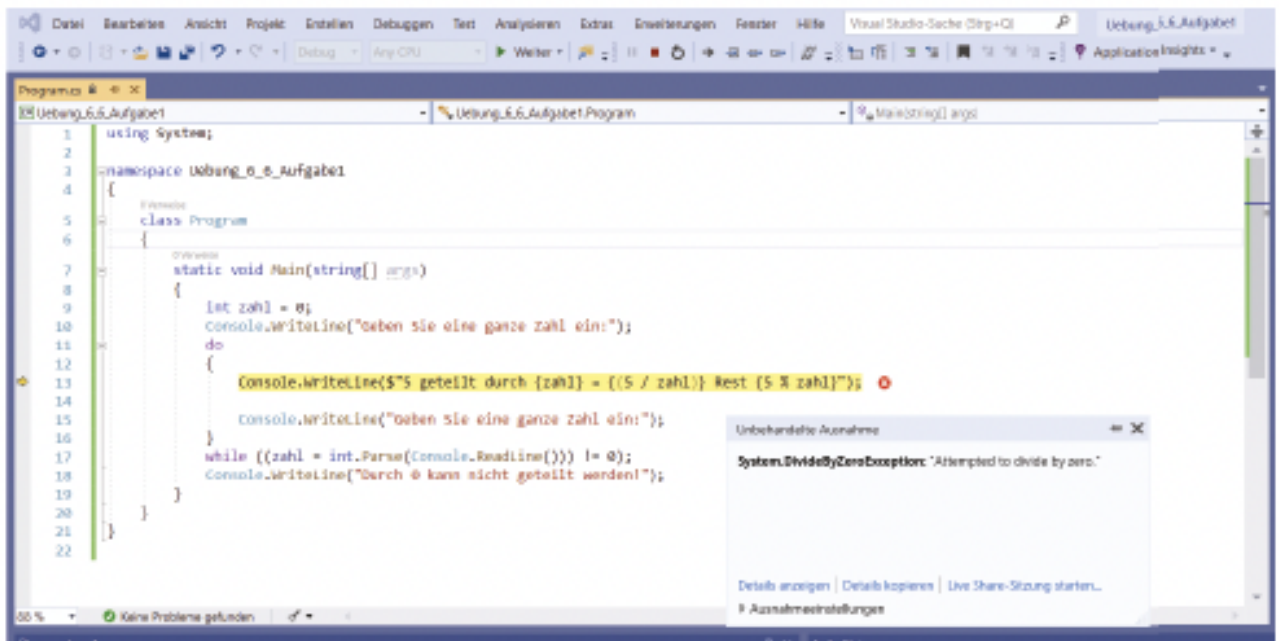


Abb. 6.6.1 DivideByZeroException nach Programmstart

Korrigieren Sie das Programm, in dem Sie nur den Schleifentyp wechseln.

## Musterlösung für Übungsaufgabe 1:

```
1 namespace Uebung_6_6_Aufgabe1
2 {
3     class Program
4     {
5         static void Main(string[] args)
6         {
7             int zahl = 0;
8             Console.WriteLine("Geben Sie eine ganze Zahl ein:");
9             while ((zahl = int.Parse(Console.ReadLine())) != 0)
10             {
11                 Console.WriteLine($"{5 geteilt durch {zahl} = {(5
12                     / zahl)} Rest {5 % zahl}");
13                 Console.WriteLine("Geben Sie eine ganze Zahl
14                     ein:");
15             }
16             Console.WriteLine("Durch 0 kann nicht geteilt
17                 werden!");
18         }
19     }
20 }
```

## Übungsaufgabe 2:

Schreiben Sie ein Programm, das eine Textzeile nach der anderen von der Tastatur einliest. Wenn der Benutzer die Textzeile „Fertig!“ eingibt, soll das Programm eine Zeile mit 10 Sternchen ausgeben und dann den gesamten eingegebenen Text, aber ohne die Zeile „Fertig!“.

## Musterlösung für Übungsaufgabe 2:

```
1  using System;
2
3  namespace Uebung_6_6_Aufgabe2
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              //Hier speichern wir den vollständigen
10             //eingegebenen Text
11             var text = "";
12             //Das Fertig-Kommando legen wir uns auch in eine
13             //Variable, damit wir es nur an einer Stelle
14             //ändern müssen falls nötig
15             var fertig = "Fertig!";
16             //Hier speichern wir die aktuelle eingegebene
17             //Zeile
18             var eingabe = "";
19
20             //Die Schleife läuft, bis der Benutzer
21             //das Fertig-Kommando eingibt
22             while((eingabe = Console.ReadLine()) != fertig)
23             {
24                 //Die neue Zeile an den Text anhängen
25                 text += eingabe;
26
27                 //Einen Zeilenumbruch an den Text anhängen
28                 text += "\r\n";
29             }
30
31             Console.WriteLine("*****");
32
33             //Text in Zeilen splitten
34             var zeilen = text.Split("\r\n");
35
36             //Die Schleife läuft über alle Zeilen
37             foreach(var zeile in zeilen)
38             {
39                 //Wenn wir die Zeile mit dem Fertig-Kommando
40                 //noch nicht erreicht haben
41                 if(zeile != fertig)
42                 {
43                     //dann geben wir die Zeile aus
44                     Console.WriteLine(zeile);
45                 }
46             }
47         }
48     }
49 }
```

## Übungsaufgabe 4:

Ein Array ist wie folgt deklariert:

```
1 int[] zahlen = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

Schreiben Sie ein Programm, das mit Hilfe dieses Arrays ein Dreieck aus „\*“-Zeichen am Bildschirm ausgibt.

## Musterlösung für Übungsaufgabe 4:

```
1 using System;
2
3 namespace Uebung_6_6_Aufgabe4
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             //Ein Array mit Zahlen von 1-9
10            int[] zahlen = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
11
12            //Schleife läuft über das gesamte Array
13            foreach(var zahl in zahlen)
14            {
15                //Die Schleife hat „zahl“ Durchgänge
16                for(var i = 0; i < zahl; i++)
17                {
18                    Console.Write("*");
19                }
20
21                //Ein Zeilenumbruch am Ende von jedem Durchgang
22                //der äußeren Schleife
23                Console.WriteLine();
24            }
25        }
26    }
27 }
```



Wie versprochen, programmieren wir jetzt schrittweise ein Tic-Tac-Toe-Spiel. Die Aufgabe ist aufgeteilt in sieben Teilaufgaben. Anschließend an die Teilaufgaben finden Sie Musterlösungen für jede Teilaufgabe, die das vollständige Programm bis einschließlich der Lösung der jeweiligen Teilaufgabe zeigt.

### Teilaufgabe 1

Deklarieren Sie in der Hauptmethode geeignete Variablen für Zeichen, die jeweils ein leeres Feld, ein Feld mit einem Kreuz und ein Feld mit einem Kreis symbolisieren. Deklarieren Sie eine Struktur, die den Zustand eines Tic-Tac-Toe-Spielfelds speichern kann. Die Struktur soll auch ein Koordinatensystem enthalten, das in horizontaler Richtung von 1 bis 3 läuft und in vertikaler Richtung von A bis C. Belegen Sie alle Variablen mit Startwerten.

### Musterlösung für Teilaufgabe 1

```
1 using System;
2
3 namespace TicTacToe
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             var leeresFeld = " ";
10            var kreuz = "X";
11            var kreis = "O";
12
13            var spielFeld = new string[4, 4]
14            {
15                {" ", "1", "2", "3"},
16                {"A", leeresFeld, leeresFeld, leeresFeld},
17                {"B", leeresFeld, leeresFeld, leeresFeld},
18                {"C", leeresFeld, leeresFeld, leeresFeld}
19            };
20
21        }
22    }
23 }
24 }
```

### Teilaufgabe 2:

Legen Sie in der Hauptmethode zwei geeignete Variablen an, um die Namen von zwei Spielern zu speichern. Schreiben Sie eine Methode, die den Namen für einen Spieler einliest und rufen Sie diese Methode zweimal in der Hauptmethode auf, um die Namen von zwei Spielern abzufragen.

### Musterlösung für Teilaufgabe 2:

```
1 using System;
2
3 namespace TicTacToe
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             var leeresFeld = "#";
10            var kreuz = "X";
11            var kreis = "O";
12
13            string spieler1, spieler2;
14
15            var spielFeld = new string[4, 4]
16            {
17                {" ", "1", "2", "3"},
18                {"A", leeresFeld, leeresFeld, leeresFeld},
19                {"B", leeresFeld, leeresFeld, leeresFeld},
20                {"C", leeresFeld, leeresFeld, leeresFeld}
21            };
22
23            spieler1 = FrageSpielerNameAb(1);
24            spieler2 = FrageSpielerNameAb(2);
25        }
26
27        static string FrageSpielerNameAb(int spielerNummer)
28        {
29            Console.WriteLine($"Geben Sie einen Namen für Spieler
30            {spielerNummer} ein:");
31            return Console.ReadLine();
32        }
33    }
34 }
35 }
```

### Teilaufgabe 3:

Schreiben Sie die Methode `Spielzug()` mit dem Rückgabebetyp `void` und folgenden Übergabeparametern:

1 `string name`, `string spielstein`, `string[,] spielfeld`, `string`  
2 `leeresFeld`

Der Übergabeparameter `name` erwartet den Namen des Spielers, der Parameter `spielstein` soll das Zeichen für den Spielstein (Kreuz oder Kreis) des Spielers enthalten. Das Array `spielfeld` soll die in der Hauptmethode definierte Struktur erhalten, die den Zustand des Spielfelds speichert. Der Parameter `leeresFeld` soll das Zeichen enthalten, das ein leeres Feld symbolisiert. Die Methode `Spielzug` soll einen

Spielzug machen. Dazu soll sie den im Parameter `name` übergebenen Benutzer nach den Koordinaten seines Zuges fragen. Die Koordinaten sollen in der Form Buchstabe, Zahl eingegeben werden: zum Beispiel A,1 oder C,3. Bei ungültigen Eingaben oder wenn das Feld an den vom Spieler eingegebenen Koordinaten schon besetzt ist, soll der Spieler auf seinen Fehler aufmerksam gemacht werden und das Programm soll

nochmal die Koordinaten für den Zug des Spielers abfragen. Bei gültigen Koordinaten soll die Methode an die eingegebenen Koordinaten den Spielstein (Kreuz oder Kreis) setzen und die Methode endet.

**ACHTUNG:** Beachten Sie, dass die Variablen `name`, `spielstein` und `leeresFeld` einen primitiven Datentyp haben. Sie werden als Kopie an die Methode übergeben. Die Variable `spielfeld` dagegen ist ein Array. Arrays werden standardmäßig nicht als Kopie, sondern als ref-Parameter übergeben, auch wenn das Schlüsselwort `ref` nicht vor dem Parameter steht. Das heißt, wenn die Methode `Spielzug` einen Wert in den Array-Parameter `spielfeld` schreibt, dann ist dieser neue Wert auch außerhalb der Methode `Spielzug` verfügbar.

## Musterlösung für Teilaufgabe 3:

```
1 using System;
2 using System.Collections.Generic;
3
4 namespace TicTacToe
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10             var leeresFeld = "#";
11             var kreuz = "X";
12             var kreis = "O";
13
14             string spieler1, spieler2;
15
16             var spielFeld = new string[4, 4]
17             {
18                 {" ", "1", "2", "3"},
19                 {"A", leeresFeld, leeresFeld, leeresFeld},
20                 {"B", leeresFeld, leeresFeld, leeresFeld},
21                 {"C", leeresFeld, leeresFeld, leeresFeld}
22             };
23
24             spieler1 = FrageSpielerNameAb(1);
25             spieler2 = FrageSpielerNameAb(2);
26
27             Spielszug(spieler1, kreuz, spielFeld, leeresFeld);
28         }
29
30         static string FrageSpielerNameAb(int spielerNummer)
31         {
32             Console.WriteLine($"Geben Sie einen Namen für Spieler
33 {spielerNummer} ein:");
34             return Console.ReadLine();
35         }
36
37         static void Spielszug(string name, string spielstein,
38 string[,] spielFeld, string leeresFeld)
39         {
40
41             var eingabeOk = false;
42             List<string> buchstaben = new List<string> { "A",
43 "B", "C" };
44             List<string> zahlen = new List<string> { "1", "2",
45 "3" };
46             string fehler = "Ungültiger Zug!";
47             string feldBesetzt = "Dieses Feld ist schon
48 besetzt!";
49
50             while(!eingabeOk)
51             {
52                 Console.WriteLine($"{name}, geben Sie die
53 Koordinaten für Ihren Zug ein (Buchstabe,Zahl)");
54
```



```

55         var eingabe = Console.ReadLine();
56
57         if (eingabe.Length != 3)
58         {
59             Console.WriteLine(fehler);
60             continue;
61         }
62
63         var koordinaten = eingabe.Split(",");
64
65         if(koordinaten.Length !=2)
66         {
67             Console.WriteLine(fehler);
68         }
69         if(!buchstaben.Contains(koordinaten[0]))
70         {
71             Console.WriteLine(fehler);
72             continue;
73         }
74         if(!zahlen.Contains(koordinaten[1]))
75         {
76             Console.WriteLine(fehler);
77             continue;
78         }
79
80         var i = buchstaben.IndexOf(koordinaten[0]) + 1;
81         var j = int.Parse(koordinaten[1]);
82
83         if(spielFeld[i,j] != leeresFeld)
84         {
85             Console.WriteLine(feldBesetzt);
86             continue;
87         }
88
89         spielFeld[i, j] = spielstein;
90
91         eingabeOk = true;
92     }
93 }
94 }
95 }

```

#### Teilaufgabe 4:

Schreiben Sie die Methode `ZeigeSpielFeld()`. Die Methode `ZeigeSpielFeld()` hat den Rückgabotyp `void` und erhält als Übergabeparameter das zweidimensionale Array `spielFeld`, das in der Hauptmethode deklariert wurde. Die Methode `ZeigeSpielFeld()` soll das Tic-Tac-Toe-Spielfeld als Quadrat von 4 mal 4 Zeichen am Bildschirm ausgeben.

### 9.8 Übungsaufgabe: Mit Objekten programmieren

Nachdem wir die Grundlagen der Objektorientierten Programmierung behandelt haben, vertiefen wir dieses Wissen mit einer umfangreicheren praktischen Übung. Wir werden Schritt für Schritt eine Anwendung zum Verwalten von Telefonnummern und E-Mail-Adressen entwickeln. Unsere Anwendung soll das Neuanlegen, Ändern, Suchen und Anzeigen von Telefonnummern und E-Mail-Adressen beherrschen. Wir werden diese Anwendung objektorientiert schreiben und dabei auf die Verwendung von public-Membervariablen verzichten.

#### Teilaufgabe 1:

Schreiben Sie eine Klasse für einen Verzeichniseintrag. Die Klasse soll Vorname, Nachname, Festnetznummer, Mobilfunknummer und E-Mailadresse für einen Eintrag speichern können. Zudem soll die Klasse die Methode `ZeigeEintrag()` die, die Daten eines Eintrags anzeigt, zur Verfügung stellen.

#### Musterlösung für Teilaufgabe 1:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace TelefonVerzeichnisObjektorientiert
6 {
7     public class Verzeichniseintrag
8     {
9         public string Vorname {get; set;}
10        public string Nachname { get; set; }
11        public string Festnetz { get; set; }
12        public string Mobilfunk { get; set; }
13        public string Email { get; set; }
14
15        public Verzeichniseintrag(string vorname, string
16        nachname, string festnetz, string mobilfunk, string
17        email)
18        {
19            Vorname = vorname;
20            Nachname = nachname;
21            Festnetz = festnetz;
22            Mobilfunk = mobilfunk;
23            Email = email;
24        }
25
26        public void ZeigeEintrag()
27        {
28            Console.WriteLine($"{Vorname} {Nachname}; Festnetz:
29            {Festnetz}; Mobilfunk: {Mobilfunk}; Email: {Email}");
30        }
31    }
32 }
```

## Teilaufgabe 2:

Schreiben Sie die Klasse Verzeichnis. Sie soll ein Dictionary zum Speichern der Verzeichniseinträge enthalten. Als Schlüssel für das Dictionary wird der String Vorname:Nachname verwendet. Die Klasse Verzeichnis soll zudem im Konstruktor das Dictionary mit ein paar Verzeichniseinträgen initialisieren. Die Klasse soll die public-Methode NeuerEintrag() enthalten die, die String-Parameter vorname, nachname, festnetz, mobilfunk und email entgegennimmt und damit einen neuen Eintrag im Dictionary erstellt. Außerdem soll die Klasse die Methode ZeigeAlleEinträge(), die alle Verzeichniseinträge am Bildschirm ausgibt, bereitstellen.

## Musterlösung für Teilaufgabe 2:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace TelefonVerzeichnisObjektorientiert
6 {
7     public class Verzeichnis
8     {
9         private Dictionary<string, Verzeichniseintrag>
10         verzeichnis = new Dictionary<string,
11         Verzeichniseintrag>();
12
13         public Verzeichnis()
14         {
15             NeuerEintrag("Robert", "Schiefele", "08912356489",
16             "01715589665", "robert.schiefele@email.de");
17             NeuerEintrag("Katia", "Japa", "0897589445",
18             "017259894", "katia.japa@email.de");
19             NeuerEintrag("Theo", "Tester", "089885595445",
20             "017365894", "theo.testster@email.de");
21         }
22
23         public void NeuerEintrag(string vorname, string
24         nachname, string festnetz, string mobilfunk, string
25         email)
26         {
27             verzeichnis.Add($"{vorname}:{nachname}", new
28             Verzeichniseintrag(vorname, nachname, festnetz,
29             mobilfunk, email));
30         }
31
32         public void ZeigeAlleEintraege()
33         {
34             foreach(var eintrag in verzeichnis.Values)
35             {
36                 eintrag.ZeigeEintrag();
37             }
38         }
39     }
```

### Teilaufgabe 3:

Erweitern Sie die Klasse Verzeichnis um die folgenden drei Methoden:

Die Methode ZeigeEintrag() nimmt den String-Parameter schlüssel entgegen und überprüft, ob dieser Schlüssel im Dictionary enthalten ist. Wenn ja, gibt die Methode den Verzeichniseintrag am Bildschirm aus, wenn nein, eine entsprechende Meldung, dass es für diesen Schlüssel keinen Eintrag im Verzeichnis gibt.

Die Methode ZeigeEintraegeFuerNachname() nimmt den String-Paramater nachname entgegen und gibt alle Eintr.ge mit diesem Nachnamen am Bildschirm aus. Werden keine Eintr.ge für den entsprechenden Nachnamen gefunden, so wird eine Meldung ausgegeben, dass es für diesen Nachnamen keine Eintr.ge im Verzeichnis gibt. Die Methode

ZeigeEintraegeFuerVorname() nimmt den String-Paramater

vorname entgegen und gibt alle Eintr.ge mit diesem Vornamen am Bildschirm aus. Werden keine Eintr.ge für den entsprechenden Vornamen gefunden, so wird eine Meldung ausgegeben, dass es für diesen Vornamen keine Eintr.ge im Verzeichnis gibt.

### Musterlösung für Teilaufgabe 3:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace TelefonVerzeichnisObjektorientiert
6 {
7     public class Verzeichnis
8     {
9         private Dictionary<string, VerzeichnisEintrag>
10             verzeichnis = new Dictionary<string,
11 VerzeichnisEintrag>();
12
13         public Verzeichnis()
14         {
15             NeuerEintrag("Robert", "Schiefele", "08912356489",
16 "01715589665", "robert.schiefele@email.de");
17             NeuerEintrag("Katia", "Japa", "0897589445",
18 "017259894", "katia.japa@email.de");
19             NeuerEintrag("Theo", "Tester", "089885595445",
20 "017365894", "theo.testers@email.de");
21         }
22
23         public void NeuerEintrag(string vorname, string
24 nachname, string festnetz, string mobilfunk, string
25 email)
26         {
```



```

27         verzeichnis.Add($"{vorname}:{nachname}", new
28         VerzeichnisEintrag(vorname, nachname, festnets,
29         mobilfunk, email));
30     }
31
32     public void ZeigeAlleEintraege()
33     {
34         foreach(var eintrag in verzeichnis.Values)
35         {
36             eintrag.ZeigeEintrag();
37         }
38     }
39
40     public void ZeigeEintrag(string schluessel)
41     {
42         if(verzeichnis.ContainsKey(schluessel))
43         {
44             verzeichnis[schluessel].ZeigeEintrag();
45         }
46         else
47         {
48             Console.WriteLine($"Eintrag: {schluessel} nicht
49             gefunden.");
50         }
51     }
52
53     public void ZeigeEintraegeFuerNachname(string nachname)
54     {

```

```

55         var gefunden = false;
56
57         foreach(var element in verzeichnis)
58         {
59             if(element.Value.Nachname == nachname)
60             {
61                 element.Value.ZeigeEintrag();
62                 gefunden = true;
63             }
64         }
65
66         if(!gefunden)
67         {
68             Console.WriteLine($"Keinen Eintrag für {nachname}
69             gefunden");
70         }
71     }
72
73     public void ZeigeEintraegeFuerVorname(string vorname)
74     {
75         var gefunden = false;
76
77         foreach (var element in verzeichnis)
78         {
79             if (element.Value.Vorname == vorname)
80             {
81                 element.Value.ZeigeEintrag();
82                 gefunden = true;
83             }
84         }
85
86         if (!gefunden)
87         {
88             Console.WriteLine($"Keinen Eintrag für {vorname}
89             gefunden");
90         }
91     }
92 }
93 }

```

#### Teilaufgabe 4:

Erweitern Sie die Klasse Verzeichnis: Schreiben Sie die Methode AktualisiereEintrag(), die die String-Parameter schluessel, festnetz, mobilfunk und email entgegennimmt. Die Methode soll den Verzeichniseintrag, der dem übergebenen Schlüssel entspricht, mit den Werten in den entsprechenden übergebenen Parametern aktualisieren. Wenn der übergebene Schlüssel nicht im Verzeichnis gefunden wird, soll die Methode eine entsprechende Meldung ausgeben. Schreiben Sie die Methode LoescheEintrag(). Die Methode soll den String-Parameter schluessel entgegennehmen und den Verzeichniseintrag, der dem übergebenen Schlüssel entspricht, aus dem Verzeichnis entfernen. Wenn der übergebene Schlüssel nicht im Verzeichnis gefunden wird, soll die Methode eine entsprechende Meldung ausgeben.

#### Musterlösung für Teilaufgabe 4

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace TelefonVerzeichnisObjektorientiert
6 {
7     public class Verzeichnis
8     {
9         private Dictionary<string, Verzeichniseintrag> verzeichnis
10         = new Dictionary<string, Verzeichniseintrag>();
11
12         public Verzeichnis()
13         {
14             NeuerEintrag("Robert", "Schiefele", "08912356489",
15                 "01715589665", "robert.schiefele@email.de");
16             NeuerEintrag("Katia", "Japa", "0897889445",
17                 "017259894", "katia.japa@email.de");
18             NeuerEintrag("Theo", "Tester", "089885595445",
19                 "017365894", "theo.testers@email.de");
20         }
21
22         public void NeuerEintrag(string vorname, string
23             nachname, string festnets, string mobilfunk, string email)
24         {
25             verzeichnis.Add($"{vorname}:{nachname}", new
26                 Verzeichniseintrag(vorname, nachname, festnets,
27                     mobilfunk, email));
28         }
29
30         public void ZeigeAlleEintraege()
31         {
32             foreach(var eintrag in verzeichnis.Values)
33             {
34                 eintrag.ZeigeEintrag();
35             }
36         }
37
38         public void ZeigeEintrag(string schluessel)
39         {
40             if(verzeichnis.ContainsKey(schluessel))
41             {
42                 verzeichnis[schluessel].ZeigeEintrag();
43             }
44             else
45             {
46                 Console.WriteLine($"Eintrag: {schluessel} nicht
47                     gefunden.");
48             }
49         }
50
51         public void ZeigeEintraegeFuerNachname(string nachname)
52         {
53             var gefunden = false;
```

```

55         foreach(var element in verzeichnis)
56         {
57             if(element.Value.Nachname == nachname)
58             {
59                 element.Value.ZeigeEintrag();
60                 gefunden = true;
61             }
62         }
63
64         if(!gefunden)
65         {
66             Console.WriteLine($"Keinen Eintrag für {nachname}
67             gefunden");
68         }
69     }
70
71     public void ZeigeEintraegeFuerVorname(string vorname)
72     {
73         var gefunden = false;
74
75         foreach (var element in verzeichnis)
76         {
77             if (element.Value.Vorname == vorname)
78             {
79                 element.Value.ZeigeEintrag();
80                 gefunden = true;
81             }
82         }
83
84         if (!gefunden)
85         {
86             Console.WriteLine($"Keinen Eintrag für {vorname}
87             gefunden");
88         }
89     }
90
91     public void AktualisiereEintrag(string schluessel,
92     string festnets, string mobilfunk, string email)
93     {
94         if(verzeichnis.ContainsKey(schluessel))
95         {
96             verzeichnis[schluessel].Festnets = festnets;
97             verzeichnis[schluessel].Mobilfunk = mobilfunk;
98             verzeichnis[schluessel].Email = email;
99         }
100         else
101         {
102             Console.WriteLine($"Eintrag: {schluessel} nicht
103             gefunden.");
104         }
105     }
106
107     public void LoescheEintrag(string schluessel)
108     {
109         if(verzeichnis.ContainsKey(schluessel))
110         {

```

```

111             verzeichnis.Remove(schluessel);
112         }
113         else
114         {
115             Console.WriteLine($"Eintrag: {schluessel} nicht
116             gefunden.");
117         }
118     }
119 }
120 }

```

### Teilaufgabe 5:

Schreiben Sie die Klasse `VerzeichnisApp`. Die Klasse besitzt die private-Membervariable `verzeichnis` von Typ `Verzeichnis`. Im parameterlosen public-Konstruktor der Klasse `VerzeichnisApp` weisen Sie der Membervariablen `verzeichnis` eine neue Instanz der Klasse `Verzeichnis` zu.

Hochschule Aalen info@hs-aalen.de

9 Grundlagen der Objektorientierten Programmierung

180

Des Weiteren besitzt die Klasse `VerzeichnisApp` die Methode `ZeigeMenu()`, welches den Bildschirm löscht und das folgende Menü am Bildschirm ausgibt:

1. Alle Einträge anzeigen.
2. Einen bestimmten Eintrag anzeigen
3. Einträge suchen nach Vornamen.
4. Einträge suchen nach Nachnamen.
5. Eintrag hinzufügen.
6. Eintrag ändern.
7. Eintrag löschen
8. Programm beenden.

-----  
Wählen Sie einen Menüpunkt (1-8)

Das Löschen des Bildschirms können Sie mit der Methode `Console.Clear()` erledigen.

Schreiben Sie die private-Methode `LeseMenuPunkt()`, welche eine Eingabe von der Tastatur einliest und einen Integer-Wert zwischen eins und acht für den entsprechenden Menüpunkt zurückgibt. Die Methode soll den Benutzer wiederholt zur Eingabe einer Zahl zwischen eins und acht auffordern, solange bis der Benutzer eine gültige Eingabe macht.

Schreiben Sie die public-Methode `Start`, welche in einer Schleife erst das Programm-Menü anzeigt und dann mit Hilfe der Methode `LeseMenuPunkt()` einen Menüpunkt von der Tastatur einliest und dann den Menüpunkt ausführt. Implementieren Sie in diesem Schritt nur die Ausführung des Menüpunkts acht: Programm beenden.

Bei Eingabe anderer Menüpunkte soll die Schleife wieder mit dem Aufruf des Programm-Menüs beginnen. Rufen Sie die Methode `Start` in der Hauptmethode der Klasse `Program` auf.