

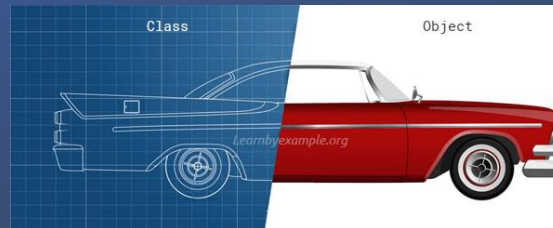


Software Development

Quellenangaben:

- Jürgen Kotz, München, Visual C# 2019, Carl Hanser Verlag München
- Michael Bonacina, C# Programmieren für Einsteiger (2. Aufl.), BMU Verlag
- <https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide>

Grundlagen OOP



Grundlagen OOP

- Klassen (Objektyp) – Objekte (Instanzen)
- Methoden
- Konstruktoren

Grundlagen OOP

- Reale Welt abbilden mit Hilfe von Objekten
- Klassen -> Abbildung der Struktur von Objekten
 - Attribute -> Eigenschaften eines Objekt (z.B. Farbe)
 - Methoden -> Funktionen eines Objekt (z.B. Berechne())
 - Events -> Ereignisse eines Objekt (z.B. Button_Click())
- Instanz -> konkretes Objekt
 - Es können viele Objekte mit einer Klassen erzeugt werden.
- Kapselung -> Saubere Trennung von öffentlichen und internen Daten und Methoden
- Wiederverwendbarkeit
 - Klasse kann in der Applikation an verschiedenen Orten verwendet werden

Klassen (Objekttyp)

- Klassen -> Abbildung der Struktur von Objekten
- Attribute -> Eigenschaften eines Objekt (z.B. Farbe)
- Methoden -> Funktionen eines Objekt (z.B. Berechne())

Student
+ Name
+ Vorname
+ Einschreiben()



Klassenname



Attribute (bestimmten Datentyp)



Methoden (Möglichkeit der Parameterübergabe)

Objekte (Instanzen)

- Instanz -> konkretes Objekt
 - Es können viele Objekte mit einer Klassen erzeugt werden.

111: Student	
Name = Harrelson Vorname = Woody	112: Student
	Name = Murray Vorname = Bill

Klassen (Objektyp) – Objekte (Instanzen)


Definition einer Klasse

C #

```
class Student
{
    string Name { get; set; }
    string Vorname { get; set; }

    void Einschreiben()
    {
        // Code .....
    }
}
```

alternative
Schreibweise
jedoch
fehleranfällig



C #

```
class Student
{
    public string name;
    public string vorname;

    public void Einschreiben()
    {
        // Code....
    }
}
```

Erstellung einer Instanz

C #

```
Student einStudent = new Student();

einStudent.Name = "Murray";
einStudent.Vorname = "Bill";

Console.WriteLine($"Hallo {einStudent.Vorname} {einStudent.Name}");
```


Übungen

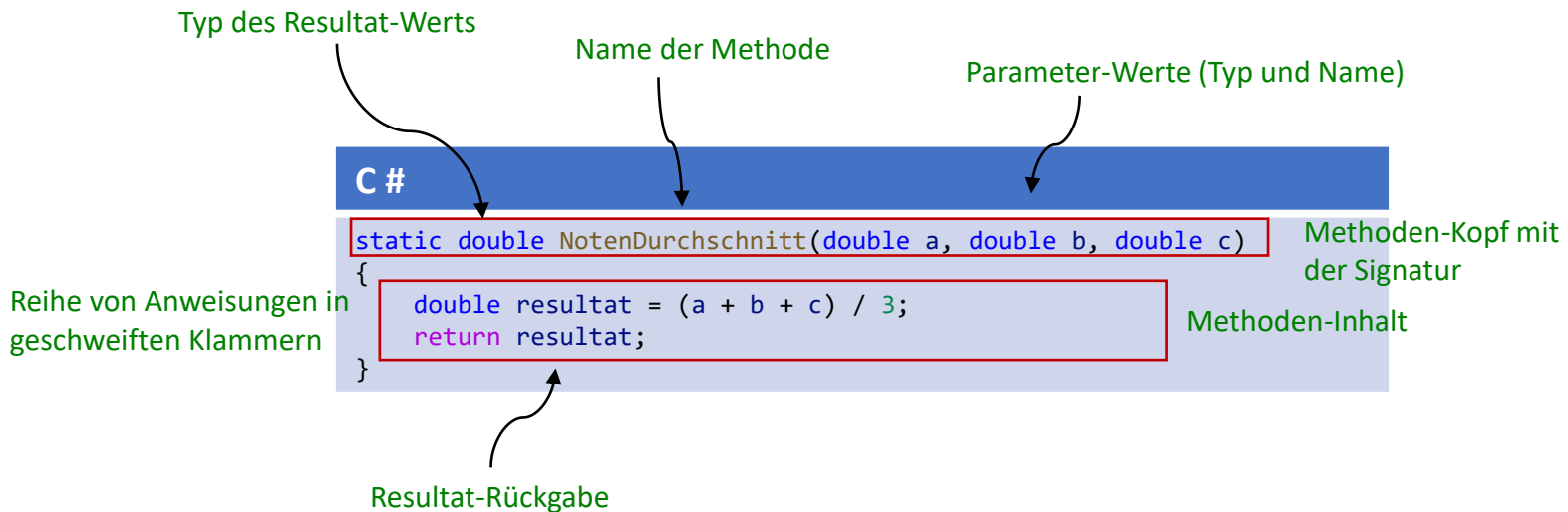
Uebungsaufgaben 8.4 s.102

Methoden

- Sind Subroutinen einer Klasse
- können Werte an den Aufrufer zurückgeben
- Verändern die Eigenschaften eines Objektes
- fassen Code zusammen die an verschiedenen Stellen im Programm aufgerufen werden können
- sind eigenständige Codeabschnitte

Methoden

Definition



Methoden

Methode "Main"

- Einstiegspunkt für Konsoleanwendung
- void Main: gibt keinen Wert an den Aufrufer zurück
- void Main(string[] args): Der Methode werden die Befehlszeilenargumente übergeben. Die Parameterliste für eine Methode beginnt und endet mit den runden Klammern
- Der zu der Methode gehörende Codeabschnitt beginnt und endet mit den geschweiften Klammern

C #

```
static void Main(string[] args)
{
    Student einStudent = new Student();

    einStudent.Name = "Murray";
    einStudent.Vorname = "Bill";

    Console.WriteLine($"Hallo {einStudent.Vorname} {einStudent.Name}");
}
```

Methoden

Statische Methoden

- werden mit dem Schlüsselwort `static` gekennzeichnet
- werden als Klassenmethoden bezeichnet
- werden unabhängig von einem Objekt genutzt
- können nur statische Variablen und Methoden aufrufen
- Bequemer und ressourcenschonender da keine Instanzierung nötig

C #

```
public class MeineFormeln
{
    public static double KreisUmfang(double radius)
    {
        return (2 * Math.PI * radius);
    }
}
```

Methoden

Benutzerdefinierte Methoden

- Methodenkopf: definiert den Aufruf einer Methode. Bietet eine Schnittstelle zu den Nutzern
- Methodenrumpf: Wird durch die geschweiften Klammern begrenzt. Innerhalb der Klammern stehen die zur Methode gehörenden Anweisungen

C #

```
static string Eingabe()
{
    string resultat;

    Console.WriteLine("Bitte geben Sie ihren Namen ein: ");
    resultat = Console.ReadLine();
    return resultat;
}

static void Ausgabe(string meldung)
{
    Console.WriteLine(meldung);
}
```

Methoden

Methodenkopf

```
void      Ausgabe(string meldung)  
[Datentyp] [Name] (parameter1, parameter2, ...)
```

- Der Datentyp legt den Rückgabebetyp fest. Der Typ void kennzeichnet eine Methode ohne Rückgabewert
- Der Name der Methode ist frei wählbar
 - wird aus den Zeichen A..Z, a..z, 0..9 und dem Unterstrich gebildet
 - sollte immer mit einem Buchstaben oder Unterstrich beginnen
 - beginnt meistens mit einem Grossbuchstaben
 - kommt einmal in einer Klasse vor
 - sollte selbstsprechend sein.
- In den runden Klammern wird der Methode Parameter, getrennt durch Komma, übergeben. Es können 0:n Parameter übergeben werden.

Methoden

Rückgabewert

C

```
static string Eingabe()
{
    string resultat;

    Console.WriteLine("Bitte geben Sie ihren Namen ein: ");
    resultat = Console.ReadLine();    → Erwartet Benutzer-Eingabe von der Konsole
    return resultat;
}
```

- Mit Hilfe des Schlüsselwortes return wird ein Wert an den Aufrufer zurückgegeben
- Der Datentyp des Rückgabewertes entspricht dem Datentyp der Methode.
- Ab C# 7 können mehrere Werte zurückgegeben werden.

C # Version > 7

```
static (string, int, string) EingabeNeu()
{
    string wert1 = "Wert1";
    int zahl1 = 234;
    string wert2 = "Wert2";

    return (wert1, zahl1, wert2);
}
```


Methoden

Methoden aufrufen

C #

```
static void Main(string[] args)
{
    string name = Eingabe();
    Ausgabe($"Ihr Name ist: {name}");
}
```

- Methoden werden immer mit ihrem Namen aufgerufen
- Falls die Parameterliste nicht leer ist, werden in den runden Klammern die Werte für die Parameter übergeben
- Falls die Methode einen Rückgabewert hat, kann dieser in einer Variablen gespeichert werden

Methoden

Parameter

- Sind Platzhalter für Werte im Speicher
- werden vom Aufrufer an die Methode übergeben
- bekommen einen Wert oder eine Speicheradresse (out, ref) zugewiesen
- werden in Abhängigkeit ihrer Position oder des Namens übergeben

Methoden

Wertparameter

- Das Argument übergibt einen Wert an den Parameter
- Der Parameter ist eine Kopie des Arguments
- Das Argument kann nicht durch die aufgerufene Methode verändert werden

```
static float Addieren(float Wert1, float Wert2)
```

```
float zahl = Addieren(34, 50);
```

Methoden

Referenzparameter (call by reference)

- Verweis auf eine bestimmte Speicherstelle
- Parameter und Argument zeigen auf die gleiche Stelle
- Innerhalb der aufgerufenen Methode kann der Wert des Arguments verändert werden

```
static void AddKunde(Kunde kunde)
```

Methoden

Schlüsselwort ref

- Das Argument muss vor der Nutzung initialisiert werden
- Im Methodenkopf und im Aufruf werden die Variablen mit dem Schlüsselwort gekennzeichnet
- Ist nur für Wertetypen relevant

```
static void Ausgabe(ref string meldung)
```

```
string meldung = "meine Meldung"; //muss initialisiert sein.  
Ausgabe(ref meldung);
```

Methoden

Schlüsselwort out

- identisch mit ref
- Ausnahme: Methode **kann** einem ref-Parameter einen Wert zuweisen, einem out-Parameter **muss** einen Wert zugewiesen werden

```
static void Ausgabe(out string meldung)
```

```
string meldung;  
Ausgabe(out meldung);
```

Methoden

Optionale Parameter

- optionaler Parameter hat einen Standardwert
- Aufruf der Methode kann Standardwert überschrieben werden

C #

```
private static string MethodeOptionalerParameter(string name = "Du")  
{  
    return ("Hallo {name}");  
}
```

Methoden

Benannte Parameter

- Reihenfolge der Parameter beim Aufruf frei wählbar
- Parameter tragen beim Aufruf denselben Namen wie in der Methodendeklaration, wobei ein Doppelpunkt angehängt wird

C #

```
private static void AddBuch(string titel, string autor = "", DateTime? termin = null)
{
    // noch zu implementieren
}

// Aufruf
AddBuch(autor: "Douglas Adams", termin: DateTime.Now, titel: "The Hitchhiker's Guide to the Galaxy");
```




Übungen

Uebungsaufgaben 9.7 s.121

Konstruktoren

- Standardmässig wird geerbter System.Object new-Standardkonstruktor verwendet.
- Ist eine Art Standardmethode der Klasse mit identischem Namen wie die Klasse
- Wird automatisch bei der Instanziierung eines Objektes (new) aufgerufen
- Dient vor allem dazu, den Feldern des neu erzeugten Objektes Anfangswerte zuzuweisen

Konstrukturen

Deklaration

C#

```
public KlasseName(Datentyp parameter)
{
    // Initialisierung der Klasse
}
```

Wie bei jeder Methode kannst du auch mehrere überladene Konstrukturen implementieren

C#

```
public class Kunde
{
    // Felder
    public string Anrede { get; set; }
    public string Name { get; set; }
    public int Plz { get; set; }
    public string Ort { get; set; }
    public bool Stammkunde { get; set; }
    public decimal Guthaben { get; set; }
}
```

...

Konstrukturen

Deklaration

C#

...

// Der erste Konstruktor initialisiert nur zwei Felder

```
public Kunde(string anrede, string name)
{
    Anrede = anrede;
    Name = name;
}
```

// Der zweite Konstruktor initialisiert alle Felder

```
public Kunde(string a, string n, int p, string o, bool s, decimal g)
{
    Anrede = a;
    Name = n;
    Plz = p;
    Ort = o;
    Stammkunde = s;
    Guthaben = g;
}
```

```
}
```

Konstrukturen

Aufruf

- Nach Deklaration eines oder mehrere Konstrukturen muss auch mindestens einer verwendet werden
- Bisherige einfache Instanziierung ist nicht mehr möglich wenn kein Standard-Konstruktor (Konstruktor ohne Parameter) definiert ist

C

```
Kunde kunde1;  
Kunde kunde2;  
Kunde kunde3;  
  
kunde1 = new Kunde("Herr", "Murray");  
kunde2 = new Kunde("Frau", "Portman", 9000, "St.Gallen", true, 200000);  
// kunde3 = new Kunde(); // Erzeugt Compilerfehler!!!
```


Konstrukturen

Statischer Konstruktor

- wird verwendet um `static`-Felder zu initialisieren oder
- um einmaligen Initialisierungscode auszuführen
- Aufruf erfolgt automatisch, bevor die erste Instanz erstellt oder auf statische Klassenmitglieder verwiesen wird

C#

```
public class Allerlei
{
    private static double _mwst;

    public static double Mwst
    {
        get { return _mwst; }
        set { _mwst = value; }
    }

    static Allerlei() //statischer Konstruktor
    {
        Mwst = 7.7;
    }
}
```