C#.NET - Kurzreferenz

Basis-Syntax

Listen

Arrays

```
// Eindimensionl
int[] zahlen1 = new int[3];
zahlen1[0] = 1;
zahlen1[1] = 2;

foreach (int zahl in zahlen1) {
  Console.WriteLine(zahl + " | ");
}

int[] zahlen3 = new int[3] { 1, 2, 3 };
int[] zahlen4 = { 1, 2, 3 };

// Mehrdimensional
  // Rechteckig
int[,] tabelle1 = new int[2, 3];
tabelle1[0, 0] = 1;
tabelle1[0, 1] = 2; // ...

int[,] tabelle2 = { { 1, 2, 3 }, { 4, 5, 6 } };

  // Ungleichförmig
int[][] metaTabelle1 = new int[5][];
for (int i = 0; i < 5; i++) {
 metaTabelle1[i] = new int[3];
}
metaTabelle1[0][0] = 1;
metaTabelle1[0][1] = 2; // ...
```

Kontrollanweisungen

Zusätzliche Steuerung

- break beendet die Ausführung der Schleife
- continue überspringt die nachfolgenden Anweisungen und springt zum nächsten Durchlauf

```
for (int i = 0; i < 10; i++){
 if (i == 2) continue;
 if (i % 2 == 0 || i == 5) {
  Console.WriteLine (i);
 }
 if (i > 8) break;
}
```

Schleifen

```
int zahl = 1;
while (zahl <= 5){
 Console.WriteLine(zahl);
 zahl++;
}

for (int i = 0; i < 10; i++){
 Console.WriteLine (i);
}


int zahl = 10;
do {
 Console.WriteLine (i);
 zahl++;
}
while (zahl <= 5);
```

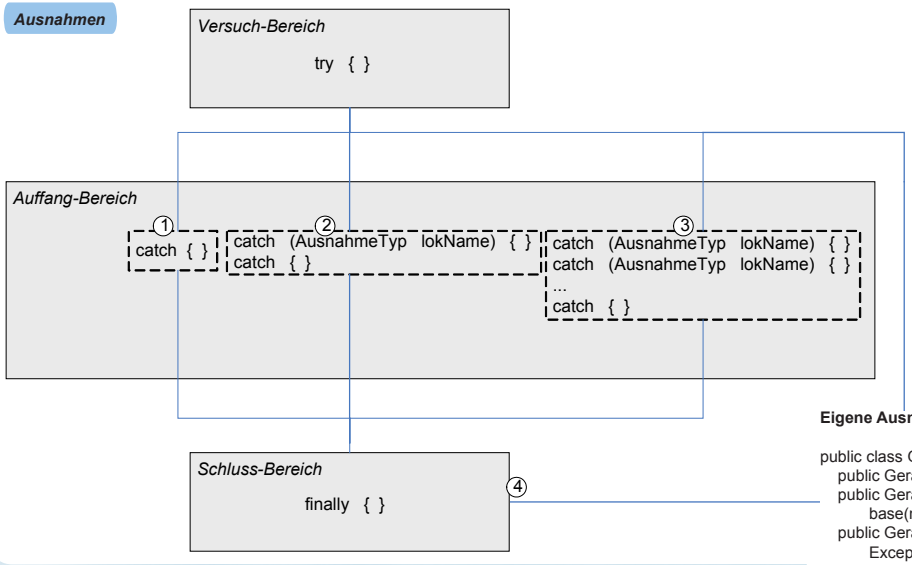
Fallunterscheidungen

```
if (ausdruck)
   // einzigeAnweisungInNaechsterZeile

int zahl1 = 5;
int zahl2 = 2;
if  (zahl1 == 4){
   // Anweisungen
}
else if (zahl1 >= 4 && zahl2 < 3){
  // Anweisungen
}
else {
 // Anweisungen
}

Enum Tier { Hund, Katze }
switch (tier) {
  case Tier.Hund:
  // Anweisungen
  break;
  case Tier.Katze:
  // Anweisungen
  break;
  default:
  // Standard-Anweisungen
  break;
}
```

Enum

```
public enum Steuersaetze : long { erm = 7, norm = 19 };
public enum Monate { Jan, Feb, Mrz, Apr }; // Start: 0
public enum Tage { Mo = 1, Di, Mit }; // Start: 1
```

Ausnahmen



Eigene Ausnahme

```
public class GeradeException : ApplicationException {
   public GeradeException() { }
   public GeradeException(string nachricht) :
         base(nachricht) { }
   public GeradeException(string nachricht,
         Exception innere) : base(nachricht, innere) { }
}
```

Klassen und ihre Elemente

Klassen

Mögliche Member einer Klasse: Konstruktoren, Destruktoren, Konstanten, Felder, Methoden, Eigenschaften, Indexer, Operatoren, Ereignisse, Delegaten, Klassen, Schnittstellen, Strukturen

```
class Person {
 // Felder
 private String vorname;
 private String nachname;
 private int nr;
 // Statisches Zähler-Feld
 private static int anzahl;
 // Eigenschaften
 public String Vorname {
  get { return vorname; }
  set { vorname = value; }
 }
 public String Nachname {
  get { return nachname; }
  set { nachname = value; }
 }
 public int Nr {
    get { return nr; }
    private set { nr = value; }
 }
 // Statische Eigenschaft
 public static int Anzahl {
  get { return Person.anzahl; }
 }
 // Konstruktor
 public Person (String v, String n){
   this.Vorname = v;
   this.Nachname = n;
   Person.anzahl++;
   this.Nr = Person.anzahl;
 }
 // Destruktor
 ~Person() {
      Person.anzahl--;
 }
 // Methoden
 public void SagHallo(){
    Console.WriteLine("Hallo, ich bin " + this.Vorname);
 }
}
```

Objekte

```
namespace Anwendung {
 class Program {
   static void Main(string[] args) {
     Person anton = new Person("Anton", "Ebenhof");
     anton.SagHallo();
     Console.WriteLine("Antons Nr.: " + anton.Nr);
     Console.WriteLine("Insgesamt: " + Person.Anzahl);
     Console.ReadLine();
   }
 }
 class Person {
   // ...
 }
}
```

Klassen-Hierarchien

Vererbung

```
public class Person {
   protected int status;
}

public class Benutzer : Person {
   private String name;
   private String passwort;

   public Benutzer(String name, String passwort) {
      this.name = name;
      this.passwort = passwort;
      this.status = 1;
   }
}
```

Abstrakte Klassen

```
abstract public class Person {
 // Felder
 public int status;
 // Abstrakte Methode
 abstract public String Begruessen();
 public String Verabschieden() {
    return "Auf Wiedersehen.";
 }
}

public class Gast : Person {
 // Konstruktor
 public Gast() {
   this.status = 0;
 }
 // Implementierte abstrakte Methode
 public override String Begruessen() {
   return "Hallo, Gast.";
 }
}
```

Verbot von Vererbung

```
public sealed class Antiquitaet {
   // Felder .. Eigenschaften ..
Methoden ..
}

public class Kunstgegenstand {
   // Felder .. Eigenschaften ..
Methoden ..
   // Versiegelte Methode
   public sealed override string
LiefereTitel(){
      return name + " " + Alter;
   }
}
```

Schnittstellen

```
public interface IKoerper {
   // Eigenschaften
   int Laenge { get; }
   int Breite { get; }
   // Methoden
   double ErrechneFlaeche();
}

public class Quadrat : IKoerper {
   // Felder
   private int _laenge;
   // Eigenschaften
   public int Laenge {
      get { return _laenge; }
   }
   public int Breite {
      get { return _laenge; }
   }
   // Konstruktor
   public Quadrat(int Laenge) {
      _laenge = Laenge;
   }
   // Methode
   public double ErrechneFlaeche() {
      return Laenge * Laenge;
   }
}
```

Methoden

Basisklassen-Methoden

Kind-Klassen rufen mit this.methode() eine lokale Methode auf, mit base.methode() eine Methode der Elternklasse.

```
public class Rechteck : Koerper {
   // Konstruktor, Methoden...
   public void Skalieren(int faktor) {
      laenge *= faktor;
      breite *= faktor;
   }
   public void Verdoppeln() {
      base.skalieren();
   }
}
```

Verdeckung

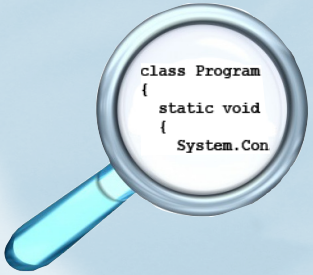
Implementierung einer Methode mit gleichem Namen, gleichem Rückgabedatentyp und gleicher Parameterliste in der Kindklasse, wobei allerdings die Anweisungen neu erstellt werden. Möglich für Methoden, Eigenschaften, Felder und statische Komponenten.

```
public class Koerper {
   protected int laenge = 0, breite = 0, hoehe = 0;
   // Verdeckte Methode
   public double ErrechneFlaeche() { return 0; }
   public Koerper() { }
}

public class Quadrat : Koerper {
   public Quadrat(int laenge) {
     this.laenge = laenge;
   }
   // Verdeckende Methode
   public new double ErrechneFlaeche() {
     return laenge * laenge;
   }
}
```

Überladung

Methodenaufruf mit unterschiedlichen Kombinationen von Parametern hinsichtlich Anzahl und Typ. Möglich innerhalb der gleichen Klasse und bei vererbten Eltern-Methoden und auch beim Konstruktor.

```
public class Koerper {
 // Felder
 private int laenge = 0;
 private int breite = 0;

 // Eigenschaften
 public double Flaeche {
   get{ //Angepasste Flächenberechnung }
 }

 // Überladene Konstruktoren
 // Quadrat
 public Koerper(int laenge) {
    this.laenge = laenge;
 }
 // Rechteck
 public Koerper(int laenge, int breite){
    this.laenge = laenge;
    this.breite = breite;
 }
}
```



Weitere OOP-Techniken

Operator-Überladung

Überladung von ein- und zweiseitigen Operatoren

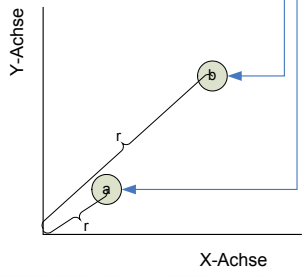
```
public static Rückgabe-Datentyp
     operator einseitiger-Operator (Typ Parametername) { }
public static Rückgabe-Datentyp
     operator zweiseitiger Operator (
                Typ Parametername,
                Typ2 Parametername2
               ) { }
```

Überladung von expliziter und impliziter Typ-Umwandlung

```
public static implicit
     operator Zieldatentyp ( Quelldatentyp  operand ) { }
public static explicit
     operator Zieldatentyp  ( Quelldatentyp operand ) { }
```

```
public class Punkt  {
  public int x, y;
  public readonly double r;
  // Konstruktor...
  public static bool operator <= (Punkt a, Punkt b)  {
    if (a.r <= b.r) { return true; }
    else { return false; }
  }
  public static bool operator >= (Punkt a, Punkt b) {
    if (a.r >= b.r) { return true; }
    else { return false; }
  }
}
```

```
// Punkte erzeugen
Punkt a = new Punkt(1, 2);
Punkt b = new Punkt(4, 8);
// Vergleichen
if (a <= b) { Console.WriteLine("a kleiner b"); }
```



Strukturen

```
public struct MyStruct {
  // Felder, Eigenschaften
  // Methoden, Ereignisse
}

// Struktur Punkt
public struct Punkt {
 // Koordinaten
 public int x, y;
 // Konstruktor
 public Punkt(int x, int y) {
   this.x = x; this.y = y;
 }
}
```

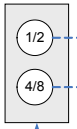
Namensräume

```
namespace name[.name1] ...] {
  // namespace
  // class, interface, struct
  // enum
  // delegate
}

using alias = typ|namensraum;
using namensraum;
```

Indexer

```
public int this [int index]  {
  get    { return Array-Feld[index];  }
  set    {  Array-Feld[index] = value;  }
}
```

```
public class Punkt {
   public int x, y;
   ArrayList punkte = new ArrayList();
   public Punkt() { }
   public Punkt(int x, int y) {
      this.x = x;
      this.y = y;
   }
   public Punkt this[int index] {
      get {
         return (Punkt)punkte[index];
      }
      set {
         punkte.Add(value);
      }
   }
}
```




```
// Punkte erstellen
Punkt a = new Punkt(1, 2);
Punkt b = new Punkt(4, 8);

Punkt gerade = new Punkt();
gerade[0] = a;
gerade[1] = b;
```

Delegates

```
// HTML-Text-Feld
public String html = "";

//Delegat
public delegate Koerper KoerperErzeugung(int[] seiten);

// Konstruktor
public KoerperFabrik()
{
   // Variable für Delegat
   KoerperErzeugung aktion;

   // Zufällige Auswahl von Aktionen
   Random zufall = new Random();
   for (int j = 0; j < 5; j++)
   {
      int zahl = zufall.Next(1, 4);
      switch (zahl)
      {
         case 1: aktion = new KoerperErzeugung(ErzeugeQuader);
                 break;
         case 2: aktion = new KoerperErzeugung(ErzeugeQuadrat);
                 break;
         default: aktion = new KoerperErzeugung(ErzeugeRechteck);
                 break;
      }
      // Ausführen einer Aktion
      Koerper koerper = aktion(new int[] { 2, 3, 5 });
      // Ausgabe von Ergebnissen
      html += koerper.GetType() + " Flaeche: "
              + koerper.ErrechneFlaeche() + "<br/>";
   }

}
```

Delegat-Typ

Delegat-Variable

Zufällige Bindung

Konkrete Methoden

Konkreter Aufruf der zufälligen Bindung

```
// Operationen
public Quader ErzeugeQuader(int[] seiten)
{
   return new Quader(seiten[0], seiten[1], seiten[2]);
}

public Quadrat ErzeugeQuadrat(int[] seiten)
{
   return new Quadrat(seiten[0]);
}

public Rechteck ErzeugeRechteck(int[] seiten)
{
   return new Rechteck(seiten[0], seiten[1]);
}
```

Zusammengestellt von Marco Skulschus
Satz und Layout Beatrice Walendy
© 2009 Comelio Medien

Comelio GmbH
Goethestraße 34, 13086 Berlin
Web: www.comelio.com

Terrashop GmbH
Lise-Meitner-Str. 8, 53332 Bornheim
Web: www.terrashop.de

ISBN 978-3-939701-30-9
9 783939 701309