



Property



Method or Function



Event **Vorschlagfunktion:** IntelliSense

Clean Code:

- Bei Datentypen Alias verwenden
- aussagekräftige Namen = Umsatz, Vorname... nicht strName, Res1... (**Ausnahme bei Schleifen i, j etc.**)
- Einheitssprache wählen (de, en...) und aufpassen wegen reservierten Wörtern
- Erste Zeichen nie eine Zahl

Methoden: Methodenname Pascal Case und Verb(Objekt) verwenden, nicht zu viele Parameter (7), Variablen in Methoden erst definieren wenn sie benötigt werden, 1 Methode = 1 Logik, wenn möglich keine Referenz als Parameter übergeben.

Klassen: Klassenname Pascal Case und Nomen verwenden, eine Klasse pro Datei,

Notationen

Camel Case (lokale Variablen innerhalb einer Methode, formaler Parameter einer Methode): Erster Buchstabe klein alle anderen Anfangsbuchstaben gross z.B. backColor.

Pascal Case (alle anderen): Jedes Wort beginnt mit einem Grossbuchstaben z.B. BackColor

GROSSSCHREIBUNG (Konstante):

Ungarische Notation: [Präfix][Datentyp]

Methoden (oder auch Funktion, Routine, Prozedur):

```
Zugriffstyp Rückgabety Methodennamen (Parameterliste)
{
    Methodenrumpf
}
```

Vordefinierter Ablauf

Variablen, welche im Methodenrumpf definiert wurden können nur in der Methode genutzt werden (übergeordnete können auch genutzt werden)!

Klassen (Konstrukt für Methoden):

Erstellen: class NAME { //Rumpf }

Variablendeklaration: NAMEKLASSE NAMEVARIABLE;

Instanziierung: NAMEVARIABLE = new NAMEKLASSE();

Nutzung: NAMEVARIABLE.METHODENNAME = oder (PARAMETER);

Modularisierung (Methoden in Klassen): bessere Übersicht, wiederverwendbar, Kapselung -> sicherer, bessere Wartbarkeit

Array Liste (es können ganz unterschiedliche Datentypen eingebunden werden):

Instanziierung: ArrayList NAME = new ArrayList();

Element hinzufügen: NAME.Add(Variable oder Wert);

Foreach: foreach (DATENTYP IETWAS in NAMELISTE) { Console.WriteLine(obj); }

Generische Listen (es wird ein Datentyp vorgegeben es gehen auch Klassen):

Definition: List<DATENTYP> NAME = new List<DATENTYP>();

Element hinzufügen: NAME.Add(Variable oder Wert);

Datenkapselung (Variablen in Klasse (werden Eigenschaften genannt)):

```
private string _Anrede;
1-Verweis
public string Anrede
{
    get { return _Anrede; }
    set { _Anrede = value; }
}
```

Get: Wird bei der Auswertung der Eigenschaft ausgeführt
Set: Wenn der Eigenschaft ein Wert zugewiesen werden soll
value: Entspricht dem Datentyp der Eigenschaft

Klassen UML

Methoden: Meistens public selten aber auch private als Werkzeugkasten, ermöglicht einen sicheren Ablauf, weil dem Benutzer Grenzen gesetzt werden und er nur die vorgegebenen Sachen ausführen kann.

Eigenschaften: Beinahe immer Private, damit es über eine Methode laufen muss

Beispiel: Beim Linken könnten z.B. alle auf einmal eingeschaltet werden, jedoch auf der rechten Seite gibt es nur zwei Methoden, welche auf die Eigenschaften zugreifen können

Speicher:

Stack: Auflistung wie z.B. primitive Datentypen -> Speicherbedarf bekannt und fixe Länge

Heap: fließende Aufteilung, Objekte können wachsen und schrumpfen wie z.B. Klassen (mit Methoden / Eigenschaften)

Klasse: Deklaration in Stack und Instanziierung mit „new“ in Heap

Name	Alias	Typ
Boolean	bool	W
Ganzzahlen		
Byte	byte	W
Int16	short	W
Int32	int	W
Int64	long	W
Fliesskommazahlen		
Single	float	W
Double	double	W
Zeichen		
Char	char	W
String	string	V
Objekte		
Object	object	V

W = Werttyp V = Verweistyp

Zugriffstyp	+ Public (von aussen verwendbar), - Private (nur innerhalb der Klasse zugreifbar), ~ Protected (innerhalb Klasse + vererbte), (static, ...)
Rückgabety	Datentyp des Rückgabewertes, falls kein Rückgabewert vorhanden, wird void eingesetzt
Methodennamen	Name der Methode in Pascal Case
Parameterliste	Die Liste kann auch leer sein! Ein Parameter besteht aus einem Namen und einem Datentyp. Mehrere Parameter werden durch Komma getrennt.
Methodenrumpf	Block bzw. Liste von Anweisungen welche von oben nach unten durchlaufen werden. Der Rumpf wird durch die geschweiften Klammer { } umgeben.

1. Variablendeklaration
2. Instanziierung
3. Nutzung

```
Gast meinGast;
meinGast = new Gast();
meinGast.Anrede = "Frau";
```

Klasse als Methodenbibliothek

```
int Methode1(int x)
string Methode2(int x)
string Methode2(string x)
```

Eigenschaft	Array	List
Startgrösse	Anzahl Elemente muss bekannt sein. (Kann aber nachträglich mit Performanceeinsparungen geändert werden)	Muss nicht festgelegt werden, passt sich automatisch an
Elemente	Beliebige Elemente (primitive Datentypen, Objekte)	Beliebige Elemente
Zugriff auf Element	Über Nummer (Index)	Über Nummer (Index)
Einfügen von Elementen	Über Nummer, es der Platz auf dieser Nummer definiert. (Die Grösse des Arrays bleibt gleich)	Über Nummer, es wird ein neues Element auf diese Position eingefügt. (Die Liste wird länger)
Löschen von Elementen	Über Nummer, der Platz an dieser Nummer wird leer	Über Nummer, die Liste wird um ein Element kürzer

```
switch(variable)
{
    case „Hallo“:
        Console.WriteLine(„“);
        break; //Muss immer geschrieben werden! Sonst meldet fehler
        default: //Optional
            Console.WriteLine(„“);
            break; }
```



Code Beispiele:

Spezielle Sachen:

```
Array: DATENTYP[] = NAME = new DATENTYP[ANZ];  
Potenz: Math.Pow(zahl1, zahl2);  
Wurzel: Math.Sqrt(zahl1);  
Umwandlung: Int VARIABLENNAME = int.Parse(Console.ReadLine());
```

Beispiel Generische Liste + Klasse

```
Public class Person {public string name; public string vorname;}  
List<Person> personen = new List<Person>();  
Person p = new Person();  
p.vorname = "Max";  
p.name = "Mustermann";  
personen.Add(p);  
foreach (Person per in personen) {Console.WriteLine("{0} {1}",  
per.vorname, per.name);
```

Anwendung Array

```
static void Main(string[] args)  
{  
    Console.WriteLine("Bitte Anzahl Gäste eingeben:");  
    int anzahlGaeeste;  
    anzahlGaeeste = int.Parse(Console.ReadLine());  
    // Array anlegen für Strings  
    string[] meineGaeeste = new string[anzahlGaeeste];  
    for (int i = 0; i < anzahlGaeeste; i++)  
    {  
        Console.WriteLine("Bitte Name für Gast Nr. " + i.ToString() + " angeben:");  
        // Zugriff mittels Index  
        meineGaeeste[i] = Console.ReadLine();  
    }  
    DruckeEinladung1(meineGaeeste);  
    Console.ReadLine();  
}  
  
//Drucker  
static void DruckeEinladung1(string[] gaeesteArray)  
{  
    for(int i=0; i < gaeesteArray.Length; i++)  
    {  
        Console.WriteLine("Hallo " + gaeesteArray[i]);  
    }  
}
```

Windows Form Anwendung (Graphical User Interface = GUI) in Public partial class Form1 : Form {

Deklaration:

```
Label titel2;  
TextBox eingabe2;  
Button knopf2;
```

Ereignis bei Klick definieren:

```
Private void knopf2_Click(object sender, EventArgs e)  
{ eingabe2.Text = "Vielen Dank für den Klick";}
```

ab hier in InitializeComponent -> FormDesigner:

Instanziierung:

```
titel2 = new Label();  
eingabe2 = new TextBox();  
knopf2 = new Button();
```

Label Einstellungen:

```
Text: titel2.Text = "Hallo";
```

```
Position: titel2.Location = new Point(150,20);
```

```
Grösse: titel2.Width = 150; or titel2.AutoSize = true;
```

TextBox Einstellungen:

```
Position: eingabe2.Location = new Point(150,40);
```

```
Grösse: eingabe2.Width = 150; or eingabe2.AutoSize = true;
```

Button Einstellungen:

```
Text: knopf2.Text = „Klicken“;
```

```
Position: knopf2.Location = new Point(150, 80);
```

```
Grösse: knopf2.Width = 150; or knopf2.AutoSize = true;
```

Ankleben auf das Fenster:

```
This.Controls.Add(titel2); //this steht hier für form1  
This.Controls.Add(eingabe2); //this steht hier für form1  
This.Controls.Add(knopf2); //this steht hier für form1
```

Weitere Sachen:

```
NAME.Height = ZAHL //höhe  
NAME.name = "name"; //Namen festlegen  
NAME.BackColor = Color.COLORNAME;
```

```
//Verknüpf  
static Single Addition(Single zahl1, Single zahl2)  
{  
    return zahl1 + zahl2;  
}
```

```
class MathOperationenMitZweiParametern {  
    public float Addition(float zahl1, float zahl2)  
    { return zahl1 + zahl2; }  
}
```

Damit es in der Klasse von aussen erreichbar ist muss es von static auf public geändert werden!

Klasse

Mirio Eggmann

```
namespace AB404_03_EinladungMitKlasse  
{  
    class Gast  
    {  
        private string _Anrede;  
        public string Anrede  
        {  
            get { return _Anrede; }  
            set { _Anrede = value; }  
        }  
    }  
}
```

In einer Klasse kann eine andere Klasse verwendet werden, indem man sie als Datentyp nimmt.
private Geschenk _Geschenke;

Klasse Nutzen

```
List<Gast> gaeeste = new List<Gast>();  
Gast meinGast = new Gast();  
meinGast.Anrede = „Herr“;  
meinGast.Geburtsdatum = new DateTime(2015, 5, 30);  
gaeeste.Add(meinGast);
```

FOREACH

```
GaeesteListe(gaeeste);  
private static void GaeesteListe(List<Gast> liste)  
{  
    foreach (Gast element in liste)  
    {  
        Console.WriteLine("Anrede: " + element.Anrede);  
        Console.WriteLine(element.Geburtsdatum.Date.Year);  
    }  
}
```

Anwendung Generische Liste

```
static void Main(string[] args)  
{  
    string nameGast;  
    int anzahlGast = 1;  
    List<string> gaeeste = new List<string>();  
    do  
    {  
        Console.WriteLine("Bitte Name für Gast Nr. " +  
anzahlGast + " angeben: ");  
        nameGast = Console.ReadLine();  
        gaeeste.Add(nameGast);  
        anzahlGast = anzahlGast + 1;  
    } while (nameGast != "");  
    DruckerEinladung1(gaeeste);  
    Console.ReadLine();  
}  
  
private static void DruckerEinladung1(List<string>  
gaesteliste)  
{  
    foreach (string element in gaesteliste)  
    {  
        if (element != "")  
        {  
            Console.WriteLine("Hallo " + element);  
        }  
    }  
}
```

Bsp: Kreisradius Klasse (Datenkapselung sicher vor Manipulation)

```
public class Kreis {  
    private double _Radius;  
    public double Radius {  
        get {  
            return _Radius;  
        }  
        set {  
            if (value >= 0)  
                _Radius = value;  
            else  
                Console.WriteLine("Unzulässiger negativer Wert");  
        }  
    }  
}
```