

DEPARTMENT WIRTSCHAFTSINFORMATIK

FACHBEREICH WIRTSCHAFTSWISSENSCHAFT

# Programmieren für Wirtschaftswissenschaftler

## SS 2015

Lucian Ionescu

Blockveranstaltung 16.03–27.3.2015

5. Arrays und Listen

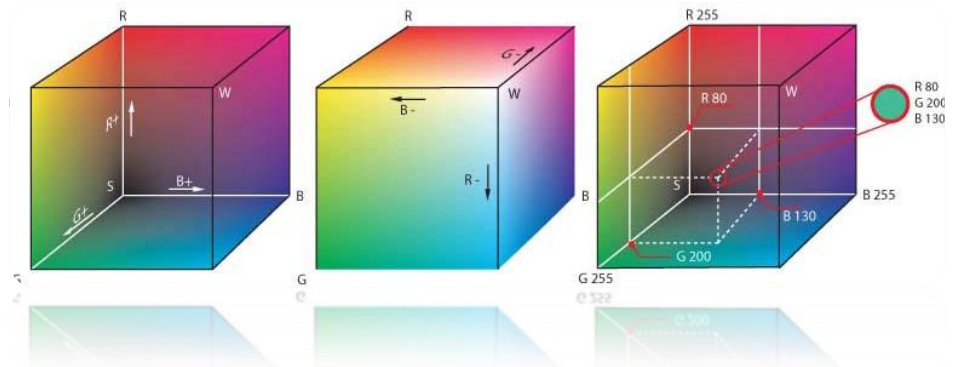
# Agenda

---

- **Arrays**
  - Motivation
  - Deklaration und Initialisierung
  - Zugriff
  - *Erweiterungen*
- Listen
- Weitere komplexe Datentypen

# Was ist ein Array?

- Ausgangsfrage
  - Wie verarbeitet man eine strukturierte große Anzahl **gleichartiger Daten**?
  - Beispiel
    - Pixel eines Bildes (1024x768): 786.432 Bildpunkte mit Farbwerten (z.B. RGB-Farben)
    - **3 · 786.432** gleichartige Variablen notwendig (Rot-, Grün-, Blau-Wert pro Pixel)
    - Idee: Variablen per *Copy&Paste* erstellen
    - Was tun bei anderen Bildgrößen?



- Lösung: Arrays!
  - **Definition:** Eine Folge von Variablen desselben Typs, die sich denselben Bezeichner teilen, werden als Array bezeichnet.
  - **Synonym:** (Daten-)Feld

# Deklaration und Initialisierung

---

- Arrays müssen wie Variablen vor Ihrer Nutzung **deklariert** werden

```
<datatype>[] <identifier>;
```

Bsp.: 

```
int[] numbers;  
string[] names;
```

- Initialisierung** mit dem Schlüsselwort `new`

```
<identifier> = new <datatype>[<length>;
```

Bsp.: 

```
numbers = new int[10];  
names = new string[20];
```

- Die einzelnen Felder des Arrays werden als **Elemente** bezeichnet
- Die Elemente werden abhängig von ihrem Datentyp automatisch initialisiert mit
  - `0`
  - `""`
  - `false`

# Deklaration und Initialisierung (2)

---

- Deklaration und Initialisierung kann auch **gleichzeitig** geschehen

Bsp.: `<datatype>[] <identifier> = new <datatype>[<length>];`  
`int[] numbers = new int[10];`  
`string[] names = new string[20];`

- Für die Länge können auch **Variablen** eines Zahlentyps genutzt werden

Bsp.: `int i = 10, j = 20;`  
`int[] numbers = new int[i];`  
`string[] names = new string[j];`

# Initialisierung vom Array und Elementen

---

- Der Begriff Initialisierung ist bei Arrays zweideutig
  - **Initialisierung des Arrays**
  - **Initialisierung der Elemente** vom Array
- Die Deklaration ist nur für das Array notwendig
- Die Deklaration der Elemente ist implizit
- **Technische Details:**
  - Was passiert bei der Deklaration des Arrays?
  - Was passiert bei der Initialisierung des Arrays?
  - Was passiert bei der Initialisierung der Elemente von Arrays?
- **Achtung! Die Größe eines Arrays ist fest**
  - Frage: Größe von Array verändern?

# Zugriff

---

- Zugriff geschieht über einen **Index**

- Syntax: **Schreibzugriff**

```
<identifizier>[<index>] = <value>;  
numbers[5] = 10;
```

- Syntax: **Lesezugriff**

```
<identifizier>[<index>];  
int i = 5;  
Console.WriteLine("Position: {0}, Value: {1}", i, numbers[i]);
```

- Der Index ist **nullbasiert**, d.h. das erste Element hat den Index 0
- Überschreitet <index> die Größe des Arrays, gibt es eine **Exception** – warum?

# Zugriff (2)

---

- Graphische Erläuterung

```
int[] numbers = new int[10];  
for (int i = 0; i < numbers.Length; i++)  
    numbers[i] = i*i;
```

Position	1	2	3	4	5	6	...	10
Index	0	1	2	3	4	5	...	9
Value	0	1	4	9	16	25	...	81

- **Wichtig:** da der Index von Arrays nullbasiert ist, gilt:
  - Wert an Position *i* hat den Index *i-1*
  - Der letzte Wert des Arrays hat den Index *Length-1*



# Wertzuweisung bei der Initialisierung

- Syntax

```
datatype[] <identifier> = new <datatype>[] {  
  <element1>, <element2>, ..., <elementn>  
};
```

- Beispiel:

```
string[] simpsons = new string[5] {  
  "Homer", "Marge", "Bart", "Lisa", "Maggie"  
};
```

Position	1	2	3	4	5
Index	0	1	2	3	4
Value	Homer	Marge	Bart	Lisa	Maggie



- Länge muss in diesem Fall nicht angegeben werden (ist implizit gegeben)

# foreach-Schleife

---

- Spezielle Schleife für Arrays und andere **Aufzählungstypen**

- einmal „über alle Elemente“ eines Arrays laufen
- nur **Lesezugriff!**

- Syntax

```
foreach(<datatype> <variable> in <array>) { <expression>; }
```

- Beispiel:

```
foreach(string character in simpsons) {  
    Console.WriteLine(character);  
}
```

- **Schleifenvariable** <variable> ist nur in der jeweiligen Iteration bekannt!

# Weitere nützliche Funktionen

---

Funktion	Beschreibung
<code>u.Length</code>	Anzahl der Elemente im Array <code>u</code>
<code>Array.Sort(u)</code>	Array <code>u</code> sortieren
<code>Array.Sort(k,u)</code>	Array <code>u</code> mit <code>k</code> verknüpfen und <code>u</code> anschließend mit <code>k</code> als Kriterium sortieren
<code>Array.Reverse(u)</code>	Die Anordnung umkehren
<code>Array.Resize(u,n)</code>	Größe des Arrays <code>u</code> auf <code>n</code> ändern

- Klasse „System.Array“
- Weitere Funktionen verfügbar unter <http://msdn.microsoft.com/en-us/library/system.array.aspx>

# Mehrdimensionale Arrays

---

- Arrays können auch mehrdimensional sein (siehe einführendes Beispiel)
- Syntax für **zweidimensionales** Array (x- und y-Koordinaten)  
`<datatype>[,] <identifier> = new <datatype>[<Length x>,<Length y>];`  
  
`byte[,] picture = new byte[1024,768];`
- Dimension **theoretisch unbegrenzt**, Beispiel: HD-Film von einer Sekunde (25 Bilder/s)  
`byte[, ,] movie = new byte[1920,1080,25];`
- Vorsicht: **Speicherverbrauch!**
  - $d1 \cdot d2 \cdot d3 = 1920 \cdot 1080 \cdot 25 = 51.840.000$  (ca. 51 MB)



# Agenda

---

- Arrays
- **Listen**
- Weitere komplexe Datentypen

# Generische Auflistungsklasse List<T>

---

- Ausgangsfrage
  - Länge eines Arrays ist nach der Erstellung fix
  - Was ist, wenn man nicht weiß, wie viele Elemente später einmal enthalten sein sollen?
    - Adressbuch, Einkaufsliste, Kontoauszug
  - Idee: großes Array erstellen → zu viel oder zu wenig reservierter Speicher
- Lösung: Listen
  - Liste mit Elementen desselben Typs
  - können wie Arrays keine Elemente unterschiedlichen Typs aufnehmen
  - vordefinierte Klasse `List<datatype>`

# Deklaration und Initialisierung

---

- Deklaration und Initialisierung

```
List<<datatype>> <identifizier> = new List<<datatype>>();
```

Bsp.: 

```
List<int> numbers = new List<int>();  
List<string> names = new List<string>();
```

- Nach dem Erstellen hat man eine leere Liste, die Elemente des angegebenen Datentyps aufnehmen kann

# Elemente aufnehmen und entfernen

---

- Neue Liste erstellen

```
List<string> simpsons = new List<string>(); // neue Liste für Strings
```

- Neues Elemente aufnehmen

```
simpsons.Add(<element>); // Element am Ende der Liste hinzufügen  
simpsons.Insert(<position>, <element>); // Element an einer Position einfügen
```

- Elemente aus der Liste entfernen

```
simpsons.Remove(<element>); // Element aus der Liste entfernen  
simpsons.RemoveAt(<position>); // Element an einer Position entfernen
```

- Aufruf eines Elements durch Index funktioniert wie bei Arrays

```
simpsons[<position>]; // gibt das Element an einer bestimmten Position zurück
```

- Wie bei Arrays gibt es eine Exception, wenn auf einen Index zugegriffen wird, der nicht in der Liste enthalten ist



# Agenda

---

- Arrays
- Listen
- **Weitere komplexe Datentypen**
  - Queues
  - Stacks

# Weitere komplexe Datentypen

---

- Schlangen: Queues

- Warteschlangen → First-in-First-out-Prinzip (FIFO)
- bspw. Supermarktkasse
- Funktionen
  - `Enqueue(<element>);` // neues Element wird am Ende eingefügt
  - `<datatype> <variable> = Dequeue();` // das erste Element wird herausgenommen

- Stapel: Stacks

- Stapel → Last-in-First-out-Prinzip (LIFO)
- Aufgabenstapel auf dem Schreibtisch
- Funktionen
  - `Push(<element>);` // neues Element wird oben draufgelegt
  - `<datatype> <variable> = Pop();` // das erste Element wird von oben abgenommen

