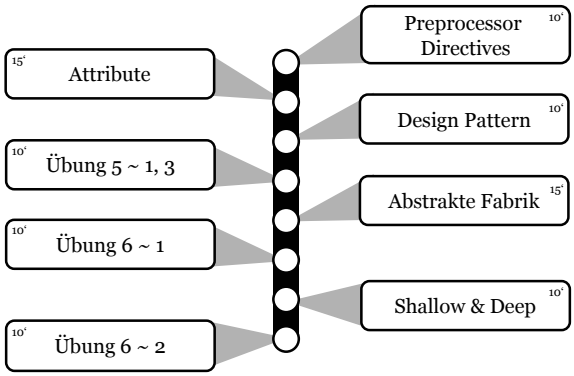
Was wir heute tun

9. Mai 2003



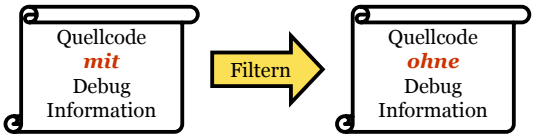
Debug Informationen

http://www.softsteel.co.uk/tutorials/cSharp/lesson18.html#1

beim Debug brauchen wir oft

```
Console.WriteLine(„Jetzt bin ich hier!“);
```

was, wenn das Produkt fertig zum Kunden kommt?



Kunde will schnelles & reines Produkt!

Preprocessor Directives

http://www.softsteel.co.uk/tutorials/cSharp/lesson18.html#1

Möglichkeit – globale Variable

```
public final bool debug = true;
if (debug) Console.WriteLine(„Jetzt bin ich hier!“);
```

→ sehr langsamer & langer Code:
einmal wird getestet,
hunderttausend mal wird es normal verwendet

Möglichkeit – Preprocessor Directives

```
#define DEBUG
#if DEBUG
 Console.WriteLine(„Jetzt bin ich hier!“);
#endif
```

→ alle Zeilen mit `#` werden vom Compiler vor der Kompilation bearbeitet & ausgewertet
→ bei der Auslieferung ändert man `#define DEBUG` zu `#undefine DEBUG` – alle Debug-Informationen weg!

Attribute

http://www.galileocomputing.de/openbook/csharp/kap2

hier sind nicht Felder (Members) & Methoden (Methods) gemeint!

Code hat Semantik – eine Bedeutung
wie teile ich dies, jemandem der Reflection macht, mit?

Möglichkeit – Interface oder bool'sche Variable

```
class A : ISehrWichtig {
 public final bool ichBinEineWichtigeKlasse = true;
 // ...
}
```

→ funktioniert leider nur bei Klassen, aber nicht bei Methoden etc.

Möglichkeit – Attribute

```
[wichtigeKlasse("Kategorie1")]
class A {
 // ...
}
```

Attribute in .NET

The common language runtime allows you to add keyword-like descriptive declarations, called attributes, to annotate programming elements such as types, fields, methods, and properties.

Attributes are saved with the metadata of a Microsoft .NET Framework file and can be used to describe your code to the runtime or to affect application behavior at run time.

While the .NET Framework supplies many useful attributes, you can also design and deploy your own.

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconextendingmetadatausingattributes.asp

Vordefinierte Attribute

```
class Rechner {
      [Obsolete(„wird in der nächsten Version entfernt")]
      public static int Summiere( int a, int b ) {
            return ( a + b );
      }
}
```

was macht **`System.ObsoleteAttribute`**?
beim Kompilieren wird bei jedem Zugriff auf die Methode **`Summiere`** eine Warnung ausgegeben

```
class MeinProgramm {
      [STAThread]
      static void Main( string[] args ) {
            // ...
      }
}
```

was macht **`System.STAThreadAttribute`**?
single-threaded apartment (STA)

Eigene Attribute schreiben

AttributeUsageAttribute anwenden

```
[AttributeUsage(AttributeTargets.Methods)]
```

Klasse deklarieren

```
public class MeinDingAttribute : System.Attribute {
```

Konstruktor deklarieren

```
public MeinDingAttribute( bool wert ) {
```

Eigenschaften deklarieren

```
private bool _wert;
public bool MeinWert {
 get ( return _wert; }
 set ( _wert = value; }
}
```

MeinDingAttribute benutzen

```
[MeinDing( true )]
public void Bla() {
```

Auf Attribute zugreifen

```
[Developer(„Hans Meiser“, „42“, Reviewed = true)]
class Bla {
 public static void Main() {
  Type t = typeof(Bla);
  DeveloperAttribute da = (DeveloperAttribute) ↵
              Attribute.GetCustomAttribute( t, ↵
                   typeof(DeveloperAttribute));
  Console.Write(da.Name);
  Console.Write(da.Level);
  Console.Write(da.Reviewed);
 }
}
```

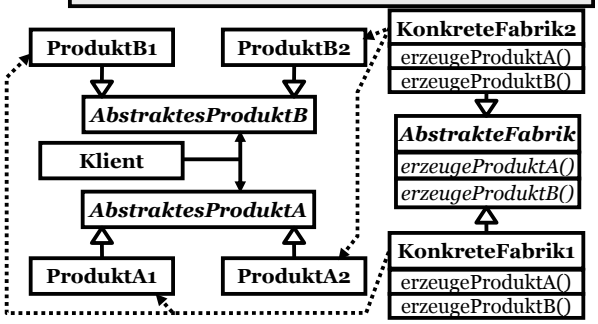
Design Pattern

Verschiedene Kategorien:

- Erzeugungsmuster – abstrakte Fabrik
  verstecken den Erzeugungsprozess
- Strukturmuster – Adapter
  befassen sich mit der Komposition von Klassen & Objekten, um grössere Strukturen zu bilden
- Verhaltensmuster – Beobachter
  befassen sich mit Algorithmen und der Zuweisung von Zuständigkeiten zu Objekten
  beschreiben nicht nur Muster von Objekten oder Klassen, sondern auch die Muster der Interaktion zwischen ihnen

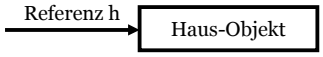
Abstrakte Fabrik

biete eine Schnittstele zum Erzeugen von Familien verwandter oder voneinander unabhängiger Objekten, ohne ihre konkreten Klassen zu benennen.

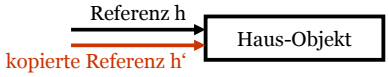


Shallow copy & deep copy

```
Haus h = new Haus(„Bahnhofstr.  1“);
```



Shallow – flache Kopie:



deep – tiefe Kopie: