Polymorphie

Unter Polymorphie versteht man, wenn eine Klasse von der anderen erbt.

Dabei nimmt eine Klasse die Rolle der Elternklasse und die anderen, die einer Kindklasse an.

Bedingung:

1. Eigenschaft/Methode muss gleich heißen
2. Die Kindklasse erbt die Elternklasse (LKW : PKW)
3. die Methode der Eltern muss virual sein
4. die Methode des Kindes muss override sein (Das heißt die Elternmethode wird überschrieben/ ergänzt)

das base.[Methodenname] implementiert die Methode aus der Elternklasse in die Kindklasse
[crayon-6766f29445440191717685/]
erstellen wir nun ein Objekt von PKW und lassen die Methode aufrufen, bekommen wir als Ausgabe
[crayon-6766f29445447185269610/]
tun wir dasselbe mit der LKW Klasse, bekommen wir als Ausgabe:
[crayon-6766f29445449396976504/]
sealed – Klasse versiegeln

Generell kann jede Klasse von einer anderen Erben. Möchte man aber vermeiden, dass von einer Klasse geerbt werden soll, nutzt man den Ausdruck sealed. Dies ist Sinnvoll, wenn man weiß, dass man in der Vererbung in der letzten Instanz angekommen ist.
[crayon-6766f2944544a714110382/]
abstract – Abstrakte Klassen

Abstrakte Klassen sind Klassen, die reine vererbbare Klassen sind. Das bedeutet man kann aus der Klasse kein Objekt mehr

erzeugen.
[crayon-6766f2944544c248108268/]

Eigene Mini Programmiersprache schaffen mithilfe von Regulären Ausdrücken

Filtert nur die Ausdrücke aus, die mit[beginnen und mit] enden.

Match liefert nur den erstgefundenen Wert,

MatchCollection dagegen alle gefundene.
[crayon-6766f294458d2252035038/]
Nun könnte man über ein switch/case eine Abfrage erstellen.

Z.B. case FELD1:

tue dies oder jenes

Quellen:

http://www.regexr.com/ – Online Editor

http://www.mycsharp.de/wbb2/thread.php?threadid=41009 – Tutorial

http://www.dotnetperls.com/regex-match – Tutorial

Strukturen – struct

Strukturen sind ähnlich wie Klassen, weisen im Gegensatz zu Ihnen aber folgende Unterschiede auf:

Die Methoden/Konstruktoren haben keine Eigenschaftswerte und Namen

Auf Strukturen kann deutlich schneller zugegriffen werden

Strukturen sind Werttypen und keine Verweistypen

Können nicht erben/vererben

Können keine Konstruktoren ohne Parameter haben

Kleines Beispiel:
[crayon-6766f29445a09736301497/]

Das Objekt x greift direkt auf die Variablen Vorwahl, Nummer zu

Das Objekt y dagegen geht den besseren und schnelleren Weg über den Konstruktor v,n und instanziert dann die Variablen dementsprechend.

Binding Elementbinding

Möchte man eine Elementeigenschaft an die andere binden, muss man in die Quelleigenschaft gehen und ein {Binding … } einsetzen.

Zuerst wird das Objekt, dann dessen Eigenschaft abgefragt.

ElementName (welches Element?)

Path(welche Eigenschaft?)

Bei Path könnte auch etwas anderes stehen. SelectedIndex z.B.

oder Items[0] für das erste Item in der ListBox
[crayon-6766f29445b27828078366/]
Das Visual Studio bietet mit dem Intellisense eine große Hilfe bei der Selektion.

Binding Allgemein

Unter Binding versteht man, eine Eigenschaft von dem Quellobjekt an eine Eigenschaft des Zielobjekts bindet.

Dabei gibt es 2 Arten:

Element an ein anderes Element / Element an Datenquelle z.B. Datenbank

Dabei bindet man eine Abängigkeits-Eigenschaft (Dependency Property) an die andere. Diese müssen gleichen Typs sein. Sind diese Anforderungen erfüllt, kann man alles binden. Breite eines Steuerelements an die Fensterbreite, selektiertes Element einer Combobox an ein Label und sogar ganze Spalte aus einer Datenbank.

Das Visual Studio bietet hierfür eine Hilfe in den Eigenschaften der Steuerelemente an.

Hier sehen wir nun, 7 Bindungstypen, die uns zur Verfügung stehen. Jedes dieser Bindungen umfasst eine Ansammlung von Eigenschaften aus seinem Gebiet.

1. **Datenkontext**: Jede WPF Anwendung hat zu Beginn 2 Schichten. Eine für UI (User Interface) und die andere für Daten. Die Datenschicht ist zu Beginn Null, das heißt ohne Inhalt. Indem wir deren Eigenschaft DataContext setzen umfasst dieser Kontext nun alle angegebenen Daten. Solange man nichts weiter angibt, erben nun die Kindobjekte (Steuerelemente wie Label, Textbox usw.) von dem Elternobjekten diese Daten.
2. **Datenquelle**:
3. **ElementName:** Umfasst alle visuelle und nicht visuelle Elemente aus dem XAML Projekt. Hier stehen einem sämtliche Eigenschaften der Steuerelemente wie Textbox, Label, Combobox und auch nicht die Eigenschaften vom

Window zur Verfügung.
4. **RelativeSource FindAncestor:** Hier werden sämtliche Eigenschaften von allen hierarchisch überliegenden Objekten zur Verfügung gestellt.
5. **RelativeSource PreviusData:** Gegenteil vom FindAncestor, werden hier alle unterliegenden Objekteigenschaften aufgelistet
6. **RelativeSource Self:** Hier bekommt man Eigenschaften vom selben Objekt zur Verfügung gestellt
7. **RelativeSource TemplatedParent:**
8. **StaticResource**: Bezeichnet eine schon zuvor in die Resources eingeschriebene Datenquelle

---

Textdateien schreiben / lesen

Textdateien schreiben:

Der erste Parameter gibt den Pfad der Textdatei an. Wird nur ein Dateiname angegeben, wird die Datei im selben Ordner erstellt, wo das Programm derzeit läuft. Der 2. Parameter gibt an, wenn es diese Datei schon gibt, ob er die dann überschreiben soll (true) oder neu erstellen soll (false)
[crayon-6766f29445c76778177244/]
Eine Textdatei auslesen funktioniert auf ähnliche Weise, nur mit dem StreamReader. Mit dieser Möglichkeit wird der Inhalt

der Textdatei in den string abc iniatilisiert.
[crayon-6766f29445c79918723432/]
Möchte man eine ganze Textdatei in einem Rutsch auslesen, geht dies auch ganz einfach mit:
[crayon-6766f29445c7b966222335/]
und schreiben mit:
[crayon-6766f29445c7c873027837/]
Um auch mit deutschen Umlauten zu arbeiten, muss man die Codierung auf UTF-8 stellen:

```
using (sw = new StreamWriter(@exportdatei, false, System.Text.Encoding.Default))
```