**Polymorphie**

Polymorphie bedeutet „**Vielgesichtigkeit**“. Eine Instanz einer Elternklasse kann in „Verkleidung“ ihrer Kindklassen auftreten aber deren Funktionalitäten unter dem eigenen Kontext ausführen.  
So kann sich jede Instanz einer abgeleiteten Klasse **polymorph** verhalten – so als sie eine Instanz der Basisklasse wäre.  
Sprich: Eine Kindklasse implementiert **in ihrer eigenen Art und Weise** eine gleichnamige Methode ihrer Elternklasse, tritt aber nach Außen auf, als ob sie Instanz der Elternklasse wäre.

**Merke:**  
Die Polymorphie und ist ein **wichtiges Konzept** der objektorientierten Programmierart. Sie ermöglicht, dass ein Objekt unterschiedlich agieren kann, **je nach logischem Kontext**, in dem es verwendet wird.

**Beispiel**

Objekt **Privatkunde** erbt von Objekt **Kunde** und überschreibt unter Verwendung des Schlüsselwortes **override** die elternklasseneigene Methode AddGuthaben.

Es darf sie überschreiben (muss aber nicht), weil AddGuthaben in der Elternklasse per Schlüsselwort **virtual** für eine etwaige Überschreibung freigegeben wurde.

**Elternklasse**:

public **virtual** void AddGuthaben(double betrag)

**Kindklasse**:

public **override** void AddGuthaben(double betrag)

{

// Die Polymorphie ermöglicht in einer überschriebenen Methode der  
// Kindklasse die Hinterlegung einer ganz anderen Logik,

// als jene, die Elternklasse in ihrer gleichnamigen Methode anbietet.

// Hier rufe ich z.B. die zugrunde liegende Methode AddGuthaben der  
// Elternklasse GAR NICHT auf

// stattdessen erhöhe ich das Guthaben, indem ich die protected-Property der Elternklasse "Guthaben" direkt verwende

base.Guthaben += (0.05 \* betrag);

}

**Instanziierung**

var privatKunde = new PrivatKunde("Herr", "Erich", "Innsbruck")

{

Stammkunde = false

};

**Anwendung**

Da privatKunde zugleich Kunde ist (privatKunde is-a Kunde), habe ich die Freiheit, AddGuthaben - Methode des Kunde -Objektes **anzusprechen**.

Welche Implementierung dieser Methode aber **wirklich ausgeführt** wird, entscheidet **zur Laufzeit** der Compiler. Grundsätzlich kommt in der Vererbungshierarchie die am **höchsten** überschriebene Implementierung zum Einsatz.

Kunde kunde = privatKunde;

kunde.AddGuthaben(12.2)

**Hinweis**

Jedes Objekt im .NET verfügt automatisch die Methode ToString. Sie können ToString() in Ihrer Klasse überschreiben und davon eine eigene Implementierung zur Verfügung stellen.

Beispiel

public override string ToString()

{

var str = base.ToString();

str += " . Diese Klasse habe ich selbst geschrieben.";

return str;

}

**Übung**

Erstellen Sie eine Basisklasse **Log** mit einer virtual-Methode **Send()**.

Zwei weitere Klassen sollen von Log erben und jeweils ihre eigene Implementierung von Send() anbieten.  
Eine weitere Klasse erbt ebenfalls von Log, doch bietet sie keine eigene Implementierung der Send()-Methode.

Wenden sie alle die Objekte in Program.cs Ihrer Konsolenanwendung so an, dass jedes Objekt **unter der Verwendung seiner Basisklasse** seine eigene Send-Methode ausführt.