***Aufgabe A1**: C# ist als Sprache eng mit dem .NET Framework verbunden. Erläutern Sie die wesentlichen Komponenten des Frameworks und setzen Sie diese in Zusammenhang mit der Sprache C#. Gehen Sie mindestens auf die folgenden Begriffe ein:*

- *CTS*
- *CLR*
- *Basisklassen*
- *IL*

Das .NET-Framework stellt eine leistungstarke Runtime zur Verfügung, die zunächst den erstellten Quellcode in die maschinen- und programmiersprachenunabhängigie Intermediate Language (IL) kompiliert, welcher dann zur Laufzeit von der Common Language Runtime (CLR) just-in-time prozessoroptimiert ausgeführt wird. Die IL ähnelt dabei in Teilen dem Java Bytecode und kann von verschiedenen .NET-Programmiersprachen erzeugt werden. Die von der CLR implementierte formale Spezifikation des Typsystems nennt sich Common Type System (CTS), welches die Grundlage für Sprachenunabhängigkeit im .NET-Framework bildet. Grundsätzlich werden zwischen Werte- und Referenztypen unterschieden. Alle anderen Datentypen werden davon abgeleitet. Basisklassen sind vordefinierte Klassen aus der Klassenbibliothek, die das Arbeiten mit dem .NET-Framework für Entwickler erheblich vereinfachen. Sie kapseln den Zugriff auf die OS-API und stellen eine Vielzahl von Methoden und Properties zur Verfügung.

***Aufgabe A2**: Skizzieren Sie kurz die wesentlichen Vorraussetzungen damit auf Elemente einer Klasse mittels foreach-Schleife zugegriffen werden kann. Welche Member müssen in der Klasse implementiert werden und welche Aufgabe besitzen diese?*

Damit eine Klasse mittels foreach durchlaufen werden kann, benötigt sie eine Methode GetEnumerator(), die einen Enumerator zurückliefert. Das Interface IEnumerable schreibt diese Methode vor, jedoch muss nicht jede Klasse dieses Interface implementieren, es genügt die angesprochene Methode. Der Enumerator durchläuft alle Elemente einer Klasse und implementiert dafür das Interface IEnumerator. Dieses Interface beinhaltet die beiden Methoden MoveNext() und Reset() sowie das Property Current. Mit Hilfe von MoveNext werden die Elemente nacheinander durchlaufen, bis das Ende erreicht ist. Mit Current wird das aktuelle Element angegeben. Reset setzt den Enumerator wieder auf das erste Element. Mit Hilfe der yield-Anweisung wird die Erstellung eines Enumerators erleichtert.

***Aufgabe B1**: Erläutern Sie die Grundlagen des Zugriffs auf Felder einer Klasse über Properties und über Indexer. Gehen Sie auch auf die Unterschiede ein. Geben Sie jeweils ein kurzes Beispiel für die Implementation eines Properties und eines Indexers.*

Mit Hilfe eines Indexers kann auf eine Klasse wie auf ein Array zugegriffen werden, in dem auf die Elemente über einen Index [i] zugegriffen wird. Ein Indexer hat immer den Namen this, die Definition ähnelt aber der eines Properties. Beide Konzepte besitzen eine get- und set-Accessormethode, um den Zugriff auf die inneren Felder zu kapseln. Während jedoch bei einem Property diese Methoden (bis auf das implizite value beim set-Accessor) keine weiteren Parameter besitzen, haben sie bei einem Indexer zusätzlich die gleichen Parameter wie der Indexer selbst. Während ein Property durch seinen Namen eindeutig identifiziert werden kann, ist bei einem Indexer nur seine Signatur eindeutig. Indexer können im Gegensatz zu Properties nicht statisch sein.

```
class Program
{
    static void Main(string[] args)
    {
        TestKlasse k1 = new TestKlasse();
        k1.MyProperty = 5;
        Console.WriteLine(k1.MyProperty);
        k1[0] = 1;
        Console.WriteLine(k1[0]);
        Console.ReadLine();
    }
}

class TestKlasse
{
    private int myProperty;
    public int MyProperty
    {
        get { return myProperty; }
        set { myProperty = value; }
    }

    private int[] arr;
    public int this[int i]
    {
        get { return arr[i]; }
        set { arr[i] = value; }
    }

    public TestKlasse()
    {
        arr = new int[5];
    }
}
```

***Aufgabe C1**: Implementieren Sie in C# eine Klasse "Vorlesungsteilnehmer", die zur Verwaltung der C# Vorlesung ab SS04 eingesetzt werden soll.*

Auf dem Klausurbogen soll lediglich der Quellcode der Datei "Vorlesungsteilnehmer.cs" zu finden sein! Es sind keine weiteren Angaben zur Konfiguration innerhalb von VS.net nötig.

Implementieren Sie die folgenden Member:

- *Capacity* — *Read-Only Property zum lesen der maximalen Anzahl von Studenten, die an dieser Veranstaltung teilnehmen können.*
- *capacity* — *das zugehörige private konstante Feld, das durch den Konstruktor initialisiert werden soll.*
- *Count Read-Only* — *Property zur Ausgabe der zzt. in der Vorlesung verwalteten StudentInnen*
- *CountM* — *Diesmal nur die männlichen.*
- *CountF* — *Diesmal nur die weiblichen.*
- *Add(StudentIn)* — *fügt eine(n) StudentIn zur Vorlesung hinzu.*
- *Remove(StudentIn)* — *entfernt eine(n) StudentIn aus der Vorlesung.*

Zusätzlich sollen ein Konstruktor und ein Indexer implementiert werden. Weitere Member – sofern benötigt – sind eigenständig dieser Klasse hinzuzufügen. Es ist freigestellt, wie die Klasse "Vorlesungsteilnehmer" die Studenten intern verwaltet.

Eine Klasse "StudentIn" ist bereits vorhanden. Objekte dieser Klasse sollen in der Klasse "Vorlesungsteilnehmer" verwendet werden. Die Klasse StudentIn verfügt über ein Property "Gender", über das das Geschlecht ausgelesen werden kann. Ein Konstruktor der Klasse StudentIn erwartet den Namen (string) und das Geschlecht (char) als Parameter.

```
class Vorlesungsteilnehmer
{
    List<StudentIn> liste;

    private int capacity;
    public int Capacity
    {
        get { return capacity; }
    }

    public Vorlesungsteilnehmer(int capacity)
    {
        this.capacity = capacity;
        liste = new List<StudentIn>(capacity);
    }

    public int Count()
    {
        return liste.Count();
    }

    public int CountM()
    {
        return liste.Count(
            delegate(StudentIn s)
            {
                return s.Gender == 'm';
            });
    }

    public int CountF()
    {
        return liste.Count(
            delegate(StudentIn s)
            {
                return s.Gender == 'w';
            });
    }

    public void Add(StudentIn studentIn)
    {
        liste.Add(studentIn);
    }

    public void Remove(StudentIn studentIn)
    {
        liste.Remove(studentIn);
    }

    public StudentIn this[int i]
    {
```

```
            get { return liste[i]; }
            set { liste[i] = value; }
        }
    }
```

***Aufgabe C2**: Implementieren Sie ein Hauptprogramm, welches ihren Code testet. Gehen Sie dabei von einer C#-Konsolenanwendung aus.*

```
class Program
{
    static void Main(string[] args)
    {
        Vorlesungsteilnehmer vt = new Vorlesungsteilnehmer(10);
        StudentIn jan = new StudentIn("jan", 'm');
        vt.Add(jan);
        Console.WriteLine("Student Jan hinzugefügt.");
        StudentIn frank = new StudentIn("frank", 'm');
        vt.Add(frank);
        Console.WriteLine("Student Frank hinzugefügt.");
        Console.WriteLine("Anzahl männlicher Teilnehmer: " +
              vt.CountM());
        StudentIn lena = new StudentIn("lena", 'w');
        vt.Add(lena);
        Console.WriteLine("Studentin Lena hinzugefügt.");
        StudentIn magy = new StudentIn("magy", 'w');
        vt.Add(magy);
        Console.WriteLine("Studentin Magy hinzugefügt.");
        Console.WriteLine("Anzahl weiblicher Teilnehmer: " +
              vt.CountF());
        Console.WriteLine("Anzahl aller Teilnehmer: " + vt.Count());
        vt.Remove(magy);
        Console.WriteLine("Studentin Magy entfernt.");
        Console.WriteLine("Anzahl weiblicher Teilnehmer: " +
              vt.CountF());
        Console.WriteLine("Anzahl aller Teilnehmer: " + vt.Count());
        Console.ReadLine();
    }
}
```

***Aufgabe D1**: Implementieren Sie eine Assembly, die ein elektronisches Buch repräsentiert.*

Innerhalb des Namespaces ElectronicLibrary sollen die Klasse Book und die Enumeration BookType implementiert werden.

Für die Klasse Book gelten die folgenden Anforderungen:

- *Alle privaten Felder sollen über öffentliche Read-Write-Properties zugreifbar sein.*
- *In einem Book sollen der Autor, der Titel, das Erscheinungsjahr und der Buchtyp gespeichert sein.*
- *Der Inhalt des Buches soll als Collection von einzelnen Kapiteln verwaltet werden. Diese sollen über einen Indexer abgerufen und über eine Add-Methode hinzugefügt werden können.*
- *Als Buchtyp soll die Enumeration BookType genutzt werden, die 5 unterschiedliche Buchtypen umfassen soll: Roman, Gedicht, Märchen, Sage, Sachbuch*
- *Implementieren Sie geeignete Konstruktoren*
  - *Standard-Konstruktor,*
  - *Copy-Konstruktor,*

- *mindestens 2 unterschiedliche Konstruktoren, die eine sinnvolle Erzeugung eines Buches durch direkte Parameterübergabe ermöglichen*

- *Ermöglichen Sie eine Ausgabe eines Buches als Zeichenkette über die in C# vorgesehenen Mechanismen. Dabei sollen ein Buch in einen „XML-String“ serialisiert werden, der die einzelnen Bestandteile der Klasse gemäß dem folgenden Beispiel enthält, wobei die Punkte durch die Werte des Objektes gefüllt werden sollen:*

  <book> <author>...</author> <title>...</title> <type>...</type> <chapters> <chapter>...</chapter> <chapter>...</chapter> <chap-ter>...</chapter> <chapter>...</chapter> <chapter>...</chapter> <chapter>...</chapter> <chapter>...</chapter> <chap-ter>...</chapter> </chapters> </book>

Erstellen Sie ein Hauptprogramm, das folgende Funktionen der Assembly testet:

- *Erstellen Sie ein Book-Objekt und füllen Sie das Buch mit Inhalt:*
  - *Setzen Sie die Felder über die Properties.*
  - *Füllen Sie das Buch mit einigen „Kapiteln“ Text.*
  - *Geben Sie das Buch auf der Konsole aus.*

Greifen Sie über den Indexer auf ein Kapitel zu und geben dieses erneut auf der Konsole aus.

```
class Program
{
    static void Main(string[] args)
    {
        Book Buch = new Book("Dan Brown","Sakrileg","2006",
             BookType.Roman);
        Buch.KapitelHinzufuegen("Kapitel 1");
        Buch.KapitelHinzufuegen("Kapitel 2");
        Buch.KapitelHinzufuegen("Kapitel 3");
        Console.WriteLine(Buch.Kapitel[0]);
        Console.WriteLine(Buch.ToString());
        Console.ReadLine();
    }
}

class Book
{
    private string autor;
    public string Autor
    {
        get { return autor; }
        set { autor = value; }
    }

    private string titel;
    public string Titel
    {
        get { return titel; }
        set { titel = value; }
    }

    private string erscheinungsjahr;
    public string Erscheinungsjahr
    {
        get { return erscheinungsjahr; }
        set { erscheinungsjahr = value; }
```

```
}

private BookType buchtyp;
public BookType Buchtyp
{
    get { return buchtyp; }
    set { buchtyp = value; }
}

private ArrayList kapitel;
public ArrayList Kapitel
{
    get { return kapitel; }
    set { kapitel = value; }
}

public Book(string autor, string titel, string erscheinungsjahr,
    BookType buchtyp)
{
    this.autor = autor;
    this.titel = titel;
    this.erscheinungsjahr = erscheinungsjahr;
    this.buchtyp = buchtyp;
    kapitel = new ArrayList();
}

public Book(Book buch) : this(buch.autor,buch.titel,
    buch.erscheinungsjahr,buch.buchtyp) { }

public void KapitelHinzufuegen(string kapitelname)
{
    kapitel.Add(kapitelname);
}

public override string ToString()
{
    string ausgabe = "<book> ";
    ausgabe += "<author>" + autor + "</author> ";
    ausgabe += "<title>" + titel + "</title> ";
    ausgabe += "<year>" + erscheinungsjahr + "</year>";
    ausgabe += "<type>" + buchtyp + "</type> ";
    ausgabe += "<chapters> ";
    foreach (string k in kapitel)
        ausgabe += "<chapter>" + k + "</chapter> ";
    ausgabe += "</chapters> ";
    ausgabe += "</book>";
    return ausgabe;
}

// wird nicht unbedingt benötigt, da ArrayList bereits einen
    Indexer implementiert
/*public string this[int i]
{
    get
    {
        return (string)kapitel[i];
    }
    set
    {
        kapitel[i] = value;
    }
}*/
```

```
}

enum BookType
{
    Roman, Gedicht, Maerchen, Sage, Sachbuch
}
```

***Aufgabe E1**: Gegeben ist eine Assembly Geometrie.dll, in der eine Klasse Point implementiert wurde. Die Klasse Point ist eine einfache Implementierung eines Punktes im zweidimensionalen Raum. Ein Punkt wird durch seine zwei Koordinaten x und y repräsentiert, die als öffentliche Felder vom Typ int in der Klasse Point implementiert wurden. Über diese Felder hinaus wurde in der Klasse keine Funktionalität implementiert.*

Implementieren Sie eine Klasse EnhancedPoint, die von der Klasse Point abgeleitet ist und um die folgenden Bestandteile erweitert wird:

- *Erzeugen Sie geeignete Konstruktoren (Standard-Konstruktor, Copy-Konstruktor, Konstruktor mit x und y als Parameter)*
- *Sorgen Sie für eine ansprechende Ausgabe des Punktes im Konsolenfenster durch die in C# vorgesehenen Mechanismen.*
- *Implementieren Sie für diese Klasse die Operatoren + und – zur Verknüpfung von Punkten sowie den Operator ==.*
- *Entwerfen Sie eine statische Methode um die Entfernung zwischen zwei Punkten zu berechnen.*
- *Definieren Sie einen expliziten Casts, der die Entfernung eines Punkts vom Ursprung in Float ausgibt.*

Erstellen Sie ein Hauptprogramm, das alle Funktionen der Klasse ausreichend testet.

Hinweise:

Die Entfernung zweier Punkte berechnet sich nach der folgenden Formel: $\sqrt{(\Delta x)^2 + (\Delta y)^2}$

Die Methode Math.Sqrt gibt die Quadratwurzel einer angegeben Zahl zurück.

```
  public static double Sqrt(doubled);
```

```
class Program
{
    static void Main(string[] args)
    {
        EnhancedPoint P1 = new EnhancedPoint(1, 3);
        EnhancedPoint P2 = new EnhancedPoint(P1);
        EnhancedPoint P3 = P1 + P2;
        EnhancedPoint P4 = P3 - P1;
        Console.WriteLine("P1: " + P1);
        Console.WriteLine("P2: " + P2);
        Console.WriteLine("P3 = P1 + P2: " + P3);
        Console.WriteLine("P4 = P3 - P1: " + P4);
        Console.WriteLine("P1 == P3: " + (P1 == P3));
        Console.WriteLine("P1 == P2: " + (P1 == P2));
        Console.WriteLine("Entfernung von P1 und P3: " +
             EnhancedPoint.Entfernung(P1, P3));
        Console.WriteLine("Entfernung von P3 vom Ursprung: " +
              (float)P3);
```

```
            Console.ReadLine();
        }
    }

    class Point
    {
        public int X;
        public int Y;
    }

    class EnhancedPoint : Point
    {
        public EnhancedPoint() : this(0,0) {}

        public EnhancedPoint(Point punkt) : this(punkt.X,punkt.Y) {}

        public EnhancedPoint(int x, int y)
        {
            this.X = x;
            this.Y = y;
        }

        public override string ToString()
        {
            return "(" + X + "," + Y + ")";
        }

        public static EnhancedPoint operator +(EnhancedPoint p1,
            EnhancedPoint p2)
        {
            return new EnhancedPoint(p1.X + p2.X, p1.Y + p2.Y);
        }

        public static EnhancedPoint operator -(EnhancedPoint p1,
            EnhancedPoint p2)
        {
            return new EnhancedPoint(p1.X - p2.X, p1.Y - p2.Y);
        }

        public static bool operator ==(EnhancedPoint p1, EnhancedPoint p2)
        {
            return p1.X == p2.X && p1.Y == p2.Y;
        }

        public static bool operator !=(EnhancedPoint p1, EnhancedPoint p2)
        {
            return !(p1 == p2);
        }

        public static double Entfernung(EnhancedPoint p1, EnhancedPoint p2)
        {
            return Math.Sqrt(Math.Pow(p1.X - p2.X, 2) + Math.Pow(p1.Y -
                  p2.Y, 2));
        }

        public static explicit operator float(EnhancedPoint p1)
        {
            return (float)Math.Sqrt((p1.X * p1.X) + (p1.Y * p1.Y));
        }
    }
```

***Aufgabe F1**: Erklären Sie den Unterschied zwischen abstrakten und virtuellen Methoden anhand einfacher Beispiele.*

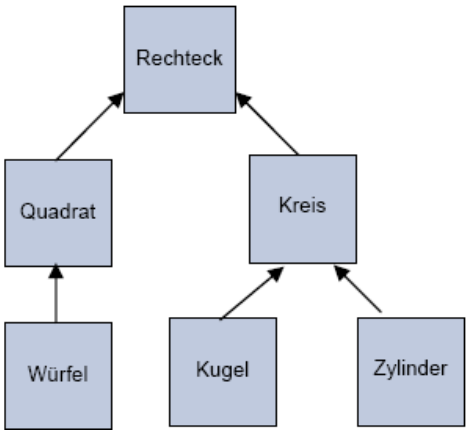
Eine abstrakte Methode definiert nur den Methodenkopf mit ihrer Signatur, ohne die Implementierung der Methode. Besitzt eine Klasse eine abstrakte Methode, so ist diese auch abstrakt. Abgeleitete Klassen müssen abstrakte Methoden implementieren, um von den Klassen Instanzen bilden zu können. Virtuelle Methode sind für ein Überschreiben (override) oder Verdecken (new) der Implementierung in einer abgleiteten Klasse vorgesehen. Beim Aufruf einer virtuellen Methode wird zur Laufzeit überprüft, ob eine überschriebene Version in einer Unterklasse existiert und diese dann ggf. aufgerufen (Late Binding). Eine abstrakte Methode ist implizit virtuell und darf nicht zusätzlich als virtual gekennzeichnet werden.

```
class Program
{
    static void Main(string[] args)
    {
        Rechteck rechteck = new Rechteck(3, 4);
        rechteck.Ausgabe();
    }
}

abstract class Dimension
{
    public int X;
    public int Y;

    public Dimension(int x, int y)
    {
        this.X = x;
        this.Y = y;
    }

    abstract public int FlaecheBerechnen();

    virtual public void Ausgabe()
    {
        Console.WriteLine("Hier kommt die Ausgabe rein.");
    }
}

class Rechteck : Dimension
{
    public Rechteck(int x, int y) : base(x, y) {}

    override public int FlaecheBerechnen()
    {
        return X * Y;
    }

    override public void Ausgabe()
    {
        Console.WriteLine("Ausgabe für ein Rechteck.");
    }
}
```

***Aufgabe F2**: Erzeugen Sie eine Klasse Dimension, die die Verwaltung von graphischen Objekten erlaubt.*

*Die Basisklasse eignet sich zur Darstellung eines Rechtecks und hat zwei Properties, die die Ausdehnung in x- und y-Richtung enthalten und eine Methode Fläche zur Berechnung des Flächeninhalts (x * y).*

Die folgenden Klassen sollen ebenfalls entsprechend der Hierarchie implementiert werden:



Als Hilfestellung die Formeln zur Flächenberechnung der einzelnen Klassen:

- *Rechteck (x*y)*
- *Quadrat (a^2)*
- *Würfel ($6*a^2$)*
- *Kreis ($2\pi r^2$)*
- *Kugel ($4\pi r^2$)*
- *Zylinder ($2*r^2*\pi + (2r*\pi*h)$)*

Legen Sie ein Array vom Typ der Basisklasse (Dimension) an und erzeugen Sie hierbei Instanzen der Kindklassen. Geben Sie anschließend den Flächeninhalt des Objekts auf der Konsole aus.

```
class Program
{
    static void Main(string[] args)
    {
        Dimension[] array = new Dimension[] {
            new Rechteck(2, 3),
            new Quadrat(2),
            new Wuerfel(4),
            new Kreis(1),
            new Kugel(1),
            new Zylinder(1,3)
        };
        foreach (Dimension d in array)
            Console.WriteLine(d.FlaecheBerechnen());
        Console.ReadLine();
    }
}

abstract class Dimension
{
    private int x;
    public int X
    {
        get
        {
```

```
            return x;
        }
        set
        {
            x = value;
        }
    }

    private int y;
    public int Y
    {
        get
        {
            return y;
        }
        set
        {
            y = value;
        }
    }

    abstract public double FlaecheBerechnen();
}

class Rechteck : Dimension
{
    public Rechteck(int x, int y)
    {
        this.X = x;
        this.Y = y;
    }

    public override double FlaecheBerechnen()
    {
        return X * Y;
    }
}

class Quadrat : Rechteck
{
    public Quadrat(int x) : base(x, x) { }
}

class Wuerfel : Quadrat
{
    public Wuerfel(int x) : base(x) { }

    public override double FlaecheBerechnen()
    {
        return X * base.FlaecheBerechnen();
    }
}

class Kreis : Rechteck
{
    public Kreis(int r) : base(r, r) { }

    public override double FlaecheBerechnen()
    {
        return 2 * Math.PI * X * X;
    }
}
```

```
class Kugel : Kreis
{
    public Kugel(int r) : base(r) { }

    public override double FlaecheBerechnen()
    {
        return 2 * base.FlaecheBerechnen();
    }
}

class Zylinder : Kreis
{
    public Zylinder(int r, int h) : base(r)
    {
        Y = h;
    }

    public override double FlaecheBerechnen()
    {
        return base.FlaecheBerechnen() + 2 * X * Y * Math.PI;
    }
}
```

***Aufgabe G1**: Definieren Sie den Begriff Assemblies!*

Eine Assembly ist eine selbstbeschreibende Installationseinheit, die alle verwendeten und referenzierten Komponenten beinhaltet. Sie wird als EXE- oder DLL-Datei gespeichert.

***Aufgabe G2**: Nennen Sie die wesentlichen Eigenschaften von Assemblies! Erläutern Sie anschließend die Probleme traditioneller Komponenten-Architekturen, die durch die Eigenschaften der Assemblies gelöst werden sollen.*

Assemblies sind selbstbeschreibend. Sie enthalten Metadaten über sich selbst und über die verwendeten Datentypen. Versionsabhängigkeiten werden im Manifest einer Assembly gespeichert. So wird sichergestellt, dass referenzierte Assemblies in der richtigen Version geladen werden. Es ist auch möglich, Assemblies side-by-side zu laden. Das bedeutet, dass ein Programm zwei verschiedene Versionen von einer Assembly laden und benutzen kann. Durch das Verwenden von Application Domains wirken sich Ausnahmen in einer Applikation nicht auf andere Applikationen aus. Das Installieren einer Assembly erfolgt durch das einfache Kopieren aller verwendeten Dateien (no-touch deployment).

Vor der Einführung von Assemblies wurden globale Funktionen in DLLs ausgelagert. Durch das Überschreiben von DLLs ohne Rücksicht auf Abwärtskompatibilität und dem verstreuten Vorhandensein von DLLs kam es zu einer wahren DLL-Hölle. Dieses Problem wurde durch die Einführung von Assemblies gelöst.

***Aufgabe G3**: Erläutern Sie den Unterschied zwischen Private Assemblies und Shared Assemblies und nennen Sie potentielle Anwendungsgebiete für die beiden Varianten.*

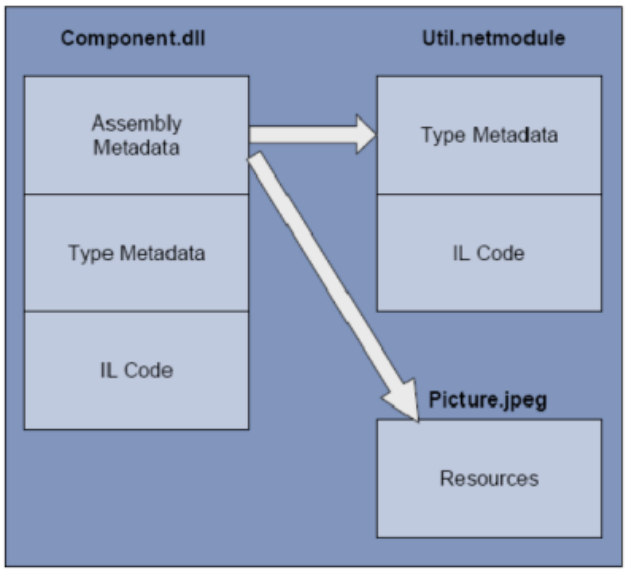
Private Assemblies speichern alle benötigten Dateien im gleichen Ordner wie das Hauptprogramm oder einem entsprechenden Unterordner. Für die meisten Programme reichen Private Assemblies aus, da sie keine Funktionen bereitstellen, die auch von anderen Programmen verwendet werden

sollen. Möchte man globale Funktionen für andere Programme bereitstellen, so wird eine Shared Assembly erstellt, die einen strong name aufweisen muss und im Global Assembly Cache gespeichert wird. Die .NET-Basisklassen sind in Shared Assemblies gespeichert.

***Aufgabe G4**: Erläutern Sie, wie Assemblies einen Beitrag zum Mehrsprachigkeitskonzept innerhalb des .NET Frameworks beitragen. Gehen Sie in diesem Zusammenhang auch kurz auf CTS und CLS ein.*

Das CTS definiert, wie Datentypen gespeichert, referenziert und verwendet werden dürfen. Da jedoch nicht jede .NET-Sprache alle angebotenen Datentypen unterstützt, definiert das CLS eine Menge von Datentypen und Operationen, welches eine Teilmenge der CTS darstellt. Diese müssen von allen .NET-Sprachen unterstützt werden. Wird der entwickelte Code als CLS-konform markiert, warnt der Compiler vor Verwendung von nicht-unterstützten Datentypen. CLS-konforme Programme können von jeder .NET-Sprache verwendet werden. So ist es möglich, ein Modul in Visual Basic zu schreiben, und in eine Assembly zu laden, die in C# geschrieben wird. In Visual Basic definierte Klassen können in C# abgeleitet werden. So erzielt das .NET-Framework eine einfache Möglich, mehrsprachige Programme zu entwickeln.

***Aufgabe G5**: Beschreiben Sie anhand der Abbildung die Bestandteile von Assemblies und schildern Sie, wie die einzelnen Bestandteile einer Multifile-Assembly einander zugeordnet werden.*



Die Assembly Component.dll besteht aus drei Elementen: in den Assembly Metadaten (Manifest) stehen Informationen über die Assembly selbst und über die referenzierten Ressourcen, in den Type Metadaten stehen selbstbeschreibende Informationen über die verwendeten Datentypen, und im IL Code ist der kompilierte, plattformunabhängige Zwischencode zu finden. Assemblies können ein oder mehrere Module beinhalten, welche selbst kein Manifest besitzen und somit nicht alle installiert werden können. Module und andere Ressourcen werden in den Assembly Metadaten referenziert. So ist es möglich, dass eine Assembly aus mehrere Dateien besteht.

***Aufgabe H1**: Im Hinblick auf die Speicherverwaltung existieren wesentliche Unterschiede zwischen den Klassen String und StringBuilder! Verbessern Sie (auf dem Aufgabenblatt) das folgende Programm derart, dass die Klasse StringBuilder in sinnvollem Umfang eingesetzt wird.*

Berücksichtigen Sie bei der Verwendung des StringBuilders auch seine Properties, um die Speicherverwaltung optimal zu steuern.

Begründen Sie zusätzlich kurz schriftlich (auf einem Klausurbogen) die getroffenen Veränderungen.

```
class MainClass
{
    static void Main(string[] args)
    {
        Encoding Encoder = new Encoding();
        string klartext = "Dies ist bzw. war ein Klartext.";
        string chiffrat = Encoder.Encode(klartext);
        Console.WriteLine(chiffrat);
        Console.ReadLine();
    }
}

class Encoding
{
    public string Encode(string text)
    {
        StringBuilder builder = new StringBuilder(text,text.Length);
        for (int i = (int)'z'; i >= (int)'a'; i--)
        {
            char alt = (char)i;
            char neu = (char)(i + 2);
            builder.Replace(alt, neu);
        }
        for (int i = (int)'Z'; i >= (int)'A'; i--)
        {
            char alt = (char)i;
            char neu = (char)(i + 2);
            builder.Replace(alt, neu);
        }
        return builder.ToString();
    }
}
```

Die Klasse String muss bei jeder Veränderung des Strings ein neues String-Objekt erzeugen. Das bedeutet, dass bei jedem Durchlauf der for-Schleife neuer Speicher für das Objekt allokiert werden muss. Durch den Einsatz der Klasse StringBuilder lässt sich das Programm effizienter gestalten. Objekte der Klasse StringBuilder lassen Veränderungen am enthaltenen String ohne Allokation von neuen Speicherbereichen zu. Beim Konstruktor der Klasse StringBuilder kann zusätzlich angegeben werden, wieviel Kapazität (also wieviel Speicherbereich) für das Objekt zu reservieren ist. Da sich die Länge des Strings in dem Beispiel nicht verändert, kann dieser auf die Länge des Strings gesetzt werden.

***Aufgabe H2**: Zum Kontext: Eine Assembly wird von einem Hauptprogramm verwendet (dieses muss nicht implementiert werden), welches Datensätze aus einer Datei zeilenweise einliest. Das Hauptprogramm übergibt jede Zeile der Reihe nach an die Methoden der Assembly und verarbeitet das Ergebnis weiter. Als Ergebnis wird bei den spezifischen Methoden jeweils nur ein Treffer erwartet.*

Implementieren Sie unter Verwendung regulärer Ausdrücke diese Assembly, die folgende Merkmale besitzt:

- *Die Assembly enthält eine Klasse „AdressParser“*

- *Die Klasse AdressParser enthält die folgenden statischen Methoden, deren Parameter und Rückgabewert den Typ string besitzen. Die Methoden geben jeweils das durch den Methodennamen spezifizierte Element zurück.*
  - *SearchStreet // Straße inkl. Hausnummer*
  - *SearchLastName // Nachname*
  - *SearchBirthDate // Geburtsdatum*
- *Der Eingabestring, der für alle Methoden identisch ist, entspricht dem folgenden Aufbau:*
  `Name;Adresse;Geburtsdatum`
- *Beispiele, mit denen die Assembly korrekt arbeiten soll:*
  `Meier, Hans;Heinrichstr. 34 45332 Essen;23.02.2001`
  `Müller-Lüdenscheidt, Otto;Berliner Str. 89 45123 Essen;12.04.1987`
  `Schmidt, Hans O.;Sommer Allee 1 17643 Lubmin;01.03.1917`
- *Zusätzlich ist gewünscht, dass ein fortgeschrittener Anwender über eine weitere Methode unter Angabe von Datenzeile und Suchpattern weitere Elemente aus den Textzeilen ermitteln kann. Daher soll eine allgemeine Methode bereitgestellt werden, die den Suchtext und das Pattern übergeben bekommt und mögliche Suchergebnisse (Plural!) als Rückgabewert liefert.*

Als Hilfestellung sind die folgenden Sprachbestandteile regulärer Ausdrücke gegeben:

Zeichen	
.	Beliebiges Zeichen außer \n
\n	New Line
\d	Dezimal-Zeichen
\s	White-Space-Zeichen
\w	Wortzeichen
Quantifizierer	
*	0-n
+	1-n
?	0-1
{n}	N

```
class Program
{
    static void Main(string[] args)
    {
        string[] strings = new string[]
        {
            "Meier, Hans;Heinrichstr. 123 45332 Essen;23.02.2001",
            "Müller-Lüdenscheidt, Otto;Berliner Str. 89 45123 Essen;
              12.04.1987",
            "Schmitdt, Hans O.;Sommer Allee 1 17643 Lubmin;01.03.1917"
        };
        foreach (string s in strings)
        {
            Console.WriteLine(s);
            Console.WriteLine("Strasse: " +
              AdressParser.SearchStreet(s));
            Console.WriteLine("Nachname: " +
              AdressParser.SearchLastName(s));
            Console.WriteLine("Geburtsdatum: " +
              AdressParser.SearchBirthDate(s));
            Console.WriteLine();
        }
        Console.ReadLine();
    }
}
```

```
class AdressParser
{
    static public string SearchStreet(string text)
    {
        string pattern = @"[a-zA-ZäöüÄÖÜß\-\s]+\.?\s\d+[a-zA-Z]?";
        Match match = Regex.Match(text, pattern);
        return match.ToString();
    }

    static public string SearchLastName(string text)
    {
        string pattern = @"^[\w\-]+";
        Match match = Regex.Match(text, pattern);
        return match.ToString();
    }

    static public string SearchBirthDate(string text)
    {
        string pattern = @"(\d{2}\.\d{2}\.\d{2,4})";
        Match match = Regex.Match(text, pattern);
        return match.ToString();
    }

    static public MatchCollection SearchPattern(string text, string
        pattern)
    {
        return Regex.Matches(text, pattern);
    }
}
```

***Aufgabe I2**: Zum Kontext: Eine Assembly wird von einem Hauptprogramm verwendet (dieses muss nicht implementiert werden). Das Hauptprogramm nutzt diese Assembly zum vaildieren von Benutzereingaben in einer Webanwendung. Die Validierung soll innerhalb der Assembly über Reguläre Ausdrücke erfolgen. Um die Komponente einfach nutzbar zu machen, soll nach außen nichts von der internen Technologie sichtbar sein.*

Implementieren Sie unter Verwendung regulärer Ausdrücke die beschriebene Assembly, die folgende Merkmale besitzt:

- *Die Assembly enthält eine Klasse „Validator“*
- *Die Klasse Validator enthält die folgenden statischen Methoden, deren Parameter die zu prüfende Zeichenkette ist und deren Rückgabewerte den Typ Boolean besitzen, um anzuzeigen, ob der Paramterausdruck dem gewünschten Schema entspricht.*
  - *IsEmailAddress Schema: z.B. hugo.mueller@mail-server.de*
  - *IsTelephoneNumber Schema: deutsche Telefonnummer inkl. Vorwahl durch „/“ getrennt.*
  - *IsPlz Schema: Deutsche Postleitzahl mit dem optionalen Präfix „D-„*
  - *IsUrl Schema: http://host-name.sub-domain.domain.topleveldomain/directory/file.html*

Zusätzlich soll eine weitere Methode ParseForEmailAddresses implementiert werden, die aus einer beliebigen gegebenen Zeichenkette Emailadressen herausfiltert und zurückgibt. Wenn in der Zeichenkette mehr als ein Treffer enthalten ist, sollen entsprechend mehrere Ergebnisse (Emailadressen) zurückgegeben werden.

Parameter und Rückgabewerte sind vom Typ string bzw. string[].

Geben Sie in jeder Methode innerhalb eines Kommentars mindestens je ein Beispiel für einen erfolgreichen und einen erfolglosen Treffer an.

Als Hilfestellung sind die folgenden Sprachbestandteile regulärer Ausdrücke gegeben:

Zeichen	
.	Beliebiges Zeichen außer \n
\n	New Line
\d	Dezimal-Zeichen
\s	White-Space-Zeichen
\w	Wortzeichen
Quantifizierer	
*	0-n
+	1-n
?	0-1
{n}	N

```
class Program
{
    static void Main(string[] args)
    {
        string[] strings = new string[]
        {
            "hugo.mueller@mail-server.de",
            "hugo-mueller@web.3",
            "0208/321423",
            "122/23431",
            "4647",
            "D-23452",
            "http://www.subdomain.domain.de/
              directory/directory/file.html",
            "http//www.subdomain.de/file.html",
            "http://www.domain.de/directory/file.html"
        };
        Console.WriteLine(strings[0]);
        Console.WriteLine("Ist E-Mail-Adresse?: " +
              Validator.IsEmailAddress(strings[0]));
        Console.WriteLine(strings[1]);
        Console.WriteLine("Ist E-Mail-Adresse?: " +
              Validator.IsEmailAddress(strings[1]));
        Console.WriteLine(strings[2]);
        Console.WriteLine("Ist Telefonnummer?: " +
              Validator.IsTelephoneNumber(strings[2]));
        Console.WriteLine(strings[3]);
        Console.WriteLine("Ist Telefonnummer?: " +
              Validator.IsTelephoneNumber(strings[3]));
        Console.WriteLine(strings[4]);
        Console.WriteLine("Ist PLZ?: " + Validator.IsPlz(strings[4]));
        Console.WriteLine(strings[5]);
        Console.WriteLine("Ist PLZ?: " + Validator.IsPlz(strings[5]));
        Console.WriteLine(strings[6]);
        Console.WriteLine("Ist URL?: " + Validator.IsUrl(strings[6]));
        Console.WriteLine(strings[7]);
        Console.WriteLine("Ist URL?: " + Validator.IsUrl(strings[7]));
        Console.WriteLine(strings[8]);
        Console.WriteLine("Ist URL?: " + Validator.IsUrl(strings[8]));
        string parse = "he fs@w34.de dsd@ mnb@dvn.de @sfdf";
        string[] erg = Validator.ParseForEmailAddresses(parse);
        foreach (string s in erg)
```

```
            Console.WriteLine(s);
        Console.ReadLine();
    }
}

class Validator
{
    static public bool IsEmailAddress(string text)
    {
        string pattern = @"^[\w\-\.]+@[\w\-\.]+\.[a-zA-Z]{2,4}$";
        return Regex.IsMatch(text, pattern);
    }

    static public bool IsTelephoneNumber(string text)
    {
        string pattern = @"^0\d{2,4}\/\d{3,7}$";
        return Regex.IsMatch(text, pattern);
    }

    static public bool IsPlz(string text)
    {
        string pattern = @"^(D-)?\d{5}$";
        return Regex.IsMatch(text, pattern);
    }

   static public bool IsUrl(string text)
    {
        string pattern = @"^[a-zA-Z]+:\/\/\w+\.(\w+\.)*\w+\/
              ((\w+\/)?)+\w+\.\w{3,4}";
        return Regex.IsMatch(text, pattern);
    }

   static public string[] ParseForEmailAddresses(string text)
   {
       string pattern = @"[\w\.\-]+@[\w\-\.]+\.[a-zA-Z]{2,4}";
       MatchCollection matches = Regex.Matches(text, pattern);
       string[] erg = new string[matches.Count];
       for (int i = 0; i < matches.Count; i++)
           erg[i] = matches[i].ToString();
       return erg;
   }
}
```

***Aufgabe J3**: Erläutern Sie am Beispiel der Zeichenkettenverarbeitung, wie sich Entscheidungen innerhalb der Softwareentwicklung auf die folgenden Bereiche auswirken können:*

- *Entwicklung*
- *Ausführung (z.B. Laufzeit & Speicher)*
- *Wartung*

Berücksichtigen Sie bzgl. der Zeichenkettenverarbeitung die Klassen String, StringBuilder und RegularExpression bzw. die mit diesen Klassen verbundenen Konzepte.

In der Sprache C war Stringverarbeitung für die Entwickler sehr aufwändig. Durch die .NET-Basisklassen String, StringBuilder und den Namespace RegularExpressions lassen sich nun auf einfachem Wege mächtige Programme zur Zeichenkettenverarbeitung entwickeln. Die Klasse String stellt eine Vielzahl an Methoden bereit, mit denen Strings ausgewertet und verarbeitet werden

können. Zur effektiven Speicherverwaltung bei Stringänderungen dient die Klasse StringBuilder. Sie hat im Vergleich zur String-Klasse den Vorteil, dass nicht bei jeder Änderung eines Zeichens ein neuer Speicherbereich allokiert werden muss. Reguläre Ausdrücke stellen eine eigenständige Sprache dar. Mit ihnen lassen sich mächtige Programme zur Suchmustererkennung in wenigen Zeilen Code ausdrücken. Mit der sinnvollen Kombination der drei Sprachkonzepte lassen sich performante und leicht wartbare Programme schreiben.

***Aufgabe K1**: Erläutern Sie die wesentlichen Aspekte des Speichermanagements in C#. Gehen Sie dabei insbesondere auf Begriffe wie "Managed Heap" und "Stack" ein. Unterstützen Sie Ihre Ausführungen an geeigneten Stellen durch Abbildungen.*

Der Speicherbereich eines C#-Programms teilt sich hauptsächlich in zwei Bereiche auf: dem Stack und dem Managed Heap. Auf dem Stack werden alle Wertetypen gespeichert. Der zuletzt erzeugte Datentyp befindet sich stets ganz oben auf dem Stack und wird auch als erstes wieder entfernt, wenn die Variable des Datentyps den Fokus verliert. Korrekterweise muss gesagt werden, dass der Wert nicht aus dem Speicher entfernt wird, sondern lediglich der Stack Pointer eine Position zurückgesetzt wird, um den freien Speicherbereich anzuzeigen. Referenztypen werden nicht auf dem Stack, sondern auf dem Heap gespeichert. Auf dem Stack befindet sich lediglich eine Referenz auf die Speicheradresse im Heap. Ist der Heap voll, so wird er vom Garbage Collector aufgeräumt. Dabei werden alle Objekte, für die es keine Referenz mehr gibt, entfernt und Speicherblöcke zusammengelegt.

***Aufgabe K2**: Erläutern Sie die Grundlagen der „Pointer“! Notieren und beschreiben Sie die syntaktischen Bestandteile der Sprache C#, die beim Einsatz von Pointern verwendet werden. Begründen Sie, warum Pointer innerhalb der Sprache C# enthalten sind.*

Pointer sind Zeiger auf Speicheradressen. Sie erlauben den schnellen Zugriff und eine einfache Manipulation von Wertetypen. Um Pointer benutzen zu können, muss der jeweilige Code-Abschnitt als unsafe deklariert werden, außerdem muss dem Compiler mitgeteilt werden, dass unsafe-Code auszuführen ist. Ein Pointer für einen Integer-Datentyp wird z.B. so deklariert:

```
int x = 10;
int* pInt = &x;
```

Ähnlich wie bei Interfaces wird auch bei Pointern bevorzugt die Hungarian Notation verwendet, d.h. das ein p dem Pointernamen vorangestellt wird.

Mit Hilfe des &-Operators kann auf die Speicheradresse eines Wertetyps zugegriffen werden, der *-Operator liefer den Inhalt eines Wertetyps.

```
*pInt = 20;
```

Pointer können direkt auf Member eines Structs zeigen. Dafür stellt C# eine dedizierte Syntax bereit:

```
long* pLong = &(pStruct->X);
```

Es ist nicht möglich, Pointer auf Klassen zeigen zu lassen. Jedoch können Pointer auf Wertetypen zeigen, die Klassenmember sind. Dafür muss das fixed-Schlüsselwort angegeben werden:

```
fixed (long* pObject = &(myObject.X)) {}
```

Das Pointer-Konzept wurde im .Net-Framework aus zwei Hauptgründen aufgenommen. Zum einen erlaubt es sehr effiziente Programme, da kein Overhead für managed code benötigt wird. Zum anderen wird die Abwärtskompatibilität gewährleistet. Zwar bietet das .Net-Framework eine Vielzahl von Basisklassen, um den Zugriff auf die Windows-API zu kapseln. Jedoch kann es trotzdem manchmal nötig sein, direkt auf API-Funktionen zuzugreifen, die als Parameter häufig einen Pointer erwarten.

***Aufgabe K3**: Erläutern Sie das Vorgehen des Garbage Collectors anhand der folgenden Abbildungen und geben Sie mindestens zwei Gründe an, warum für den Garbage Collector dieses Vorgehen gewählt wurde.*

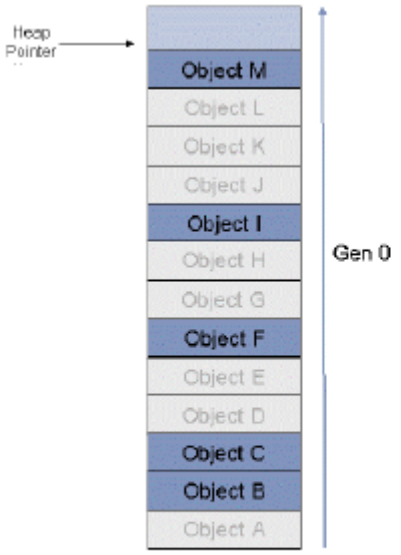


Abbildung 1

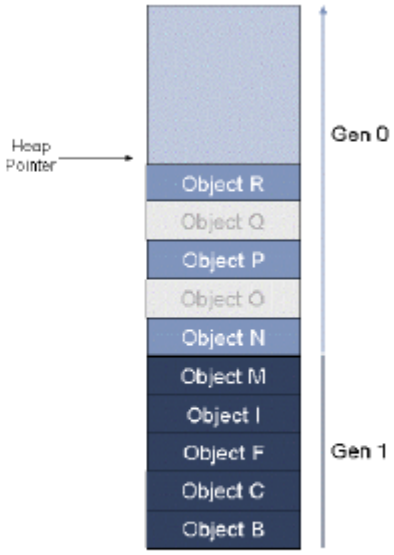


Abbildung 2

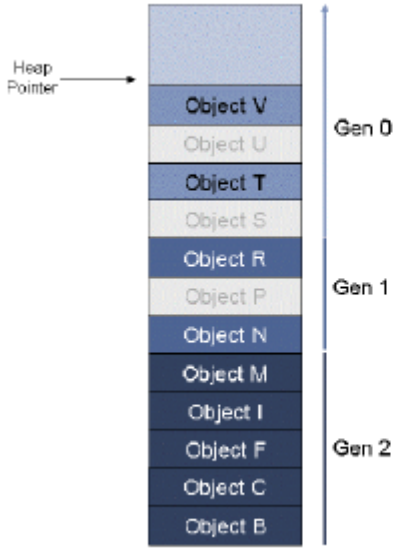


Abbildung 3

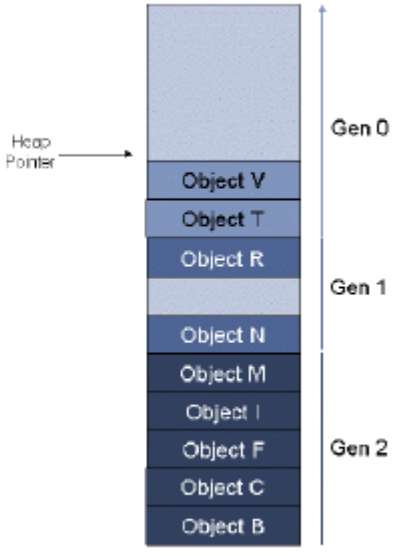


Abbildung 4

Auf dem Heap werden Referenztypen gespeichert. Der Heap Pointer zeigt den nächsten freien Speicherbereich an. In Abbildung 1 ist der Speicherbereich für den Heap fast voll, so dass der Garbage Collector aufgerufen wird. Alle Objekte, auf die es keine Referenz mehr gibt, werden entfernt. Da der Speicherbereich nun sehr fragmentiert ist, werden die noch vorhandenen Objekte der Generation 0 zusammengefasst und als Generation 1 gekennzeichnet. Alle neu hinzugefügten Objekte werden nun wiederum als Generation 0 auf dem Heap abgelegt (Abbildung 2). Wird der Gargabe Collector wieder ausgeführt, so werden wieder alle nicht mehr referenzierten Objekte gelöscht, die Objekte der Generation 1 werden zu Generation 2 und die der Generation 0 werden zu Generation 1. Wie man in der Abbildung 4 sieht, defragmentiert der Garbage Collector zunächst nur die Elemente der Generation 0. Erst wenn das nicht mehr genügend freien Speicherplatz schafft, räumt er die nächst ältere Generation auf. Da es nur drei Generationen gibt, verbleiben die Objekte der Generation 2 solange in dieser Generation, bis auf sie keine Referenz mehr existiert (hier liegt übrigens ein Fehler in der Abbildung der Aufgabensammlung!).

Die Gründe für dieses Vorgehen des Garbe Collectors sind vielseitig:

- es ist performanter, nur einen Teil des Speicherbereichs zu defragmentieren
- es wird davon ausgegangen, dass neuere Objekte eine kürzere Lebenszeit besitzen im Vergleich zu Objekten, die schon zu Beginn des Programms erstellt wurden
- neue Objekte gehören häufig zusammen, deswegen werden diese immer in zusammenhängede Speicherbereiche geschrieben

---

***Aufgabe L2**: Beschreiben Sie im Detail, welche Vorgänge im Bezug auf das Speichermanagement während der Ausführung des folgenden Code-Abschnitts stattfinden.*

```
int x = 20;
{
      int y = 30;
}
int * pX = &x;
*pX = 50 ;
```

Zunächst wird ein Objekt des Datentyps int auf dem Stack abgelegt. Diesem wird der Wert 20 zugeordnet. Der Stackpointer zeigt auf dieses Objekt, da es das oberste Element des Stacks ist. Im nächsten Schritt wird ein neues Objekt des Datentyps int erzeugt. Wiederum wird der benötigte Speicherplatz im Stack allokiert und der Stackpointer eine Position weitergesetzt. Im Speicherbereich des neuen Objekts wird der Wert 30 eingetragen. Nach dem Ende des Codeblocks verliert die Variable y den Fokus, deshalb wird der Stackpointer eine Position zurückgesetzt, so dass der Speicherbereich, wo vormals die Variable y gespeichert war, als frei gekennzeichnet wird. Im nächsten Schritt wird ein int-Pointer angelegt, der auf dem Stack gespeichert wird, womit der Stack-Pointer wieder eine Position weitergesetzt wird. Der int-Pointer beinhaltet statt einem Wert für einen Wertetyp lediglich eine Referenz (also eine Speicheradresse) auf einen Wertetypen. Dem int-Pointer wird die Speicheradresse der Variable x über den address-Operator & zugewiesen. Über den indirection-Operator * wird auf den Inhalt des Speicherbereichs zugegriffen, auf den der Pointer zeigt. Da der Pointer auf die Speicheradresse von x zeigt, wird der Wert für x von 20 auf 50 geändert.

***Aufgabe L3**: Erläutern Sie im Hinblick auf die Speicherverwaltung die Unterschiede zwischen den Klassen String und Stringbuilder! Gehen Sie dabei auch auf die Properties des Stringbuilders ein!*

In welchen Situationen ist der Einsatz eines Strings bzw. eines Stringbuilders vorzuziehen?

Unterstützen Sie Ihre Erläuterungen bei Bedarf durch eine Abbildung!

Die Klasse String bietet eine Vielzahl von Möglichkeiten zum Ausgaben und Verarbeiten von Strings. Jedoch ist ein String-Objekt unveränderbar. Das heißt, das bei Veränderungen an einem String-Objekt stets ein neues String-Objekt angelegt werden muss. Hier bietet die Klasse StringBuilder Abhilfe. Sie allokiert nicht nur den Speicherplatz, den ein String benötigt, sondern etwas mehr (kann über das Property Capacity angegeben werden) und kann auch dynamisch vergrößert werden. Dadurch muss nicht für jede Veränderung an einem String ein neues Objekt erstellt werden. Für das Ausgeben und Übergeben von Strings ist die String-Klasse jedoch die geeignete Wahl, da hier mehr Funktionen für den Nutzer angeboten werden und kein unnötiger Speicherplatz allokiert wird. Da der Speicherplatz für einen StringBuilder frei wählbar ist, kann aber auch dieser so gewählt werden, dass er genau so groß ist wie für den beinhalteten String (Capacity = Length).

***Aufgabe N1**: Erläutern Sie, wo bei der Entwicklung mit dem .NET-Framework Metadaten zum Einsatz kommen. Gehen Sie dabei u.a. die folgenden Aspekte ein:*

- *Welche Relevanz besitzen Metadaten im Kontext der Software-Entwicklung?*
- *Welche Metadaten werden bei der Entwicklung mit C# „automatisch“ bereitgestellt?*
- *Über welche Möglichkeiten zur Nutzung und zur Einflussnahme verfügt der Entwickler?*

Metadaten reichern Programme um zusätzliche Informationen an, so dass sich Programmelemente (Klassen, Assemblies) selbst beschreiben können. Die Klasse Type enthält eine Vielzahl von Methoden und Properties, mit denen Datentypen in C# über das Konzept der Reflektion ausgelesen werden können. Analog dazu enthält die Klasse Assembly die Möglichkeit, Informationen über Assemblies bereitzustellen. Damit können Entwickler andere Assemblies in ihr Programm laden und darin enthaltene öffentliche Member ausführen. Zusätzlich zu den vom .Net-Framework bereitgestellten Metadaten können Entwickler benutzerspezifische Attribute ihren Programmelementen hinzufügen, die ebenfalls über das Konzept der Reflektion ausgewertet und verwendet werden können. Diese Attribute können z.B. den Entwicklungsprozess unterstützen, in dem mit ihnen Änderungen an Programmteilen dokumentiert und ausgewertet werden können.

***Aufgabe N2**: Im .NET-Framework wird dem Entwickler ein Mechanismus zur Fehlerbehandlung bereitgestellt.*

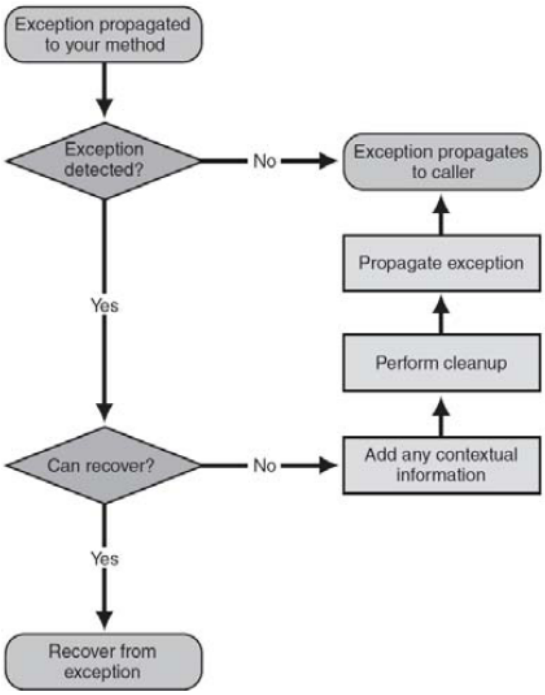
- *Skizzieren Sie die wesentlichen Aspekte dieses Mechanismus.*
- *Gehen Sie dabei sowohl auf systemseitige als auch auf benutzerdefinierte Fehlerbehandlung ein.*
- *Formulieren Sie ein kurzes Code-Beispiel zur Fehlerbehandlung, um die in C# eingesetzte Syntax darzustellen.*

Ein Fehler ist in C# immer ein Objekt der Klasse System.Exception. Von dieser Klasse werden zwei Hauptklassen abgeleitet: SystemException für allgemeine, in jedem Programm auftretende Fehler und ApplicationException für sehr spezifische Fehler, die meist nur in bestimmten Programmen auftreten. Die Fehlerbehandlung läuft in der Regel in bis zu drei Schritten ab. Zunächst einmal wird

der Codeblock, indem potentiell Ausnahmen auftreten können, in einen try-Block geschrieben. Tritt dort ein Fehler auf, durchsucht der Compiler, ob es einen entsprechenden catch-Block gibt. Der erste catch-Block, der die Ausnahme behandelt, wird ausgeführt. Zusätzlich kann noch ein finally-Block definiert werden, der immer durchgeführt wird, unabhängig vom Auftreten von Fehlern. Hier können Säuberungsarbeiten wie das Schließen von externen Ressourcen durchgeführt werden. Es ist möglich, mehrere catch-Blöcke hintereinander zu definieren, so dass verschiedene Ausnahmen verschieden behandelt werden können. Dabei ist jedoch darauf zu achten, die speziellsten Fehler zuerst zu behandeln, da der Compiler stets die erste catch-Anweisung nimmt, auf den der Fehler zutrifft. Des weiteren ist es möglich, geschachtelte try-catch-Blöcke zu schreiben. Wird im inneren Block eine Ausnahme erzeugt und nicht behandelt, so wird diese an den äußeren Block weitergegeben. Der innere finally-Block wird aber in jedem Fall noch vorher ausgeführt.

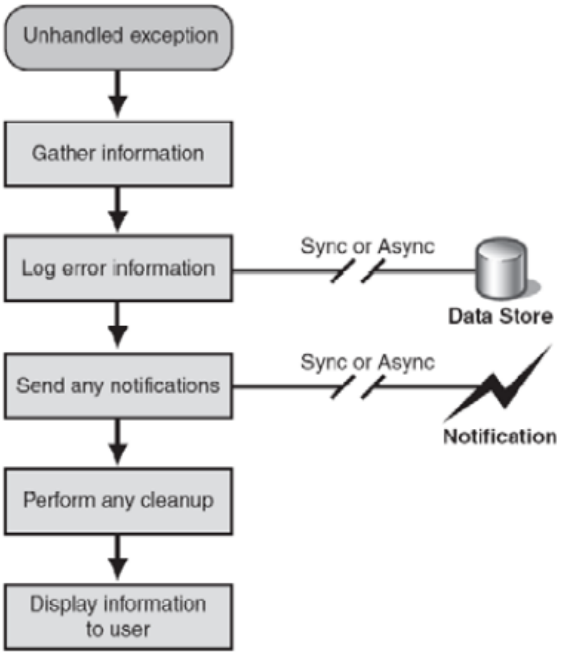
```
static void Main(string[] args)
{
    int[] array = new int[4];
    try
    {
        array[4] = 1;
    }
    catch (IndexOutOfRangeException e)
    {
        Console.WriteLine(e.Message);
    }
    finally
    {
        Console.WriteLine("Finally Statement.");
    }
}
```

***Aufgabe P1**: Im .NET-Framework wird dem Entwickler ein Mechanismus zur Fehlerbehandlung bereitgestellt. Skizzieren Sie zunächst kurz die grundlegenden Aspekte der Fehlerbehandlung im .NET Framework! Erläutern Sie anschließend anhand der folgenden Abbildung ausführlich die Vorgehensweise bei der Implementierung dieses Mechanismus! Erwähnen Sie dabei ebenfalls konkrete inhaltliche Anwendungsgebiete, um die Ausführungen zu stützen! (Code-Beispiele sind nicht erforderlich).*



In einer Methode tritt ein Fehler auf. Wird dieser nicht in einem try-catch-Block behandelt, so wird der Fehler zu der aufrufenden Methoden zurückgereicht. Andernfalls sucht der Compiler nach einem entsprechenden catch-Bereich, um die Ausnahme zu behandeln. Wird ein entsprechender catch-Block gefunden, kann die Ausnahme dort gegebenfalls verarbeitet werden und das Programm kann anschließend wie gewohnt weiterarbeiten. Es ist aber auch möglich, dass die Ausnahme an dieser Stelle nicht behandelt werden kann oder soll. Dann wird die Ausnahme mit weiteren Informationen angereichert, es wird der nachfolgende finally-Block für Aufräumarbeiten ausgeführt und anschließend wird die Ausnahme zur aufrufenden Methode weitergereicht.

***Aufgabe P2**: Die Abbildung zeigt die Bearbeitungsschritte im Falle eines nicht behandelten Fehlers. Erläutern Sie die einzelnen Schritte und nennen Sie einige Informationsarten, die im Fehlerfall erfasst werden, um hilfreiche Meldungen zusammenzustellen!*



Wenn ein unbehandelter Fehler solange an die aufrufenden Methoden weitergeleitet wird, bis der letzte Punkt erreicht ist, wo das Programm den Fehler verarbeiten kann und anschließend die Kontrolle an den Endbenutzer zurückgegeben wird, hat das Programm die Aufgabe, zunächst so viele Informationen wie möglich zu sammeln, die die Ursache des Fehlers beschreiben könnten. Darunter fällt die Art des Fehlers, die ausführende Assembly oder das aktuelle Datum. Die Informationen werden dann in eine persistente Datenbank gespeichert, um Log-Informationen bereitzustellen. Anschließend werden an alle beteiligten Systeme (z.B. an eine Monitoring-Applikation) Benachrichtigungen geschickt, damit die Ausnahme überhaupt bemerkt werden kann. Das Speichern der Log-Informationen und das Versenden der Benachrichtigungen kann dabei je nach Anwendungsfall synchron (das heißt, es wird auf eine Antwort gewartet) oder asynchron erfolgen. Im Anschluss daran werden Aufräumarbeiten durchgeführt, das heißt es werden z.B. Datei- oder Netzwerkhandle geschlossen. Und zu guter Letzt wird der Endbenutzer über die Ausnahme informiert.

***Aufgabe Q2**: Welche Zielsetzung verfolgt die Enterprise Library?*

Welche Bestandteile umfasst die Enterprise Library? (Bitte aggregieren und nur durch zwei charakterisierende Beispiele ergänzen. Eine Nennung jedes einzelnen Elementes ist nicht erforderlich.)

Die Enterprise Library ist kein Bestandteil des .NET-Frameworks, sondern eine Erweiterung. Es bietet eine Vielzahl von wiederverwendbaren, generischen Codeabschnitten. Sie stellen sozusagen die Best Practices aus vielen Projekten dar und sollen helfen, Projekte wohlgeformt und effizient zu entwickeln. Die Enterprise Library besteht aus den Application Blocks Caching, Data Access, Logging, Exception Handling, Policy Injection, Validation, Security und Cryptography. Zusätzlich gibt es die Core-Funktionen Config Helpers & Design, Instrumentation und Object Builder.

***Aufgabe R1**: Implementieren Sie eine Assembly, die einen Bestellauftrag abbildet.*

Innerhalb des Namespaces Finance sollen die Klassen PurchaseOrder, Address und OrderItem implementiert werden. Beachten Sie bei der Implementierung die zweite Teilaufgabe, damit Sie die Anforderungen frühzeitig bedenken oder vor den jeweiligen Elementen entsprechend eine Zeile frei lassen.

Für die Klassen gelten die folgenden Anforderungen:

- *Die Klasse Address ist gegeben und muss nicht implementiert werden. Sie enthält die folgenden öffentlichen Felder vom Typ string*
  - *Name, Street, Zip, City*
- *Die Klasse OrderItem enthält die folgenden Felder*
  - *ItemName vom Typ string*
  - *Description vom Typ string*
  - *UnitPrice vom Typ decimal*
  - *Quantity vom Typ int*
- *Die Klasse OrderItem enthält das folgende Property zur Berechnung des Gesamtpreises*
  - *LineTotal vom Typ decimal*
- *Die Klasse PurchaseOrder enthält dir folgenden Felder*

  - *ShipTo vom Typ Address*
  - *OrderDate vom Typ string*
  - *OrderedItems vom Typ Array von OrderItem*
  - *ShipCost vom Typ decimal*
- *Die Klasse PurchaseOrder enthält die folgenden Properties zur Berechnung der Gesamtpreise*
  - *SubTotal vom Typ decimal (Summe aller enthaltenen OrderItems)*
  - *TotalCost vom Typ decimal (Summe von SubTotal und ShipCost)*
- *Alle Felder können öffentlich verfügbar sein. Es müssen keine zusätzlichen Properties implementiert werden.*

***Aufgabe R2**: Implementieren Sie innerhalb der Klasse PurchaseOrder zwei Methoden zur Serialisierung bzw. Deserialisierung der Klasse. Die Methoden sollen den folgenden Signaturen entsprechen:*

- *void ExportToXml(string fileName)*
- *PurchaseOrder ImportFromXml(string fileName)*

Passen Sie zusätzlich die Implementierung der Klasse an, damit die Ausgabe des XML-Dokumentes der Abbildung 1 entspricht. Ohne den Einsatz von Attributen entspricht die Ausgabe der Abbildung 2. In der Tabelle können Sie mögliche Attribute nachschlagen.

```
<?xml version="1.0" encoding="utf-8" ?>
- <PurchaseOrder xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" OrderDate="2005-08-02">
    <ShipTo Name="Hugo Müller" Street="Zum Irrtum 12" Zip="37121" City="Velbert-Neviges" />
  - <Items>
      <Item ItemName="Asus MyPal 610" Description="PDA" UnitPrice="270.65" Quantity="1" />
      <Item ItemName="Rey Color A4 100" UnitPrice="8.99" Quantity="5" />
    </Items>
    <ShipCost>4.55</ShipCost>
  </PurchaseOrder>
```

Abbildung 1 - Zielformat der XML-Datei

```
<?xml version="1.0" encoding="utf-8" ?>
- <PurchaseOrder xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ShipTo Name="Hugo Müller" Street="Zum Irrtum 12" Zip="37121" City="Velbert-Neviges" />
    <OrderDate>2005-08-02</OrderDate>
  - <OrderItems>
    - <OrderItem>
        <ItemName>Asus MyPal 610</ItemName>
        <Description>PDA</Description>
        <UnitPrice>270.65</UnitPrice>
        <Quantity>1</Quantity>
      </OrderItem>
    - <OrderItem>
        <ItemName>Rey Color A4 100</ItemName>
        <UnitPrice>8.99</UnitPrice>
        <Quantity>5</Quantity>
      </OrderItem>
    </OrderItems>
    <ShipCost>4.55</ShipCost>
  </PurchaseOrder>
```

Abbildung 2 - Format der XML-Datei ohne Einsatz von Attributen

```
class Program
{
    static void Main(string[] args)
    {
        Address adresse = new Address("Hugo Müller","Zum Irrtum 12",
            "37121","Velbert-Neviges");
```

```
            PurchaseOrder auftrag = new PurchaseOrder(adresse,"2005-08-02",
                  4.55m);
            OrderItem item1 = new OrderItem("Asus MyPal 610","PDA",
                  270.65m,1);
            OrderItem item2 = new OrderItem("Rey Color A4 100",8.99m,5);
            auftrag.OrderedItems[0] = item1;
            auftrag.OrderedItems[1] = item2;
            auftrag.ExportToXML("purchaseorder.xml");
            PurchaseOrder auftrag2 =
                  auftrag.ImportFromXML("purchaseorder.xml");
        }
    }

    public class PurchaseOrder
    {
        public Address ShipTo;
        public string OrderDate;
        [XmlArray("Items")]
        [XmlArrayItem("Item")]
        public OrderItem[] OrderedItems;
        public decimal ShipCost;
        public decimal SubTotal
        {
            get
            {
                decimal subTotal = 0;
                foreach (OrderItem item in OrderedItems)
                    subTotal += item.UnitPrice;
                return subTotal;
            }
        }
        public decimal TotalCost
        {
            get { return SubTotal + ShipCost; }
        }
        public PurchaseOrder() {}
        public PurchaseOrder(Address shipTo, string orderDate, decimal
            shipCost)
        {
            ShipTo = shipTo;
            OrderDate = orderDate;
            ShipCost = shipCost;
            OrderedItems = new OrderItem[2];
        }

        public void ExportToXML(string fileName)
        {
            XmlSerializer mySerializer = new
                  XmlSerializer(typeof(PurchaseOrder));
            StreamWriter myWriter = new StreamWriter(fileName);
            mySerializer.Serialize(myWriter, this);
            myWriter.Close();
        }

        public PurchaseOrder ImportFromXML(string fileName)
        {
            XmlSerializer mySerializer = new
                  XmlSerializer(typeof(PurchaseOrder));
            FileStream myFileStream = new FileStream(fileName,
                  FileMode.Open);
            return (PurchaseOrder) mySerializer.Deserialize(myFileStream);
        }
```

```
}

public class Address
{
    [XmlAttribute]
    public string Name;
    [XmlAttribute]
    public string Street;
    [XmlAttribute]
    public string Zip;
    [XmlAttribute]
    public string City;
    public Address() {}
    public Address(string name, string street, string zip, string city)
    {
        Name = name;
        Street = street;
        Zip = zip;
        City = city;
    }
}

public class OrderItem
{
    [XmlAttribute]
    public string ItemName;
    [XmlAttribute]
    public string Description;
    [XmlAttribute]
    public decimal UnitPrice;
    [XmlAttribute]
    public int Quantity;
    public decimal LineTotal
    {
        get { return UnitPrice * Quantity; }
    }
    public OrderItem() { }
    public OrderItem(string itemName, decimal unitPrice, int quantity)
    {
        ItemName = itemName;
        UnitPrice = unitPrice;
        Quantity = quantity;
    }
    public OrderItem(string itemName, string description, decimal
        unitPrice, int quantity)
    {
        ItemName = itemName;
        Description = description;
        UnitPrice = unitPrice;
        Quantity = quantity;
    }
}
```

***Aufgabe S1**: Implementieren Sie eine Assembly, die das Pendant zum abgebildeten XML-Dokument bildet. Das XML-Dokument entspricht einem Austauschformat mit einem internationalen Parterunternehmen. Im Gegensatz zum englischsprachigen XML-Dokument erfolgt die Implementierung unter Berücksichtigung der unternehmens-internen Namensgebung, die deutsche Bezeichner für Klassen, Felder, etc. vorschreibt.*

Berücksichtigen Sie bei der Implementierung die Einordnung in einen geeigneten Namespace, eine geeignete Bezeichnung der Felder, Klassen, etc. Treffen Sie für komplexe Elemente der XML-Struktur entsprechende Implementierungsentscheidungen, um eine möglichst hohe Deckung zwischen XML-Struktur und Klasse zu erreichen.

Beachten Sie bei der Implementierung, dass einige Einträge innerhalb einer Klasse als Methoden bzw. Properties bereitgestellt werden und dynamisch berechnet werden sollen. Treffen Sie angemessene Entscheidungen bzgl. der Datentypen. Es ist nicht erforderlich, einfache Felder als Properties zu kapseln.

Die Klasse soll zwei Methoden bereitstellen, die die Serialisierung und Deserialisierung kapseln. (ExportToXml(string fileName); ImportFromXml(string fileName))

Nutzen Sie XML-Attribute um zu erreichen, dass die Serialisierung dem abgebildeten Ergebnis entspricht.

```xml
<?xml version="1.0" encoding="utf-8"?>
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.cpandl.com">
    <ShipTo Surname="Schuler" Forename="Peter M.">
        <Street>Universitätsstr. 9</Street>
        <City>Essen</City>
        <Zip>45141</Zip>
    </ShipTo>
    <OrderDate>2006-02-09</OrderDate>
    <Items>
        <OrderedItem>
            <ItemName>Braun WK210 Sahara</ItemName>
            <Description>Wasserkocher - kabellos</Description>
            <UnitPrice>30.00</UnitPrice>
            <Quantity>5</Quantity>
            <LineTotal>150.00</LineTotal>
        </OrderedItem>
        <OrderedItem>
            <ItemName>Philips HP 4841 Compact</ItemName>
            <Description>Haartrockner - kabelgebunden</Description>
            <UnitPrice>12.50</UnitPrice>
            <Quantity>2</Quantity>
            <LineTotal>25.00</LineTotal>
        </OrderedItem>
    </Items>
    <SubTotal>175.00</SubTotal>
    <ShipCost>6.95</ShipCost>
    <TotalCost>181.95</TotalCost>
</PurchaseOrder>
```

Abbildung 3 - Zielformat der XML-Datei

```
class Program
{
    static void Main(string[] args)
    {
        Adresse adresse = new Adresse("Müller", "Hugo", "Zum Irrtum
              12", "37121", "Velbert-Neviges");
        Auftrag auftrag = new Auftrag(adresse, "2005-08-02", 4.55m);
        RechnungsPosten posten1 = new RechnungsPosten("Asus MyPal 610",
              "PDA", 270.65m, 1);
```

```
            RechnungsPosten posten2 = new RechnungsPosten("Rey Color A4
                  100", 8.99m, 5);
            auftrag.BestelltePosten[0] = posten1;
            auftrag.BestelltePosten[1] = posten2;
            auftrag.ExportToXML("purchaseorder.xml");
            Auftrag auftrag2 = auftrag.ImportFromXML("purchaseorder.xml");
        }
    }

    [XmlRoot("PurchaseOrder")]
    public class Auftrag
    {
        [XmlElement("ShipTo")]
        public Adresse LieferAdresse;
        [XmlElement("OrderDate")]
        public string LieferDatum;
        [XmlArray("Items")]
        [XmlArrayItem("OrderedItem")]
        public RechnungsPosten[] BestelltePosten;
        [XmlElement("SubTotal")]
        public decimal NettoKosten
        {
            get
            {
                decimal nettoKosten = 0;
                foreach (RechnungsPosten posten in BestelltePosten)
                    nettoKosten += posten.StueckGesamtPreis;
                return nettoKosten;
            }
            set { }
        }
        [XmlElement("ShipCost")]
        public decimal VersandKosten;
        [XmlElement("TotalCost")]
        public decimal Gesamtkosten
        {
            get { return NettoKosten + VersandKosten; }
            set { }
        }
        public Auftrag() { }
        public Auftrag(Adresse lieferAdresse, string lieferDatum, decimal
            versandKosten)
        {
            LieferAdresse = lieferAdresse;
            LieferDatum = lieferDatum;
            VersandKosten = versandKosten;
            BestelltePosten = new RechnungsPosten[2];
        }

        public void ExportToXML(string fileName)
        {
            XmlSerializer mySerializer = new
                  XmlSerializer(typeof(Auftrag));
            StreamWriter myWriter = new StreamWriter(fileName);
            mySerializer.Serialize(myWriter, this);
            myWriter.Close();
        }

        public Auftrag ImportFromXML(string fileName)
        {
            XmlSerializer mySerializer = new
                  XmlSerializer(typeof(Auftrag));
```

```
            FileStream myFileStream = new FileStream(fileName,
                  FileMode.Open);
            Auftrag auftrag =
                  (Auftrag)mySerializer.Deserialize(myFileStream);
            myFileStream.Close();
            return auftrag;
        }
    }

    public class Adresse
    {
        [XmlAttribute("Surname")]
        public string NachName;
        [XmlAttribute("Forename")]
        public string VorName;
        [XmlElement("Street")]
        public string Strasse;
        [XmlElement("City")]
        public string Stadt;
        [XmlElement("Zip")]
        public string Plz;
        public Adresse() { }
        public Adresse(string nachName, string vorName, string strasse,
            string plz, string stadt)
        {
            NachName = nachName;
            VorName = vorName;
            Strasse = strasse;
            Plz = plz;
            Stadt = stadt;
        }
    }

    public class RechnungsPosten
    {
        [XmlElement("ItemName")]
        public string PostenName;
        [XmlElement("Description")]
        public string Beschreibung;
        [XmlElement("UnitPrice")]
        public decimal StueckPreis;
        [XmlElement("Quantity")]
        public int Anzahl;
        [XmlElement("LineTotal")]
        public decimal  StueckGesamtPreis
        {
            get { return StueckPreis * Anzahl; }
            set { }
        }
        public RechnungsPosten() { }
        public RechnungsPosten(string postenName, decimal stueckPreis, int
            anzahl)
        {
            PostenName = postenName;
            StueckPreis = stueckPreis;
            Anzahl = anzahl;
        }
        public RechnungsPosten(string postenName, string beschreibung,
            decimal stueckPreis, int anzahl)
        {
            PostenName = postenName;
            Beschreibung = beschreibung;
```

```
            StueckPreis = stueckPreis;
            Anzahl = anzahl;
        }
    }
```

***Aufgabe T1**: Implementieren Sie eine Assembly „Warehouse“, die für eine Software eingesetzt werden soll, die die oben abgebildete Struktur zur Verwaltung von Waren einsetzt.*

Die ersten 4 Ebenen sollen als Klassenhierarchie implementiert werden; die fünfte Ebene stellt die Instanzen dar; die letzte Ebene die Felder der Klasse.

- *Die ‚Funktionalität' der einzelnen Elemente soll in der abstrakten Basisklassse „Ware“ implementiert werden.*
- *Alle abgeleiteten Klassen dienen lediglich der eindeutigen Typisierung und des Aufbaus der Hierarchie.*
- *Für die ersten drei Ebenen soll verhindert werden, dass Instanzen der Klassen erzeugt werden können.*
- *Für die Klassen in der Hierarchie müssen Sie keinen Konstruktor implementieren! (Gehen Sie davon aus, dass diese Aufgabe durch einen anderen Entwickler erledigt wird.)*
- *Neben den aus der Abbildung ersichtlichen Klassen soll eine Klasse „WarehouseAdministration“ erstellt werden, die Waren über eine Collection verwaltet. Die Klasse soll zwei Methoden bereitstellen, die die Serialisierung und Deserialissierung kapseln (`ExportToXml(string fileName); ImportFromXml(string fileName)`).*
- *Nutzen Sie in der Assembly „Warehouse“ XML-Attribute um zu erreichen, dass*
  - *der Name einer Ware als Attribut serialisiert wird und*
  - *die Bestandteil e der Collection als „Ware“ serialisiert werden.*

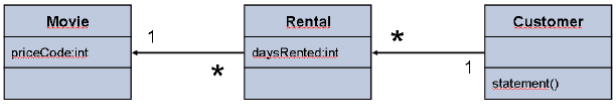
```
class Program
{
    static void Main(string[] args)
    {
        Kaese Gouda = new Kaese();
        Gouda.Name = "Gouda";
        Gouda.Preis = 1.99m;
        Gouda.Waehrung = "euro";
        Gouda.Herkunftsland = "Niederlande";
        Kaese Emmentaler = new Kaese();
        Emmentaler.Name = "Emmentaler";
        Emmentaler.Preis = 2.49m;
        Emmentaler.Waehrung = "euro";
        Emmentaler.Herkunftsland = "Belgien";
        Wurst TuehringerRostbratwurst = new Wurst();
        TuehringerRostbratwurst.Name = "Tuehringer Rostbratwurst";
        TuehringerRostbratwurst.Preis = 1.39m;
        TuehringerRostbratwurst.Waehrung = "euro";
        TuehringerRostbratwurst.Herkunftsland = "Deutschland";

        WarehouseAdministration WarenListe = new
              WarehouseAdministration();
        WarenListe.AddItem(Gouda);
        WarenListe.AddItem(Emmentaler);
        WarenListe.AddItem(TuehringerRostbratwurst);
        WarenListe.ExportToXML("warehouse.xml");
```

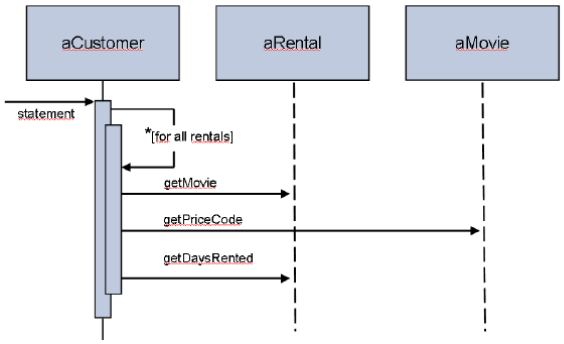
```
            WarehouseAdministration WarenListe2 =
                  WarenListe.ImportFromXML("warehouse.xml");
        }
    }

    public class WarehouseAdministration
    {
        [XmlElement(typeof(Kaese)), XmlElement(typeof(Yoghurt)),
            XmlElement(typeof(Wurst)), XmlElement(typeof(Parfuem))]
        public List<Ware> liste;

        public WarehouseAdministration()
        {
            liste = new List<Ware>();
        }

        public void AddItem(Ware ware)
        {
            liste.Add(ware);
        }

        public void ExportToXML(string fileName)
        {
            XmlSerializer mySerializer = new
                  XmlSerializer(typeof(WarehouseAdministration));
            StreamWriter myWriter = new StreamWriter(fileName);
            mySerializer.Serialize(myWriter,this);
            myWriter.Close();
        }

        public WarehouseAdministration ImportFromXML(string fileName)
        {
            XmlSerializer mySerializer = new
                  XmlSerializer(typeof(WarehouseAdministration));
            FileStream myFileStream = new FileStream(fileName,
                  FileMode.Open);
            WarehouseAdministration warenliste =
                  (WarehouseAdministration)mySerializer.
                  Deserialize(myFileStream);
            myFileStream.Close();
            return warenliste;
        }
    }

    abstract public class Ware
    {
        [XmlAttribute]
        public string Name;
        public decimal Preis;
        public string Waehrung;
        public string Herkunftsland;
    }
    abstract public class Nahrungsmittel : Ware { }
    abstract public class Kosmetik : Ware { }
    abstract public class Milchprodukte : Nahrungsmittel { }
    abstract public class Fleischwaren : Nahrungsmittel { }
    abstract public class Duft : Kosmetik { }
    public class Kaese : Milchprodukte { }
    public class Yoghurt : Milchprodukte { }
    public class Wurst : Fleischwaren { }
    public class Parfuem : Duft { }
```

***Aufgabe U1**: Erläutern, was sich hinter dem Konzept des Refaktorisierens verbirgt. Gehen Sie bei Ihren Ausführungen auch auf wichtige Regeln ein, die beim Refaktorisieren befolgt werden sollen.*

Refaktorisieren ist die Optimierung des Codes im Hinblick auf leichteres Verständnis und bessere Wartbarkeit, ohne das Verändern des externen Verhaltens. Zu Zeiten von agilen Softwareentwicklungsprozessen wird immer mehr versucht, schnell lauffähigen Code zu erhalten, der dem Kunden präsentiert wird. Durch Refaktorisieren kann der Code im Nachhinein optimiert werden. Lange Methoden sollten in logische Einheiten aufgeteilt werden. Die entstandenen Codeabschnitte sollten in die Klasse verschoben werden, deren Daten verwendet werden. Lokale Variablen können durch Methodenaufrufe ersetzt werden, falls dies nicht zu Lasten der Performanz geschieht. Auf jeden Fall sollten Variablen und Methoden selbsterklärende Namen erhalten, damit andere Entwickler sich schneller in den Code reinversetzen können. Vor dem Refaktorisieren ist es wichtig, genügend Testfälle zu definieren und diese beim Refaktorisieren immer wieder durchzuführen. Das Vorgehen in kleinen Schritten hilft, Fehler zu vermeiden und schnell zu entdecken. Tests sollten am besten selbstüberprüfend sein, um nicht lange Zeit für den Abgleich der Testfälle zu benötigen.

***Aufgabe U2**: Erläutern Sie anhand der gegebenen Materialien (2 Abbildungen, 1 Codeauszug), welchen Einschränkungen dieses „pseudo-objektorientierten" Programm in Hinblick auf die Wartbarkeit und Erweiterbarkeit unterliegt.*





```
public class Customer
{
    private string name;
    private ArrayList rentals = new ArrayList();

    public Customer (string name)
    {
        this.name = name;
    }
    public void addRental(Rental arg)
    {
        rentals.Add(arg);
    }
    public string Name
    {
        get {return name;}
    }

    public string Statement()
    {
        double totalAmount = 0;
        int frequentRenterPoints = 0;

        string result = "Rental Record for " + this.Name + "\n";

        foreach (Rental aRental in rentals)
        {
            double thisAmount = 0;

            //Beträge pro Zeile ermitteln
            switch (aRental.Movie.PriceCode)
            {
                case Movie.Type.REGULAR:
                    thisAmount += 2;
                    if (aRental.DaysRented > 2)
                        thisAmount += (aRental.DaysRented -2) * 1.5;
                    break;
                case Movie.Type.NEW_RELEASE:
                    thisAmount += aRental.DaysRented * 3;
                    break;
                case Movie.Type.CHILDRENS:
                    thisAmount += 1.5;
                    if (aRental.DaysRented > 3)
                        thisAmount += (aRental.DaysRented -3) * 1.5;
                    break;
            }
            // Bonuspunkte aufaddieren
            frequentRenterPoints ++;
            // Bonuspunkte für zweitägige Ausleihe einer Neuerscheinung
            if ((aRental.Movie.PriceCode == Movie.Type.NEW_RELEASE) &&
                aRental.DaysRented > 1) frequentRenterPoints ++;
            // Zahlen für diese Ausleihe ausgeben
            result += "\t" + aRental.Movie.Title + "\t" +
                thisAmount + "\n";
            totalAmount += thisAmount;
        }
        // Fußzeilen einfügen
        result += "Amount owed is " + totalAmount + "\n";
        result += "You earned " + frequentRenterPoints +
            " frequent renter points";
        return result;
    }
}
}
```

Wie man in dem Beispiel sieht, gibt es nur eine lange Methode Statement, die mehrere Aufgaben auf einmal übernimmt: berechnen der Beträge pro Ausleihe, Berechnung der Bonuspunkte und Ausgabe der Rechnung. Soll nun eine Änderung in einem Teil der Methode Statement erfolgen, muss sich der Entwickler die komplette Logik der Methode anschauen, um die Änderungen vorzunehmen. Ändert sich etwas an den Filmtypen, so hat dies auch Auswirkungen auf die gezeigte Methode, da in der Switch-Anweisung auf den Typ geprüft wird. Es ist also dringend anzuraten, den gegebenen Code zu optimieren, um ihn für spätere Erweiterbarkeit und Wartbarkeit vorzubereiten.

***Aufgabe U3**: Erläutern Sie kurz 2 Veränderungen des Anwendungsdesigns, die im Rahmen eines Refaktorisierens durchgeführt werden können. Stellen Sie für jede Veränderung kurz die Nachteile der Ausgangssituation dar, die wesentlichen Aspekte der Durchführung der Veränderung sowie die Charakteristika der Zielsituation dar.*

Ein möglicher Ansatzpunkt ist das Zerlegen von langen Methoden. Dies fördert die Übersichtlichkeit und erleichtert die Pflege der einzelnen Programmteile. Die entstandenen Codeabschnitte sollten anschließend der Klasse zugeordnet werden, deren Daten verwendet wird. Wird an einer Klasse

etwas an den Membern geändert, sollte das im Idealfall keine Auswirkungen auf andere Klassen mit sich bringen.

Ein weiterer Ansatzpunkt ist das Umbenennen von lokalen Variablen. Es sollte auf Anhieb ersichtlich sein, welchen Zweck eine Variable erfüllt. Dies erleichtert die Arbeit für spätere Entwickler enorm.

***Aufgabe V1**: Nennen und Erläutern Sie die Schlüsselworte, die innerhalb der Vererbung in C# zum Einsatz kommen.*

Erklären Sie dabei auch den Unterschied zwischen abstrakten und virtuellen Methoden anhand einfacher Beispiele.

Gehen Sie zusätzlich auf darauf ein, wie sich die Mechanismen des Überschreibens (override) und des Verbergens (hide) in C# auswirken, wenn zu einem späteren Zeitpunkt Veränderungen an der Basisklasse vorgenommen werden.

C# unterstützt eine einfache Vererbungshierarchie. Das heißt, dass jede Klasse von einer Basisklasse erbt. Implizit erbt jede erstellte Klasse von der systeminternen Klasse System.Object. Durch die Angabe einer anderen Basisklasse kann die Oberklasse aber auch selbst gewählt werden. Mit Hilfe des base-Schlüsselwortes kann auf Member der direkten Oberklasse zugegriffen werden. So ruft base() den Standardkonstruktor der Oberklasse auf. Soll eine Methode der Oberklasse in der abgeleiteten Klasse überschrieben werden, so muss diese Methode in der Oberklasse als virtual und in der Unterklasse als override deklariert werden. Es ist auch möglich, die Methode nur zu verstecken, dann wird das Schlüsselwort new verwendet. Das Deklarieren einer Methode als virtual wird auch als Late Binding bezeichnet. Dabei prüft der Compiler erst zur Laufzeit den Aktualtyp eines Objektes und schaut nach, ob es eine überschriebene Version der aufgerufenen Methode gibt. Im Gegensatz dazu wird ohne das virtual-Schlüsselwort ein Early Binding durchgeführt. Hierbei wird schon bei der Compilierung der Typ des Objekts bestimmt. Das nachfolgende Codebeispiel verdeutlicht den Effekt:

```
class Program
{
    static void Main(string[] args)
    {
        A a1 = new A();
        A a2 = new B();
        a1.Methode1();
        a1.Methode2();
        a2.Methode1();
        a2.Methode2();
    }
}

class A
{
    public void Methode1()
    {
        Console.WriteLine("Klasse A, Methode1");
    }

    public virtual void Methode2()
    {
        Console.WriteLine("Klasse A, Methode2");
```

```
        }
    }

    class B : A
    {
        public new void Methode1()
        {
            Console.WriteLine("Klasse B, Methode1");
        }

        public override void Methode2()
        {
            Console.WriteLine("Klasse B, Methode2");
        }
    }
```

Da bei der Methode2 das Late Binding erzwungen wird, prüft die Runtime zur Laufzeit den Aktualtyp, in diesem Beispiel ist es also ein Objekte der Klasse B. Da Methode2 in dieser Klasse überschrieben wurde, wird die überschrieben Methode aufgerufen und nicht die der Oberklasse. Im Gegensatz dazu wird bei der Methode1 das Early Binding durchgeführt. Da die Variable a1 der Klasse A zugeordnet wird, wird in jedem Fall die Methode1 der Oberklasse durchgeführt.

Ändert man zu einem späteren Zeitpunkt die Signatur einer virtuellen Methoden, so führt dies zu einem Compilerfehler, da in der Unterklasse die Methode als override deklariert wird, es aber nun keine Entsprechung mehr in der Oberklasse gibt. Durch diese sehr strikte Compilerregelung werden unbeabsichtigte Änderungen ohne Überprüfung der Auswirkungen in den abgeleiteten Klassen vermieden.

Definiert man eine Methode nicht als virtual, sondern als abstract, so kann die Methode nicht instanziiert werden. Klassen, die mindestens eine abstrakte Methode beinhalten, müssen ebenfalls als abstract deklariert werden. Abstrakte Methoden enthalten lediglich die Signatur, eine Implementierung erfolgt erst in einer abgeleiteten, nicht abstrakten Unterklasse.

```
    abstract class AbstractClass
    {
        public abstract int Sum(int a, int b);
    }

    class DerivedClass : AbstractClass
    {
        public override int Sum(int a, int b)
        {
            return a + b;
        }
    }
```

Deklariert man einen Member als sealed, so kann sie in den abgeleiteten Klassen nicht überschrieben werden. Ebenfalls können keine statischen Klassen vererbt werden.

Des weiteren gibt es noch diverse Zugriffsmodifizierer, die Auswirkungen auf die Vererbung haben:

- public: überall sichtbar
- private: nur in der eigenen Klasse sichtbar
- protected: nur in der eigenen und in abgeleiteten Klassen sichtbar
- internal: nur in der verwendeten Assembly sichtbar

***Aufgabe V2**: Zeichnen Sie eine frei zu wählende - für die folgende Aufgabe angemessene - Vererbungshierarchie.*

Erläutern Sie anschließend anhand dieser Darstellung die Objekterzeugung in Vererbungshierarchien! Was ist bei der Implementierung von Konstruktoren zu beachten? Gehen Sie darüber hinaus kurz darauf ein, welche Restriktionen bei Type Casts innerhalb von Vererbungshierarchien zu beachten sind!

Nehmen wir an, wir implementieren zwei Klassen: BaseClass und DerivedClass. Wie der Name schon vermuten lässt, ist BaseClass die Oberklassse von DerivedClass. Wird in einer Klasse kein Konstruktor erstellt, so erstellt der Compiler automatisch einen Standardkonstruktor. Erzeugt man ein Objekt von DerivedClass, so wird zunächst nach einem passenden Konstruktor in dieser Klasse geschaut. Da keiner implementiert wurde, wird der Standardkonstruktor der Klasse aufgerufen, welcher automatisch von System erzeugt wird. Dieser ruft wiederrum den Standardkonstruktor der Oberklasse BaseClass auf, welcher wiederrum den Standardkonstruktor der Oberklasse System.Object aufruft. Nachdem dieser durchlaufen wurde, kehrt das System zum aufrufenden Standardkonstruktor von BaseClass zurück und führt diesen aus. Erst danach wird der Konstruktor von DerivedClass ausgeführt. Nehmen wir nun an, es wird ein Konstruktor mit einem Parameter in der Klasse BaseClass implementiert. Jetzt wird vom System kein Standardkonstruktor mehr bereitgestellt. Will man nun ein Objekt von DerivedClass erzeugen, schlägt dies aus einem Grunde fehl: Der Standardkonstruktor von DerivedClass versucht, den Standardkonstruktor von BaseClass aufzurufen, welcher nun aber nicht mehr existiert. Es gibt nun zwei Möglichkeit, dieses Problem zu lösen: Es wird ein leerer Standardkonstruktor in BaseClass hinzugefügt, oder es wird ein neuer Konstruktor für DerivedClass erstellt, der den selbst erstellten Konstruktor von BaseClass aufruft.

Ein Type Cast von DerivedClass zu BaseClass ist implizit möglich, da jedes Objekt von DerivedClass auch immer ein Objekt von BaseClass ist. Andesrum ist es jedoch nicht implizit, nur explizit möglich. Für einen impliziten Type Cast müsste dieser in einer der Klassen definiert werden.

***Aufgabe W**: Sie sind an einem Projekt beteiligt, in dem ein Framework für O/R-Mapping implementiert werden soll. Ihr Vorschlag, Metadaten zur Steuerung des Frameworks einzusetzen wurde von der Projektleitung angenommen. Die Methoden zur Verarbeitung der Metadaten sind bereits von einem Kollegen auf Basis einer theoretischen Spezifikation implementiert worden.*

Die Implementierung könnte u.a. folgendes SQL-Script erzeugen:

```
2  CREATE TABLE [dbo].[contact] (
3      [id] [int] IDENTITY (1, 1) NOT NULL,
4      [email] [varchar] (80) NOT NULL,
5      [address] [varchar] (80) NOT NULL,
6      [city] [varchar] (50) NOT NULL,
7      [name] [varchar] (50) NOT NULL,
8      [age] [int] NOT NULL
9  ) ON [PRIMARY]
10 GO
11
```

***Aufgabe W1**: Ihre Aufgabe ist es, drei für das O/R-Mapping der Klasse Contact benötigte Attribu-te in einer Assembly zu implementieren. Die Attribute sollen die benötigten Metadaten abbilden, um ein Mapping zwischen Objekten und relationalen Datentabellen zu erreichen.*

Die Implementierung der Attribute soll den Anforderungen an Attributimplementierung genügen (Serialisierbarkeit ist nicht erforderlich) und abhängig vom Attribut die benötigten Member bereitstellen. Versäumen Sie nicht den gezielten Einsatz des AttributeUsage-Attributes.

```
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]
public class FeldtypAttribute : Attribute
{
    private string typ;
    public string Typ
    {
        get { return typ; }
    }
    private int min;
    public int Min
    {
        get { return min; }
    }
    private int max;
    public int Max
    {
        get { return max; }
    }

    public FeldtypAttribute(string typ, int min, int max)
    {
        this.typ = typ;
        this.min = min;
        this.max = max;
    }

    public FeldtypAttribute(string typ, int max) : this(typ, -1, max){}

    public FeldtypAttribute(string typ) : this(typ, -1, -1) {}
}

[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]
public class NotNullAttribute : Attribute {}

[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]
public class IdentityAttribute : Attribute {}
```

Aufgabe W2: Implementieren zusätzlich Sie eine Klasse Contact, die mit den Attributen versehen ist und die das Pendant zum SQL-Skript darstellen kann. Neben den Feldern, Properties und Attributen sind keine weiteren Funktionen erforderlich - Es genügt die Klasse. Assembly-Informationen (using, namespace, etc.) sind an dieser Stelle nicht erforderlich.

```
public class Contact
{
    private int id;
    [Feldtyp("int", 1, 1)]
    [Identity]
    [NotNull]
    public int Id
    {
        get { return id; }
        set { id = value; }
    }

    private string email;
    [Feldtyp("varchar", 80)]
```

```
        [NotNull]
        public string Email
        {
            get { return email; }
            set { email = value; }
        }

        private string address;
        [Feldtyp("varchar", 80)]
        [NotNull]
        public string Address
        {
            get { return address; }
            set { address = value; }
        }

        private string city;
        [Feldtyp("varchar", 50)]
        [NotNull]
        public string City
        {
            get { return city; }
            set { city = value; }
        }

        private string name;
        [Feldtyp("varchar", 50)]
        [NotNull]
        public string Name
        {
            get { return name; }
            set { name = value; }
        }

        private int age;
        [Feldtyp("int")]
        [NotNull]
        public int Age
        {
            get { return age; }
            set { age = value; }
        }
    }
```

***Aufgabe X2**: Erläutern Sie die wesentlichen Grundlagen der Delegates! Beschreiben Sie dazu zunächst, was Delegates sind. Gehen Sie anschließend auf drei Einsatzgebiete für Delegates ein.*

Delegates sind typsichere Zeiger auf Funktionen. Damit ist es möglich, als Methodenparameter dynamisch Methoden zu übergeben, die dann ausgeführt werden. Bei der Definition von Delegates müssen die benötigten Parameter und der Rückgabetyp angegeben werden. Delegates sind nützlich, um parallele Threads zu starten. Im Konstruktor der Thread-Klasse wird als Parameter die auszuführende Methode benötigt. Ebenfalls müssen bei Events die nachfolgende Methode als Parameter übergeben werden. Das dritte Einsatzgebiet sind generische Klassen, die zum Beispiel ein sortierbares Array von beliebigen Objekten bereitstellen. Das Sortieren kann nur vom aufrufenden Client vorgenommen werden, da nur er weiß, wie die Elemente zu vergleichen sind.

***Aufgabe X3**: Ergänzen Sie Ihre Ausführungen aus Teilaufgabe X2 mit einem kurzen Code-Snippet, das die Deklaration und den Einsatz eines Delegates aufzeigt!*

```
class Program
{
    delegate string GetString();
    delegate int GetSum(int a,int b);

    static void Main(string[] args)
    {
        int i = 50;
        GetString DelegateString = i.ToString;
        Console.WriteLine(DelegateString());
        GetSum DelegateSumme = Summe.Sum;
        int u = DelegateSumme(5, 6);
        Console.WriteLine(u);
        Console.Read();
    }

}

class Summe
{
    public static int Sum(int a, int b)
    {
        return a + b;
    }
}
```

***Aufgabe Y1**: Nennen Sie drei Vorteile von Generics.*

- Wiederverwendung: es muss nur eine generische Liste für verschiedene Datentypen geben
- Datentypsicherheit: keine Verwendung mehr von der Klasse object, der Compiler kann falsche Zuweisungen schon zur Laufzeit erkennen
- Performance: es erfolgt kein Boxing und Unboxing mehr

***Aufgabe Y2**: Notieren Sei ein kurzes Code-Snippet, das den Einsatz von Generics anhand einer Collection aus den Basisklassen veranschaulicht und sowohl*

- *Deklaration,*
- *Instanziierung der Liste,*
- *Hinzufügen von Elementen und*
- *den Aufruf einer Methode eines Elements aus der Liste*

umfasst. Wählen Sie für das Beispiel einen geeigneten Typ als Element aus.

```
List<int> liste = new List<int>();
liste.Add(4);
liste.Add(2);
foreach (int i in liste)
      Console.WriteLine(i);
```

***Aufgabe Y3**: Implementieren Sie eine Assembly, die eine generische Klasse DocumentManager enthält. Die Klasse DocumentManager soll über die Mechanismen der Generics Dokumente unterschiedlichen Typs in einer Collection verwalten können. Als öffentliche Member sollen*

- *eine Add-Methode zum Hinzufügen von Dokumenten*
- *eine Get-Methode zum Auslesen von Dokumenten*
- *eine IsDocumentAvaillable, die false zurückgibt, wenn die Collection leer ist (und umgekehrt)*

implementiert werden.

```
class Program
{
    static void Main(string[] args)
    {
        DocumentManager<int> docs = new DocumentManager<int>();
        Console.WriteLine(docs.IsDocumentAvailable());
        docs.AddDocument(1);
        docs.AddDocument(2);
        Console.WriteLine(docs.GetDocument(1));
        Console.WriteLine(docs.IsDocumentAvailable());
        Console.ReadLine();
    }
}

public class DocumentManager<T>
{
    private List<T> liste;

    public DocumentManager()
    {
        liste = new List<T>();
    }

    public void AddDocument(T doc)
    {
        liste.Add(doc);
    }

    public T GetDocument(int i)
    {
        return liste.ElementAt(i);
    }

    public bool IsDocumentAvailable()
    {
        return liste.Count > 0;
    }
}
```