Übungsaufgaben
Software-Entwicklung in Games

SS 2015

BLOCK 1

Aufgabe 1

```
public struct Foo
{
  public int a;
}

public class Bar
{
  public int b;
}

void TakeFoo(Foo f)
{
   f.a = 12;
}

void TakeBar(Bar b)
{
   b.b = 12;
}

void DoSomething()
{
  var foo = new Foo() { a = 3 };
  var bar = new Bar() { b = 3 };

  TakeFoo(foo);
  TakeBar(bar);

  // foo.a = 3
  // bar.b = 12
}
```

Welchen Inhalt haben `foo` und `bar` am Ende von `DoSomething`?

Wie könnte man erreichen, dass `TakeFoo` sich so verhält wie `TakeBar`, ohne die Deklaration von `Foo` zu ändern?

Wie würden sich analoge Versionen von `TakeInt(int i)`, `TakeString(string s)` und `TakeArray(int[] a)` verhalten?

Aufgabe 2

```
public struct uv2
{
    public float u;
    public float v;
}

public struct vector3
{
    public float x;
    public float y;
    public float z;
}

public struct vertex
{
    public vector3 position;
    public vector3 normal;
    public uv2 texcoord;
}
```

```
public class uv2
{
    public float u;
    public float v;
}

public class vector3
{
    public float x;
    public float y;
    public float z;
}

public class vertex
{
    public vector3 position;
    public vector3 normal;
    public uv2 texcoord;
}
```

Führen Sie den Laufzeitvergleich beim Zugriff auf einen großen Array von vertex-Objekten durch. Wie verhält sich die Laufzeit, wenn die vertex-Objekte Klassen sind, wie, wenn die vertex-Objekte Strukturen sind?

Aufgabe 3

Schreiben Sie einen kleines C#-Konsolen-Testprogramm, das einen bestimmten Algorithmus ausführt (Beispiele: Berechnung der Fibonaccizahlen, KgV, GgT, Primfaktorzerlegung, Palindrom-Erkenner, …).

Kompilieren Sie das Programm und geben Sie die ausführbare Datei (.exe) ohne Quellcode an Ihren rechten Nachbarn. Finden Sie durch Dekompilieren mit DotPeek oder Resharper heraus, welcher Algorithmus implementiert wurde.

Aufgabe 4

```
public class ParentA
{
    public virtual void DoSomething() {
        Console.WriteLine("I’m a type A Parent");
    }
}

public class ChildA : ParentA
{
    public override void DoSomething() {
        Console.WriteLine("I’m a type A child");
    }
}

public class ParentB
{
    public void DoSomething() {
        Console.WriteLine("I’m a type B Parent");
    }
}

public class ChildB : ParentB
{
    public new void DoSomething() {
        Console.WriteLine("I’m a type B child");
    }
}

void Main()
{
    ParentA a = new ChildA();
    a.DoSomething();

    ParentB b = new ChildB();
    b.DoSomething();
}
```

Welche Ausgaben werden jeweils erzeugt? Lesen Sie die Verwendung der Schlüsselworte virtual, override und new (NICHT OBJEKT-Konstruktor) nach.

BLOCK 2-4

Aufgabe 5

Nennen Sie Verwendungszwecke/Beispielanwendungen für die im Folgenden gelisteten Standard-C# Generic Colllection Classes. Finden Sie durch Dekompilieren heraus, mit welchen Speicher-Strukturen (z.B. Array, lineare Liste, einfach/doppelt verkettet/Hash-Table, …) die jeweiligen Klassen intern arbeiten

- Dictionary<TKey, TValue>
- List<T>
- Queue<T>
- SortedList<TKey, TValue>
- Stack<T>

Aufgabe 6

Implementieren Sie eine Container-Klasse BinTree, die Objekte sortiert in einem binären Baum abspeichert.

- Finden Sie einen passenden generic type constraint für den generic parameter
- Implementieren Sie einen Enumerator, der die in einem BinTree gespeicherten Objekte in sortierter Reihenfolge ausgibt. Eine BinTree-Instanz soll dabei als Enumerator verwendet werden können
    - Ohne coroutine/yield
    - Als coroutine mit yield
- Implementieren Sie ein Property Backward von BinTree, mit dem in umgekehrter Reihenfolge über die im BinTree gespeicherten Objekte iteriert werden kann.
- Erzeugen Sie Test-Code, der mit foreach über Ihre Enumerator iteriert. Dekompilieren Sie den entstehenden Code, um herauszufinden, wie der C#-Compiler eine Coroutine compiliert, sowie eine foreach-Schleife in eine normale for-Schleife auflöst.

Aufgabe 7

Melden Sie sich bei github an und installieren Sie sich github for [windows|mac]. Clonen Sie sich FUSEEProjectTeam/Fusee an einen geeigneten Ort auf Ihrer Festplatte und versuchen Sie das Projekt zu bauen. Erzeugen Sie wie im Unterricht gezeigt, eine Test-Applikation.

Implementieren Sie eine Baumstruktur, die Cubes in unterschiedlichen Größen und mit unterschiedlichen Farben an beliebigen Positionen und Orienitierungen rendern kann. Versuchen Sie, Iterator zum Traversieren des Baums und zum Rendern zu verwenden.

BLOCK 5

Aufgabe 8

Erstellen Sie ein FUSEE-Beispiel, das mit Hilfe des SceneRenderer Objektes aus Components/SimpleScene Projektes eine aus CINEMA 4D exportierte .fus Datei einlesen und darstellen kann. Der SceneRenderer enthält (u.A.) die beiden Methdoen Render und GetAABB, die beide über den in einer .fus Datei enthaltenen Szenen-Graphen iterieren und dafür jeweils eigene Iterationsmethoden implementieren

Fügen Sie eine Möglichkeit hinzu, einen Szenen-Graph-Knoten auf Grund seines Namens zu identifizieren. Damit können Sie in CINEMA 4D hierarchische Objekte bauen, diese als .fus exportieren und dann in FUSEE auf die Einzelbestandteile zugreifen (um z.B. deren Transformationskomponente zu ändern). Sie können eine neue Methode Find hinzufügen oder z.B. einen Enumerator implementieren, der alle Szenengraph-Objekte einmal besucht.

Aufgabe 9

Schreiben Sie ein Programm, das eine beliebige .Net Assembly (.dll oder .exe) einlesen kann und sämtliche in dieser Assembly enthaltenen Typen auf der Konsole ausgeben kann. Zu jedem gefundenen Typ sollen die enthaltenen Methoden, Properties und Felder mit ausgegeben werden. Zu Feldern und Properties sollen die Typen angegeben werden, zu Methoden die Rückgabewerte, sowie die Liste der Parameter samt Typen.

Aufgabe 10

Suchen Sie sich einen Partner / eine Partnerin. Erstellen Sie eine DLL „Unknown“, die im globalen (leeren) Namespace eine Klasse namens Unknown im mit einer statischen Methode „GetUnknown()“ enthält:

```
public class Unknown
{
    public static object GetUnknown()
    {
      // implement me…
    }
}
```

Implementieren Sie die Methode und schicken Sie die kompilierte DLL zu einem vereinbarten Termin ohne Source Code an Ihren Partner.

Erstellen Sie ein Testprogramm, dass ein beliebiges Objekt entgegen nimmt und mit Reflection-Mechanismen so viel Information wie möglich von dem bestehenden Objekt abfragt.

Für Fortgeschrittene: Erlauben Sie Benutzern das Interaktive Beschreiben und Auslesen von Fields/Properties des Unknown-Objektes. Erlauben Sie Benutzern, das Setzen von Parameter-Werten und das Aufrufen von Methoden.

BLOCK 6

Aufgabe 11

Die folgende Aufgabe verwendet das Calculator-Beispiel des Managed Extensibility Framework:

http://msdn.microsoft.com/de-de/library/dd460648(v=vs.110).aspx

(a) Laden Sie das Projekt „Step0-NoDepInj“ von der Kursseite. Hierbei handelt es sich um einen Kommandozeilen-Rechner, der jeweils zwei Operanden mit einer Operation verknüpft. Derzeit sind die vier Grundrechenarten als Operationen implementiert. Bauen Sie das Projekt und verwenden Sie es. Fügen Sie weitere (binäre) Operationen hinzu, z.B. Fakultät, Potenzieren, n-te Wurzel.

(b) Nun soll es möglich sein, den Rechner auch „von außen“ mit neuen Operationen auszustatten. So gesehen ist der Rechner die Hauptapplikation und neue Operationen sollen über DLLs als Plug-Ins hinzugefügt werden können. Implementieren Sie diese Funktionalität mit Hilfe des Managed Extensibility Framework.

Aufgabe 12

Fügen Sie der im Unterricht erarbeiteten Lösung zum „Calculator-Beispiel“ ein neues DLL-Projekt hinzu, in dem weitere Operatoren implementiert sind. Zur Laufzeit sollen diese Operatoren auch dem Calculator zur Verfügung stehen. Erweitern Sie daher Ihren Catalog um einen „DirectoryCatalog“:

```
catalog.Catalogs.Add(new DirectoryCatalog(<Direcotry-Pfad zu DLLs>));
```

Aufgabe 13

Vergleichen Sie die im Unterricht erarbeitete Lösung zum „Calculator-Beispiel“ mit der in Aufgabe 18 angegebenen Link. Wo sind die Unterschiede?

Die im Microsoft-Beispiel verwendete Lösung benutzt das im MEF (Managed Extensibility Framework) definierte Generic Lazy<T, TMetadata>. Dieses ist von der allgemeineren Klasse Lazy<T> abgeleitet. Welchen Nutzen hat die Klasse Lazy<T>? Warum ist dieser Nutzen für die Verwendung im MEF von Vorteil?

Aufgabe 14

Betrachten Sie die Klasse ImpFactory im Fusee.Core Projekt. Wie könnte hier Dependency Injection verwendet werden?

Betrachten Sie das FUSEE-Example AppBrowserWinForms. Wie könnte hier Dependency Injection verwendet werden?

Versuchen Sie Lösungen mit dem MEF Dependency Injection Container zu skizzieren.

Aufgabe 15

Betrachten Sie das FUSEE-Teilprojekt „SerializationContainer“. In den Projekteigenschaften unter „Build Events“ finden Sie den Aufruf des Protobuf-Precompiler, der nach dem Build-Vorgang der SerializationContainer-DLL die DLL nach [Protcontract] und [Protomember] Attributen durchsucht und daraus den eigentlichen Serialisierungscode erzeugt.

Erstellen Sie nach diesem Vorbild eigene Beispiel-Datenklassen, die dann mit Protobuf serialisiert werden. Verwenden Sie Containerklassen (z.B. List).

Aufgabe 16

Lesen Sie den Artikel von Martin Fowler, dem „Erfinder von Dependency Injection“:

http://martinfowler.com/articles/injection.html

Lesen Sie die einleitenden Artikel über das Unity-Dependency-Injection-Framework

http://msdn.microsoft.com/en-us/library/dn170416.aspx

BLOCK 7

Aufgabe 17

Ersetzen Sie in den Beispielen zum Visitor Pattern die Klassen mit Dummy-Implementierungen durch Interfaces.

Aufgabe 18

Betrachten Sie das Visitor-Pattern OHNE Reflection. Versuchen Sie sich klar zu machen, an welcher Stelle entschieden wird, welche der unterschiedlichen Visit-Methoden eines Visitors aufgerufen wird. Warum funktioniert das Visitor-Pattern nicht, wenn die Methode Accept nur in der Klasse GraphicsObject implementiert wird? Mit anderen Worten: Warum muss in jeder von GraphicsObject abgeleiteten Klasse die Methode Accept noch mal implementiert werden?

Aufgabe 19

Übertragen Sie einen Stand des Visitor-Pattern Ihrer Wahl (mit oder ohne explizites Dispatching/Typüberprüfung) aus dem Unterricht in ein FUSEE-Projekt. Fügen Sie der „GraphicsObject“-Klasse eine „Transform-Komponente“ in Form einer Transformationsmatrix hinzu. Implementieren Sie einen „echten“ RenderingVisitor, der die Objekte rendert, und dabei für jedes Objekt die jeweilige Transformationsmatrix an die aktuelle RenderContext.ModelView-Matrix dranmultipliziert und nach dem Rendern der eigenen Geometrie wieder vom RenderContext „entfernt“.

BLOCK 8

Aufgabe 20

Für Event-basierte Programmierung gibt es in C#/.Net Unterstützung auf Programmiersprachenebene UND IL-Code durch die Schlüsselworte event, delegate, sowie durch die Möglichkeit, anonyme Methoden zu implementieren. In Java gibt es für die Implementierung Event-basierter Konzepte die Klasse java.util.EventObject, sowie ein Pattern bzw. eine Konvention, wie EventSource-Objekte zu implementieren sind. Für die Implementierung EventListener gibt es anonyme Klassen. Stellen Sie die Konzepte in C# und Java gegenüber. Wo sind die Unterschiede?

Aufgabe 21

Betrachten Sie das FUSEE-Beispiel zur UI-Programmierung (SpotTheDiff) und erweitern Sie das Beispiel um eigene UI-Elemente und Events.

Aufgabe 22

Kovarianz und Kontravarianz funktioniert auch für generische Interfaces. Finden Sie ein Beispiel für IList<T>

BLOCK 10

Aufgabe 23

Vollziehen Sie das Account-Beispiel im C#-Referenzhandbuch für das lock Keyword nach.

http://msdn.microsoft.com/en-us/library/c5kehkcz.aspx

Probieren Sie unterschiedliche Mechanismen der Thread-Synchronisierung nach.

Das lock-Keyword ist eine vom C#-Compiler implementierte verkürzende Schreibweise für die Anwendung der Monitor-Klasse. Betrachten Sie hierzu den Artikel „Thread Synchronization (C# and Visual Basic)“ aus dem MSDN

http://msdn.microsoft.com/en-us/library/ms173179.aspx

Wie würde das Account-Beispiel ohne „lock“ Keyword aussehen?

Aufgabe 24

Verwenden Sie FileStream.ReadAsync() um eine größere Datei einzulesen, so dass in der Zwischenzeit, so lange eingelesen wird, andere Dinge vom Haupt-Thread erledigt werden können. Referenz für ReadAsync:

http://msdn.microsoft.com/de-de/library/hh158566(v=vs.110).asp

Lesen Sie den Artikel über asynchrone I/O im MSDN

http://msdn.microsoft.com/en-us/library/windows/desktop/aa365683(v=vs.85).aspx#

ACHTUNG: Der Artikel bezieht sich NICHT auf die .NET-Runtime oder C#. Versuchen Sie die Art der asynchronen Programmierung, die mit der Win32-API-Funktion ReadFile und der Verwendung der OVERLAPPED-Struktur möglich ist, mit den Möglichkeiten, die async, await und FilesStream.ReadAsync bieten, zu vergleichen.

Aufgabe 25

Was ist eigentlich, wenn während einer asynchronen Operation eine Exception auftritt? Z.B. wenn beim Lesen von Datei etwas schief geht?

BLOCK 11

Aufgabe 26

Lesen Sie den Artikel über die Task Parallel Library, die einen Einblick in die Möglichkeiten der TPL liefert und vollziehen Sie die dort genannten Beispiele nach.

http://msdn.microsoft.com/en-us/library/dd537609(v=vs.110).aspx

Aufgabe 27

Finden Sie eigene Beispiele für Methoden im .NET-Framework, die asynchron arbeiten. Klassischerweise enden diese auf „Async“. Welche davon können mit async/await verwendet werden und welche arbeiten ohne async/await?

Aufgabe 28

Verändern Sie das Beispiel T07-AsynchronousDownloadAsyncAwait so, dass der Aufruf von client.DownloadStringTaskAsync nicht mit dem await-Schlüsselwort aufgerufen wird. Dazu muss statt dem Ergebnis string-Objekt ein Task<string> Objekt als Rückgabewert entgegen genommen werden (siehe auch die MSDN-Referenz-Seite zur Methode DownloadStringTaskAsync – wie ist deren Rückgabewert). Jetzt wartet der Methodenaufruf nicht auf das Beenden des Tasks. Verifizieren Sie das durch Test-Code, der nach dem Aufruf von DownloadStringTaskAsync eingefügt wird. Um schließlich doch noch auf das Ergebnis zuzugreifgen (und das Beenden des Tasks abzuwarten), können sie mit der Await-Anweisung auf die Beendigung des zurückgegebenen Task<string> Objektes warten.

Aufgabe 29

Decompilieren Sie das Beispiel T07-AsynchronousDownloadAsyncAwait mit dem DotPeek Decompiler. Schalten sie unter Tools->Options->Decompiler die Option „Show compiler-generated code“ ein. Jetzt sehen Sie, was der C#-Compiler aus den async/await Statements baut. Versuchen Sie, den Code zu verstehen.

Aufgabe 30

Für eifrige: Verändern Sie Beispiel T08-AsyncAwaitOwnTask so, dass CalculateAsync mehrfach „gleichzeitig“ aufgerufen wird (wie in Aufgabe 29 angedeutet). Wie müssen die Resultate abgefragt werden? Welche Möglichkeiten gibt es, auf alle asynchronen ergebnisse zu warten. Ist das sequentielle Warten mit await optimal? Schauen Sie sich die Referenz-Seite zur Methode Task.WaitAll an und versuchen Sie diese zu verwenden.

BLOCK 12

Aufgabe 31

Finden Sie Beispiele für die Anwendungen folgender Extensionmethods

GroupBy, OrderBy, Average

Betrachten Sie die Liste der Enumerable Extension-Methods:

http://msdn.microsoft.com/de-de/library/system.linq.enumerable_methods(v=vs.110).aspx

finden Sie weitere Anwendungsbeispiele

Aufgabe 32

Erzeugen Sie einen anonymen Typ. Versuchen Sie mit DotPeek herauszufinden, welchen Code der C#-Compiler daraus erzeugt (u.U. mit der Option „Show compiler-generated code“ (siehe oben)).

Ist der erzeugte Typ eine Klasse oder ein Struct?

Aufgabe 33

Lesen Sie den MSDN-Artikel zu Anonymen Typen

http://msdn.microsoft.com/en-us/library/bb397696.aspx

Welche Beschränkungen bestehen für anonyme Typen (gegenüber expliziten Typen). Lesen Sie den „Remarks“-Abschnitt.

Aufgabe 34

Vollziehen Sie das Beispiel des MS-Walkthrough durch LINQ nach

http://msdn.microsoft.com/en-us/library/bb397900.aspx

Aufgabe 35

Wandeln Sie mit Hilfe des ReSharper die Queries aus o.G. Beispiel aus der Query-Syntax in Methodensyntax um. Machen Sie sich klar,

1. Wie die Query-Schlüsselworte „where“, „oderby“, „groupby“, „select“ und so weiter durch LINQ in geschachtelte Methodenaufrufe umgewandelt werden.
2. Wie in diesen Methoden verwendete Einschränkungen/Anweisungen als Lambda-Ausdrücke, also letztendlich als Methoden in den generierten Code eingebunden werden.

Dekompilieren Sie ein paar LINQ-Anweisungen mit Hilfe von DotPeek (u.U. mit der Option „Show compiler-generated code“ (siehe oben)), um zu sehen, was der C#-Compiler aus einer LINQ-Anweisung macht.