WS – Collections, File IO und Testing

1. Boxing und Unboxing

a) Welches Resultat liefert der folgende Code? Weshalb?

```
int i = 123;
object o = i;
object x = o;
i = 456;
Console.WriteLine($"The value-type value = {i}");
Console.WriteLine($"The object-type value = {o}");
o = 987;
Console.WriteLine($"The object-type value = {x}");
```

b) Welches Resultat liefert der folgende Code? Weshalb?

```
double e = 2.718281828459045;
double d = e;
object o1 = d;
object o2 = e;
Console.WriteLine(d == e);
Console.WriteLine(o1 == o2);
```

c) Welche Probleme kann Boxing und Unboxing mit sich bringen?

2. Properties

Wir haben bereits letzte Woche die Verwendung von Properties im PersonAdminSystem eingeführt. Dass dahinter keine grosse «Magic» ist, erkennt man, wenn man sich den dafür generierten Intermediate Code anschaut.

a) Starten Sie das ILDasm-Tool aus dem „Developer Command Prompt“ (verwenden Sie diese, da damit alle notwendigen Pfade gesetzt werden).

Hinweis: Die VS Console ist Teil Ihrer VisualStudio Installation.

b) Laden Sie nun die Assembly in den ILDasm (wo finden Sie diese?) und finden Sie heraus, in welchem Feld der Firstname und der Surname einer Person abgelegt werden und wie die jeweiligen Zugriffs-Methoden heissen.

Bemerkung: Sie können sich auch die komplette Methode in IL-Code anschauen. Auf die einzelnen Details gehen wir nicht ein; sie dürfen sich aber selbst natürlich schlau darüber machen. Wissen schadet nie! 😊

3. Indexers und Collections

Fügen Sie nun dem Projekt `PersonAdminLib` eine neue Klasse `PersonRegister` hinzu. Die Klasse soll eine Liste von Personen verwalten können. Erstellen Sie dazu ein privates Feld:

```
private List<Person> personList;
```

a) Implementieren Sie für die Klasse einen **Indexer**, so dass Lese-Zugriff auf die Personen im Personenregister mittels Index möglich ist. Z.B.

```
var personRegister = new PersonRegister();
Console.WriteLine($"Person: {personRegister[0].Firstname} {personRegister[0].Surname}");
```

b) Erweitern Sie die Klasse `PersonRegister` zusätzlich um ein `Count`-Property, welches die Anzahl der Einträge im Register zurückgibt.

z.B. für die Ausgabe der letzten Person im Register:

```
Console.WriteLine("Person: {0} {1}",
    personRegister[personRegister.Count - 1].Firstname,
    personRegister[personRegister.Count - 1].Surname);
```

c) Füllen Sie zu Testzwecken einige Personen in die Liste ab (z.B. im Konstruktor der PersonRegister-Klasse). Führen Sie die obigen Code Snippets in der Main-Method der Applikationsklasse auf.

4. File IO

Um mit grossen und flexiblen Datenmengen arbeiten zu können, sollen Sie nun die Personen von einem File einlesen.

a) Erweitern Sie die Klasse PersonRegister um die Methode

```
public int ReadPersonsFromFile(string filename)
```

welche die Personennamen zeilenweise aus dem übergebenen Textfile liest und in die interne Liste abfüllt. Sie finden das Textfile Persons.txt auf dem Netzlaufwerk, es enthält die Vor- und Nachnamen, jeweils durch einen Tabulator getrennt. Die Methode soll die Anzahl eingelesener Personen zurückgeben.
Die Methode soll die Anzahl der eingelesenen Personen als Rückgabewert zurückgeben.
Sie können für das Einlesen die Klasse StreamReader oder den Helper File.Open() verwenden.
Tipp: Mit der string.Split()-Methode können Sie einen String in ein Array von Teilstrings aufteilen.

b) Sie finden auf dem Active Directory das Text-File "Persons.txt". Fügen Sie File ihrem Projekt "PersonAdmin" hinzu.
WICHTIG: Damit das File zur Laufzeit-Umgebung deployed wird, müssen Sie dies für das File explizit spezifizieren.
Sie machen das, indem Sie bei den File-Eigenschaften die Option "Copy to Output Directory" auf "Copy always" setzen.

c) Testen Sie nun die nun Funktionalität, in dem Sie Ihre PersonRegister Klasse nicht mehr hart codiert mit den Personen initialisieren, sondern in der Main-Methode zunächst die ReadPersonsFromFile() Method aufrufen.