WS – Delegates & Events

1. Sortieren über Delegates

Sie haben in der letzten Stunde des PersonAdmin-Projekt um die Klasse `PersonenRegister` erweitert, die es erlaubt eine Liste von Personen zu verwalten. Erweitern Sie die Klasse schrittweise so, dass nun Personen beliebig sortiert ausgegeben werden können. Spezifizieren Sie die Sortier-Kriterien dazu als Delegates.

a) Erweitern Sie zunächst die Klasse `PersonRegister` um das Property `Persons`, das eine Referenz auf die Personenliste zurückgibt und um die folgende Methode, die alle Personen auf die Konsole ausgibt:

```
public void PrintPersons()
{
    foreach(var p in Persons)
        Console.WriteLine($"{p.Surname}, {p.Firstname}");
}
```

Testen Sie Ihre Implementation, indem Sie in der Konsolenapplikation alle Personen ausgeben lassen.

b) Fügen Sie der Klasse `PersonRegister` eine neue Methode `Sort` hinzu, welcher das Sortier-Kriterium als Delegate übergeben wird. Verwenden Sie dabei den in .NET-Core bereits definierten Delegate-Typ `Comparison<T>` als Parameter der Sort-Methode, und passen Sie den Generischen Type-Parameter `T` auf Ihren Anwendungsfall an. Rufen Sie in der Implementierung der `Sort`-Methode die `Sort`-Methode der internen Liste auf, welche den übergebenen `Comparison`-Delegate als Parameter entgegennimmt.

c) Erweitern Sie die Applikations-Klasse, so dass einmal die Personen nach Vornamen sortiert, einmal nach Nachnamen sortiert ausgegeben werden. D.h. Implementieren Sie jeweiligen Delegates für das Sortieren in der Applikationsklasse und übergeben Sie diese jeweils der Sort-Methode vor der Ausgabe.
Frage: Warum wird der Delegate nicht in der Klasse PersonRegister implementiert?

Hinweis: Ein Comparison-Delegate könnte etwa folgendermassen aussehen:

```
static int CompareByFirstname(Person p1, Person p2)
{
    return p1.Firstname.CompareTo(p2.Firstname);
}
```

2. Observer

Erweitern Sie das Programm dahingehend, dass sich Clients bei der Klasse `PersonRegister` registrieren können, so dass sie informiert werden, wenn neue Personen hinzugefügt werden (Observer-Pattern).

a) Fügen Sie der Klasse `PersonRegister` eine neue Methode Add hinzu, die eine übergebene Person der Liste hinzufügt:

```
int Add(Person newPerson)
```

Die Methode soll die Anzahl der Personen in der Liste nach dem Hinzufügen zurückgeben. Verwenden Sie diese neue Methode auch in `ReadPersonsFromFile()`.

b) Definieren und implementieren Sie für das Observer-Pattern dieses Delegate

```
delegate void PersonAddedHandler(object source, PersonEventArgs args);
```

und den notwendigen Event inkl. `PersonEventArgs`-Klasse dazu. Rufen Sie den Event an der geeigneten Stelle auf.

c) Implementieren und registrieren Sie in der Konsolen-Applikation zwei Observer-Methoden, von denen eine die neu hinzugefügte Person auf die Konsole schreibt (Vor- und Nachname), die zweite dieselben Daten, zusätzlich jedoch Datum und Uhrzeit (`DateTime.Now`) in das Logfile `logfile.txt` schreibt.

3. Predicates

a) Fügen Sie die folgende Suchfunktion der Klasse `PersonRegister` hinzu, die eine Person mit dem übergegebenen Nachnamen:

```
public Person FindPerson(string surname)
{
    foreach (var p in personList)
        if (p.Surname == surname.Trim())
            return p;

    return null;
}
```

b) Die Methode aus a) erlaubt nur die Suche nach einem Kriterium. Schöner wäre es, nach beliebigen Kriterien suchen zu können. Dies ist mit einem Predicate-Delegate als Parameter der Such-Methode möglich. Schreiben Sie nun einen neue Methode `FindPerson`, welcher ein beliebiges Kriterium in Form eines Predicate Delegates für die Suche übergeben werden kann. Führen Sie die Such-Funktion für den Nachnamen damit aus.

In welcher Klasse implementieren Sie den Predicate Delegate? Wieso?

c) Suchen Sie nun nach der ersten Person, die sowohl ein «a» im Vornamen, als auch im Nachnamen hat. Implementieren Sie den entsprechenden Predicate-Delegate dazu.