HAWK Fakultät Naturwissenschaften und Technik Göttingen

Windowsprogrammierung – WiSe17

Übung 2

Aufgabe 2.1 Verweis- und Wertetypen, Teil 1

Diese Aufgabe hat das Ziel, Sie für die Unterschiede zwischen Verweis- und Wertetypen zu sensibilieren.
Definieren Sie zu diesem Zweck eine Klasse **Test**, die über ein privates Feld **testfeld** vom Typ *int* sowie eine zugehörige Eigenschaft **TestEigenschaft** verfügt, mit der man den Wert von **testfeld** zurückgeben und festlegen kann. Definieren Sie einen allgemeinen Konstruktor, mit dessen Hilfe **testfeld** initialisiert werden kann. Überschreiben Sie außerdem die Methode *ToString*, so dass sie den Wert von **testfeld** in String-Form zurückgibt.
Definieren Sie des Weiteren eine Struktur, die den gleichen Namen und den gleichen Inhalt wie die Klasse **Test** hat. Um einen Namenskonflikt zu vermeiden, definieren Sie die Klasse **Test** im Namensraum **MeineKlassen** und die Struktur **Test** im Namensraum **MeineStrukturen**.
Betrachten Sie im Anschluss das folgende Programm:

```
using System;
using MeineKlassen;
//using MeineStrukturen;

class MainClass
{
    static void Main()
    {
        Test t1 = new Test(13);
```

```
        Test t2 = new Test(13);
        Test t3 = t1;
        Vergleiche(t1, t2, t3);            //Stelle 1
        t3.TestEigenschaft = 42;
        Vergleiche(t1, t2, t3);            //Stelle 2
        ÄndereEigenschaft(t1, 121);
        Vergleiche(t1, t2, t3);            //Stelle 3
        Console.ReadKey();
    }
    static void Vergleiche(Test t1, Test t2, Test t3)
    {
        Console.WriteLine("t1: {0}, t2: {1}, t3: {2}", t1, t2, t3);
        Console.WriteLine("Vergleich zwischen t1 und t2 ergibt: {0}", t1.Equals(t2));
        Console.WriteLine("Vergleich zwischen t1 und t3 ergibt: {0}", t1.Equals(t3));
        Console.WriteLine("Vergleich zwischen t2 und t3 ergibt: {0}", t2.Equals(t3));
    }
    static void ÄndereEigenschaft(Test t, int neuerWert)
    {
        t.TestEigenschaft = neuerWert;
    }
}
```

Mit Hilfe der using-Direktiven „`using MeineKlassen;`“ bzw. „`using MeineStrukturen;`“ können Sie einfach zwischen der Verwendung der Klasse **Test** und der Struktur **Test** wechseln. Lassen Sie das Programm in beiden Varianten durchlaufen und erläutern Sie ausführlich, wie die unterschiedlichen, von der gewählten Variante abhängigen Programmausgaben zu Stande kommen. Nehmen Sie dabei Bezug auf die 3 im Code markierten Stellen.

Beantworten Sie abschließend die folgenden Fragen:
Wie kann man erreichen, dass die *Equals*-Methode bei Verweistypen (also Klassen) nicht auf referenzielle, sondern auf inhaltliche Gleichheit prüft?
Welche Änderung muss bei der Struktur-Variante des Programms an der Methode **ÄndereEigenschaft** vorgenommen werden, so dass die Variable **t1** (bzw. deren privates Feld) nach Aufruf der Methode den Wert 121 hat?

Aufgabe 2.2 Verweis- und Wertetypen, Teil 2

Bei welchen der folgenden .NET-Typen handelt es sich um Verweis-, bei welchen um Wertetypen?

int	short[]
double	System.Collections.Generic.List<string>
string	struct A{string s;}
bool	class B{bool b;}
object	enum E{X, Y, Z}
Int32	System.Nullable<long>

Aufgabe 2.3 Verweis- und Wertetypen, Teil 3

Gegeben ist das folgende C#-Programm:

```
class MainClass
```

```
{
    static void Main()
    {
        int i, j;
        object obj;

        i = 42;
        obj = i;            //Stelle 1

        //Stelle 3

        j = (int)obj;       //Stelle 2
        System.Console.WriteLine(j);
    }
}
```

Welcher Mechanismus greift an den Stellen 1 und 2? Welche Daten befinden sich zur Laufzeit an Stelle 3 im Stack und welche im Heap? Was würde sich an der Programmausgabe ändern, wenn an Stelle 3 folgende Anweisung stünde:

```
i = 13;
```

Aufgabe 2.4 Die generische Klasse List<T>

Schreiben Sie ein interaktives Konsolenprogramm, mit dem ein Benutzer eine Liste von Filmen verwalten kann. Verwenden Sie als Liste die generische Klasse *List<T>* aus dem Namensraum *System.Collections.Generic*. Verwenden Sie als generischen Parameter die Klasse **Film**, die Sie in Aufgabe 1.2 von Übung 1 definiert haben. Dem Benutzer sollen folgende Aktionen angeboten werden, die sich (mit Ausnahme des letzten Punktes) alle auf eine Instanz von *List<Film>* beziehen:

- Film hinzufügen: Zunächst werden die für die Erzeugung einer neuen **Film**-Instanz erforderlichen Parameter eingelesen, anschließend wird eine neue **Film**-Instanz der Liste hinzugefügt.
- Alle Filme auflisten: Benutzen Sie eine *foreach*-Schleife zum Iterieren über alle Filme in der Liste. Zur Ausgabe aller relevanter Informationen eines Films können Sie die Eigenschaft **LangeBeschreibung** verwenden, die in Anhang A definiert wird.
- Liste vollständig leeren
- Statistik ansehen: Es sollen folgende Informationen über die Liste ausgegeben werden:
    - Länge des längsten Filmtitels
    - Anzahl von Allzeit-Favoriten
    - Durchschnittliche Filmlänge
    - Gesamte Abspielzeit
- Film auswählen (siehe unten)
- Liste speichern / Liste laden: Diese Funktionen sind bereits vollständig in der Vorlage implementiert. Alles, was Sie noch tun müssen, ist die **Film**-Klasse mit dem Attribut *Serializable* auszuzeichnen.
- Programm beenden

Bei der Aktion „Film auswählen“ soll der Benutzer nach einem Filmtitel gefragt werden. Befindet sich ein Film mit dem eingebenen Titel in der Liste, soll ein Untermenü mit folgenden Aktionspunkten angezeigt werden, die sich (ebenfalls mit Ausnahme des letzten Punktes) alle auf den ausgewählten Film beziehen:

- Film anschauen: Es wird zunächst das Geburtsdatum des Benutzers abgefragt. Ist dieser alt genug, wird der Film angeschaut, andernfalls wird eine Fehlermeldung ausgegeben.
- Film aus Liste löschen: Nach dieser Aktion wird das Untermenü automatisch geschlossen.
- Bewertung ändern: Der Benutzer kann eine neue Bewertung zwischen 1 und 5 festlegen.
- Untermenü beenden

Verschaffen Sie sich einen Überblick über den Aufbau der Vorlage aus Anhang B und ergänzen Sie das Programm an den markierten Stellen.

Anhang A

```
    //innerhalb der Klasse Film:
    public string LangeBeschreibung
    {
        get
        {
            return string.Format(
                "{0,-25}{1}\n{2,-25}{3}\n{4,-25}{5:dd\\.MM\\.yyyy}\n{6,-
25}{7:hh\\:mm\\:ss}\n" +
                "{8,-25}{9}\n{10,-25}{11}\n{12,-25}{13}\n{14,-25}{15:%d} Tage\n{16,-
25}{17:hh\\:mm\\:ss}\n" +
                "{18,-25}{19}{20}",
                "Titel:", Titel,
                "Regisseur:", Regisseur,
                "Release-Datum:", ReleaseDatum,
                "Spielzeit:", Spielzeit,
                "Mindestalter:", Mindestalter,
                "Bewertung:", Bewertung.HasValue ? Bewertung.Value.ToString() : "nicht
vorhanden",
                "Anzahl Abspielungen:", WieHäufigAbgespielt,
                "Alter:", Alter,
                "Gesamtspielzeit:", Gesamtspielzeit,
                "Qualitätskategorie:", Qualitätskategorie,
                IstAllzeitFavorit ? "\nDer Film ist ein Allzeit-Favorit!" : "");
        }

    }
```

Anhang B

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;


class MainClass
{
    static void Main()
    {
        Hauptmenü hauptmenü = new Hauptmenü();
        hauptmenü.Run();
    }
}


class Hauptmenü
{
    List<Film> filmliste = new List<Film>();
    bool beenden = false;

    public void Run()
    {
        do
        {
            ZeigeMenü();
            char eingabe = Console.ReadKey(true).KeyChar;
            switch(eingabe)
            {
                case '1':
                    FilmHinzufügen();
                    break;
                case '2':
                    AlleFilmeAuflisten();
                    break;
                case '3':
                    ListeLeeren();
                    break;
                case '4':
                    FilmAuswählen();
                    break;
                case '5':
                    StatistikAusgeben();
                    break;
                case '6':
                    ListeSpeichern();
                    break;
                case '7':
                    ListeLaden();
                    break;
                case '8':
                     MenüBeenden();
                    break;
                default:
```

```
                Console.WriteLine("Ungültige Eingabe!");
                Console.ReadKey(true);
                break;
            }
        } while (!beenden);
    }


    void ZeigeMenü()
    {
        Console.Clear();
        Console.WriteLine(  "Willkommen! Was wollen Sie tun?\n" +
                            "(1) Film Hinzufügen\n" +
                            "(2) Alle Filme Auflisten\n" +
                            "(3) Liste leeren\n" +
                            "(4) Film auswählen\n" +
                            "(5) Statistik ausgeben\n" +
                            "(6) Liste speichern\n" +
                            "(7) Liste laden\n" +
                            "(8) Menü beenden\n");
    }

    void FilmHinzufügen()
    {
        //Ergänzen
    }

    void AlleFilmeAuflisten()
    {
        //Ergänzen
    }

    void ListeLeeren()
    {
        //Ergänzen
    }

    void FilmAuswählen()
    {
        Film gefundenerFilm;

        //Ergänzen: Filmtitel eingeben lassen, Film suchen

        if (gefundenerFilm == null)
        {
            Console.WriteLine("Der Film konnte nicht gefunden werden!");
            Console.ReadKey(true);
            return;
        }
        Console.WriteLine("Starte Untermenü...");
        Console.ReadKey(true);
        Untermenü untermenü = new Untermenü(filmliste, gefundenerFilm);
        untermenü.Run();
    }

    void StatistikAusgeben()
    {
        //Ergänzen
    }

    void ListeSpeichern()
    {
```

```
        if (IstListeLeer())
            return;
        Console.Write("Bitte den Pfad zum Speichern eingeben: ");
        string pfad = ConsoleTools.LeseZeichenkette();
        BinaryFormatter formatter = new BinaryFormatter();
        try
        {
            using (System.IO.FileStream fs = new System.IO.FileStream(pfad,
FileMode.Create))
            {
                formatter.Serialize(fs, filmliste);
                Console.WriteLine("Die Liste wurde erfolgreich gespeichert!");
            }
        }
        catch(Exception exc)
        {
            Console.WriteLine("Ein Fehler ist aufgetreten: " + exc.Message);
        }
        Console.ReadKey(true);
    }

    void ListeLaden()
    {
        if (!IstListeLeer(false))
        {
            Console.Write("Die aktuelle Liste geht verloren, wenn eine andere geladen wird.
Fortfahren (j/n)? ");
            if (!ConsoleTools.LeseJaNein())
                return;
        }
        Console.Write("Bitte den Pfad zum Laden eingeben: ");
        string pfad = ConsoleTools.LeseZeichenkette();
        BinaryFormatter formatter = new BinaryFormatter();
        try
        {
            using (System.IO.FileStream fs = new System.IO.FileStream(pfad, FileMode.Open))
            {
                filmliste = (List<Film>)formatter.Deserialize(fs);
                Console.WriteLine("Die Liste wurde erfolgreich geladen!");
            }
        }
        catch (Exception exc)
        {
            Console.WriteLine("Ein Fehler ist aufgetreten: " + exc.Message);
        }
        Console.ReadKey(true);
    }

    bool IstListeLeer(bool druckeInfo = true)
    {
        if (filmliste.Count == 0)
        {
            if (druckeInfo)
            {
                Console.WriteLine("Es sind keine Filme vorhanden!");
                Console.ReadKey(true);
            }
            return true;
        }
        else
            return false;
    }
```

```
    void MenüBeenden()
    {
        Console.Write("Soll das Menü wirklich beendet werden (j/n)? ");
        beenden = ConsoleTools.LeseJaNein();
    }
}

class Untermenü
{
    List<Film> filmliste;
    Film ausgewählterFilm;
    bool beenden = false;

    public Untermenü(List<Film> filmliste, Film ausgewählterFilm)
    {
        this.filmliste = filmliste;
        this.ausgewählterFilm = ausgewählterFilm;
    }

    public void Run()
    {
        do
        {
            ZeigeMenü();
            char eingabe = Console.ReadKey(true).KeyChar;
            switch (eingabe)
            {
                case '1':
                    FilmAnschauen();
                    break;
                case '2':
                    FilmLöschen();
                    break;
                case '3':
                    BewertungÄndern();
                    break;
                case '4':
                    MenüBeenden();
                    break;
                default:
                    Console.WriteLine("Ungültige Eingabe!");
                    Console.ReadKey(true);
                    break;
            }
        } while (!beenden);
    }

    void ZeigeMenü()
    {
        Console.Clear();
        Console.WriteLine(  "Derzeit ausgewählt\n: " + ausgewählterFilm.LangeBeschreibung +
"\n\n. Was wollen Sie tun?\n" +
                            "(1) Film gucken\n" +
                            "(2) Film aus Liste löschen\n" +
                            "(3) Bewertung ändern\n" +
                            "(4) Zurück zum Hauptmenü\n");
    }

    void FilmAnschauen()
    {
        //Ergänzen
    }
    void FilmLöschen()
```

```
    {
        //Ergänzen
    }
    void BewertungÄndern()
    {
        //Ergänzen
    }
    void MenüBeenden()
    {
        Console.Write("Zum Hauptmenü zurückkehren (j/n)? ");
        beenden = ConsoleTools.LeseJaNein();
    }
}

static class ConsoleTools
{
    public static DateTime LeseDatum()
    {
        return LeseIrgendwas<DateTime>(str => DateTime.ParseExact(str, @"d.M.yyyy",
System.Globalization.CultureInfo.InvariantCulture), "22.9.2014");
    }
    public static TimeSpan LeseZeitspanne()
    {
        return LeseIrgendwas<TimeSpan>(str => TimeSpan.ParseExact(str, @"h\:mm\:ss",
System.Globalization.CultureInfo.InvariantCulture), "1:42:08");
    }
    public static byte LeseByte()
    {
        return LeseIrgendwas<byte>(str => byte.Parse(str), "Der Wert muss zwischen 0 und
255 liegen.");
    }
    public static bool LeseJaNein(string ja = "j", string nein = "n")
    {
        return LeseIrgendwas<bool>(str =>
            {
                string lower = str.ToLower();
                if (lower == ja.ToLower())
                    return true;
                if (lower == nein.ToLower())
                    return false;
                throw new InvalidCastException();
            },
            ja + " oder " + nein + ", mehr Auswahl gibt es nicht!");
    }


    public static string LeseZeichenkette()
    {
        return Console.ReadLine();
    }
    private static T LeseIrgendwas<T>(Func<string, T> parser, string beispiel = null)
    {
        T ergebnis = default(T);
        bool richtigEingelesen = false;
        do
        {
            try
            {
                string zeile = Console.ReadLine();
                ergebnis = parser(zeile);
                richtigEingelesen = true;
            }
            catch
```

```
            {
                Console.WriteLine("Die Eingabe ist fehlerhaft!");
                if (beispiel != null)
                    Console.WriteLine("Beispiel für korrekte Eingabe: " + beispiel);
                Console.Write("Bitte noch einmal versuchen: ");
            }
        } while (!richtigEingelesen);

        return ergebnis;
    }
}
```