Ostfalia
Hochschule für angewandte Wissenschaften

Prüfungszeitraum SoSe 2016

Studienleistung Objektorientierte Programmierung - Teil 3

Name: _____

Matr.-Nr. _____

Regularien

Bitte melden Sie sich zur Studienleistung (SL) **5152** in der EPV an.

Die Studienleistung besteht aus insgesamt drei Teilen, die im Verlauf des Semesters ausgegeben und bearbeitet werden sollten. Die Lösungen sind jeweils spätestens zu den genannten Terminen abzugeben.

	Ausgabe	Abgabe
Teil 1	Mi 20.4.2016	Mi 4.5.2016
Teil 2	Mi 4.5.2016	Mi 25.52016
Teil 3	Mi 25.5.2016	Mi 8.6.2016

Die Lösungen sind als 7z Datei per Mail an m.maiwald@ostfalia.de und b-u.rogalla@ostfalia.de zu senden. Im Dateinamen stehen ihr Nachname und ihre Matrikelnummer! (Nachname-Matr-Teil1.7z)

Abzugeben sind eingescannte, handschriftliche Algorithmen, UML Diagramme usw. aus denen der Programmentwurf hervorgeht und die die zugrundeliegenden Ideen der Implementierung verdeutlichen. Falls Sie für die Erstellung der UML Diagramme ein Programm verwenden (Draw.io, Visio o.ä.) sind PDF-Ausgaben beizufügen. Zusätzlich sind der gesamte Quellcode und die Projektdateien der Visual Studio Projekte beizufügen, so dass die Programme getestet werden können.

Die Lösungen werden zeitnah von den Tutoren durchgesehen und es gibt ein Feedback zu ihren Ergebnissen.

Die Studienleistung ist bestanden, wenn Sie etwa 70 % der Aufgaben im Wesentlichen fehlerfrei bearbeitet haben.

Aufgabe 1 – Der Weltraum, unendliche Weiten

In Ihrer Ausbildung als Offizier der Sternenflotte lernen Sie u.a. den Umgang mit einem Shuttle. Vor jedem Start muss eine Checkliste durchlaufen werden, die größere Schäden an Ihnen, Ihrem Shuttle, dem Mutterschiff (hier Galaxy-Klasse) und an dumm Rumstehenden verhindern soll.

Die Checkliste sieht die folgenden Aktionen vor: Vor dem Start muss geklärt werden, ob die Schutzschilde des Mutterschiffes „oben“ sind. Wenn ja, gibt es zwei Möglichkeiten:

1. Die Schilde müssen gesenkt werden
2. Die Schilde müssen oben bleiben, z.B. beim Angriff durch eine unbekannte Spezies mit echt unangenehmen Disruptorstrahlen

Im letzteren Fall sollten Sie lieber „zu Hause“ bleiben. Müssen Sie doch fliegen, muss die Frequenz Ihres Shuttle-Antriebs (Warp) an die Resonanzfrequenz der Deflektorschilde des Mutterschiffs angepasst werden.

Um das Tor der Shuttlerampe, zu deren Systemen auch das Rampentor gehört, zu öffnen, muss ein Kraftfeld direkt vor dem Tor aktiviert werden. Danach kann das Tor ohne Druckverlust geöffnet und durchflogen werden. Während das Tor geöffnet ist, muss aus Sicherheitsgründen ein akustischer und optischer Alarm aktiv sein. Kraftfeld und Alarm werden von der Shuttlerampe automatisch gesteuert (auf den Ausbildungsschiffen will sonst keiner in der Shuttlerampe arbeiten).

Der eigentliche Start des Shuttles wird durch den Druck des Startknopfes im Shuttle ausgelöst. (Den Kurs haben Sie schon vorher eingegeben.) Nach dem erfolgreichen Start bleibt Ihnen nur noch das Rampentor zu schließen und den Gefahren des Weltraums ins Auge zu sehen. Eindämmungsfeld und Alarm werden automatisch abgeschaltet.

a) Modellieren Sie den Start Ihres Shuttles während eines Angriffs mit Hilfe eines UML-Sequenzdiagramms. (Ups, falls Sie vergessen haben, die Schilder des Shuttles nach Verlassen des Mutterschiffs zu aktivieren, sind Sie leider jetzt schon tot!)

b) Modellieren Sie die am Start beteiligten Raumschiffteile mit Hilfe eines UML-Klassendiagramms. Berücksichtigen Sie dabei zusätzlich die folgenden Punkte.
   - Shuttle und Mutterschiff haben Deflektorschilde, einen Impuls- und einen Warpantrieb
   - Deflektorschilde und Eindämmungsfelder sind beides besondere Schilde

Achten Sie auf Konsistenz zwischen Sequenz- und Klassendiagramm.

Aufgabe 2 – Arztpraxis

In der modernen Arztpraxis melden sich eintreffende Patienten selbst mit Nachname, Vorname und Geburtsdatum am Terminal an. Sie erhalten eine laufende Nummer - morgens beginnend mit 1 - angezeigt und die Information, wie viele Patienten noch vor ihnen angemeldet sind, aber noch nicht behandelt wurden. Patienten können dann entscheiden, ob Sie den Termin annehmen oder wieder löschen wollen.

Der Arzt kann im Terminal des Behandlungszimmers jeweils den Namen des nächsten Patienten sehen und aus dem Wartezimmer (ggf. über ein Display dort) aufrufen. Zusätzlich erkennt er, wie lange dieser Patient seit seiner Anmeldung warten musste. Diese Wartezeit soll auch für die Tages-Endauswertung abrufbar sein. Auch soll ablesbar sein, wie viele weitere Patienten noch warten.

Aufgabenbeschreibung:

a) Entwerfen Sie für dieses System ein UML-Klassendiagramm, das u. a. die Klassen Praxis, Arzt und Patient enthalten sollte.

b) Ergänzen Sie in ihrem Entwurf die Methoden Anmelden() und Behandeln() und stellen den Ablauf mit UML Aktivitätsdiagrammen dar.

c) Implementieren Sie alle Klassen und testen Sie das Programm mit mindestens 10 Patienten. Der zeitliche Abstand zwischen dem Eintreffen der Patienten beträgt 5 Min. Eine Behandlung durch den Arzt dauert 15 Min.

d) Erzeugen Sie Bildschirmabzüge, mit dem zeitlichen Verlauf der Anmeldungen und Behandlungen. (siehe nachstehendes Beispiel)

e) Stellen Sie die Klassen und Assoziationen in Visual Studio in einem UML-Klassendiagramm dar.

5 Patienten melden sich an.

```
--- Anmeldung Patient:Müller, Max
Warteliste um 15:13
Das Warteliste ist leer
Soll ich Sie in die Warteliste eintragen?
J/N:j

--- Anmeldung Patient:Krüger, Renate
Warteliste um 15:18
  1. Müller       Max          15:13
Soll ich Sie in die Warteliste eintragen?
J/N:j

--- Anmeldung Patient:Fischer, Klaus
Warteliste um 15:23
  1. Müller       Max          15:13
  2. Krüger       Renate       15:18
Soll ich Sie in die Warteliste eintragen?
J/N:j

--- Anmeldung Patient:Schneider, Marianne
Warteliste um 15:28
  1. Müller       Max          15:13
  2. Krüger       Renate       15:18
  3. Fischer      Klaus        15:23
Soll ich Sie in die Warteliste eintragen?
J/N:j

--- Anmeldung Patient:Schmidt, Heinrich
Warteliste um 15:33
  1. Müller       Max          15:13
  2. Krüger       Renate       15:18
  3. Fischer      Klaus        15:23
  4. Schneider    Marianne     15:28
Soll ich Sie in die Warteliste eintragen?
J/N:j
```

Patientin nimmt den Termin nicht an

```
--- Anmeldung Patient:Meier, Hanna
Warteliste um 15:38
  1. Müller       Max          15:13
  2. Krüger       Renate       15:18
  3. Fischer      Klaus        15:23
  4. Schneider    Marianne     15:28
  5. Schmidt      Heinrich     15:33
Soll ich Sie in die Warteliste eintragen?
J/N:n
```

Arzt behandelt

```
>>> Arzt behandelt:Müller, Max
Warteliste um 15:58
  2. Krüger       Renate       15:23
  3. Fischer      Klaus        15:28
  4. Schneider    Marianne     15:33
  5. Schmidt      Heinrich     15:38
```

Zusätzlicher Patient

```
--- Anmeldung Patient:Weber, Mathias
Warteliste um 16:03
  2. Krüger       Renate       15:23
  3. Fischer      Klaus        15:28
  4. Schneider    Marianne     15:33
  5. Schmidt      Heinrich     15:38
Soll ich Sie in die Warteliste eintragen?
J/N:j
```

Arzt behandelt

```
>>> Arzt behandelt:Krüger, Renate
Warteliste um 16:18
  3. Fischer      Klaus        15:28
  4. Schneider    Marianne     15:33
  5. Schmidt      Heinrich     15:38
  6. Weber        Mathias      16:03
```

Arzt behandelt ein Patient

```
>>> Arzt behandelt:Fischer, Klaus
Warteliste um 16:33
  4. Schneider    Marianne     15:33
  5. Schmidt      Heinrich     15:38
  6. Weber        Mathias      16:03
```

2 Patienten melden sich an.

```
--- Anmeldung Patient:Becker, Peter
Warteliste um 16:45
  4. Schneider    Marianne     15:40
  5. Schmidt      Heinrich     15:45
  6. Weber        Mathias      16:10
Soll ich Sie in die Warteliste eintragen?
J/N:j

--- Anmeldung Patient:Hoffmann, Axel
Warteliste um 16:50
  4. Schneider    Marianne     15:40
  5. Schmidt      Heinrich     15:45
  6. Weber        Mathias      16:10
  7. Becker       Peter        16:45
Soll ich Sie in die Warteliste eintragen?
J/N:j
```

Arzt behandelt drei Patienten

```
>>> Arzt behandelt:Schneider, Marianne
Warteliste um 17:05
  5. Schmidt      Heinrich     15:45
  6. Weber        Mathias      16:10
  7. Becker       Peter        16:45
  8. Hoffmann     Axel         16:50

>>> Arzt behandelt:Schmidt, Heinrich
Warteliste um 17:20
  6. Weber        Mathias      16:10
  7. Becker       Peter        16:45
  8. Hoffmann     Axel         16:50

>>> Arzt behandelt:Weber, Mathias
Warteliste um 17:35
  7. Becker       Peter        16:45
  8. Hoffmann     Axel         16:50
```

Patientin nimmt den Termin nicht an

```
--- Anmeldung Patient:Bauer, Karin
Warteliste um 17:46
  7. Becker       Peter        16:51
  8. Hoffmann     Axel         16:56
Soll ich Sie in die Warteliste eintragen?
J/N:j
```

Arzt behandelt

```
>>> Arzt behandelt:Becker, Peter
Warteliste um 18:01
  8. Hoffmann     Axel         16:56
  9. Bauer        Karin        17:46

>>> Arzt behandelt:Hoffmann, Axel
Warteliste um 18:16
  9. Bauer        Karin        17:46

>>> Arzt behandelt:Bauer, Karin
Warteliste um 18:31
Die Warteliste ist leer

>>> Arzt behandelt: kein Patient mehr da
```

Aufgabe 3 – Programmausgaben

Gegeben ist das Interface ICalculate und die abstrakte Basisklasse A. Die Klassen B und C liegen teilweise vor. Ergänzen Sie die Klassen wie angegeben

```
public interface ICalculate {
    void G( double value );
}

public abstract class A {

    protected   double  _x;
    protected   int     _y;
    public      double  X { get { return _x; } }
    public      int     Y { get { return _y; } }

    public A( double x )  {
        _y = (int) x;
        _x += _y;
    }
    public virtual int F( int x ) {
        return 2;
    }
    public int F( float x ) {
        return 3;
    }
}
```

Die Klasse B sieht folgendermaßen aus:

```
public class B : A, ICalculate {

    public B( double x ) : base( x ) {
        _y = 2 * (int) x;
    }

    // Überschreiben der Methode int F( int x), Rückgabewert ist immer 4
    // ToDo …

    // Implementieren der Interface Methode void G( double value), die den
    // Wert value auf die Objektvariable _x addiert.
    // ToDo…
}
```

und die Klasse C:

```
public class C : B {

    public C( double x ) : base( x ) {
        _y = 3 * (int) x;
    }

    // Überschreiben der Methode int F( int x), Rückgabewert ist immer 5
    // ToDo …
```

```
}
```

Das Programm zum Testen lautet:

```
void Run() {

    A a = new C(1.5);
    Console.WriteLine( a.X + " " + a.Y );   // Ausgabe 1

    int retA = a.F(1);
    Console.WriteLine( retA );              // Ausgabe 2

    B b = new B(6);
    a   = b;
    Console.WriteLine( b.X );               // Ausgabe 3
    Console.WriteLine( a.X + " " + a.Y );   // Ausgabe 4

    int retB = b.F(1f);
    Console.WriteLine( retB );              // Ausgabe 5

    // Aufruf der Methode G mit einem Parameter von 1.9
    // ToDo …
    Console.WriteLine( a.X + " " + a.Y );   // Ausgabe 6

    int retAB = a.F(1);
    Console.WriteLine( retAB );             // Ausgabe 7
}
```

Geben Sie die Ausgabe des Programms an (Ausgabe 1: …, Ausgabe 2: …. Usw)

Implementieren Sie das Programm und überprüfen ihre Ausgabe.

Stelle Sie das UML-Klassendiagramm dar.

Zeichnen Sie ein Sequenzdiagramm für die Programmzeile `A a = new C(1.5);`