Universität Regensburg

PD Dr. J. Reischer 23.02.2017

Wiederholungsklausur "ADP" WS 2016/2017

Nachname, Vorname	
Abschluss (BA, MA, FKN etc.)	
Matrikelnummer, Semester	
Versuch (1/2/3)	

Bitte füllen Sie zuerst den Kopf des Angabenblattes aus!

Die Klausur dauert 90 Minuten.
Es sind maximal 50 Punkte zu erreichen.
Es sind keine Hilfsmittel zugelassen.
Die Klausur besteht aus 8 Seiten.

Bitte beantworten Sie alle Fragen direkt auf das Angabenblatt.
Nutzen Sie ggf. die Rückseite und kennzeichnen Sie dies entsprechend.
Eigene Schmierblätter sind nicht erlaubt.

Bei mehreren oder mehrdeutigen Lösungen wird die schlechtere Lösung gewertet. Streichen Sie daher ungültige Lösungen eindeutig durch.

Verwenden Sie nur Java, C# oder Pseudokode für Programmieraufgaben.

Viel Erfolg!

Aufgabe 1: *Konzeptionelle Fragen* (8 Punkte)

Bitte fassen Sie sich bei der Beantwortung der Fragen kurz. Überflüssige Ausführungen und Begründungen werden negativ gewertet.

Aufgabe 1.1: (2 Punkte)

Was unterscheidet Wertetypen von Referenztypen? Nennen Sie zwei Aspekte.

Aufgabe 1.2: (2 Punkte)

Welche Datentypen sind in Java/C#/Pseudokode als Werte-, welche als Referenztypen realisiert? Nennen Sie je zwei (insgesamt also vier).

Aufgabe 1.3: (2 Punkte)

Könnten Wertetypen auch als Referenztypen und umgekehrt realisiert werden (Begründung!)?

Aufgabe 1.4: (2 Punkte)

Nennen Sie je zwei Vor- und Nachteile von Werte- vs. Referenztypen (insgesamt also vier Aspekte).

Aufgabe 2: *Iteration Synthese* (8 Punkte)

Schreiben Sie eine Funktion **int** `ReplaceInMatrix(`**int**`[][] M,` **int** `X,` **int** `Y)`, die alle Vorkommnisse des Wertes `X` in `M` durch `Y` ersetzt; alle anderen Werte bleiben unverändert. Als Ergebnis soll die Anzahl vorkommender bzw. ersetzter Werte `X` zurückgegeben werden (0 bis N); ist die Matrix nicht initialisiert, soll dies durch den Rückgabewert −1 angezeigt werden. Achten Sie darauf, dass die Matrix nicht unbedingt rechteckig sein muss und auch uninitialisierte Zeilen enthalten kann.

Beispiel: `M = {{1,2,3,4},{2,3,4},{3,4},{4}}` mit Suchwert `X = 3` und Ersetzungswert `Y = 0` ergibt `{{1,2,0,4},{2,0,4},{0,4},{4}}` mit Rückgabewert `3` (da drei Vorkommnisse der Zahl `3` durch `0` ersetzt wurden).

Aufgabe 3: *Objektorientierte Programmierung* (20 Punkte)

Aufgabe 3.1:

Definieren Sie eine Klassenhierarchie mit zwei Ebenen aus den unten beschriebenen drei Klassen für die Darstellung verschiedener Mitglieder der Familie Duck (Donald, Dagobert, Tick, Trick und Track). Entscheiden Sie dabei selbst, welche Member welche Modifikationen (Eigenschaften) besitzen. Für nicht sinnvolle Modifikatoren gibt es Punktabzug.

1) abstrakte Oberklasse `Duck`: (10 Punkte)
   a. Attribut `Count`, das alle erzeugten Instanzen der abgeleiteten Unterklassen `AdultDuck` und `YoungDuck` mitzählt (s. Aufgabe 3.2; eine Zugriffsfunktion/ Getter sei hier vernachlässigt);
   b. Attribut `PreName` und `SurName` für den Vor- und Nachnamen eines Ducks, die auch in den Unterklassen zur Verfügung stehen sollen; beachten Sie, dass der Nachname für alle Ducks identisch ist;
   c. Attribut `Adult`, das den binären Status eines Duck-Mitglieds als erwachsen vs. kindlich beschreibt (muss in den Unterklassen nicht zugreifbar sein);
   d. Instanzkonstruktor, der als Parameter den Vornamen eines Ducks und den Status als Erwachsener vs. Kind erhält und die entsprechenden Attribute initialisiert (und evtl. weitere Aktionen durchführen muss); überlegen Sie, ob weitere Konstruktoren notwendig sind;
   e. Methode `ToText()`, die Vorname plus Leerzeichen plus Nachname zurückliefert und in den Unterklassen überschrieben werden muss.
2) zwei Unterklassen `AdultDuck` und `YoungDuck`, die beide von `Duck` abgeleitet und strukturell identisch sind: (6 Punkte)
   a. jeweils Instanzkonstruktor pro Klasse, der jeweils den Vornamen als Parameter erhält und jeweils den Instanzkonstruktor der Oberklasse `Duck` benutzt, um zugleich Vorname und Erwachsenen- vs. Kindstatus zu initialisieren (man beachte die Konstruktorparameter der Oberklasse!);
   b. jeweils überschriebene Methode `ToText()`, die den Erwachsenen- vs. Kindstatus vor dem Vor- und Nachnamen ausgibt, wobei Vor- plus Nachname durch die passende Methode der Oberklasse ermittelt werden soll, wo die entsprechende Funktionalität bereits implementiert wurde (Ergebnis z. B. `"Adult Donald Duck"` oder `"Young Tick Duck"`).

Aufgabe 3.2: Schreiben Sie eine Methode `Duck[] CreateDucks()`, die fünf Instanzen für die Mitglieder der Famile `Duck` erzeugt und als Array entsprechender Größe zurückgibt: Erwachsene Ducks Donald und Dagobert, kindliche Ducks Tick, Trick und Track. Die Zählung für `Count` muss nach den Instanzierungen korrekt sein. (4 Punkte)

Aufgabe 4: *Rekursion Analyse* (7 Punkte)

Gegeben sind folgende rekursive Funktionen F1 und F2:

```
// Rekursive Funktion F1
int F1(int N)
{
  if (N % 2 == 0) return F2(N + 1);
  else return F2(N – 1);
}

// Rekursive Funktion F2
int F2(int N)
{
  N = N – 1;
  if (N <= 0) return N;
  else return F1(N – 1);
}
```

4.1: Welche Ergebnisse liefert die Funktion F1 für die Aufrufe N = 1 bis 6 zurück? Tragen Sie die Werte für F1 in unten stehende Tabelle ein (Fehlversuche bitte streichen).

(3 Punkte)

N	1	2	3	4	5	6
F1(N)						

4.2: Welchen Komplexitätsgrad bezüglich des Laufzeitverhaltens O(?) weist die Funktion F1 in Abhängigkeit von N generell auf (Begründung!)? (3 Punkte)

4.3: Welche Formen der Rekursion treten in F1 auf? Nennen Sie zwei verschiedene.

(1 Punkt)

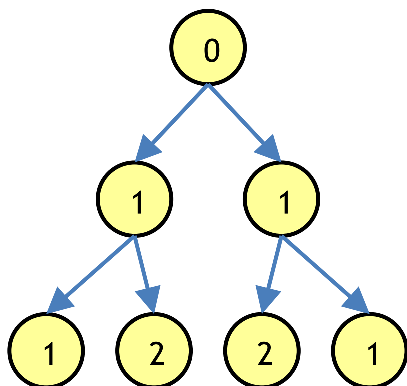
Aufgabe 5: *Rekursion Synthese* (7 Punkte)

Gegeben ist folgende Definition eines Baumknotens:

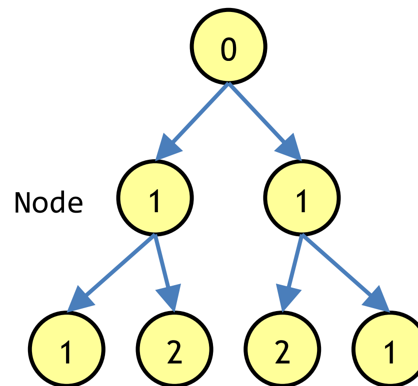
```
class TreeNode
{
  private int Cont = 0;
  private TreeNode Left = null;
  private TreeNode Right = null;

  public TreeNode(int _Cont, TreeNode _Left, TreeNode _Right)
  { Cont = _Cont; Left = _Left; Right = _Right; }
  public int CountCont(int X) { … }             // Ihre Aufgabe!
}
```

Implementieren Sie obige Methode **int** `CountCont(`**int** `X)`, die *rekursiv* für einen Knoten die Anzahl der Werte `X` im aktuellen Knoten und seinem linken bzw. rechten Unterbaum ermittelt (s. Beispiel u. `Root` bzw. `Node`); d. h. es soll die Anzahl vorkommender Werte `X` im Teilbaum ab dem betrachteten Knoten ermittelt werden. Beachten Sie, dass der Baum beliebig groß sein kann und nicht unbedingt balanciert ist. Bei der Implementierung sind weder Schleifen noch Änderungen an Member-Definitionen erlaubt!

Beispiel: `Root`



Node



`Root.CountCont(1)` ergibt 4 `Node.CountCont(1)` ergibt 2

Hinweis: `Root` und `Node` sind nur beispielhaft angegeben und stehen nicht zur Verfügung!