Programmieren 2

Test 2, Frühlingssemester 2012

R. Diehl, M. Klaper, R. Meier Version 1.0

Name: Rohrer Kurs: 3

(Bitte mit Druckbuchstaben schreiben)

Vorname: Felix

Rahmenbedingungen:

1. Zeit: **70 Minuten**
2. Im Test können maximal **68** Punkte erreicht werden. Jeder Aufgabe ist eine maximal erreichbare Punktezahl zugeordnet.
3. Schreiben Sie Ihren Namen und Ihren Kurs auf dieses Blatt. Blätter ohne Namensangabe werden nicht bewertet.
4. Es handelt sich um einen schriftlichen Test ohne Einsatz des Computers oder elektronischer Hilfsmittel. Sie dürfen keine Unterlagen verwenden.
5. Sollte die Problemstellung Unklarheiten aufweisen, dürfen Sie eigene Annahmen treffen. Führen Sie diese in der Lösung auf.
6. Schreiben Sie möglichst verständlich und gut leserlich. Missverständliche Lösungen werden nicht berücksichtigt.
7. Benutzen Sie den Freiraum unter den Aufgaben für Ihre Lösung.

Für die Korrektur (nicht ausfüllen!)

1	2	3	4	5	6	7	Punkte
5	12/15	6/7	6/10	3	5	6	58/68
8	**9**	**10**	**11**				
3/4	4/5	4	4				

Inhaltsverzeichnis:

Aufgabe 1: Textdatei schreiben (5 Punkte) 3
Aufgabe 2: Queue mit Exceptions (15 Punkte) 4
Aufgabe 3: Ordnung (7 Punkte) 6
Aufgabe 4: Thread Lebenszyklus (10 Punkte) 7
Aufgabe 5: Monitore in Threads (3 Punkte) 9
Aufgabe 6: Threads starten (5 Punkte) 10
Aufgabe 7: Qualitätsaspekte (6 Punkte) 11
Aufgabe 8: Anforderungen (4 Punkte) 12
Aufgabe 9: HTAgil (5 Punkte) 13
Aufgabe 10: Meilensteine und Iterationsplan (4 Punkte) 14
Aufgabe 11: Projektmanagement (4 Punkte) 14

Hinweis zu den Multiple Choice Fragen:

Kreuzen Sie alle Aussagen an, die richtig sind. Wenn bei der Aufgabe nichts anderes angegeben ist, so gilt: Pro richtiges Kreuz erhalten Sie Punkte. Für ein falsches Kreuz erhalten Sie einen Abzug. Sie können dabei nicht weniger als 0 Punkte erhalten.

Aufgabe 1: Textdatei schreiben (5 Punkte)

Das folgende Programm erstellt und beschreibt eine Textdatei. Ergänzen Sie die lückenhaften Zeilen direkt im Programm.

```
import java.io.*;

public class TextFileWriter
{
    public static void main(String[] args)
    {
        String fileName = "testfile.txt";
        File aFile = new File(fileName);

        if (!aFile.exists()) {
            try {
                PrintWriter aPrintWriter =

                                    new PrintWriter(aFile) ;

                BufferedWriter aBufferedWriter =

                                    new BufferedWriter(aPrintWriter);

                for (int j = 7; j < 15; j++) {
                    aBufferedWriter.write(j + " ");
                }

                aBufferedWriter. flush() ;

                aBufferedWriter. close() ;
            }

            catch (IOException ioe ) {
                System.out.println("Exception: " + ioe.getMessage());
                return;
            }
        }
    }
}
```

Aufgabe 2: Queue mit Exceptions (15 Punkte)

Gegeben ist die folgende Klasse Queue mit den Methoden `insert` und `extract`. Bei voller Queue soll eine *checked* Exception, bei leerer Queue eine *unchecked* Exception ausgelöst werden.

a) Ergänzen Sie die Methoden `insert` und `extract` und implementieren Sie die Klassen `QFullException` und `QEmptyException`. (10 Pkt.)

```
public class Queue
{
    private int size;
    private int n = 0;
    private int in = 0;
    private int out = 0;
    private Object[] queue;

    public Queue(int s)
    {
        size = s;
        queue = new Object[size];
    }

    public void insert(Object o) throws QFullException ✓
    {
        if (isFull() )
        {
            throw new QFullException(); ✓

        }
        n++;
        if (in == size) {in = 0;}
        queue[in++] = o;
    }

    public Object extract() throws QEmptyException (-1) (✓) ✓
    {
        Object obj = null;
        if (isEmpty())
        {
            throw new QEmptyException(); ✓

        }
        n--;
        if (out == size) {out = 0;}
        obj = queue[out++];
        return obj;
    }
}
```

```
public class QEmptyException extends RuntimeException
{
   public QEmptyException()
   {
   //System.out.println("Queue is empty!");
   super("Error...");

}
public class QFullException extends Exception
{
   public QFullException()
   {
   //System.out.println("Queue is Full!");
   super("Error...");

}
```

b) Implementieren Sie je einen **möglichst einfachen** Aufruf der Queue Methoden `insert` und `extract`. (5 Punkte)

```
Queue q = new Queue(5);
try {
   q.insert("test");
}
catch (QFullException e) {}

System.out.println(q.extract());
```

Aufgabe 3: Ordnung (7 Punkte)

```
public class Fahrzeug implements Comparable<Fahrzeug>
{

    private String marke;
    private int baujahr;

    public Fahrzeug(String marke, int baujahr)
    {
        this.marke = marke;
        this.baujahr = baujahr;
    }

    public int getBaujahr()
    {
        return baujahr;
    }

    public int compareTo(Fahrzeug other)
    {
      if (this == other){
        return 0;
      }
      return (this.marke.compareTo(other.marke));

    }
}
```

a) Implementieren Sie direkt oben im Code die Methode `compareTo()`. Die natürliche Ordnung sei die alphabetisch aufsteigende Marke der Fahrzeuge, z.B. "BMW" < "Toyota". (2 Punkte)

b) Man möchte Fahrzeuge auch nach ihrem Baujahr ordnen können. Alte Fahrzeuge werden als "kleiner" definiert. Implementieren Sie einen `FahrzeugBaujahr`-Komparator mit Hilfe des Interfaces `Comparator<T>`. (5 Punkte)

```
import java.util.Comparator;
public class AgeComparator implements Comparator<Fahrzeug>
{
  public int compare(Fahrzeug f1, Fahrzeug f2){
    return (f1.getBaujahr() - f2.getBaujahr());
  }

  (equals ...)  public boolean equals(Object o){
                  // implementation
                }

}
```

Aufgabe 4: Thread Lebenszyklus (10 Punkte)

Ein praktischer Synchronisationsmechanismus ist das Latch. Latches sperren so lange, bis sie einmal ausgelöst werden und danach sind sie frei passierbar. Das folgende Programm implementiert ein Latch.

```
 1  public interface Synch
 2  {
 3      public void acquire();
 4      public void release();
 5  }
 6
 7  public class Latch implements Synch
 8  {
 9      private boolean latched = false;
10
11      public synchronized void acquire()
12      {
13          while (!latched)
14          try {
15              this.wait();
16          } catch (InterruptedException e) {
17              e.printStackTrace();
18          }
19      }
20
21      public synchronized void release()
22      {
23          latched = true;
24          this.notifyAll();
25      }
26  }
27
28  public class RaceHorse implements Runnable
29  {
30      private Synch startSignal;
31      private int nr;
32
33      public RaceHorse(int nr, Synch startSignal)
34      {
35          this.nr = nr;
36          this.startSignal = startSignal;
37      }
38
39      public void run()
40      {
41          startSignal.acquire();
42          try {
43              Thread.sleep((long)(3000.0*Math.random()));
44          } catch (InterruptedException e) {
45              e.printStackTrace();
46          }
47          System.out.println("Rennpferd "+nr+" ist im Ziel.");
48      }
49  }
50
```

```
51  public class Turf
52  {
53      public static void main(String[] args)
54      {
55          Synch starterBox = new Latch();
56          for (int i = 1; i < 6; i++) {
57              new Thread(new RaceHorse(i,starterBox)).start();
58          }
59          starterBox.release();
60      }
61  }
```

Für den Test des Latch veranstalten wir ein kleines Pferderennen. Dazu gibt es Rennpferde (`RaceHorse`) und eine Rennstrecke (`Turf`). Wie Sie vielleicht wissen werden die Rennpferde in eine Starterbox gestellt und nachdem alle Rennpferde in der Box sind, geht das Rennen los.

Ordnen Sie, falls dies möglich ist, den folgenden Aussagen die dazugehörige Programmzeile oder wenn nötig mehrere Programmzeilen zu.
Je richtige Zuordnung 1 Punkt

		Zeile(n)
a)	Ein Thread wird erzeugt.	57 ✓
b)	Der Thread wird in den Ready-Zustand versetzt.	57 ✓
c)	Diese Operation(en) des Threads werden (quasi)parallel abgearbeitet.	39-48 42-47 (✓)
d)	Der Thread wird immer in den Blocked-Zustand versetzt	43 ✓
e)	Der Thread kommt vom Running-Zustand in den Objects-Lock-Pool. Er will einen Lock bekommen.	11 ✓ (41) 21 (55)
f)	Der Thread kommt vom Running-Zustand in den Objects-Wait-Pool.	15 ✓
g)	Der Thread kommt vom Objects-Lock-Pool in den Running-Zustand. Er hat den Lock erhalten.	~~42~~ nicht möglich → zuerst ready! f
h)	Operation die den Thread vom Objects-Wait-Pool in den Objects-Lock-Pool versetzt.	~~24~~ f
i)	Operation die den Thread vom Objects-Lock-Pool in den Objects-Wait-Pool versetzt.	~~24~~ nicht möglich f
k)	Der Thread wird in den Dead-Zustand versetzt.	~~—~~ 48 f (Exception: 45)

Aufgabe 5: Monitore in Threads (3 Punkte)

Gegeben ist der folgende Quellcode einer Klasse **Account**:

```
public class Account
{
     private static int accountNumb;
     private double balance;

     public Account( double amount )
     {
          balance = amount;
          accountNumb++;
     }

     public synchronized void deposit( double amount )
     {
          balance += amount;
     }

     public double withdraw( double amount )
     {
          synchronized(accountNumb) {
               if (balance >= amount) {
                    balance -= amount;
                    return amount;
               }
               else
                    return 0.0;
          }
     }

     public static synchronized int getAccountNumb()
     {
          return accountNumb;
     }

     //...weitere Methoden zur Kontoverwaltung
}
```

a) Welchen Monitor (lock) verwendet die Methode `deposit` ? (1 Punkt)

Objekt-Monitor (this) ✓

b) Welchen Monitor verwendet die Methode `getAccountNumb` ? (1 Punkt)

Klassen-Monitor (Account) ✓

c) Wer stellt den Monitor in der Methode `withdraw` zur Verfügung ? (1 Punkt)

keiner, Syntax Error, accountNumb ist kein Objekt ✓

Aufgabe 6: Threads starten (5 Punkte)

Man möchte Objekte der Klasse `Ballon` wie ein Thread-Objekt starten (siehe `main`-Methode). Ergänzen Sie die Klasse `Ballon` so, dass dies möglich ist.

```
public class Balloon extends Component implements Runnable
{
    private int position;
    private int altitude;
    private Thread thread;

    public Balloon() {
        position = 0;
        altitude = 0;
    }

    private void newLocation(int wind) {
        position += wind;
        altitude++;
    }

    public void run() {
        for (int i = 1; i <= 1000; i++) {
            newLocation(5);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {}
        }
    }

    public void start() {
      if (thread == null) {
        thread = new Thread(this);
        thread.start();
      }
    }


    //...weitere Methoden zur grafischen Darstellung

    public static void main(String[] args)
    {
        Ballon t = new Ballon();
        t.start();
    }
}
```

Aufgabe 7: Qualitätsaspekte (6 Punkte)

Hinweis: Kreuzen Sie alle Aussagen an, die richtig sind. Pro richtiges Kreuz erhalten Sie 1 Punkt. Für ein falsches Kreuz erhalten Sie -1 Punkt. Bei dieser Aufgabe können Sie nicht weniger als 0 Punkte erhalten. Es gibt insgesamt 6 verschiedene richtige Kreuze.

a) Qualität ist definiert als Übereinstimmung mit den Anforderungen bezüglich …

- ☐ Wiederverwendbarkeit, Übertragbarkeit, Erweiterbarkeit.
- ☐ Funktion, Effizienz, Benutzbarkeit.
- ☒ Zeit, Funktion, Kosten
- ☐ Zeit, Funktion, Effizienz.
- ☐ Funktion, Zeit, Benutzbarkeit.

b) Folgende Qualitätsaspekte stehen aus Entwicklungssicht im Vordergrund …

- ☐ Erweiterbarkeit, Effizienz, Funktionserfüllung.
- ☐ Wiederverwendbarkeit, Übertragbarkeit, Benutzbarkeit.
- ☐ Effizienz, Funktionserfüllung, Zuverlässigkeit.
- ☒ Übertragbarkeit Wartbarkeit, Wiederverwendbarkeit.
- ☐ Effizienz, Sicherheit, Wiederverwendbarkeit.

c) Folgende Qualitätsaspekte stehen aus Anwendungssicht im Vordergrund …

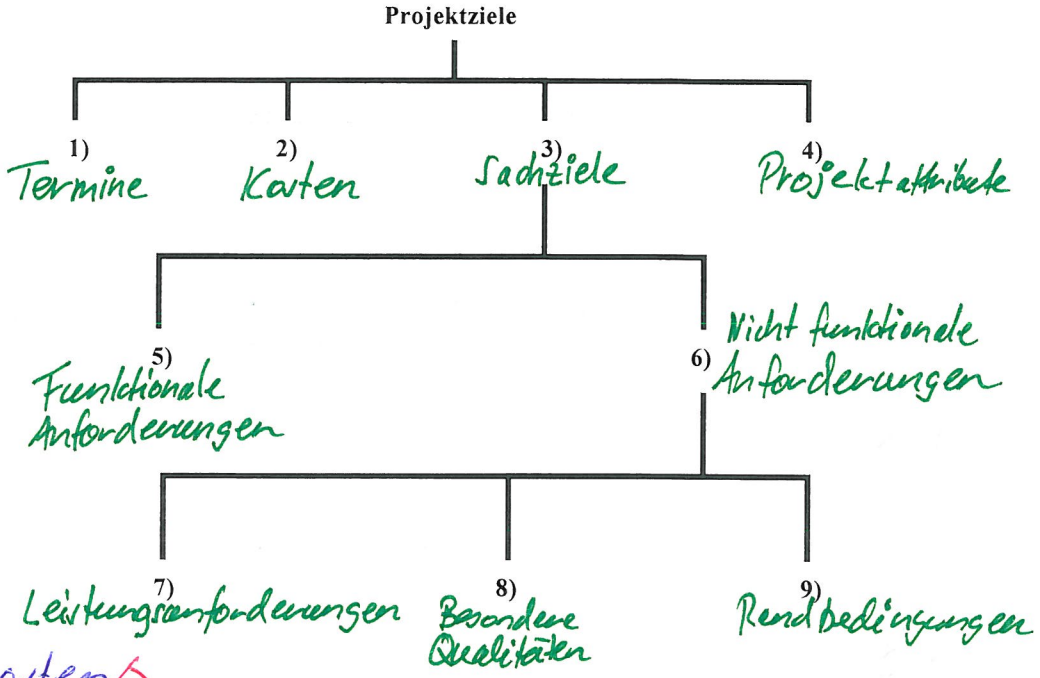
- ☐ Erweiterbarkeit, Funktionserfüllung, Effizienz.
- ☐ Übertragbarkeit, Benutzbarkeit, Wiederverwendbarkeit.
- ☒ Effizienz, Funktionserfüllung, Zuverlässigkeit.
- ☐ Übertragbarkeit, Wartbarkeit, Wiederverwendbarkeit.
- ☐ Sicherheit, Effizienz, Wiederverwendbarkeit.

d) Welche Begriffe gehören zu einem agilen Entwicklungsprozess …

- ☐ inkrementell und intellektuell.
- ☐ sequentiell.
- ☒ Phasen.
- ☒ Meilensteine.
- ☐ Wasserfallmodell.
- ☒ iterativ und inkrementell.
- ☐ iterativ und sequentiell.

Aufgabe 8: Anforderungen (4 Punkte)

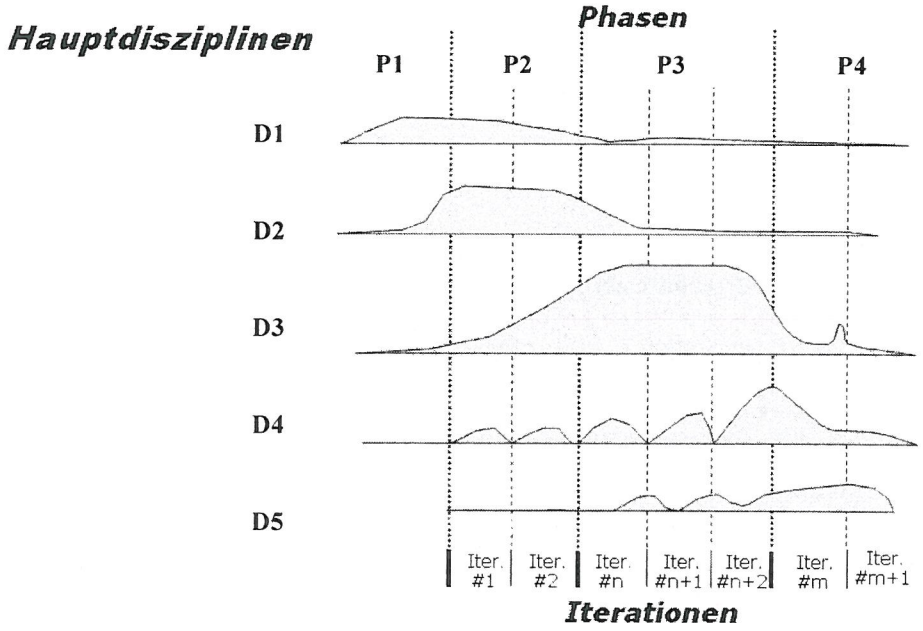
Nennen Sie die verschiedenen Arten von Anforderungen (Requirements) indem Sie das untenstehende Diagramm vervollständigen:



1) Kosten
2) Zeit
3) Funktionsumfang (✓)
4) Projektaspekte (✓)
5) funktionale Anforderungen ✓
6) nicht funktionale Anforderungen ✓
7) Performance (✓)
8) Zuverlässigkeit
9) Sicherheit

Aufgabe 9: HTAgil (5 Punkte)

Benennen Sie vier Phasen (P1 … P4) und die fünf fehlenden Hauptdisziplinen (D1 … D5) von HTAgil. Welche weitere Hauptdisziplin (D6) kennen Sie, die nicht graphisch im Diagramm dargestellt ist?



Phasen:
P1: Vorbereitung ✓
P2: Design (v) Ausarbeitung
P3: Implantierung (v) Konstruktion
P4: Rollout (v) Übergang

Hauptdisziplinen:
D1: Kundenanforderungen ✓
D2: Systemspezifikation ✓
D3: Entwicklung (v) Realisierung
D4: Testen ✓
D5: ~~Wartung~~ Verteilung & Einsatz

Weitere Hauptdisziplin:
D6: ~~Dokumentation~~ Projektmanagement

Aufgabe 10: Meilensteine und Iterationsplan (4 Punkte)

a) Was gehört zu jedem Meilenstein? Nennen Sie zwei konkrete Beispiele (1 Punkt).

Dokumentation ✓, Projekt-Mgmt-Plan, Testprotokoll + (-n)

b) Ergänzen Sie die folgende Aussage (2 Punkte):

Der Meilenstein ist erreicht, wenn die geforderten Artefakte ✓

vorliegen und ihre Überprüfung (Test.) ✓

erfolgreich war.

c) Was ist das Ergebnis einer Iteration?
(1 Punkt)?

Ein lauffähiges Prototyp ✓

Aufgabe 11: Projektmanagement (4 Punkte)

a) Nennen Sie zwei Kernrisiken, die bei vielen Projekten gemeinsam sind (2 Punkte):

Kundenanforderungen ungenügend spezifiziert ✓

Moving-Target, sich ändernde Anforderungen ✓

b) Wie quantifizieren Sie Risiken? (2 Punkte)

Eintrittswahrscheinlichkeit

Auswirkung (Schaden)

(-.) (✓)

---- Ende Test 2, Frühlingssemester 2012 ----