Technische Fachhochschule Berlin
Fachbereich Informatik
Technische Informatik
Luxemburgstr. 10
13353 Berlin

Prof. Dr. F.-M. Reisin

KLAUSUR
Programmieren2
Medieninformatik
15. Februar 2007

Tragen Sie bitte auf diesem Deckblatt und auf allen weiteren Blättern, die Sie während der Klausur verwenden, Ihren Namen und Ihre Matrikelnummer ein.
Verwenden Sie für Ihre Lösung jeweils das Blatt (einschließlich Rückseite), auf dem die Aufgabe steht. Sollte der Platz nicht reichen, verlangen Sie Zusatzblätter.

- **Insgesamt** sind 162 **Punkte** erzielbar.
- Für eine **1.0** genügen 140 **Punkte**.
- Für eine **4.3** müssen **70 Punkte** erzielt werden.

Die **Bearbeitungszeit** beträgt maximal **120 Minuten**.

Viel Erfolg!

Name, Vorname: _____

Matrikelnummer: _____

Wiederholungszahl: 0

Aufgabe	A1	A2	A3	A4	A5	A6	A7	A8	A9	Σ
Maximal	15	10	30	13	14	10	25	20	25	162
Erreicht	14	10	26,75	13	14	10	20	19	24	151

1.0 Reisin
Bravo!

Aufgabe 1(15P)

Die Klasse **StArLiCo (StringArrayListComp)** sei

1. eine für **String**-Elemente generierte **ArrayList<>-Extension**
2. eine Implementierung des **Comparator**-Interface für den Vergleich von je zwei String-Elementen

Geben Sie bitte die vollständige Spezifikation
der ***Importe*** sowie ***des Kopfs*** und ***Rumpfs*** der Klasse an.

```
import java.util.ArrayList;                                          1
import java.util.Comparator;                                         1

class StArLiCo extends ArrayList<String> implements Comparator {    4
                                                  <String> ?
 public int compare (Object o1, Object o2) {
    String s1=(String)o1, s2=(String)o2;                             2   gut!
    return s1.compareTo(s2);                                         2
 }
 public boolean equals (Object o1) {                                 1
    if(o1 instanceof StArLiCo) return true;                          3
    else return false;
 }

}                                                                    14
```

Aufgabe 2 (10 P)

Color-Objekte sind durch drei int-Werte zwischen 0 und 255 definiert, die die Red-, Green- und Blue-Anteile (RGB) repräsentieren.

Geben Sie eine Color-Funktion **randomColor()** an, die ein Color-Objekt zu zufallsgenerierten Rot-, Grün-, Blauwerten erzeugt und zurückliefert.

```
public java.awt.Color randomColor(){
   java.util.Random r = new java.util.Random();   ✓
   return new Color(r.nextInt(255), r.nextInt(255), r.nextInt(255));

}
```

10

Aufgabe 3(30P)
Die folgende Aufgabe besteht aus vier Abschnitten. Jeder Abschnitt weist eine Punktzahl aus.

3.1 (6P) Geben Sie bitte die vollständigen Spezfikationen samt Modifier zu den folgenden Schnittstellen an:

`interface Ein`	mit einer einzigen Funktion	`int e(String str)`
`interface Zwei`	**extendiere** `Ein` um eine Funktion	`String z(Object o)`
`interface Drei`	**extendiere** `Zwei` um eine Funktion	`int e()`

```
interface Ein { public abstract int e(String str); }

interface Zwei { extends Ein
{ public abstract String z(Object o); }

interface Drei extends Zwei {
public abstract int e();

}
```

1
2
2
5

3.2 (6) Geben Sie die **abstrakte** Klasse `Impl2` an, die das Interface `Zwei` implementiere und wie folgt spezifiziert ist:

- String-Attribut `str` sei **zur Vererbung** an weitere Extensionsklassen deklariert und mit "GOOD" initialisiert.
- `Public String z (Object o)` liefere den Wert von `o.toString()` zurück.

```
abstract class Impl2 implements Zwei {
protected String str = "GOOD";

public String z(Object o) {
   return o.toString();   <- if (o != null)
   else return null;
}
```

2
2
2
6

3.3(9) Die Klasse `KxY` extendiere `Impl2`, implementiere `Drei`, und stelle folgenden Funktionen bereit:

- `int e ()` liefere als Ergebnis die Länge des String-Attributs zurück.
- `int e(String str)` liefere das Resultat des Vergleichs zwischen dem aktuellen Wert des-Parameters `str` und dem Wert des `String`-Attributs `str` zurück.

Geben Sie bitte die vollständige Klasse einschließlich Kopf an.

```
class KxY extends Impl2 implements Drei {

public int e() {
  return str.length();
}

public int e(String str) {
  return str.compareTo(super.str);

}
```

2
2
2
3

Reißen Sie bitte die vorliegende Seite zur Beantwortung der Fragen aus.

9

3.4 (9)

Geben Sie für die vorgegebene void-Funktion `test()` der Klasse `KTest` an, ob die Deklaration oder Anweisung vor dem jeweiligen Kommentar zulässig **(Z)** oder unzulässig **(U)** ist. Begründen **(Begr.)** Sie Ihre Antwort **knapp**. Geben Sie für zulässige Anweisungen den **Effekt** auf dem Bildschirm an.

```
class KTest{

    private static KxY xyObj = new KxY();
    private static final String STR = "GOOD";

 public static void test(){

   class KvW extends KxY{ // Z|U  BEGR.:
    KvW(){
     super();
     prt(super.z("Viel Spass")); // Z|U  BEGR.:                EFFEKT:
    }
  }//KvW

  Drei d1 = new KvW(); // Z|U  BEGR.:                          EFFEKT:

  prt (xyObj.str); // Z|U BEGR.:                              EFFEKT:

  prt (xyObj.e(STR)); // Z|U  BEGR.:                         EFFEKT:

  prt(xyObj.e()); // Z|U       BEGR.:                        EFFEKT:

  prt(xyObj.z(STR + new String("LUCK!!!")));
 // Z|U  BEGR.:                                              EFFEKT:

  prt(d1.z(new Integer(42))); // Z|U  BEGR.:                  EFFEKT:
 }
}
```

Innere Klasse in Funktion möglich.

KvW normale Extension der Klasse KxY 0,75

Funktionsaufruf auf Erbklasse Hyperklasse von KxY 0,5

Viel ~~Viel Spass~~ Klasse Spass 0,5

KvW ist durch Extension von KxY vom Typ Drei

Zugriff auf protected-Attribut der Erbklasse. GOOD 1

Funktionsaufruf aus KxY 0 1

Funktionsaufruf aus KxY 4 1

Funktionsaufruf aus Impl2 + Objektübergabe EFFEKT: GOODLUCK!!! 1

Funktionsaufruf aus Impl2 EFFEKT: 42 1

6,75

Aufgabe 4(13)

Schauen Sie sich bitte **Exception_Demo_Init** an. Unterstellt sei ein Aufruf von **main()** Geben Sie bitte unten alle ***Bildschirmausgaben*** an.

```
class Exception_Demo {
   public void level1() throws NullPointerException {
     p.p("start l1");
     try      {
       level2();
     }
     catch (NullPointerException problem) {
        System.out.println (problem.getMessage());
        throw new NullPointerException("null == NoReference!");
     }
     p.p("endl1");
   }
 //-------------
    public void level2()    {
      p.p("start l2");
      level3 (null);
      p.p("end l2");
    }
 //-------------
    public void level3 (Object o) throws NullPointerException    {
      NullPointerException fatal = new NullPointerException("NoObject");

      p.p("start l3");
      try {
               o.toString();
    }
     catch (NullPointerException e) {
               p.p(fatal.getMessage());
               throw fatal;
         }
          p.p("end l3");
     }
}//Exception

public class Exception_Demo_Init {
  public static void main (String[] args) {
    Exception_Demo demo = new Exception_Demo();
    p.p("START");
    try  {
          demo.level1();
    }
    catch(Exception ex) {
      p.p(ex.getMessage());
    }
    p.p("END");
  }//main()
}//Exception_Demo_Init
```

Ausgaben:

START
Start l1
Start l2
Start l3
NoObject
NoObject
null == NoReference!
END

~~Aufgabe 6 (12)~~

Aufgabe 5 (14)

Begriff	Erläuterung	
Exception definieren	Eine eigene Exception-Klasse erstellen, extendiert Exception.	2
Exception-Objekt deklarieren	Ein Exception-Objekt zu einer bestimmten Exception-Klasse erzeugen.	2
Exception aufwerfen	Falls ein irregulärer Programmablauf eintritt, wird entweder automatisch von der Laufzeitumgebung oder vom Programmierer per throw ein Exception-Objekt erzeugt.	2
Exception behandeln	Bei Auftritt eines irregulären Programmablaufs wird per try und catch ein alternativer Programmablauf ausgeführt um eine Exception zu behandeln. (automatisch, wenn nicht vor Ort behandelt)	2
Exception propagieren	Die Exception wird per throw an die nächsthöhere Aufrufstelle weitergereicht um dort eine Behandlung durchzuführen.	2
call stack	Das ist der Aufruf- bzw. Laufzeitstack auf dem alle Aufrufe gestapelt werden.	2
call stack trace	Das ist die Spur ('trace') auf dem call stack bis zum Auftritt einer Exception. Man kann sie also zurückverfolgen	2

14

Aufgabe 6(10)

Geben Sie bitte zur Berechnung der Potenz für int b (basis) und int e (exponent) einen iterativen und einen rekursiven Algorithmus an.

// Iterative Berechnung von b hoch e 4P

```
int powerIterativ (int b, int e){
  int erg=1;

  while(e>0){
      erg*=b;
      e-=1;
  }
  return erg;
}
```

4

// Rekursive Berechnung von b hoch e 6P

```
int powerIterativ (int b, int e){
  if(e<=0) return 1;
  if(e==1) return b;
  return b * powerIterativ(b,e-1);

}
```

6

Aufgabe 7(25)
Gegeben sei die folgende Signatur:

`String liesLiLi(LinkedList<String> inLi, int idX)`
Zurückgeliefert wird ein String-Objekt, das über alle String-Elemente aufgebaut ist, die aus der übergebenen LinkedList von der Indexstelle idX bis zum letzten Element vorhanden sind und ausgelesen werden können.
Geprüft werden muss der null-Fall (keine LinkedList) sowie das Erreichen des letzten Elements.

7A (10P)
Geben Sie einen iterativen Algorithmus unter Verwendung des ListIterators ab der Stelle idX.
TIPP: `ListIterator<E> listIterator(int index)` wird von LinkedList bereitgestellt.

`String liesLiLi(LinkedList<String> inLi, int idX){`

```
if((inLi!=null)&&(inLi.size()-1>=idX)){
  StringBuffer sB = new StringBuffer();
ListIterator it = inLi.listIterator(idX);
  while(it.hasNext()){
     sB.append(it.next());
  }
  return sB.toString();
}
return null;
}//liesLiLi
```

fast!
✓
✓

10

7b (15P)
Geben Sie einen rekursiven Algorithmus OHNE Verwendung des ListIterators ab der Stelle idX.
TIPP: Verändern Sie den Index, idX, für den rekursiven Aufruf.

`String liesLiLi(LinkedList<String> inLi, int idX){`

```
if((inLi!=null)&&(inLi.size()-1>=idX)){
     str = inLi.get(idX);
  return str+liesLiLi(inLi,idX+1);
}
return str;
}//liesLiLi
```

`String str = "";` ✓

+= 2

str + inLi.get(idX) + Lies...

Es fehlt hier der Basisfall und dann die ordentliche Terminierung

str + inLine.get(idX)

10

Aufgabe 8(20)

Name: Matr. Nr.
Wiederholung:

Gegeben sei das folgende Stack-Interface zur Umsetzung der LIFO Organisation von Collections:

```
interface MyStack<Object>{
    public void    push(Object obj)
    public Object  top()
    public void    pop()
    public boolean isEmpty();
} //interface MyStack
```

Geben Sie bitte eine `LinkedList<Object>` **Extensionsklasse** `StackLiLi`, die das vorgegebene Stack-Interface und damit das LIFO-Prinzip vollständig implementiert.

Zur Erinnerung: Die Collection LinkedList< > stellt **unter Anderem** die folgenden nützlichen Funktionen bereit:

- `public void addFirst(E o)` und `public void addLast(E o)`
  fügt o vor das erste (bzw. hinter das letzte) vorhandene Listenelement ein.
  Erzeugt eine neue Liste falls LinkedList<Object> leer ist.
- `public E getFirst()` und `public E getLast()`
  liefert das erste (bzw. letzte) Element der Liste, ohne die Liste zu ändern.
  Falls die Liste leer ist: NoSuchElementException
- `public E peek()` double Element
  liefert das erste (bzw. letzte) Element der Liste, ohne die Liste zu ändern.
  Falls die Liste leer ist: NoSuchElementException
- `public E removeFirst()` und `public E removeLast()`
  entfernt das erste (bzw. letzte) Element aus der Liste und liefert es zurück oder
  NoSuchElementException falls Liste leer ist.
- `public int size()`
  liefert die Anzahl der Elemente in der Liste Element der Liste, ohne die Liste zu ändern.

Geben Sie unter Verwendung ausgewählter Methoden von java.util.LinkedList<E> die vollständige Implementierung von MyStack<Object> an

`class` StackLiLi extends LinkedList<Object> { implements MyStack<Object> { 4

```
public void push(Object obj){
    addLast(obj);
}
```
4

```
public Object top() {
  if(!(isEmpty())) return getLast();
   else return null;
}
```
4

```
public void pop() {
  if(!(isEmpty())) removeLast();
}
```
4

```
public boolean isEmpty(){
  if(size()==0) return true;
    else return false;
}
```
if true size()==0,
3

19

Aufgabe 9(25)
Gegeben sei eine vordefinierte `JPanel`-Extension `VideoPanel` zum Abspielen von VideoClips.
Zur Steuerung seien im VideoPanel die folgenden ereignissensitiven GUIs deklariert.

3 `Buttons` mit den Bezeichnern
- `play`
- `pause`
- `stop`

1 `JPanel` mit dem Bezeichner
- `ControlPanal`

Die Bezeichner der Buttons im Programm sollen bei der Erzeugung auch als Label-Strings dienen, mit denen die Buttons initialisiert werden.

Des Weiteren enthalte `VideoPanel` eine private innere Klasse **VBL – VideoButtonListener**.
VBL implementiere das **ActionListener-Interface.**
Quellbereiche und **Ziel**bereiche der Ereignisse sind die Buttons.

Zwischen den Buttons **play** auf der einen Seite sowie **pause und stop** auf der anderen Seite besteht ein Zusmmenspiel.

Sofern der **play-Button** durch ein click gedrückt worden ist, müssen
- der **pause**- und der **stop-Button** ***selektierbar*** sein, d. h. aktiviert werden.
- Der **play-Button** muss ***un*selektierbar** , d.h. de-aktiviert werden.

Wird demgegenüber der **stop-Button** durch ein click gedrückt, so muss umgekehrt
- der **play-Button** ***selektierbar*** sein, d. h. aktiviert werden.
- Der **pause-** sowie der **stop-Button** müssen jeweils ***un*selektierbar** , d. h. deaktiviert werden.

Zum Aktivieren und Deaktivieren stelle VideoPanel die vordefinierten und daher geerbten Funktionen bereit. Sie müssen NICHT deklariert, sondern NUR AUFGERUFEN werden.

- **activateB (Button b)**
- **deactivateB (Button b)**

Beim click auf dem **pause-Button** soll der Bezeichner des Buttons wie folgt geändert werden:

- Ist der Label des pause-Buttons **"PAUSE"**, so soll es zu **"CONTINUE"** überwechseln.
- Ist es umgekehrt **"CONTINUE"**, so soll es zu **"PAUSE"** überwechseln.

Dazu müssen Sie die von `Button` bereitstehende Funktion `setLabel(String lab)` aufrufen.
Geben Sie die Klasse VBL und insbesondere den
- Konsturktor sowie
- die Funktion **ActionPerformed()** zur Behandlung der o. a. Action-Ereignisse an.

Ordnen Sie den Buttons jeweils ein dynamisch erzeugtes, d. h. NICHT EXTRA deklariertes VBL-Objekt zu.
Fügen Sie
- die 3 ereignissensitiven `Button`-Objekte dem `ControlPanel` und das
- `ControlPanel` dem `VideoPanel` zu.

```
class VideoPanel extends JPanel{

  public void activateB(Button b) {}  //muss nur aufgerufen werden

  public void deactivateB (Button b){} //muss nur aufgerufen werden

// Geben Sie  alle Deklarationen an.
private JPanel ControlPanel; ~~= new JPanel();~~
private Button play, pause, stop;


  VideoPanel() {
  play = new Button("PLAY");
  play.addActionListener(new VBL());
  pause = new Button("PAUSE");
  pause.addActionListener(new VBL());
  stop = new Button("STOP");
  stop.addActionListener(new VBL());

  ControlPanel = new JPanel();

  ControlPanel.add(play);
  ControlPanel.add(pause);
  ControlPanel.add(stop);

  add(ControlPanel);
  }
~~[illegible]~~


private class VBL implements ActionListener {

  public void actionPerformed(ActionEvent e) {
     if (e.getSource() == play) {
       activateB(pause);
       activateB(stop);
       deactivateB(play);
     }
     if (e.getSource() == stop) {
       activateB(play);
       deactivateB(pause);
       deactivateB(stop);
     }
     if (e.getSource() == pause) {
       if (pause.getLabel() == "PAUSE") pause.setLabel("CONTINUE");
        else pause.setLabel("PAUSE");
     }
   } //actionPerformed
  } //VBL
} //VideoPanel
```

Bemerkung: Hier benötige ich keinen Konstruktor für VBL

Mem!

5/6

4

2

1

1

3

1

3

4

24