Übungsklausur

Aufgabe 1: JDK

a)

help und version sind zwei mögliche Optionen, mit denen der Java-Compiler **javac** aufgerufen werden kann. Nennen Sie zwei weitere und beschreiben Sie deren Wirkung.

Lösung:

```
C:\Windows\system32>javac -help
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are u
                             sed
  -classpath <path>          Specify where to find user class files and annotati
                             on processors
  -cp <path>                 Specify where to find user class files and annotati
                             on processors
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -endorseddirs <dirs>       Override location of endorsed standards path
  -proc:{none,only}          Control whether annotation processing and/or compil
                             ation is done.
  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors
                             to run; bypasses default discovery process
  -processorpath <path>      Specify where to find annotation processors
  -d <directory>             Specify where to place generated class files
  -s <directory>             Specify where to place generated source files
  -implicit:{none,class}     Specify whether or not to generate class files for
                             implicitly referenced files
  -encoding <encoding>       Specify character encoding used by source files
  -source <release>          Provide source compatibility with specified release
  -target <release>          Generate class files for specific VM version
  -version                   Version information
  -help                      Print a synopsis of standard options
  -Akey[=value]              Options to pass to annotation processors
  -X                         Print a synopsis of nonstandard options
  -J<flag>                   Pass <flag> directly to the runtime system
  -Werror                    Terminate compilation if warnings occur
  @<filename>                Read options and filenames from file
```

b)

Der Java-Compiler **javac** und der Java-Interpreter **java** sind zwei Programme des Java Development Kits (JDK). Nennen Sie drei weitere und beschreiben Sie deren Aufgaben.

Lösung:

- **jar** zum Erstellen von Jar Archiven

- **javadoc** zum Erzeugen von HTML Dokumentationen aus den javadoc Kommentaren im Quellcode
- **jarsigner** zum Signieren von Jar Archiven
- **appletviewer** Browser zum "Testen" von Java Applets (stellt nur Applets da)
- ...

Aufgabe 2: GUI und Datenstrukturen

Beschreiben Sie kurz die Aufgaben der folgenden Klassen/Schnittstellen der **Swing**-Bibliothek bzw. des **JCF** (Java Collection Frameworks).

JLabel:	Darstellung von Texten und Bildern
JComboBox:	Eine aufklappbare Liste von Elementen, bei der ein bis mehrere Elemente selektiert werden können
JList:	Liste von Elementen (nicht aufklappbar), bei der ein bis mehrere Elemente selektiert werden können
JTextField:	Textfelt zur Eingabe von einer Zeile
JScrollbar:	Scrollbar, welches die Darstellung von einem Objekt steuert
Collection:	Zusammenfassung von endlich vielen Elementen zu einer Einheit
Set:	Interface für Mengen (mit allen bekannten Mengenoperationen)
SortedSet:	Interface für sortierte (total) Mengen, ergbt von Set
List:	Interface für Listen

Aufgabe 3: Enumeration, Packages, Annotationen, Parallelprogrammierung

		wahr	falsch
1.	Enum-Klassen können instanziiert werden.		☑
2.	Enum-Werte können mit dem Operator == auf Gleichheit getestet werden.	☑	
3.	Wenn eine Klasse keine package-Anweisung enthält, wird sie dem Default-Paket zugeordnet.	☑	
4.	Klassen des Default-Pakets können ohne explizite import-Anweisung verwendet werden.	☑	
5.	Annotationen stehen vor dem Programmelement, das sie annotieren.	☑	
6.	Annotationen beginnen mit dem Zeichen @.	☑	
7.	Der Algorithmus von Dekker verwendet aktives Warten zur Synchronisation der Prozesse.	☑	
8.	Ein Semaphor kann nur Werte, die größer als 0 sind, annehmen.		☑

Anmerkungen:

1. Falsch, da Emums durch die JVM verwaltet werden. Beim Laden von Enum Klassen wird automatisch im Hintergrund ein Enum Objekt erstellt und ist für den Programmierer zur Laufzeit nicht änderbar.
2. Wahr, da jedes Enum Objekt nur einmal im Speicher existiert, somit müssen auch die Speicherzellen gleich sein.
3. Wahr, oder habt ihr jemals einen Paketnamen verteilen müssen?
7. Wahr: while (turn != 0) {} ist aktives Warten
8. Falsch, da bei 0 die Threads schlafen gelegt werden.

Aufgabe 4: Zuverlässigkeit von Programmen

Welche der beiden folgenden Regeln zum Nachweis der partiellen Korrektheit sind korrekt?

1. $\frac{\{p\}\ S\ \{r\}, r \wedge e \rightarrow p,\ r \wedge \neg e \rightarrow q}{\{p\}\ while\ e\ do\ S\ od\ \{q\}}$

2. $\frac{\{p\}\ S\ \{r\}, r \wedge e \rightarrow p,\ r \wedge \neg e \rightarrow q}{\{p\}\ while\ e \vee \neg e\ do\ S\ od\ \{q\}}$

Lösung:

1. Kann nicht korrekt sein, da nicht sichergestellt wurde, das S ausgefürht wird.

2. Da $while\ e \vee \neg e$ immer True ergibt, "hängt" das Programm in einer Endlosschleife fest. Somit terminiert es nie, daswegen ist es partiell korrekt.

Aufgabe 5: Zuverlässigkeit von Programmen

```
1  static int f(int m, int n) {
     assert m <= n; // Vorbedingung P
3    int s = m+n,
     i = m;
5    assert ... // Schleifeninvariante Q
     while ( i <= n ) {
7      s = s - 2*i;
       i = i + 1;
9      assert ... // Schleifeninvariante Q
     }
11   assert ... // Nachbedingung R
     return s;
13 }
```

a)

Es wird der Wert $m + n - \sum_{j=m}^{n} 2j$ berechnet. Eine entsprechende Nachbedingung ist daher R : $m + n - \sum_{j=m}^{n} 2j$.

b)

Eine zum Nachweis der partiellen Korrektheit von f bezüglich P und R geeignete Schleifeninvariante ist Q : $m \leq i \leq n + 1 \wedge s = m + n \sum_{j=m}^{i-1} 2j$.

c)

```
1  R: s == m * m - n * n
   Q: m <= i && i <= n + 1 && s == n + m * m - i * i + i
```

Aufgabe 6: Programmverständnis, Fehlerkorrektur

Bei der Ausführung des folgenden Programms soll ein JFrame geöffnet werden, das einen JButton im oberen Fensterteil enthält. Bei Betätigung des JButtons soll die Hintergrundfarbe blau werden. Das Programm enthält Fehler.

```
   import java.awt.event.*;
2  import javax.swing.*;

4  public class Test extends JFrame {
     JButton button;
6
     public Test() {
8      c = getContentPane(); // c nicht Deklariert
       button = new JButton("TestButton");
10     c.add(button, BorderLayout.SOUTH); // import fuer BorderLayout fehlt
       // der Button sollte nach North statt South
12     Listener l = new Listener();
       button.addActionListener(l);
14   }

16   class Listener implements ActionListener {
       public void actionPerformed(ActionEvent e) {
18       Color farbe = new Color(000,255,000); // import fuer Color fehlt
         // es sollte auch Blau statt Gruen verwendet werden
20       c.setBackground(farbe); // c muss sich erst "geholt" werden
       }
22   }
```

```
24    public static void main(String[] args) {
        Test t = new Test();
26      t.setTitle("TestFrame");
        t.setSize(400,200);
28      t.setLocation(200,200);
        t.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
30      // die GUI wird nie mit setVisible(true) angezeigt
    }
32  }
```

Aufgabe 7: Datenstrukturen, Programmierung

Gegeben seien die beiden Klassen Node und List zur Implementierung verketteter Listen:

```
public class Node {
2   public int item;
    public List next;
4
    public Node(int item) {
6     this.item = item;
      this.next = new List();
8   }
}
10
public class List {
12  public Node node = null;

14  public boolean isEmpty() {
      return node == null;
16  }

18  public void insert(int x) {
      if (isEmpty())
20      node = new Node(x);
      else {
22      Node n = new Node(x);
        n.next.node = node;
24      node = n;
      }
26  }

28  public String toString() {
      String s = "<";
30    if (!isEmpty()) {
        s = s+node.item+"; ";
32      return s = s+node.next;
      }
34    return s+">";
    }
36  }
```

a)

Was gibt das folgende Programmfragment aus?

```
List l = new List();
2 l.insert(2); l.insert(-4); l.insert(3); l.insert(6);
```

```
System.out.println(l);
```

< 6; < 3; < −4; < 2; <>

b)

Schreiben Sie eine Methode boolean **isPermutation()**, die den Wert true genau dann liefert, falls die aktuelle Liste jede der Zahlen $1, 2, \ldots, n$ genau einmal enthält, wobei n die Anzahl der Listenelemente ist. Beispiel: Die Methode soll true für die Listen $4, 2, 1, 3$ und $5, 1, 2, 3, 4$ sowie false für die Listen $3, 3, 1, 2$ und $1, 2, 4$ liefern.

gibt sicherlich bessere Implementierungen

```
boolean isPermutation() {
    int max = 0;
    Node cur = node;
    while (cur != null) {
        if (cur.item < 1) { // kein element darf kleiner als 1 sein
            return false;
        }
        if (cur.item > max) { // suche maximales element
            max = cur.item;
        }
        cur = cur.next.node;
    }

    boolean[] check = new boolean[max + 1];

    cur = node;
    while (cur != null) { // teste ob doppelt
        if (!check[cur.item]) {
            check[cur.item] = true;
        } else {
            return false;
        }
        cur = cur.next.node;
    }

    for (int i = 1; i < max; i++) { // test ob alle enthalten sind
        if (!check[i]) {
            return false;
        }
    }
    return true;
}
```

oder einfacher:

```
boolean isPermutation() {
  int length = this.length();
  for (int i = 1; i <= length; i++) {
    if (!this.isIn(i))
      return false;
  }
  return true;
}

int length() {
  if (this.isEmpty())
    return 0;
  return 1 + this.node.next.length();
```

```
}

boolean isIn(int x) {
  if(this.isEmpty())
    return false;
  if(this.node.item == x)
    return true;
  return this.node.next.isIn(x);
}
```

c)

Schreiben Sie eine Methode **void reverse()**, die die Reihenfolge der Listenelemente umkehrt. Beispiel: Falls l die Liste $5, 1, 2, -4, 3$ ist, dann soll l nach Aufruf der Methode die Liste $3, -4, 2, 1, 5$ sein.

```
void reverse() {
    if(isEmpty()) // wenn leer mache nichts
        return;
    Node currNode,nextNode , loopNode;
    currNode = node;
    nextNode = node.next.node;
    node.next.node = null;

    while(nextNode != null) {
        loopNode = nextNode.next.node;
        nextNode.next.node = currNode;
        currNode = nextNode;
        nextNode = loopNode;
    }
    node = currNode;
}
```

Klausur 2 a

Aufgabe 1: Programmverständnis

Das folgende Programm benutzt eine zirkuläre Liste.

```
class Node {
  int val;
  Node next;

  Node (int val) {
    this.val = val;
  }
}

class Josephus {
  public static void main(String[] args) {
    int n = 19,
    m = 5;
    Node t = new Node(2),
    x = t;
    for (int i = n-1; i >= 0; i=i-3) {
      x = (x.next = new Node(i+1));
    }
    System.out.print(x.val); // (*)
    x.next = t.next;
```

```
    while (x != x.next) {
22      for (int i = 0; i <= m; i++)
          x = x.next;
24      System.out.print(";" + x.val);
        x.next = x.next.next;
26    }
      System.out.println();
28    System.out.println("Nr. " + x.val + " ist der Gewinner.");
  }
30 }
```

a)

Welche Elemente besitzt die Liste bei der Ausführung von (*)? Geben Sie die Listenelemente in der Reihenfolge an, wie sie eingefügt wurden.

Es wird 1 an der Position (*) ausgegeben.

Die Reihenfolge ist: 2, 19, 16, 13, 10, 7, 4, 1

b)

Was gibt das Programm aus?

1;4;4;16;7;7;7
Nr. 7 ist der Gewinner.

a) Aufgabe 2: Aufzählungstypen, Pakete, Annotationen, Parallelprogrammierung

Bitte kreuzen Sie an. Für jede richtige Antwort erhalten Sie einen Punkt, für jede falsche Antwort wird ein Punkt abgezogen. Kein Kreuz bzw. zwei Kreuze bedeuten 0 Punkte. Die minimale Gesamtpunktzahl für diese Aufgabe beträgt 0 Punkte. Alle Fragen dieser Aufgabe beziehen sich auf Java.

		wahr	falsch
1.	Enum-Werte können mit equals auf Gleichheit getestet werden.	☑	
2.	Enum-Klassen implementieren die Schnittstelle Comparable.	☑	
3.	Durch statischen Import können Konstanten aus Interfaces importiert werden.	☑	
4.	Die Anweisung package ordnet eine Klasse einem Paket zu.	☑	
5.	Annotationen dürfen annotiert werden.		☑
6.	Annotationen sind vorgegeben und können nicht selbst programmiert werden.		☑
7.	Ein Aufruf der Methode run der Klasse Thread erzeugt einen Thread.		☑
8.	Die Klasse Thread gehört zum Paket java.lang und braucht daher nicht importiert zu werden.	☑	

Anmerkungen:

1. Wahr, equals leitet auf (==) ab und es Existiert immer nur ein einziges Enum Objekt..
2. Wahr, All Implemented Interfaces: Serializable, Comparable¡E¿
3. Wahr, aber sollte nicht oft verwendet werden da sich Math.PI einfacher bzw verständlicher liest als PI
5. Falsch, mir ist kein Beispiel bekannt bei dem es zulässig ist
6. Falsch, implementieren des Interfaces: java.lang.annotation.Annotation
7. Falsch, Run() startet die Run() des Thrads auf dem Aufrufer. Nur Start() startet Run() auf dem neuen Thread.
8. Wahr, java.lang.Thread.

Aufgabe 3: GUI

Beschreiben Sie kurz die Aufgaben der folgenden Klassen/Schnittstellen der Swing-Bibliothek.

JLabel:	Darstellung von Texten und Bildern
JTextArea:	Darstellung einer mehrzeiligen Texteingabe
JTextField:	Darstellung einer einzeiligen Texteingabe
JRadioButton:	Button Gruppe bei der immer nur maximal ein Button selektiert ist
Nennen Sie zwei Layout-Klassen:	FlowLayout, BorderLayout, GridLayout, GridBagLayout...
Nennen Sie zwei mögliche Darstellungen der Farbe „Schwarz“:	Color.black, Color.Black (ist der Verweiß auf black), new Color(0,0,0)

Aufgabe 4: Fehlerkorrektur, Generizität

```
    public static <T extends Comparable<? super T>> void sort(T[] a) {
2       for (int j=1; j<=a.length-1; j++) {
          int k = a[j]; // k muss T statt int sein
4         int i = j-1;

6         while (i>=0 && a[i].compareTo(k)>null) { // > 0 statt > null
            a[i+1] = a[i];
8           i--;
          }
10        a[i+1] = k;
        }
12  }

14  public static void main(String[] args) {
      int[] d = {9,4,1,5,6}; // d muss Integer[] statt int[] sein,
16    //da nur Integer Comparable implementiert
      sort(d);
18  }
```

Aufgabe 5: Programmzuverlässigkeit

```
static int f(int m, int n) {
2  assert m>=n; // Vorbedingung P
   int i = m+1,
4  x = 0;
   assert ... // Schleifeninvariante Q
6  while (i > n+1) {
     x = x+m+n;
8    i = i-1;
     assert ... // Schleifeninvariante Q
10 }
   assert ... // Nachbedingung R
12 return x;
}
```

a)

Welchen Wert berechnet diese Methode? Formulieren Sie eine entsprechende Nachbedingung R.

$R: x = \sum_{j=n}^{m} m + n$

b)

Geben Sie eine geeignete Schleifeninvariante Q an, mit deren Hilfe die partielle Korrektheit der Methode bezüglich P und R nachgewiesen werden kann. Sie brauchen den Nachweis nicht zu führen.

$Q: n+1 \leq i \leq m+1 \wedge x = \sum_{j=0}^{m+1-i} m + n$

c)

Formulieren Sie Q und R als Java-Ausdrücke, sodass diese in den obigen assert-Anweisungen verwendet werden können.

```
Q : assert (n + 1 <= i && i <= m + 1) && x == (m + 1 - i) * (m + n);

R : assert x == (m - n) * (m + n);
```

Aufgabe 6:Datenstrukturen

Gegeben seien die folgenden Klassen Knoten und Suchbaum zur Implementierung binärer Suchbäume:

```
public class Knoten {
  public int wert;
  public Suchbaum links, rechts;

  public Knoten(int wert) {
    this.wert = wert;
    this.links = new Suchbaum();
    this.rechts = new Suchbaum();
  }
}

public class Suchbaum {
  private Knoten wurzel = null;

  public boolean isEmpty() {
    return wurzel == null;
  }

  public void insert(int x) {
    if (isEmpty())
      wurzel = new Knoten(x);
    else
      if (x < wurzel.wert)
        wurzel.links.insert(x);
      else
        wurzel.rechts.insert(x);
  }

  public String toString() {
    if (isEmpty())
      return "_";
    else
      return ":"+wurzel.links+
    wurzel.rechts+wurzel.wert;
  }
}
```

Die Knotenwerte 4 und 5 sind mehr als einmal im Suchbaum enthalten. Als Einfachsumme des Baumes bezeichnen wir die Summe aller Knotenwerte, dabei werden Werte, die mehrfach im Suchbaum enthalten sind, nur einmal addiert. Die Einfachsumme für den obigen Suchbaum beträgt also $0 + 2 + 3 + 4 + 5 + 8 = 22$.

a)

Was gibt das folgende Programmfragment aus?

```
Suchbaum t = new Suchbaum();\\
```

```
t.insert(4); t.insert(5); t.insert(4); t.insert(2); t.insert(1);\\
System.out.println(t);\\
```

:::__1_2::__4_54

b)

Zeichnen Sie den Graphen des Baumes aus a) wie im obigen Beispiel.

Habt ihr in der Übung gesehen.

c)

Schreiben Sie eine Methode boolean negativ(), die genau dann den Wert true liefert, falls mindestens ein Knotenwert des aktuellen Suchbaumes kleiner als 0 ist

```
boolean negativ() {
    if (this.isEmpty()) // testen ob leer
        return false;
    if (this.wurzel.wert < 0) // testen ob kleiner 0
        return true;
    return this.wurzel.links.negativ();// wenn nicht teste links weiter
}
```

d)

Schreiben Sie eine Methode int einfach(), die die Einfachsumme des aktuellen Suchbaumes zurückgibt

```
int einfach() {
    return einfach(new Suchbaum());
}

int einfach(Suchbaum b) {
    if (this.isEmpty())
        return 0;
    if (b.insert2(wurzel.wert)) {
        return this.wurzel.links.einfach(b) + this.wurzel.wert
        +  this.wurzel.rechts.einfach(b);
    }
    return this.wurzel.links.einfach(b) + this.wurzel.rechts.einfach(b);
}

public boolean insert2(int x) {
    if (isEmpty()) {
        wurzel = new Knoten(x);
        return true;
    } else {
        if (x < wurzel.wert)
            return wurzel.links.insert2(x);
        else {
            if (x > wurzel.wert)
                return wurzel.rechts.insert2(x);
            else
                return false;
        }
    }
}
```