1. Grundlegende Konzepte in Java (6 Punkte)

a) Welches der folgenden Literale ist **korrekt** und wenn ja, von welchem **Typ** ist es**?**

`"true"`	☐ nicht korrekt	X korrekt vom Typ String
`'true'`	X nicht korrekt	☐ korrekt vom Typ _____
`true`	☐ nicht korrekt	X korrekt vom Typ Boolean
`5`	☐ nicht korrekt	X korrekt vom Typ int
`5.`	☐ nicht korrekt	X korrekt vom Typ double
`5.e1`	☐ nicht korrekt	X korrekt vom Typ double
`5.ef`	X nicht korrekt	☐ korrekt vom Typ _____

b) **Type-Casting**

Welche der folgenden Zuweisungen bzw. Zuweisungsfolgen sind syntaktisch korrekt?

	Korrekt	Fehlerhaft
`String s = "anton";`	X	☐
`char c = '2';` `int i = c;`	X	☐
`float f = 2.1;`	☐	X
`short sh = 2.0;`	☐	X
`double d = 65.1;` `char c2 = (char)d;`	X	☐

c) **Arithmetische Berechnungen**

Schreiben Sie eine Methode `istVielfaches` in dem unten vorgegebenen Rahmen, welche testet, ob für zwei ganze Zahlen `z1` und `z2` gilt, daß die eine **ein ganzzahliges Vielfaches** der anderen ist. Ist dies der Fall, soll die Methode den `boolean`-Wert `true` zurückgeben, `false` sonst.
Beispiele: `istVielfaches(4,7)==false, istVielfaches(4,8)==true`

```
boolean istVielfaches(int z1, int z2)
{

      if ((z1==0) | (z2==0)) return true;
      return (z1 % z2 == 0) | (z2 % z1 == 0);









}
```

d) **Arrays**

Schreiben Sie eine Methode `zufallsArray` in dem unten vorgegebenen Rahmen, welche ein `int`-Array mit ganzzahligen Zufallszahlen erzeugt, die alle zwischen einer vorgegebenen unteren Grenze `unten` und oberen Grenze `oben` liegen. Diese beiden Grenzen und die Größe des Arrays `größe` werden als Parameter übergeben; die Methode gibt das gefüllte Array zurück. So liefert etwa der Aufruf `zufallsArray(1, 6, 10)` ein Array mit 10 zufällig gewählten Zahlen zwischen 1 und 6 (einschließlich der Grenzen).

Verwenden Sie die Methode `double Math.random()`, die bei jedem Aufruf einen Zufallswert `z` liefert mit $0.0 \leq z < 1.0$. Sie können beim Programmieren der Methode davon ausgehen, dass die übergebenen Parameter plausibel sind (also `unten<oben`, `größe>0`).

```
int [] zufallsArray(int unten, int oben, int größe)
{

        int [] a = new int[größe];

        for (int i=0; i<a.length; i++)

           a[i] = (int)(Math.random()*(oben+1-unten))+unten;

        return a;

}
```

2. Iteration und Rekursion (3 Punkte)

a) **Iterative und rekursive Methode**

Schreiben Sie je eine iterative und eine rekursive Methode, die als Parameter eine ganze Zahl `n>1` erhält und den Wert der Folge

$$f(n) = 1 + 1/2 + 1/3 + \ldots + 1/(n-1) + 1/n$$

berechnet. Es ergibt sich `f(1)=1, f(2)=1.5, f(3)=1.833`... usw.

```
double iterationIterativ (int n)
{


      if (n<1) return 0.0;

       double ergebnis = 1.0;

       for (int i=2; i<=n; i++) ergebnis += 1.0/i;

       return ergebnis;









}


double iterationRekursiv (int n)
{



       if (n<1) return 0.0;

       if (n==1) return 1.0;

       else return iterationRekursiv(n-1)+1.0/n;

}
```

b) **Verschiedene Formen von Iteration**

Schreiben Sie ein Java-Programmfragment, das die gleiche Wirkung erzielt wie die folgende `for`-Schleife, aber dafür eine `while`-Schleife verwendet:

```
for (int y = -100; y < 10; y+=7) System.out.print(y+" ");
```

```
int y = -100;
while (y < 10)
{
        System.out.print(y+" ");
        y+=7;
}
```

3. Suchen und Sortieren (3 Punkte)

a) Schreiben Sie eine Java-Methode `alleVerschieden`, die prüft, ob die Zahlen in einem als Parameter übergebenen `int`-Array alle paarweise voneinander verschieden sind. Die Methode gibt einen `boolean`-Wert zurück: `true`, wenn alle Zahlen paarweise voneinander verschieden sind; `false` sonst.

```
boolean alleVerschieden (int [] a)
{


      for (int i=0; i<a.length-1; i++)
         for (int j=i+1; j<a.length; j++)
            if (a[i]==a[j]) return false;
      return true;









}
```

b) Schreiben Sie eine Methode `arraySort` zum aufsteigenden Sortieren eines `float`-Arrays, indem Sie das folgende einfache Verfahren anwenden (das Array heiße `a`):

> Solange es im Array `a` einen Index `i` mit der Eigenschaft `a[i] > a[i+1]` gibt, vertausche die beiden Elemente.

```
static float[] arraySort(float[] a)
{
   int i=0;
   while (i<a.length-1)
   {
      if (a[i]>a[i+1])
      {
         float f = a[i]; a[i] = a[i+1]; a[i+1] = f;
         i=0;
      }
      else i++;

   }
   return a;
}
```

4. Dynamische Datenstrukturen (6 Punkte)

a) **Lineare Listen**

In einem Java-Programm seien Objekte der Klasse `Knoten` verwendet worden, um eine einfach verkettete Liste mit `int`-Zahlen zu speichern. Die Klasse `Knoten` sei wie folgt gegeben und darf nicht verändert werden:

```
class Knoten
{
   private int wert;
   private Knoten nächster;

   Knoten(int w, Knoten n)
   {
      setWert(w);
      setNächster(n);
   }

   public int getWert() { return wert; }
   public void setWert(int w) { wert = w; }
   public Knoten getNächster() { return nächster; }
   public void setNächster(Knoten n) { nächster = n; }
}
```

Schreiben Sie eine Methode `anzahl`, die prüft, wie oft eine als Parameter übergebene `int`-Zahl `x` in der mit `Knoten`-Objekten aufgebauten Liste vorkommt. Deren Anfangsknoten `liste` und die Zahl `x` werden dieser Methode übergeben. Sie gibt als `int`-Zahl zurück, wie oft `x` in der Liste auftaucht.

```
static int anzahl(Knoten liste, int x)
{


      int gefunden = 0;

      while (liste != null)

      {

         if (liste.getWert()==x) gefunden++;

         liste = liste.getNächster();

      }

      return gefunden;



}
```

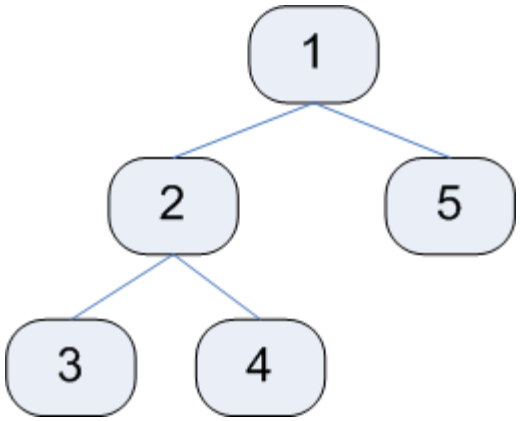
b) **Bäume**

In einem Java-Programm soll mit Objekten der Klasse `BaumKnoten` ein binärer Baum aufgebaut worden, der als Daten Zahlen enthält.

```
class BaumKnoten
{
   private int wert;
   private BaumKnoten links, rechts;

   BaumKnoten(BaumKnoten l, int w, BaumKnoten r)
   {
      links = l; wert = w; rechts = r;
   }

   int getWert() { return wert; }
   BaumKnoten getLinks() { return links; }
   BaumKnoten getRechts() { return rechts; }
}
```

b.1) Bauen Sie zunächst im Speicher durch geschachtelte Aufrufe des `new`-Operators den folgenden Baum auf (die Zahlen entsprechen dem jeweiligen `wert` im Baumknoten). Legen Sie dabei die Referenz auf den Wurzelknoten in der Variablen `wurzel` ab.

```
BaumKnoten wurzel =

new BaumKnoten(
    new BaumKnoten(
        new BaumKnoten(
            null,
            3,
            null),
        2,
        new BaumKnoten(
            null,
            4,
            null)
        ),
    1,
    new BaumKnoten(
        null,
        5,
        null)
    );
```



b.2) Schreiben Sie eine Methode namens `infixDrucken`, die die Zahlenwerte eines derart gespeicherten Baum in der **Infix-Reihenfolge** ausgibt.

```
static void infixDrucken(BaumKnoten b)
{

      if (b == null) return;

      if (b.getLinks()!=null)
      {
         System.out.print(" (");
         infixDrucken(b.getLinks());
         System.out.print(") ");
      }

      System.out.print(b.getWert());

      if (b.getRechts() != null)
      {
         System.out.print(" (");
         infixDrucken(b.getRechts());
         System.out.print(") ");
      }









}
```

5. Klassen, Interfaces, Vererbung (4 Punkte)

a) Interfaces zur Programmstrukturierung

In einem Java-Programm sollen grafische Objekte, und zwar u.a. **Kreise** und **Rechtecke** verwaltet werden. In dem Programm sei ein Interface `flächig` definiert, welches erlauben soll auszurechnen, wie groß die Fläche ist, die ein grafisches Objekt einnimmt.

```
interface flächig
   {    double fläche();    }
```

a.1) Ergänzen Sie die unten gegebenen Klassen `Kreis` und `Rechteck` so, dass sie dieses Interface implementieren. Der **Kreis** wird in der Klasse durch die Angabe des Mittelpunkts (`double mx, my`) und des Radius (`double r`) gespeichert. Die Fläche eines Kreises mit Radius r beträgt πr^2. Dabei steht die Kreiszahl π in der automatisch importierten Klasse `Math` als Konstante `PI` zur Verfügung. Ein **Rechteck** wird durch zwei gegenüberliegende Eckpunkte `(x1, y1)` und `(x2, y2)` definiert, die in `double`-Variablen in der Klasse `Rechteck` gespeichert werden. Die relative Lage der beiden Eckpunkte zueinander sei in keiner Weise eingeschränkt.

```
class Kreis implements flächig
{

   private double mx, my, r;

   Kreis (double mx, double my, double r)
   {  this.r = r; this.mx = mx;  this.my = my; }

   public double fläche()
   { return Math.PI*r*r; }









}
```

```
class Rechteck implements flächig
{

   private double x1, y1, x2, y2;

   Rechteck (double x1, double y1, double x2, double y2)
   {  this.x1=x1; this.y1=y1;  this.x2=x2; this.y2=y2; }


   public double fläche()
   {
      return (x1<x2 ? (x2-x1):(x1-x2)) *
             (y1<y2 ? (y2-y1):(y1-y2));
   }









}
```

a.2) Schreiben Sie eine Methode `flächenSumme`, die bei Übergabe eines Arrays mit `flächig`-Objekten die Summe aller Flächen berechnet und zurückgibt:

```
double flächenSumme(flächig[] fo)
{


   double summe = 0;
   for (int i=0; i<f.length; i++) summe += f[i].fläche();
   return summe;









}
```

b) **Klassen und Objekte**

Ein Java-Programm enthalte folgende Klassenhierarchie:

```
class A
{
   void x() { y(); z(); }
   void y() { System.out.print("yA"); }
   void z() { System.out.print("zA"); }
}

class B extends A
{
   void y() { System.out.print("yB"); }
   void z() { System.out.print("zB"); }
}

class C extends B
{
   void x() { System.out.print("xC"); }
   void y() { x(); z(); }
}
```

- Welche Ausgabe erzeugen die folgenden Programmfragmente?

`A a = new A(); a.x();`	yAzA
`B b = new B(); b.x();`	yBzB
`C c = new C(); c.y();`	xCzB

- Welche Programmfragmente sind korrekt, welche führen zu einem Fehler ?

	Korrekt	Fehlerhaft
`B b = new A();`		X
`A a = new B();`	X	
`B b = new C(); C c = (C)b;`	X	
`B b = new C(); C c = b;`		X

6. Bedienoberfläche (5 Punkte)

a) Objektbaum zeichnen (2,0 Pkt)

Zeichnen Sie einen Objektbaum, der die Hierarchie der verwendeten GUI-Komponenten darstellt:

Jeder Knoten repräsentiert eine Instanz einer GUI-Komponente.

Die Kanten des Baumes geben an, welche Komponenten die Container enthalten.

" Schreiben Sie den Namen der Klasse der Komponente an den Knoten.

Analog zu der Aufgabe 4 vom Blatt 2

b) LayoutManager (1,0 Pkt)

Geben Sie zu jedem der vorkommenden Container-Objekte an, welcher LayoutManager verwendet wird. Begründen Sie, woran Sie ihn erkannt haben.

1, BorderLayout: NORTH,CENTER,SOUTH übereinander, volle Breite, Höhe ungleich2

2,3,4: FlowLayout:

Zeilenweise ausgerichtet und zentriert

Hinweis: 1,2,3,4 sind Labels.

c) Eigenschaften des BorderLayout Managers (1,0 Pkt)

Geben Sie für den BorderLayout-Manager an, welche Eigenschaften unverändert bleiben, wenn die Form des Containers verändert wird.

- NORTH über CENTER und CENTER über SOUTH
- WEST links von CENTER und CENTER links von EAST
- NORTH und SOUTH ganze Breite
- EAST und WEST gleiche Höhe

d) Programmieraufgabe (1,0 Pkt)

Ergänzen Sie das folgende Programmstück so, dass jede Eingabe in das TextField in dem Label out angezeigt wird:

out = new Label ("nichts");

TextField input = new TextField ("", 5);

```
Input.AddActionListener(

     new ActionListener(){

public void actionPerformed(ActionEvent e){

out.setText(e.getActionCommand());

}

} );
```

7. Synchronisation der Prozesse (6 Punkte)

Ein Fahrradhändler verleiht Fahrräder an Ausflugsgruppen.

k Fahrräder stehen zum Verleih zur Verfügung. Eine Gruppe benötigt je nach Anzahl der Personen unterschiedlich viele Fahrräder und gibt sie nach ihrem Ausflug wieder zurück.

Hinweis !

Bevor Sie in (d) eine Monitorklasse zur Simulation der Abläufe implementieren, beantworten Sie die Fragen (a) bis (c):

a) Welche Rollen spielen die Monitore und die Prozesse in dieser Aufgabe? (1,0 Pkt)

Ein Monitor-Objekt – ein Fahrradhändler.

Jede Ausfluggruppe – ein Prozess.

b) Welche Monitor-Operationen mit welchen Parametern benötigen Sie hier? (1,0 Pkt)

```
void leiheFahrräder(int n)

void gibFahrräderZurück(int n)
```

(oder in Worten)

c) Welche der Operationen benötigen Wartebedingungen? Geben Sie die Wartebedingungen zunächst in Sätzen an. Durch Ausführen welcher Operationen können sie erfüllt werden? (1,0 Pkt)

leiheFahrräder(int n){

warten bis mindestens n Fahrräder verfügbar sind

}

Achtung: erfüllbar durch Aufrufe von gibFahrräderZurück!

d) Implementieren Sie eine Monitor-Klasse für diese Aufgabe. Sie soll nur die synchronisierte Ressourcevergabe leisten. (2,0 Pkt)

```
class FHandel{
private int bestand;
FHandel(int n){bestend = n;}
Synchronized void leiheFahrräder(int n){
     While (bestand <n){
     try{wait();}
     catch(interruptedException e){}}
     bestand -=n;
}
synchronised void gibFahrräderZurück(int n){
     bestand +=n;
     notifyAll();
}}
```

e) Nehmen Sie an, Sie sollen die Software für eine Brückenkontrolle entwickeln. An der Auffahrt und der Ausfahrt werden die Fahrzeuge gewogen. Die Kontroll-Software hindert Fahrzeuge am Auffahren auf die Brücke, wenn dadurch ihre Tragfähigkeit überschritten würde. Begründen Sie, weshalb Sie für diese Aufgabe den Monitor aus (d) wiederverwenden können. (1,0 Pkt)

- beides sind Aufgaben der Ressourcenvergabe
- Die Ressourcen sind Fahrräder bzw. das Recht die Brücke zu belasten.
- zu beiden Fällen werden mehrere Einheiten der ressourcen zugleich benötigt:

  n Fahrräder bzw. n kg. Lastfreigabe.

8. Verständnisfragen (2 Punkte aus SWE1)

1. Was bedeutet „dynamische Methodenbindung“?

Die Festlegung, welche Objektmoethode gerufen wird, wird erst zur Laufzeit (abhängig vom tatsächlichen Typ des gerufenen Objektes) getroffen und nicht zur Übersetzungszeit.

2. Was bedeutet das Schlüsselwort `static` in einer Variablendeklaration?

Die Variable ist dann eine Klassenvariable.

3. Was bedeutet Type-Casting?

Dadurch wird der Wert eines Ausdrucks explizit in einen anderen Typ konvertiert (durch Davorschreiben des Zieltyps).

4. Was haben eine abstrakte Klasse und ein Interface gemeinsam?

Von beiden kann man keine Instanzen bilden.

5. Was ist der Unterschied zwischen einer abstrakten Klasse und einer nicht-abstrakten Klasse?

Von abstrakten Klassen kann man keine Instanzen bilden. Nicht-abstrakte Klassen dürfen keine abstrakten Methoden enthalten.

6. Was ist der Unterschied zwischen einer Objekt- und einer Klassenvariablen?

Klassenvariablen existieren genau einmal pro Klasse, unabhängig von der Existenz von Instanzen. Objektvariablen existieren einmal pro Instanz.

7. Nennen Sie drei verschiedene klassen für Komponenten von graphischen Benutzungsoberflächen.

Label, Button, TextField

8. Was bedeutet „gegenseitiger Ausschluss“?

Ein kritischer Abschnitt darf zu jedem Zeitpunkt von höchstens einem Prozess ausgeführt werden.

9. Welche Einschränkung gilt für die Aufrufe von Methoden eines Monitors?

Für jedes Monitor.objekt werdebn die als synchronised gekennzeichneten Methoden unter gegenseitigem Ausschluss aufgerufen.