Praktische Informatik (Software) Vorlesung Softwareentwicklung 1
Prof. Dr. A. Ferscha Hauptklausur am 01. 02. 2001

Zuname _____ Vorname _____ Matr. Nr. _____

Stud. Kennz. ____ Sitzplatz HS __ / ___ / ___ / ___ Punkte ____ Note _____ korr. ___

Fügen Sie fehlende Teile in den folgenden Aufgaben in die umrandeten Kästen ein.

1. Typen und Literale (6 Punkte)

Welche *Typen* haben die folgenden *Literale* aus Java?

Literal	**Typ**
`'a'`	`char`
`"A"`	`String`
`314`	`int`
`false`	`boolean`
`3.14f`	`float`
`3e14`	`double`

2. Zuweisungen (6 = 3 + 3 Punkte)

Welche Ausgabe erzeugt das folgende Programmstück?

Programm:

```
int[] y= new int[3];
int[] x= y;
y[2]= 5;
System.out.println("x="+x[2]);
System.out.println("y="+y[2]);
```

Ausgabe des Programms:

```
x=5
y=5
```

3. Schleifen (10 Punkte)

Ergänzen Sie das angegebene Programm so, daß für die Zahlen von 1 bis 10 ausgegeben wird, ob die Zahl gerade oder ungerade ist. Benutzen Sie dazu **eine Schleife**.

Gewünschte Ausgabe:

```
1 ist ungerade
2 ist gerade
3 ist ungerade
…
10 ist gerade
```

Programm:

```
class Zahlentabelle {
  public static void main(String[] args) {
    for (int i= 1; i<=10; i++) {
      if (i%2==0)
        System.out.println(i+"ist gerade");
      else
        System.out.println(i+"ist ungerade");
    }
  }
}
```

4. Strings (7 Punkte)

Geben Sie jeweils an, was der jeweilige *String*-Methodenaufruf zurückgibt.

Beispiel: `"guido".length()` → ***5***

```
String klaus= "Klausur SWE1";
```

`klaus.indexOf('u')` →	`3`
`klaus.indexOf('k')` →	`-1`
`klaus.indexOf('u', 4)` →	`5`
`klaus.indexOf("SWE1")` →	`8`
`klaus.substring(1)` →	`"lausur SWE1"`

`klaus.substring(0, 5)+" hat die "+klaus+" geschafft!"`

→ `"klaus hat die Klausur SWE1 geschafft!"`

`(klaus.compareTo("Klaus")>0)` →	`true`

5. Binäre Suche rekursiv (13 = 3 + 5 + 5 Punkte)

Das gegebene Programm soll ein Element *x* in einem aufsteigend sortierten Array *a* suchen und den Index des gesuchten Elements oder –1, wenn das Element nicht vorhanden ist, zurückgeben.

Beachten Sie: Ein Aufruf für ein leeres Array soll immer –1 zurückgeben.

```
public int binSearch(int[] a, int x, int l, int r) {
  int midElem= (l+r)/2;                          // Element in der Mitte
  if (a==null || a.length==0 || r<1)
    return –1;                                    // Feld null oder leer
  if (a[midElem]==x)
    return midElem;                               // Element gefunden
  else
    if (x<a[midElem])
      return binSearch( a, x, 1, midElem-1 ); // Suche links weiter
    else
      return binSearch( a, x, midElem+1, r ); // Suche rechts weiter
}
```

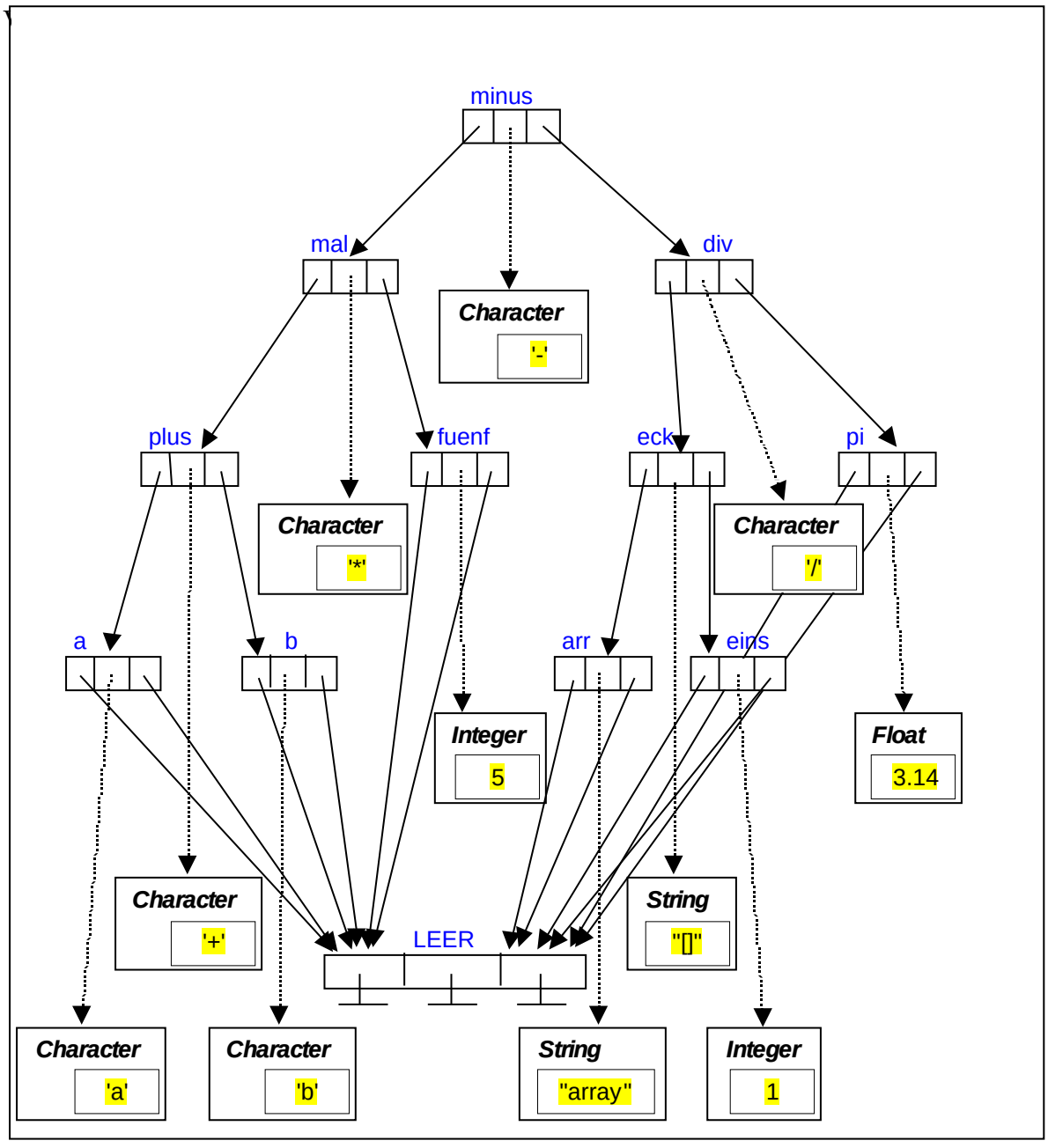
6. Konstruktoren (22 = 11 * 2 Punkte)

Eine Klasse für Bäume von Objekten sei wie folgt festgelegt:

```
public class Baum {
  Object inhalt;
  Baum links, rechts;
  public final static Baum LEER= new Baum();

  public Baum() {
    inhalt= links= rechts= null;
  }

  public Baum(Object x) {
    this(LEER, x, LEER);
  }

  public Baum(Baum l, Object x, Baum r) {
    inhalt= x; links= l; rechts= r;
  }
}
```

Zeichnen Sie die Verweisstruktur, wie Sie durch folgende Aufrufe erzeugt wird:

```
Baum a=     new Baum(new Character('a'));
Baum b=     new Baum(new Character('b'));
Baum plus=  new Baum(a, new Character('+'), b);
Baum fuenf= new Baum(new Integer(5));
Baum mal=   new Baum(plus, new Character('*'), fuenf);
Baum arr=   new Baum("array");
Baum eins=  new Baum(new Integer(1));
Baum eck=   new Baum(arr, new String("[]"), eins);
Baum pi=    new Baum(new Float(3.14));
Baum div=   new Baum(eck, new Character('/'), pi);
Baum minus= new Baum(mal, new Character('-'), div);
```



7. Dynamische Datenstrukturen (Stack) (10 Punkte)

Die Klasse *Stack* sei wie folgt implementiert. Erweitern Sie die Implementierung um eine Methode *contains*, die *true* zurückgibt, wenn ein gesuchtes Objekt *x* im Stack enthalten ist, und *false*, falls nicht.

```
public class Stack {

    private class Node {
       Object element;
       Node   next;
    }

    private Node top;

    public Stack() { top= null; }

    public boolean empty() {
       return top == null;
    }

    ...  // push, pop, …

    public boolean contains(Object x) {
       Node n= top;
       while (n!=null && !(n.element.equals(x)))
          n= n.next;
       return (n!=null);
    }
}
```

Achtung: nicht Verweise vergleichen, sondern Objektinhalte!

8. Abstrakte Klassen und Vererbung (26 = 3 * (3 + 4) + 5 Punkte)

Gegeben sei eine abstrakte Klasse *Geometrie*, die die abstrakte Methode *flaeche*() zur Flächenberechnung bereitstellt.

```
abstract class Geometrie {

   public abstract double flaeche();


}
```

a) Implementieren Sie die erbenden Klassen (Konstruktor und Methoden) Rechteck, Kreis und Quadrat, sodaß eine Verwendung im Sinne der Aufrufe unten möglich wird.

```
class Rechteck extends Geometrie {
   double laenge, breite;

   Rechteck (double laenge, breite) {
      this.laenge= laenge;
      this.breite= breite;
   }

   public double flaeche(){
      return(laenge*breite);
   }

}
```

```
class Kreis extends Geometrie {
   double radius;

   Kreis (double radius) {
      this.radius= radius;
   }

   public double flaeche(){
      return(radius*radius*3.14159);
   }

}
```

```
class Quadrat extends Geometrie {
  double seite;

  Quadrat (double seite) {
    this.seite= seite;
  }

  public double flaeche(){
    return(seite*seite);
  }

}

...
Geometrie[] tabelle= new Geometrie[5];
tabelle[0]= new Kreis(1.0);
tabelle[1]= new Rechteck(3.0, 5.2);
tabelle[2]= new Quadrat(4.0);
tabelle[3]= new Kreis(0.0);
tabelle[4]= new Rechteck(1.0, 2.5);
double gesamtflaeche= 0.0;
for (int i=1; i<tabelle.length; i++) {  // Achtung: Laufweite!
  gesamtflaeche+= tabelle[i].flaeche();
}
System.out.println("Gesamtflaeche:  "+gesamtflaeche);
...
```

b) Welche Ausgabe produziert das obige Programm?

```
Gesamtflaeche: 34.1
```