



Zuname:	<input type="text"/>
Vorname:	<input type="text"/>
Matr. Nr.:	<input type="text"/>
SKZ:	<input type="text"/>

Hörsaal:	<input type="text"/>		
Sitzplatz:	<input type="text"/>	<input type="text"/>	<input type="text"/>
Punkte / Note:	<input type="text"/>	<input type="text"/>	

- Erlaubte Hilfsmittel sind alle (aus-)gedruckten Unterlagen (z.B. Vorlesungsfolien, Übungen, Bücher).
- Nicht erlaubt sind elektronische Hilfsmittel wie Handy, PDA, Notebook, iPad.
- Legen Sie bitte den Studierendenausweis bereit.
- Sie benötigen kein eigenes Papier für die Klausur.
- Bitte verwenden Sie nur **Kugelschreiber oder Füller, auf keinen Fall Bleistifte, rote oder grüne Stifte.**
- Fügen Sie Ihre Antworten bzw. die fehlenden Teile der Aufgaben in die umrandeten Kästchen (Platzhalter) ein. Ausschließlich die in den Kästchen angegebenen Lösungen werden berücksichtigt bzw. korrigiert.
- Die Punktezahl der einzelnen Beispiele ist ein Hinweis auf die vorgesehene Bearbeitungsdauer (**insgesamt 90 Punkte / 90 Minuten**).

Viel Erfolg!

1. Arrays und Schleifen (2+2+1+1+2+2+1+3=14 Punkte)

Ergänzen Sie in den folgenden Klassendefinitionen die fehlenden Teile in den dafür vorgesehenen Kästchen. Hinweise zur Implementierung finden Sie direkt als Kommentare in den Klassendefinitionen.

```
public class Package
{
    private int id;
    private float weight; // weight in kilos
    private float[] xyz = new float[3]; // width, height, depth in meters

    // constructor
    public Package (int id, float weight, float width, float height, float depth)
    {
        this.id = id;
        this.weight = weight;
        this.xyz[0] = width;
        this.xyz[1] = height;
        this.xyz[2] = depth;
    }

    // returns the package id
    public int getId()
    {
        return id;
    }

    /* Determines whether a package is a special package. It is a special package if
    * it is larger than 1 cubic meter, heavier than 50 kilos or in at least one
    * dimension (width, height, depth) larger than 2 meters.
    */
    public boolean isSpecialPackage()
    {
        boolean result = false;

        if (  ) // too much volume or weight?
        {
            result = true;
        } else
        {
            for (int i = 0; i < xyz.length; i++) {
                // any dimension > 2 meters?
                
            }
        }
    }
}
```

```
    }  
    }  
    return result;  
}  
  
// returns the total volume of the package in cubic meters  
public float volume()  
{  
    return xyz[0] * xyz[1] * xyz[2];  
}  
}  
  
public class Logistics  
{  
    /* Calculates the total volume of all packages passed to this method  
    * as array parameter.  
    */  
    public static float entireVolume( Package [] packages )  
    {  
        float result = 0;  
  
        for (int i = 0; i < packages.length; i++) {  
            result += packages[i].volume();  
        }  
  
        return result;  
    }  
  
    /* Returns a newly created array of package, which contains all package of  
    * the inbound array, that are special package.  
    */  
    public static Package[] allSpecialPackages (Package[] packages)  
    {  
        int numSpecialPackages = 0;  
        for (int i = 0; i < packages.length; i++) // count the special packages  
        {  
            if (packages[i].isSpecialPackage()) {  
                numSpecialPackages++;  
            }  
        }  
  
        // create new package-array  
        Package[] specialPackages = new Package[numSpecialPackages];  
        int index = 0;  
        for (int i = 0; i < packages.length; i++) // insert special packages  
        {  
            if (packages[i].isSpecialPackage()) {  
                specialPackages[index]= packages[i];  
                index++;  
            }  
        }  
  
        return specialPackages;  
    }  
}
```

2. Exception Handling

(5 * 2 = 10 Punkte)

Eingabe-/Ausgabeströme („Streams“) müssen in Java richtig geschlossen werden wenn sie nicht länger benötigt werden. Das wird im Normalfall durch die Methode `close()` sichergestellt:

```
public class InputExceptions
{
    public static void main( String[] args )
    {
        InputStream input = new FileInputStream("c:\\data\\input.txt");
        int data = input.read();

        while(data != -1) {
            //do something with data...
            doSomethingWithData(data);
            data = input.read();
        }

        input.close();
    }
}
```

Der Codeausschnitt sieht auf den ersten Blick korrekt aus. Was passiert aber nun wenn beispielsweise innerhalb der Methode `doSomethingWithData(data)` ein Problem auftritt und eine Exception geworfen wird? Richtig erkannt, der `InputStream` wird nicht geschlossen!

Ergänzen Sie nachfolgend den ursprünglichen Code durch eine Ausnahmebehandlung (`IOException`). Beachten (und berücksichtigen) Sie dabei auch, dass die `close()`-Methode selbst auch eine Exception werfen kann (z.B. falls Sie `close()` aufrufen obwohl der `InputStream` bereits geschlossen worden ist...).

```
public class InputExceptions
{
    public static void main( String[] args )
    {
        InputStream input = null;

        try
        {
            input = new FileInputStream("c:\\data\\input.txt");
            int data = input.read();

            while(data != -1) {
                //do something with data...
                doSomethingWithData(data);
                data = input.read();
            }
        }
        catch (IOException e)
        {
            //do something with e... log, perhaps rethrow etc.
        }
        finally
        {
            try
            {
                if(input != null) input.close();
            }
            catch (IOException e)
            {
                //do something, or ignore.
            }
        }
    }
}
```

3. Listen**(5+3+3=11 Punkte)**

Die Klasse `List` dient zur Repräsentation von Listen von Zahlen. Jedes Listenelement wird als Objekt der Klasse `Element` dargestellt. Ein Listenelement enthält eine Zahl (`value`) und einen Verweis auf den Nachfolger. Die Zahl wird in dem Attribut `'value'` gespeichert, das Attribut `'next'` zeigt auf das nächste Element der Liste. Das letzte Element einer Liste hat keinen Nachfolger (Attribut `'next'` zeigt auf `null`).

Objekte der Klasse `List` haben ein Attribut `'start'`, das auf das erste Element der Liste zeigt. Eine leere Liste hat kein erstes Element (Attribut `'start'` zeigt auf `null`).

```
public class List
{
    public Element start;
    public List(Element start)
    {
        this.start = start;
    }
}
```

```
public class Element
{
    public int value;
    public Element next;
    public Element(int value, Element next)
    {
        this.value = value;
        this.next = next;
    }
}
```

Ergänzen Sie die Methode `public void duplicate()` der Klasse `List`. Die Methode `'duplicate'` soll die jeweilige Liste dahingehend verändern, dass von jedem Element eine Kopie (neues Element mit gleichem Wert) erzeugt und diese Kopie direkt als Nachfolger des ursprünglichen Elements eingefügt wird.

Hinweise:

- Falls die aktuelle Liste leer ist, soll sie durch `'duplicate'` auch nicht verändert werden
- Bsp. die Liste **7-3-11-6** wird durch `'duplicate'` zur Liste **7-7-3-3-11-11-6-6**

```
// duplicates each list element (original list is extended)
public void duplicate ()
{
    Element current = this.start; // object to traverse the list
    Element element = null;       // place holder for the duplicated object

    // traverse the list until the end (null) is reached
    while (current != null)
    {
        // create new list element as a copy of current element
        element = new Element(current.value, current.next);

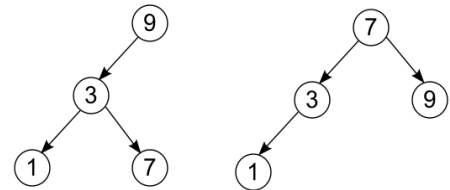
        // add the new element to the existing list (right after the 'current' element)
        current.next = element;

        // move forward in the list
        current = element.next;
    }
}
```

4. Rekursion/Bäume

(2 * 3 + 2 * 4 = 14 Punkte)

Bei geordneten Binärbäumen ist es möglich, dass zwei Instanzen die gleichen Elemente enthalten, sich jedoch trotzdem hinsichtlich ihrer Struktur unterscheiden – wie es nebenstehende Grafik verdeutlicht. Beide Bäume enthalten die gleichen Elemente und auch die Ordnungsrelation ist in beiden korrekt erfüllt, der Aufbau unterscheidet sich jedoch.



Die Klasse `BinaryTree` definiert die Methode `equals`, welche überprüft ob zwei Bäume strukturell identisch sind. Dazu benutzt sie eine weitere Methode `serializeInorder`, welche einen beliebigen Teilbaum `inorder` traversiert und alle besuchten Knoten in dieser Reihenfolge in einen `StringBuffer` ablegt. Strukturelle Gleichheit zweier Bäume liegt genau dann vor, wenn ihre `String`-basierten serialisierten Darstellungen identisch sind.

Ergänzen Sie die fehlenden Codeabschnitte in der Klasse `BinaryTree`, sodass die gewünschte Funktionalität realisiert wird.

Gegeben ist folgende Knotenklasse (Elemente des Binärbaums):

```

public class Node
{
    int content; //value of the node
    Node left;
    Node right;

    public Node (int value)
    {
        this.content = value;
        this.left = null;
        this.right = null;
    }
}
  
```

Klasse `BinaryTree`:

```

public class BinaryTree {
    Node root = null;

    // buffer for the tree serialization
    private static StringBuffer buffer = null;

    // constructor
    public BinaryTree(Node root) {
        this.root = root;
    }

    // Compares the two trees t1 and t2 regarding their structure. If both have the
    // same structure return 'true', otherwise 'false'.
    static boolean equals(BinaryTree t1, BinaryTree t2) {
        String strT1, strT2; // serialized trees to be stored here

        // create new buffer, serialize t1 and store it in strT1
        buffer = new StringBuffer();
        serializeInorder(t1.root);
        strT1 = buffer.toString();

        // create new buffer, serialize t2 and store it in strT2
        buffer = new StringBuffer();
        serializeInorder(t2.root);
        strT2 = buffer.toString();

        return strT1.equals(strT2);
    }
}
  
```

```

/* Traverses through a partial tree beginning with node 'n' as root. Each of the
 * nodes, which are visited in inorder-sequence is stored into a
 * 'StringBuffer'. The single node-values are separated from each other
 * by the character '-'. The whole serialized representation is stored in the
 * static variable 'buffer'.
 */
publ if (n.right != null) {
        serializeInorder(n.right);
    }

    buffer.append(n.content);
    buffer.append('-');

    if (n.left != null) {
        serializeInorder(n.left);
    }
}
    
```

5. Methoden, Vererbung, Polymorphismus (3+3=6 Punkte)

Welche Ausgaben produzieren die folgenden Java-Programme?

a) Vererbung und Polymorphismus

(3 Punkte)

Quellcode	Ausgabe
<pre> class A { public A() { this("hello", " world"); } public A(String s) { System.out.println(s); } public A(String s1, String s2) { this(s1 + s2); } } class B extends A { public B() { super("good bye"); } public B(String s) { super(s, " world "); } public B(String s1, String s2){ this(s1 + s2 + "!"); } } Was gibt das folgende Programm aus? public class TestClass { public static void main(String args[]) { A b = new B("good bye"); } } </pre>	<p style="color: red; text-align: center;">"good bye world"</p>

b) Überschriebene Methoden

(3 Punkte)

Quellcode	Ausgabe
<pre>class BaseClass { public void print(String s) { System.out.println("BaseClass :" + s); } } class SubClass extends BaseClass { public void print(String s) { System.out.println("SubClass :" + s); // call method } }</pre> <p>Durch welche Anweisung kann an der Stelle „call method“ die überschriebene Methode der Basisklasse aufgerufen werden?</p>	<pre>super.print(s);</pre>

6. Stringverarbeitung

(10 Punkte)

Vervollständigen Sie die unten angegebene Funktion `StringBuffer compressSpace(StringBuffer s)`, die zwei oder mehr als zwei Leerzeichen in der übergebenen Zeichenkette zu einem einzigen Leerzeichen verschmelzen lässt.

Hinweise:

- Die Klasse `StringBuffer` bietet einige Methoden an, die Sie hier natürlich verwenden dürfen:
 - `public int lastIndexOf (String str)`
Returns the index within this string of the rightmost occurrence of the specified substring. The rightmost empty string "" is considered to occur at the index value `this.length()`.
Parameters: `str` - the substring to search for.
Returns: if the string argument occurs one or more times as a substring within this object, then the index of the first character of the last such substring is returned. If it does not occur as a substring, -1 is returned.
 - `public StringBuffer deleteCharAt (int pos)`
Removes the char at the specified position in this sequence. This sequence is shortened by one char.
Parameters: `index` - Index of char to remove
Returns: This object.

```
public class CompressSpace {

    static StringBuffer compressSpace(StringBuffer sb) {
        int index = sb.lastIndexOf(" "); //2 blanks

        while ( index >= 0 )
        {
            sb.deleteCharAt(index);
            index = sb.lastIndexOf(" "); //2 blanks
        }

        return sb;
    }

    public static void main( String[] args )
    {
        StringBuffer sb = compressSpace(new StringBuffer("Hello. This is a Example"));
        System.out.println(sb.toString()); //Output: "Hello. This is a Example."
    }
}
```

7. Rekursion in Binärbäumen

(8 Punkte)

Implementieren Sie in der unten angeführten Klasse `Tree` eine rekursive Methode `height(TreeNode node)`, die die Höhe des Baums berechnet.

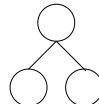
Beispiele:

(leerer Baum)

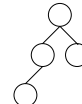


h=0

h=1



h=2



h=4

Gegeben ist folgende Knotenklasse (Elemente des Binärbaums):

```
public class TreeNode
{
    int value; //value of the node
    TreeNode left;
    TreeNode right;

    ...
}
```

```
public class Tree {
    private TreeNode root;

    public int height()
    {
        return height(root); // first call
    }

    // recursive height method; hint: call recursion for both subtrees!
    private int height (TreeNode node)
    {
        if (node == null)
        {
            return 0;
        }

        int h1 = height(node.left) + 1;
        int h2 = height(node.right) + 1;

        return h1 >= h2 ? h1 : h2;
    }
}
```

8. Palindrom-Test durch Stackreversierung (4+5=9 Punkte)

In der folgenden Aufgabe sollen Sie die Methode `reverseString()` vervollständigen, die einen übergebenen `String s1` unter Zuhilfenahme eines push/pop-Stacks `st` (first in-last out; FILO) reversiert (umdreht) und zurückgibt.

„Stack“ bietet folgende Funktionalität, die Sie natürlich verwenden dürfen:

Constructor Summary	
<code>Stack()</code>	Creates an empty Stack.

Method Summary	
boolean <code>empty()</code>	Tests if this stack is empty.
Object <code>peek()</code>	Looks at the object at the top of this stack without removing it from the stack.
Object <code>pop()</code>	Removes the object at the top of this stack and returns that object as the value of this function.
Object <code>push(Object item)</code>	Pushes an item onto the top of this stack.
int <code>search(Object o)</code>	Returns the 1-based position where an object is on this stack.

```

public class ReverseStack {

    private static Stack<Character> st = new Stack();

    public static void main(String[] args)
    {
        String s1;

        System.out.println("Enter text: ");
        s1 = Input.readString();

        if ((reverseString(s1)).equals(s1))
            System.out.println(s1 + " is a palindrome.");
        else
            System.out.println(s1 + " is not a palindrome.");
    }

    public static String reverseString (String s1)
    {
        StringBuilder sb = new StringBuilder ();

        // add (push) single characters into the stack
        for (int i=0; i < s1.length(); i++) {
            st.push (s1.charAt(i));
        }

        // pop stack elements into 'sb', return the reversed string
        while (!st.empty()) {
            char c = st.pop();
            sb.append(c);
        }

        return sb.toString();
    }
}

```

9. Bool'sche Algebra**(2+2*3=8 Punkte)**

Formulieren Sie für folgende in Textform gegebene Aufgabe eine korrekte if-Kaskade unter Verwendung des Integer-Parameters `traffic_light` (-1: keine Ampel, 0: grün, 1: gelb, 2: rot), sowie der beiden bool'schen Parameter `main_road` und `right_vehicle`.

„Falls es bei einer Kreuzung (Crossing) eine Ampel (`traffic_light`) gibt, darf man immer dann fahren wenn sie nicht rot ist. Falls es keine Ampel gibt, darf man nur dann fahren wenn man sich auf einer Vorrangstraße(`main_road`) befindet oder kein Fahrzeug von rechts (`right_vehicle`) kommt.“

```
public class Crossing {  
  
    public static boolean drivingAllowed (int traffic_light, boolean main_road,  
                                         boolean right_vehicle)  
    {  
  
        boolean driving = false;  
  
        if ( traffic_light != -1 ) // crossing with traffic light  
        {  
  
            if (traffic_light != 2) {  
                driving = true;  
            }  
  
        } else // no traffic light  
        {  
  
            if (main_road || !right_vehicle) {  
                driving = true;  
            }  
  
        }  
  
        return driving;  
    }  
}
```