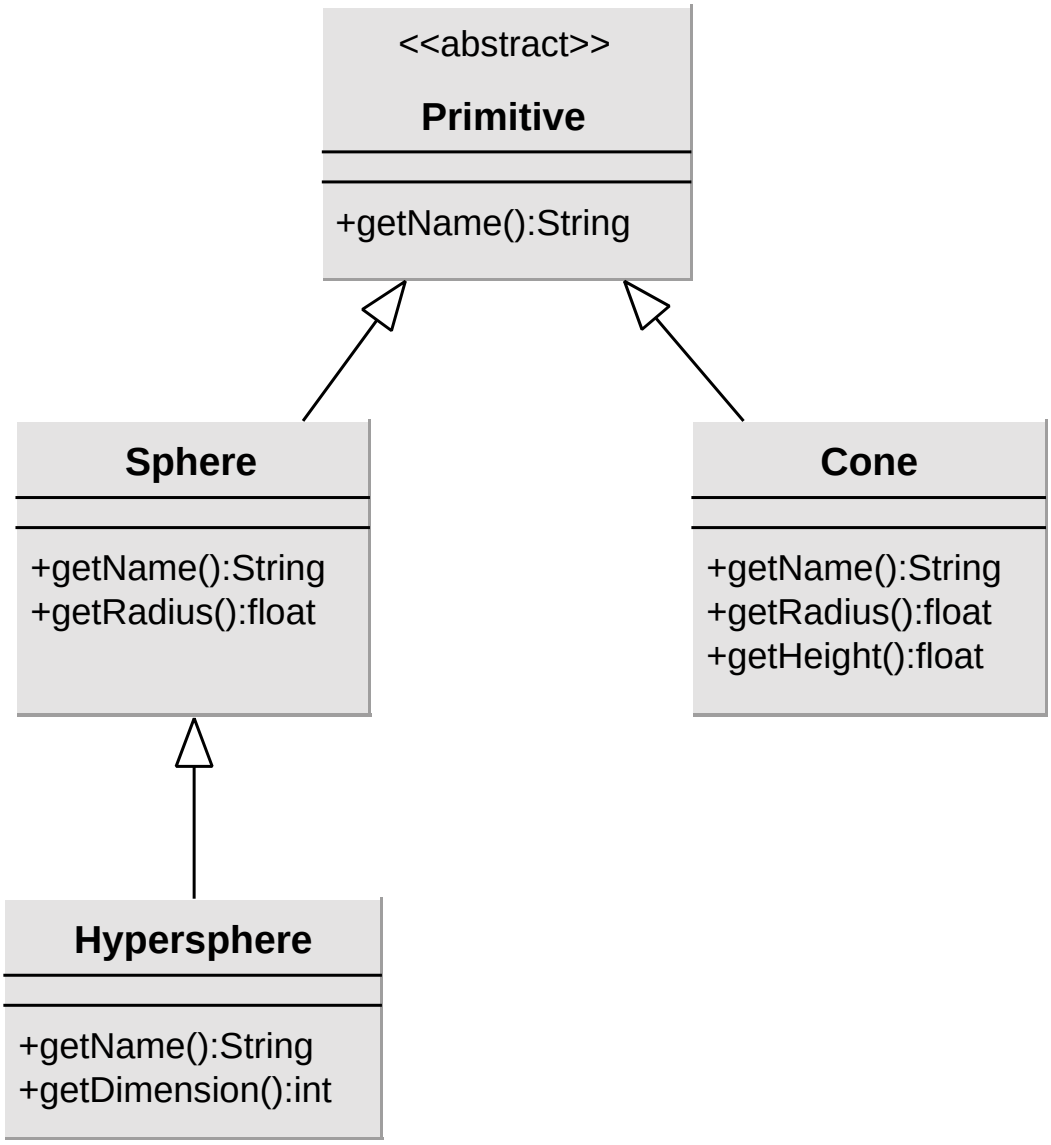
Übungsgruppe 3; Zweiergruppe 23
274196 Lucas Brutschy & 273784 Philipp Fischer

Übung Programmierung
Blatt 8

Aufgaben 1 – Geometrieklasse / Klassenhierarchie

a)



Übungsgruppe 3; Zweiergruppe 23
274196 Lucas Brutschy & 273784 Philipp Fischer

b)

Anweisung 1:
Es wird "5" ausgegeben

Anweisung 2:
Es wird "`cone`" ausgegeben

Anweisung 3:
Compiliert nicht, da `Primitive` keine Methode `getRadius()` implementiert

Anweisung 4:
Es wird "`h-sphere`" ausgegeben.

Anweisung 5:
Compiliert nicht, da eine `Sphere` nicht implizit in eine `HyperSphere` gecasted werden kann. Es muss ein explizites casting erfolgen.

Anweisung 6:
Es wird "`h-sphere`" ausgegeben.

Anweisung 7:
Zur Laufzeit tritt ein Fehler auf, da ein Objekt vom internen Typ `Sphere` nicht in ein Objekt vom Typ `Cone` gecasted werden kann, auch wenn der Verweis vom Typ `Primitive` ist.

Anweisung 8:
Es wird "`h-sphere`" ausgegeben.

Aufgabe 2 – Binärer Suchbaum

Die Quelltexte zu dieser Aufgabe befinden sich im Anhang.

Knoten.java

```java
/**
 * SHEET 8 - EXERCISE 2
 * Knoten, represents one node of a binary tree.
 * @see Baum
 * @author 273784 Philipp Fischer & 274196 Lucas Brutschy
 * @version 17.12.2006
 * platform: J2RE 1.5.0_08-b03, Linux 2.6.17-gentoo
*/
public class Knoten {
    // The value of this node
    private int value;

    // Its children
    private Knoten left, right;

    /**
     * The constructor sets the new nodes value and sets
     * the children to null
     */
    public Knoten (int value) {
        this.value = value;
        left = right = null;
    }

    /**
     * Returns the left child
     */
    public Knoten getLeft() {
        return left;
    }

    /**
     * Sets the left child
     */
    public void setLeft(Knoten left) {
        this.left = left;
    }

    /**
     * Returns the right child
     */
    public Knoten getRight() {
        return right;
    }

    /**
     * Sets the right child
     */
    public void setRight(Knoten right) {
        this.right = right;
    }

    /**
     * Returns the nodes value
     */
    public int getValue() {
        return value;
    }

    /**
     * Sets the nodes value
     */
    public void setValue(int value) {
        this.value = value;
    }
}
```

Baum.java

```java
/**
 * SHEET 8 - EXERCISE 2
 * Baum, an implementation of a binary tree.
 * @see Knoten
 * @author 273784 Philipp Fischer & 274196 Lucas Brutschy
 * @version 17.12.2006
 * platform: J2RE 1.5.0_08-b03, Linux 2.6.17-gentoo
*/
public class Baum {
    /**
     * The root. All nodes (Knoten) of the tree (Baum) can
     * be accessed by traversing the tree starting with this node
     */
    private Knoten root;

    /**
     * The standard constructor initializes a tree without nodes
     */
    public Baum () {
        root = null;
    }

    /**
     * Adds a value to the tree
     * @param value the value of the new node
     */
    public void einfuegen(int value) {
        if(root == null) {
            root = new Knoten(value);
        } else {
            einfuegen(root, value);
        }
    }

    /**
     * Inserts a value into a certain sub-tree
     * @param insertionRoot the sub-tree in which to insert the value
     * @param value the value of the new node
     */
    private boolean einfuegen(Knoten insertionRoot, int value) {
        if(insertionRoot == null) return true;
        if(value > insertionRoot.getValue()) {
            if(einfuegen(insertionRoot.getRight(), value)) {
                insertionRoot.setRight(new Knoten(value));
            }
        } else {
            if(einfuegen(insertionRoot.getLeft(), value)) {
                insertionRoot.setLeft(new Knoten(value));
            }
        }
        return false;
    }

    /**
     * Searches for a specific node in the tree
     * @param wert the wert to be searched for
     * @return true if the wert exists. false if it doesnt.
     */
    public boolean suche (int value) {
        Knoten iterator = root;
        while((iterator != null) && (iterator.getValue() != value)) {
            // decide which way to go
            if(iterator.getValue() > value) {
                iterator=iterator.getLeft();
            } else {
                iterator=iterator.getRight();
            }
        }
        return ((iterator != null) && (iterator.getValue() == value));
    }

    /**
     * Searches for the first node with the given value and deletes it
     * @param value the value to be searched for
     */
    public void loesche (int value) {
        Knoten iterator = root, parent = null;
        boolean rightSide = false;
        while((iterator != null) && (iterator.getValue() != value)) {
            parent = iterator;
            if(iterator.getValue() > value) {
                rightSide = false;
                iterator = iterator.getLeft();
            } else {
                rightSide = true;
                iterator = iterator.getRight();
            }
        }

```

Baum.java

```
90          if((iterator != null) && (iterator.getValue() == value)) {
91              if(iterator.getRight() == null) {
92                  // Replace the deleted node by its left subtree
93                  setChild(parent, rightSide, iterator.getLeft());
94              } else {
95                  if(iterator.getLeft() != null) {
96                      // Attach the left subtree to the leftmost node of the right subtree
97                      Knoten anotherIterator = iterator.getRight(), anotherParent = iterator.getRight();
98                      while(anotherIterator != null) {
99                          anotherParent = anotherIterator;
100                         anotherIterator = anotherIterator.getLeft();
101                     }
102                     setChild(anotherParent, false, iterator.getLeft());
103                 }
104                 // Replace the deleted node by its right subtree
105                 setChild(parent, rightSide, iterator.getRight());
106             }
107         }
108
109     }
110
111     /**
112      * Helper function. It attaches a child node to a parent node
113      * @param parent the knoten to which the child will be attached. null will replace root by child
114      * @param rightSide if true, the child will be attached to the right subtree of the parent. Otherwise left.
115      * @param child the knoten which will attached to the parent
116      */
117     private void setChild(Knoten parent, boolean rightSide, Knoten child) {
118         if(parent == null) {
119             root = child;
120         } else {
121             if(rightSide) {
122                 parent.setRight(child);
123             } else {
124                 parent.setLeft(child);
125             }
126         }
127     }
128
129     /**
130      * Returns a string that represents the tree in text-form
131      */
132     public String traverse() {
133         return traverse(root, 0) + "\n";
134     }
135
136     /**
137      * Returns a string that represents the given sub-tree in text-form
138      */
139     public String traverse(Knoten node, int indentation) {
140         String output = new String();
141
142         if(node == null) return output;
143         output += traverse(node.getRight(), indentation+1);
144
145         for(int i = 0; i < indentation+1; i++) {
146             output += "-";
147         }
148
149         output += node.getValue() + "\n";
150
151         output += traverse(node.getLeft(), indentation+1);
152
153         return output;
154     }
155
156     /**
157      * toString can be used to display the trees content in text-form
158      */
159     public String toString() {
160         return traverse();
161     }
162 }
```