Physikalisch–Technische Lehranstalt Wedel
Prof. Dr. Uwe Schmidt

17. August 1999

Aufgaben zur Klausur **Objektorientierte Programmierung** im SS 99 (IA 252)

Zeit: 60 Minuten erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen ausschließlich an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Viel Erfolg !

Diese Klausur besteht einschließlich dieses Deckblattes aus 10 Seiten

---

Aufgabe 1:

Gegeben seien die folgende Schnittstelle

```
public interface Dictionary {
    public abstract Dictionary insert(Object key, Object attr);

    public abstract Object lookup(Object key);

    public abstract boolean isEmpty();

    public abstract int card();
}
```

und eine Schnittstelle für Vergleichsfunktionen, die ähnlich wie die *strcmp*–Funktion aus der C–Bibliothek arbeiten.

```
public interface Compare {
    // cmp arbeitet wie strcmp in C
    // o1 < o2 : -1
    // o1 = o2 : 0
    // o1 > o2 : +1
    public abstract int cmp(Object o1, Object o2);
}
```

Diese Schnittstelle für einfache ADTs für Verzeichnisse (dictionaries) soll von einer Klasse *Bintree* implementiert werden. Teile der Klasse *Bintree* sind vorgegeben, diese Klasse arbeitet mit Delegation. Der eigentliche binäre Baum, in dem die Schlüssel–Wert-Paare gespeichert werden, ist als Datenfeld vorhanden, genauso wie eine Referenz auf die Vergleichsfunktion, mit der im Verzeichnis gearbeitet werden soll.

Der eigentliche binäre Baum wird aus Objekten der lokalen Klassen *Node* und *Empty* aufgebaut, wobei der leere Baum durch eine Referenz auf ein Objekt der Klasse *Empty* repräsentiert wird. Ein Konstruktoraufruf von *Bintree* erzeugt ein leeres Verzeichnis.

Entwickeln Sie die fehlenden Methodenrümpfe so, daß in *Bintree* die Daten wie in einem binären Baum gespeichert werden.

Die *insert*–Methode soll ein neues Wertepaar in das Verzeichnis eintragen, wenn der Schlüssel schon vorhanden ist, soll das Attribut aktualisiert werden.

Die *lookup*–Methode soll, wenn der Schlüssel existiert, das zugehörige Attribut zurückgeben, sonst `null`.

isEmpty soll testen, ob das Verzeichnis leer ist, *card* soll die Anzahl der gespeicherten Paare berechnen.

Hinweis: Versuchen sie erst, das Zusammenspiel der Klassen zu verstehen, dann die einfachen Methoden zu entwickeln und zum Schluß die komplexeren Methoden.

```
public class Bintree implements Dictionary {

    private Compare c;
    private Empty e;
    private Dictionary d;

    public Bintree(Compare c) {
        this.c = c;
        this.e = new Empty();
        this.d = e;
    }

    class Node implements Dictionary {
        private Object key;
        private Object attr;
        private Dictionary left;
        private Dictionary right;

        Node(Object key, Object attr) {

            ..........................

            ..........................

            ..........................

            ..........................
        }
```

```
// Bintree.Node (cont.)
        public Dictionary insert(Object key, Object attr) {
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
        }
```

```
// Bintree.Node (cont.)
        public Object lookup(Object key) {
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
            ..........................
        }
        public boolean isEmpty() {
            ..........................
            ..........................
        }
        public int card() {
            ..........................
            ..........................
            ..........................
        }
    }
```

```
// Bintree.Empty
    class Empty implements Dictionary {
        public Dictionary insert(Object key, Object attr) {

            ..........................

            ..........................
        }

        public Object lookup(Object key) {

            ..........................

            ..........................
        }

        public boolean isEmpty() {

            ..........................

            ..........................
        }

        public int card() {

            ..........................

            ..........................
        }
    }
```

```
// Bintree (cont.)
    public Dictionary insert(Object key, Object attr) {

        ..........................

        ..........................
    }

    public Object lookup(Object key) {
        return
            d.lookup(key);
    }

    public boolean isEmpty() {

        ..........................

        ..........................
    }

    public int card() {

        ..........................

        ..........................
    }
}
```

Aufgabe 2:

Gegeben sei das folgende Testprogramm:

```
public class ExcTest {

    public static void main(String[] argv) {
        int i = 0;
        try {
            i = Integer.parseInt(argv[0]);
        }
        catch ( Exception e ) { }
        test(i);
    }

    public static void test(int i) {

    tryblock:
        try {
            switch ( i ) {
            case 1:
                System.out.println("test:  call wow");
                wow();
                break;
            case 2:
                System.out.println("test:  call foo");
                foo();
                break;
            default:
                System.out.println("test:  normal exit");
            }
        }
        catch ( MyException e ) {
            System.out.println("test:  caught MyException");
        }
        catch ( Exception e ) {
            System.out.println("test:  caught Exception");
        }
        finally {
            System.out.println("test:  exec finally");
        }
        System.out.println("test:  normal exit");
    }
```

```
    public static void foo()
        throws MyException {
        try {
            System.out.println("foo :  throw MyException");
            throw
                new MyException("foo test");
        }
        finally {
            System.out.println("foo :  exec finally");
        }
    }

    public static void wow()
        throws MyException {
        int a, b=1, c=0;
        try {
            System.out.println("wow :  division by 0");
            a = b/c;
        }
        catch (Exception e) {
            System.out.println("wow :  caught Exception");
            System.out.println("wow :  throw MyException");

            throw
                new MyException("wow test");
        }
    }
}

class MyException extends Exception {
    public
        MyException(String s) {
        super(s);
    }
}
```

Welche Ausgabe wird bei einem Aufruf von `java ExcTest 1` erzeugt:

.....

.....

.....

.....

.....

.....

.....

Welche Ausgabe wird bei einem Aufruf von `java ExcTest 2` erzeugt:

.....

.....

.....

.....

.....

.....

.....
