Physikalisch–Technische Lehranstalt Wedel
Prof. Dr. Uwe Schmidt

25. Januar 2002

Aufgaben zur Klausur **Objektorientierte Programmierung** im WS 2001/02 (IA 252)

Zeit: 75 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:

Die folgenden Klassen, **List** und deren Unterklasse dienen zur Implemetierung einer Listen–Datenstruktur.

In diesen Klassen sind einige Methodenrümpfe zu entwickeln, und zwar zur einheiltlichen und flexiblen Verarbeitung aller Elemente einer Liste. Hierzu werden zwei Schnittstellen **Function** und **Function2** verwendet. Diese deklarieren jeweils eine Methode **apply** zum Anwenden einer 1– und einer 2–stelligen Funktion.

Die Methode **map** soll aus einer Liste durch Anwenden einer Funktion **f** auf alle Elemente eine neue Liste berechnen.

Die Methode **zipWith** soll aus zwei Listen durch Anwenden einer 2-stelligen Funktion auf die einzelnen Elemente der beiden Listen eine neue Liste berechnen. Sind die beiden Listen unterschiedlich lang, so sollen die zusätzlichen Elemente der längeren Liste ignoriert werden.

Nimmt man als Beispiel für die 2–stellige Funktion eine Addition und wendet diese mit **zipWith** auf zwei Listen von Zahlen an, so erhält man eine Vektoraddition.

Tipp: Bitte lesen Sie alle Programmteile einschließlich des Testprogramms sorgfältig durch, bevor Sie mit der Bearbeitung der Aufgabe beginnen.

Die abstrakte Klasse für die Listen–Struktur und ihrer Ausprägungen:

```
public abstract class List
{
    private static final
        List el = new EmptyList();

    public static List empty()
    {
        return el;
    }

    public abstract Object head();

    public abstract List tail();

    public abstract boolean isEmpty();

    public List cons(Object head)
    {
        return new Node(head, this);
    }

    public final List cons(int i)
    {
        return cons(new Integer(i));
    }

    public abstract List map(Function f);

    public abstract List zipWith(Function2 f, List l2);
```

```
private static final class EmptyList extends List
{
    public Object head()
    {
        throw
            new RuntimeException("head of empty list");
    }

    public List tail()
    {
        throw
            new RuntimeException("tail of empty list");
    }

    public boolean isEmpty()
    {
        return true;
    }

    public String toString()
    {
        return "nil";
    }

    public List map(Function f)
    {

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

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

    public List zipWith(Function2 f, List l2)
    {

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

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

```
private static final class Node extends List
{
    private Object head;
    private List tail;

    Node(Object head, List tail)
    {
        this.head = head;
        this.tail = tail;
    }

    public Object head()
    {
        return head;
    }

    public List tail()
    {
        return tail;
    }

    public boolean isEmpty()
    {
        return false;
    }

    public String toString()
    {
        return
            head.toString() +
            "  .  " +
            tail.toString();
    }
```

```
        public List map(Function f)
        {
        ..........................
        ..........................
        ..........................
        ..........................
        ..........................
        ..........................
        }

        public List zipWith(Function2 f, List l2)
        {
        ..........................
        ..........................
        ..........................
        ..........................
        ..........................
        ..........................
        ..........................
        ..........................
        ..........................
        }
    }
}
```

Die Schnittstellen

```
interface Function
{
    Object apply(Object x);
}
```

und

```
interface Function2
{
    Object apply(Object x1, Object x2);
}
```

und eine Beispiel–Klasse für eine Increment–Funktion

```
public class Plus1 implements Function
{
    public Object apply(Object x)
    {
        return
            new Integer(((Integer)x).intValue() + 1);
    }
}
```

Vervollständigen Sie folgende Klasse für eine Funktion zum Erhöhen um einen festen ganzzahligen Wert:

```
public class PlusN implements Function
{
    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

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

Vervollständigen Sie folgende Klasse für eine 2-stellige Funktion zum Addieren zweier ganzer Zahlen:

```
public class Add implements Function2
{
    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

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

Eine einfache kleine Testklasse für die Listenimplementierung:

```
public class Test
{
    public static void main(String [] argv)
    {
        List l1
            = List.empty().cons(1).cons(2).cons(3);

        List l2
            = List.empty().cons("abc").cons(5).cons(3).cons(1);

        List l3
            = List.empty().cons(5).cons(3).cons(1).cons("abc");

        Function plus2
            = new PlusN(2);

        Function2 add
            = new Add();

        System.out.println(l1);
        System.out.println(l1.map(plus2));

        System.out.println(l1.zipWith(add, l1));
        System.out.println(l1.zipWith(add, l2));
        System.out.println(l1.zipWith(add, l3));
    }
}
```

Welche 5 Ausgabezeilen erzeugt das Testprogramm:

1) .....

2) .....

3) .....

4) .....

5) .....