Technische Universität München
Fakultät für Informatik
Prof. Dr. Seidl

WS 2004/2005
12. Februar 2005

Lösungsvorschläge der Zwischenklausur zu Einführung in die Informatik I

Aufgabe 1 Fehlerbehandlung mit Exceptions

a) Was berechnet dieses Programmstück?
Ganzzahlige Division

b) Welche Ausnahmen/Exceptions können hier auftreten? Wo können diese auftreten?

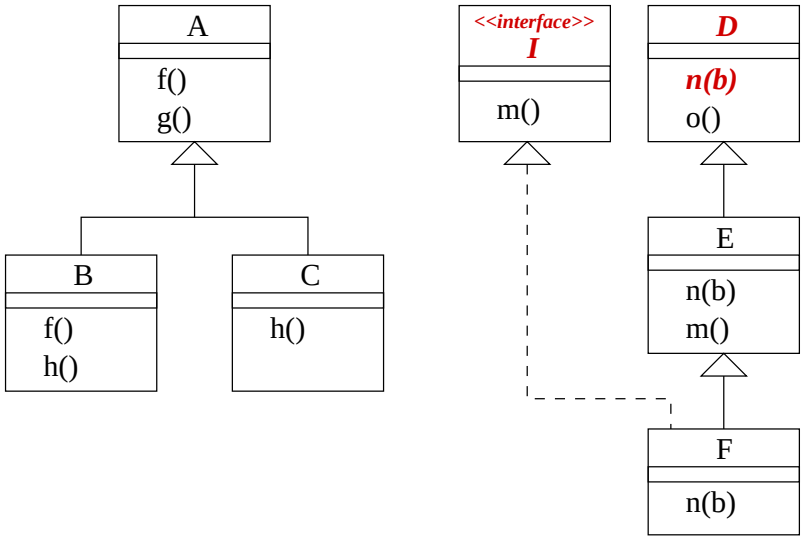
```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
String userinput;
int a, b;

System.out.println("1. Zahl eingeben: ");
userinput = in.readLine();                  // IOException
a = Integer.parseInt(userinput);            // NumberFormatException

System.out.println("2. Zahl eingeben: ");
userinput = in.readLine();                  // IOException
b = Integer.parseInt(userinput);            // NumberFormatException

int result = a / b;                         // ArithmeticException (/ by zero)
System.out.println("Das Ergebnis ist " + result);
```

c) Erweitern Sie das Programmstück um eine sinnvolle Fehlerbehandlung.

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
try {
    String userinput;
    int a, b;
    while (true) {
        try {
            System.out.println("1. Zahl eingeben: ");
            userinput = in.readLine();
            a = Integer.parseInt(userinput);
            break;
        } catch (NumberFormatException e) {
            System.out.println("Ungültige Eingabe für die 1. Zahl.");
        }
    }
    while (true) {
        try {
            System.out.println("2. Zahl eingeben: ");
            userinput = in.readLine();
            b = Integer.parseInt(userinput);
            int result = a / b;
            System.out.println("Das Ergebnis ist " + result);
            break;
        } catch (NumberFormatException e) {
            System.out.println("Ungültige Eingabe für die 2. Zahl.");
        } catch (ArithmeticException e) {
            System.out.println("Bitte eine andere Zahl als 0 eingeben.");
        }
    }
} catch (IOException e) {
    System.out.println("Fehler beim Lesen der Eingabe.");
}
```

Aufgabe 2 Vererbung

a) Stellen Sie die Klassen-Hierarchie mithilfe eines UML-Diagramms dar!



b) Welche der folgenden Zuweisungen sind erlaubt und welche nicht?

(i) `A a = new A();` – erlaubt

(ii) `B b = new B(); A t = b;` – erlaubt

(iii) `B b = new A();` – nicht erlaubt: incompatible types

(iv) `D d = new D();` – nicht erlaubt: cannot be instantiated

(v) `D d = new F();` – erlaubt

(vi) `F f = new F(); I i = f;` – erlaubt

c) Welche Ausgaben produzieren die folgenden Anweisungen?

(i)
```
A a = new A(); a.g();
```
g in A
f in A

(ii)
```
B b = new B(); b.h();
```
h in B
f in B
g in A
f in B

(iii)
```
C c = new C(); c.g();
```
g in A
f in A

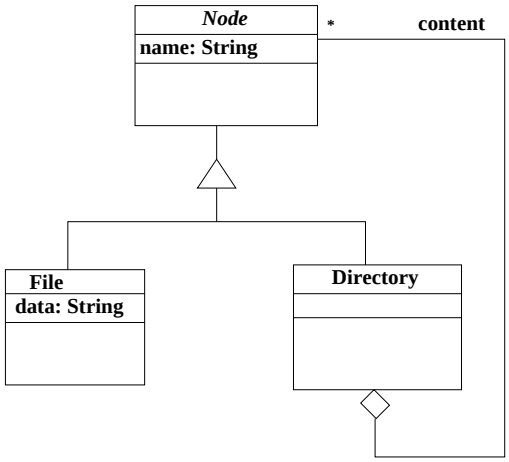
(iv)
```
B b = new B(); A a = b; a.g();
```
g in A
f in B

(v)
```
D d = new E(); d.n(true);
```
n in E
o in D
n in E

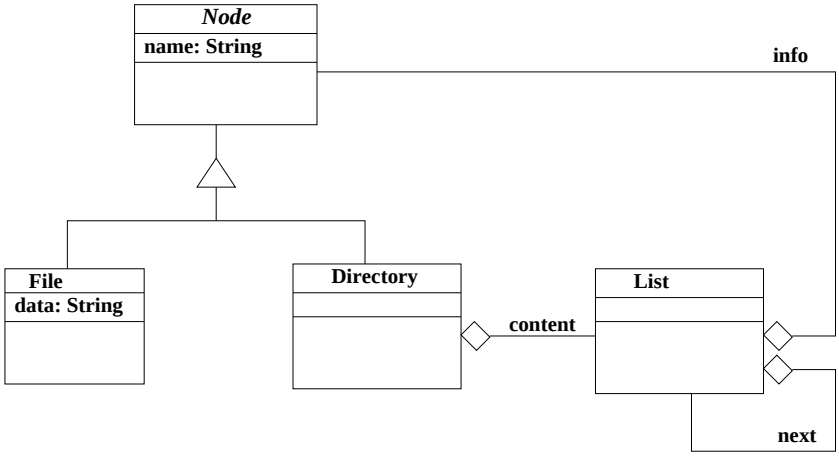
(vi)
```
F f = new F(); f.o();
```
o in D
n in F

Aufgabe 3 Datenstrukturen

a) UML Diagramm:



Oder ausführlicher:




```
abstract class Node{
    String name;
}

class File extends Node {
    String data;
}

class Directory extends Node{
    List<Node> content;
}
```

b) Die Klasse `Node`:

```
abstract class Node{
    String name;

    public Node(String n){
        name = n;
    }
```

```
    public String getName(){return name;}

    public abstract boolean isDirectory();
}
```

Die Klasse `File`:

```
class File extends Node {
    String data;

    public File(String name, String d){
        super(name);
        data = d;
    }

    public boolean isDirectory(){return false;}
}
```

Die Klasse `Directory`:

```
class Directory extends Node{
    List<Node> content;

    public Directory(String name){
        super(name);
    }

    public boolean isDirectory(){return true;}

    public Node find(String name){
        List<Node> currentNode = content;
        while (currentNode != null){
            if (currentNode.info.getName().compareTo(name) == 0)
                return currentNode.info;
            currentNode = currentNode.next;
        }
        return null;
    }

    public void makeDir(String name){
        if (find(name) != null){
            System.out.println("Directory already exists.");
            return;
        }
        content.insert(new Directory(name));
    }

    public void makeFile(String name, String inhalt){
        if (find(name) != null){
            System.out.println("File already exists.");
            return;
        }
        content.insert(new File(name,inhalt));
```

```
    }

    public Directory changeDir(List<String> path){
        if (path==null) return this;
        Node n = find(path.info);
        if (n == null){
            System.out.println("Directory not found.");
            return null;
        }
        if (n.isDirectory()) return ((Directory)n).changeDir(path.next);
        System.out.println("Not a directory.");
        return null;
    }
}
```

Aufgabe 4 **Ampelsteuerung**

a)
```
package Ampeln;
public class Ampel extends Thread {
    private boolean fussgangerGruen = false;

    public void run() {
        try {
            while (!isInterrupted()) {
                schalteAmpel();
                sleep(60000);
            }
        } catch (InterruptedException e) {
        }
    }

    public synchronized boolean getFussgaengerAmpelIstGruen() {
        return fussgangerGruen;
    }

    public synchronized boolean getAutoAmpelIstGruen() {
        return !fussgangerGruen;
    }

    private synchronized void schalteAmpel() {
        fussgangerGruen = !fussgangerGruen;
        System.out.println(fussgangerGruen ? "Auto: rot - Fussgänger: grün" : "Auto
        notifyAll();
    }

    public static void main(String[] args) {
        Ampel ampel = new Ampel();
        ampel.start();
        Auto a1 = new Auto("VW", ampel);
        a1.start();
        Fussgaenger f1 = new Fussgaenger("Martin", ampel);
        try {
            sleep(3000);
        } catch (InterruptedException e) {
        }
        f1.start();
        Fussgaenger f2 = new Fussgaenger("Thomas", ampel);
        try {
            sleep(3000);
        } catch (InterruptedException e) {
        }
        f2.start();
        Auto a2 = new Auto("Audi", ampel);
        a2.start();
        Fussgaenger f3 = new Fussgaenger("Frank", ampel);
```

```
        try {
            sleep(3000);
        } catch (InterruptedException e) {
        }
        f3.start();
        Auto a3 = new Auto("Opel", ampel);
        try {
            sleep(3000);
        } catch (InterruptedException e) {
        }
        a3.start();
        a1.join();a2.join();a3.join();f1.join();f2.join();f3.join();
        ampel.interrupt();
    }
}
```

b) Im folgenden ist eine Lösung angegeben, die in sinnvoller Weise Instanzen der beteiligten Objekte generiert und die Threads startet. Für eine korrekte Lösung der Klausur genügte die Instanziierung einer Ampel und zweier Fussgänger, sowie deren Start.

```
package Ampeln;
public class Fussgaenger extends Thread {
    String name;
    Ampel ampel;

    public Fussgaenger(String name, Ampel ampel) {
        this.name = name;
        this.ampel = ampel;
    }

    public void run() {
        System.out.println(name + " wartet");
        synchronized(ampel) {
           while (!ampel.getFussgaengerAmpelIstGruen()){
               try {
                   ampel.wait();
               } catch (InterruptedException e) {
               }
           }
           System.out.println(name + " geht");
}
    }
}
```