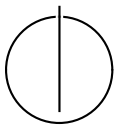
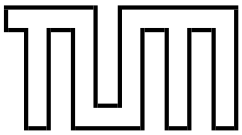
Name: | Vorname: | Matr.–Nr.:



TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK



Lehrstuhl für Sprachen und Beschreibungsstrukturen WS 2007/2008
Praktikum Grundlagen der Programmierung 16. Februar 2008
Prof. Dr. Helmut Seidl

Klausur zu Einführung in die Informatik I

Hinweis: In dieser Klausur können Sie insgesamt 70 Punkte erreichen. Zum Bestehen der gesamten Prüfung (Zwischen- und Endklausur) benötigen Sie mindestens 40 von insgesamt 100 Punkten.

Aufgabe 1 Applet (10+4=14 Punkte)

Schreiben Sie ein Applet, das 100 Knöpfe anzeigt. Im Ausgangszustand sollen alle Knöpfe die Beschriftung ’O’ haben. Die Beschriftung eines Knopfes können Sie mit der Methode `setLabel(String beschriftung)` der Klasse `Button` festlegen.

a) Bei einem Mausklick auf einen Knopf soll dieser die Beschriftung ’X’ erhalten.

b) Wird ein Knopf angeklickt, wenn bereits 5 andere Knöpfe mit einem ’X’ beschriftet sind, sollen alle Knöpfe wieder mit ’O’ beschriftet werden.

Anmerkung: Eine HTML-Seite, die das Applet aufruft, ist **NICHT** nötig.

Aufgabe 2 Vererbung **(4 + 5 + 6 = 15 Punkte)**

Wichtig: Lösen Sie diese Aufgabe direkt auf **diesem Blatt**!

Gegeben seien die im **Anhang** angegebenen Interface- und Klassen-Definitionen.

a) Erstellen Sie das UML–Klassendiagramm gemäss den Konventionen aus der Vorlesung.

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

	Erlaubt!	**Nicht erlaubt!**
(i) `class G extends A,E{ ... }`	☐	☐
(ii) `class G implements A,B{ ... }`	☐	☐
(iii) `A a = new A();`	☐	☐
(iv) `E<A> e = new E<C>();`	☐	☐
(v) `E<B> e = new E<B>();`	☐	☐
(vi) `A c1 = new C(); C c2 = c1.a1()`	☐	☐
(vii) `B b = new C();`	☐	☐
(viii) `A a = new F(); a.a1(true);`	☐	☐
(ix) `Y y = new X(1);`	☐	☐
(x) `class G<T>{ static void g(T t){ ... } }`	☐	☐

c) Welche Ausgaben produzieren die folgenden Anweisungen?

Hinweis: In manchen Fällen kommt es zu einer Exception.

(i)
```
X x = new X(1);
```
.....

.....

.....

.....

(ii)
```
A c = new C(); c = c.a1();
```
.....

.....

.....

.....

(iii)
```
F f = new D(); try { f.a1(true); }
catch (Y y) { System.out.println("Y"); }
catch (X x) { System.out.println("X"); }
```
.....

.....

.....

.....

(iv)
```
F f = new F(); try { f.a1(true); } catch (X x) { f.a2().a1(); }
```
.....

.....

.....

.....

(v)
```
B c = new C(); E<D> e = new E<D>(new D()); e.e1(c);
```
.....

.....

.....

.....

(vi)
```
A c = new C(); c.a3();
```
.....

.....

.....

.....

Aufgabe 3 Modellierung **(3+6+2+2=13 Punkte)**

In dieser Aufgabe soll ein Klassenmodell für Mietwohnungen implementiert werden, das es erlaubt den Mietpreis einer Mietwohnung zu bestimmen. Eine Mietwohnung besteht aus mehreren Räumen, die eine Länge und Breite in Metern haben. Die Grundmiete beträgt 10 EUR pro m^2. Wir unterscheiden zwei Arten von Räumen:

- **Wohnräume:** können möbliert sein. Die Möblierung hat einen Wert. Der Mietpreis erhöht sich durch eine Möblierung um 1 % des Möblierungswerts.
- **Funktionsräume:** können einen Wasseranschluss haben. Ist dies der Fall, verdoppelt sich die Grundmiete für den Raum.

a) Definieren Sie die Klasse `Raum`. Die Werte für Länge und Breite des Raumes sollen im Konstruktor gesetzt werden können. Der Zugriff auf die Attribute soll zusätzlich über entsprechende `get`- und `set`-Methoden ermöglicht werden. Weiterhin soll die Klasse eine Methode **`public int`** `berechneMietpreis()` anbieten, die die Grundmiete des Raumes berechnet.

b) Erweitern Sie Ihr Klassenmodell entsprechend der Beschreibung aus der Aufgabenstellung um die beiden Klassen `Wohnraum` und `Funktionsraum`. Nutzen Sie in Ihrer Implementierung möglichst geschickt die Konzepte von Objektorientierung, Vererbung und Polymorphie aus.

c) Definieren Sie die Klasse `Mietwohnung`. Die Klasse soll eine Methode **`public void`** `fuegeRaumHinzu(Raum r)` anbieten, die einen Raum `r` zur Mietwohnung hinzufügt. Unmittelbar nach der Erzeugung eines `Mietwohnung`-Objekts soll die Mietwohnung leer sein. **Hinweis:** Verwenden Sie in Ihrer Implementierung die unten angegebene generische Klasse `List`.

d) Erweitern Sie die Klasse `Mietwohnung` um eine Methode **`public int`** `berechneMietpreis()`. Als Ergebnis soll die Methode die Summe der Mietpreise der einzelnen Räume der Mietwohnung zurückliefern.

öffentlicher Konstruktor der Klasse `List`	
`List<T>()`	Erzeugt eine neue Liste für Objekte vom Typ `T` der Länge 0.
öffentliche Methoden der Klasse `List`	
`void` `add(T t)`	Fügt das Objekt `t` vom Typ `T` an das Ende der Liste an.
`T get(`**`int`**` i)`	Liefert das Objekt an der Stelle `i` in der Liste zurück.
`int` `size()`	Liefert die Anzahl der Elemente in der Liste zurück.

Aufgabe 4 IO und Exceptions **(7+3+4=14 Punkte)**

In dieser Aufgabe soll die Klasse `HtmlWriter` implementiert werden, die den Inhalt eines Telefonbuches in HTML-Dateien schreibt. Das Telefonbuch ist als ein `PersonReader` gegeben (ein Stream). Dabei werden Personen repräsentiert durch Objekte der Klasse `Person`:

```
public class Person {
    public String name;
    public String tel;
}
```

Die Klasse `PersonReader` stellt lediglich die Methode

```
Person readPerson() throws EndOfFileException
```

zur Verfügung, die die nächste Person im Telefonbuch (ein Stream) zurückliefert. Existiert keine weitere Person, so wird eine `EndOfFileException` geworfen.

a) Definieren Sie die Klasse `HtmlWriter`, die Personen aus einem `PersonReader` ausliest und in einer HTML-Datei abspeichert. Dazu soll der `PersonReader` einem `HtmlWriter`-Objekt über den Konstruktor übergeben werden. Weiterhin soll die Klasse `HtmlWriter` eine Methode

   ```
   void output(String file)
   ```

   zur Verfügung stellen, die bis zu 100 Personen aus dem `PersonReader` im HTML-Format in eine Datei mit dem Namen `file` schreibt. Ein Beispiel der zu generierenden HTML-Datei sehen Sie in der nachfolgenden Abbildung. Falls der `PersonReader` bei Aufruf der Methode `output()` keine Person enthält, so soll keine Ausgabe-Datei erzeugt werden. Stattdessen soll eine `KeineAusgabeException` geworfen werden (siehe Teil b)).

b) Implementieren Sie eine `KeineAusgabeException`. Diese Exception soll als Attribute den `PersonReader` und den Namen der nicht geschriebenen Datei enthalten und Methoden bereitstellen, um diese Attribute auszulesen. Gesetzt werden sollen die Attribute über einen geeigneten Konstruktor.

c) Schreiben Sie eine **statische** Methode

   ```
   void outputAll(PersonReader r, String file),
   ```

   die sämtliche Personen aus dem `PersonReader` in eine Folge von nicht-leeren HTML-Dateien (Benennung: `file0.html`, `file1.html`, `file2.html`, ...) schreibt. Verwenden Sie dazu die Methode `output()` aus Teil a).

```
<html>
<body>
<table>
<tr><td>Name</td><td>Telefonnummer</td></tr>
<tr><td>Hans Maier</td><td>89237865</td></tr>
<tr><td>Alfred Huber</td><td>171452119</td></tr>
<tr><td>Josef Schmidt</td><td>463756</td></tr>
</table>
</body>
</html>
```

Aufgabe 5 Java-Threads **(14 Punkte)**

In dieser Aufgabe soll die Situation in einem Geschäft mit mehreren Kassen aber **genau einer Warteschlange** simuliert werden. Es gibt **beliebig viele** Kassierer und Kunden, die als Java-Threads modelliert werden. Die entsprechenden Klassen `Kunde` und `Kassierer` sind wie folgt vorgegeben:

```
class Kunde extends Thread {
    Geschaeft g;
    String name;

    public Kunde(Geschaeft g, String name) {
        this.g = g;
        this.name = name;
    }

    public void run() {
        g.anstellen(this);
    }
}
```

```
class Kassierer extends Thread {
    Geschaeft g;
    String name;

    public Kassierer(Geschaeft g, String name) {
        this.g = g;
        this.name = name;
    }

    public void run() {
        while (true)
            g.kassieren();
    }
}
```

Ihre Aufgabe ist es die nachfolgend beschriebene Klasse `Geschaeft` zu implementieren.

Klasse `Geschaeft`	
öffentliche Methoden der Klasse `Geschaeft`	
`void` `kassieren()`	Entfernt den ersten Kunden aus der Warteschlange und bedient ihn. Die Bedienung soll mindestens 10 Sekunden in Anspruch nehmen. **Wichtig:** Ist kein Kunde in der Warteschlange, so muss der Kassierer warten bis sich ein Kunde anstellt. Dies darf **nicht** dadurch geschehen, dass andauernd überprüft wird, ob die Schlange leer ist (**busy-waiting**).
`void` `anstellen(Kunde k)`	Reiht den Kunden `k` in die Warteschlange ein. Danach ist seitens des Kunden keine Aktivität mehr erforderlich.

Wichtig: Verwenden Sie bei Ihrer Implementierung die angegebene Klasse `Queue`. Sie müssen allerdings darauf achten, dass **nicht** mehrere Threads **gleichzeitig** auf ein `Queue`-Objekt zugreifen.

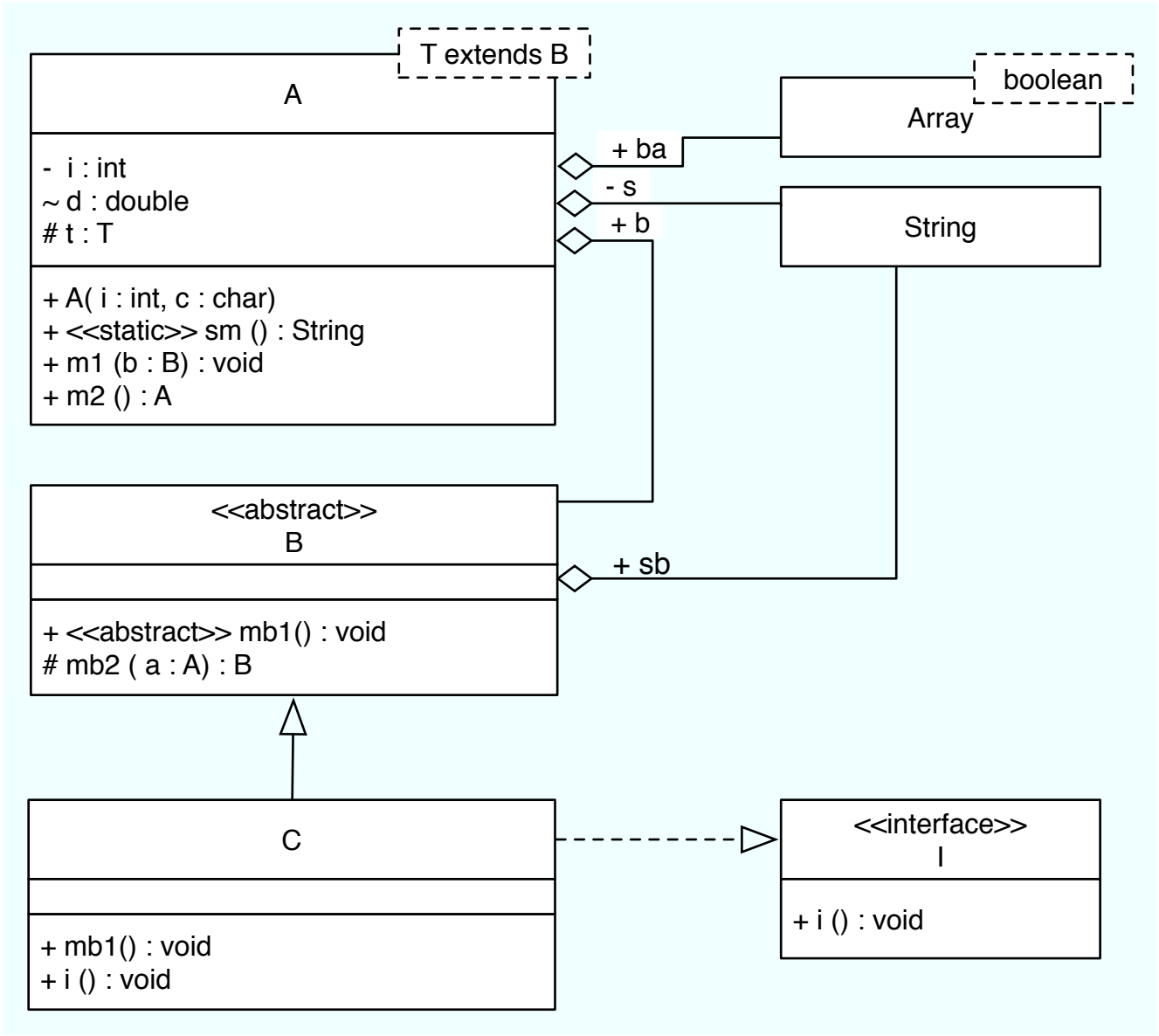
öffentlicher Konstruktor der Klasse `Queue`	
`Queue()`	Erstellt eine leere Schlange.
öffentliche Methoden der Klasse `Queue`	
`boolean` `isEmpty()`	Liefert genau dann **`true`** zurück, falls die Schlange leer ist.
`void` `enqueue(Kunde k)`	Fügt einen Kunden hinten in die Schlange ein.
`Kunde dequeue()`	Entfernt den ersten Kunden aus der Schlange und liefert ihn zurück.

Anhang zur Aufgabe 2 (Vererbung)

```
abstract class A{
    public abstract A a1();
    public A a2(){
        System.out.println("a2()␣in␣A");
        return a1();
    }
    public void a3(){
        System.out.println("a3()␣in␣A");
        s();
    }
    public void s(){
        System.out.println("s()␣in␣A");
    }
}
interface B{
    public void b1(B b);
}
class C extends A implements B{
    public C a1(){
        System.out.println("a1()␣in␣C");
        return new C();
    }
    public void b1(B b){
        System.out.println("b1(B)␣in␣C");
        a2();
    }
    public void s(){
        System.out.println("s()␣in␣C");
    }
}
class F extends A {
    public F(){
        System.out.println("F␣wurde␣erzeugt");
    }
    public F a1(){
        System.out.println("a1()␣in␣F");
        return null;
    }
    public void a1(boolean b) throws X{
        throw new X(4);
    }
}
class D extends F implements B{
   public void a1(boolean b) throws Y{
       throw new Y(4);
   }
    public void b1(B b){
        System.out.println("b1(B)␣in␣D");
    }
}
class E<T extends B>{
    public E(T t){
        System.out.println("E␣wurde␣erzeugt␣");
    }
    public void e1(B b){
        b.b1(b);  System.out.println("e1(B)␣in␣E");
    }
    public void e1(C c){
        c.b1(c);  System.out.println("e1(C)␣in␣E");
    }
}
class X extends Exception{
    public X(int x){
        super("Exception␣X␣an␣stelle␣"+x);
    }
}
class Y extends X{
    public Y(int y){
        super(y);
        System.out.println("Exception␣Y");
    }
}
```

UML Referenz

```
public class A<T extends B>{
    private int i;
    double d;
    protected T t;
    public B b;
    private String s;
    public boolean[] ba;

    public A(int i, char c) {
        ...
    }

    public void m1(B b) {
        ...
    }

    public A m2() {
        ...
    }

    public static String sm() {
        ...
    }
}

public abstract class B {
    public String sb;
    public abstract void mb1();
    protected B mb2(A a) {
        ...
    }
}

public interface I {
    public void i();
}

public class C extends B implements I {
    public void mb1() {
        ...
    }
    public void i() {
        ...
    }
}
```


T extends B
A
- i : int
~ d : double
# t : T
+ A(i : int, c : char)
+ <<static>> sm () : String
+ m1 (b : B) : void
+ m2 () : A
boolean
Array
+ ba
- s
+ b
String
<<abstract>>
B
+ sb
+ <<abstract>> mb1() : void
# mb2 (a : A) : B
C
+ mb1() : void
+ i () : void
<<interface>>
I
+ i () : void