TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Praktikum Grundlagen der Programmierung
F. Forster, T. Gawlitza, A. Flexeder

WS 2007/2008
Lösungsvorschläge zu Blatt 9

Übungen zu Praktikum Grundlagen der Programmierung

Aufgabe 44 (Ü) Generizität

```
public class GenericEintrag<S>{
    private S info;
    public GenericEintrag<S> next = null;
    public GenericEintrag<S> prev;

    public S getInfo(){
    return info;
    }
    public GenericEintrag<S> getNext(){
    return next;
    }
    public GenericEintrag<S> getPrev(){
    return prev;
    }
    public GenericEintrag(S elem){
    info = elem;
    }
    public GenericEintrag(S elem, GenericEintrag<S> n, GenericEintrag<S> p){
    info = elem;
    next = n;
    prev = p;
    }
}

public class GenericVerketteteListe<T>{

    //interne Listenstruktur der Verketteten Liste
    private GenericEintrag<T> list = null;

    GenericVerketteteListe(){
    list = new GenericEintrag<T>(null, null, null);
    list.prev = list.next = list;
    }

    GenericEintrag<T> next(GenericEintrag<T> e) {
    if (e.next == list)
        return null;
    return e.next;
    }
    GenericEintrag<T> prev(GenericEintrag<T> e){
    if (e.prev == list)
        return null;
    return e.prev;
    }
    T getInfo(){
    return list.getInfo();
    }

    private void addBefore(T i, GenericEintrag<T> l) {
    if (i!=null){
```

```
        GenericEintrag<T> newEintrag = new GenericEintrag<T>(i, l, l.prev);
        newEintrag.prev.next = newEintrag;
        newEintrag.next.prev = newEintrag;
    }
    }

    void add(T o) {
    addBefore(o, list);
    }

    void remove(GenericEintrag<T> e) {
    if (e != list){
        e.prev.next = e.next;
        e.next.prev = e.prev;
    }
    }

    public  String toString() {
    StringBuffer buf = new StringBuffer("[");
    for (GenericEintrag<T> e = next(list); e != null; e = next(e))
        {
        buf.append(e.getInfo() + "");
        if (next(e) != null)
            buf.append(",");
        }
    buf.append("]");
    return buf.toString();
    }

    /*
     * iterator ueber die einzelnen Listen-Elemente
     */
    Iterierer<T> iterator(){
    return new MyIterierer<T>(list);
    }
}

public interface Iterierer<T> {
    public T next();
    public boolean hasNext();
}

//Iterator ueber die Listenelemente
public class MyIterierer<S> implements Iterierer<S>{
    private GenericEintrag<S> elems; //zu iterierende Liste

    public MyIterierer(GenericEintrag<S> list){
    elems = list.next; //Beginn nach Sentinel Element
    }

    /*
     * pruefe ob naechstes Element vorhanden
     * Abbruch bei sentinel Element
     */
    public  boolean hasNext(){
    return elems.getInfo() != null;
    }

    /*
     * hole naechstes Element der Liste
     * @return null -> falls keine Elemente mehr vorhanden
     */
    public S next(){
    if(elems != null){
        S i = elems.getInfo();
        elems = elems.getNext();
        return i;
    } else {
        return null;
    }
    }
}

public interface Vergleichbar<T>{
    public int compareTo(T t);
}
```

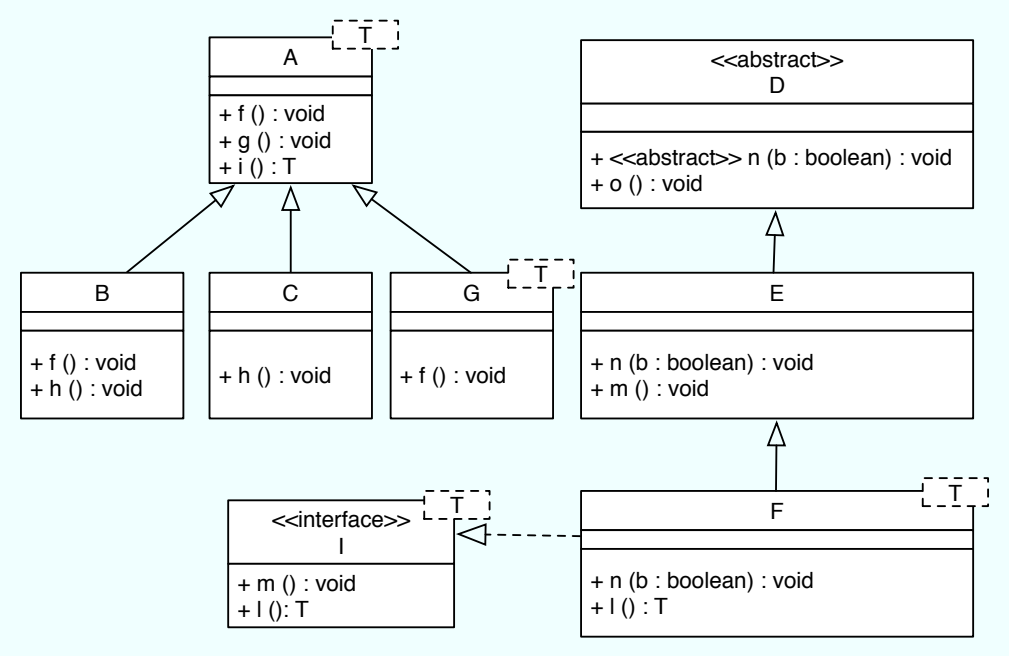
```
public class SortierPruefer{
    /*
     * prueft ob Liste in aufsteigender Reihenfolge sortiert ist
     */
    static <T extends Vergleichbar<T>>  boolean isSorted(GenericVerketteteListe<T> l){
    Iterierer<T> it = l.iterator();
    if(!it.hasNext()) return true;
    T elem = it.next();

    while (it.hasNext()) {
        T n = it.next();
        if (elem.compareTo(n) > 0){ return false;}
        elem = n;
    }
    return true;
    }
}

public class Ganzzahl implements Vergleichbar<Ganzzahl>{
    Integer i;
    public Ganzzahl(Integer i){this.i=i; }
    public int compareTo(Ganzzahl b){
    return i.compareTo(b.i);
    }
    public String toString(){return i.toString();}
}

public class Zeichenkette implements Vergleichbar<Zeichenkette>{
    String s;
    public Zeichenkette(String s){this.s=s; }
    public int compareTo(Zeichenkette b){
    return s.compareTo(b.s);
    }
    public String toString(){return s;}
}

public class ListenTest{

    public static void main(String[] args){
    ////////////////liste von Integer - Objekten//////////////////////////
    GenericVerketteteListe<Ganzzahl> l = new GenericVerketteteListe<Ganzzahl>();
    l.add(new Ganzzahl(5));
    l.add(new Ganzzahl(new Integer(87)));
    l.add(new Ganzzahl(13));
    System.out.println("liste:␣"+l.toString());
    Ganzzahl i = l.iterator().next();
    Iterierer<Ganzzahl> iter = l.iterator();
    Ganzzahl wert;
    while(iter.hasNext()){
    wert = iter.next();
    System.out.println("listeninhalt:␣"+wert);
    }
    System.out.println("normale␣iteration␣beendet␣");
    if(SortierPruefer.isSorted(l)) System.out.println("diese␣Liste␣ist␣in␣aufsteigender␣Reihenfolge␣sortiert!")
    else System.out.println("diese␣Liste␣ist␣NICHT␣in␣aufsteigender␣Reihenfolge␣sortiert!"+l);
    ////////////////////liste von Zeichenkettes ////////////////////////////
    GenericVerketteteListe<Zeichenkette> s = new GenericVerketteteListe<Zeichenkette>();
    s.add(new Zeichenkette("hello"));
    s.add(new Zeichenkette("world"));
    System.out.println("liste:␣"+s.toString());
    Zeichenkette str = s.iterator().next();
    Iterierer<Zeichenkette> iters = s.iterator();
    Zeichenkette val;
    while(iters.hasNext()){
    val = iters.next();
    System.out.println("listeninhalt:␣"+val);
    }

    if(SortierPruefer.isSorted(s)) System.out.println("diese␣Liste␣ist␣in␣aufsteigender␣Reihenfolge␣sortiert!")
    else System.out.println("diese␣Liste␣ist␣NICHT␣in␣aufsteigender␣Reihenfolge␣sortiert!"+s);

    }
}
```

Aufgabe 45 (Ü) Vererbung und Interfaces

Hinweise zu Generics:

- Konzept der raw types: um Kompatibilität zu altem Code zu gewährleisten, wurde das Prinzip der raw types eingeführt. Dies sind schlicht unparametrisierte Typen (siehe Beispiel i) `A a = new A()`). Der Compiler hat an dieser Stelle keine Information über den Typ und kann somit nicht verifizieren, dass der raw typ tatsächlich eine Instanz des angegebenen konkreten Typs ist. (Somit schlägt viii fehl, bzw. hier wird meist eine Warnung erzeugt) Durch einen expliziten Cast kann das Problem behoben werden (`A<B> a = (A<B>) new A()`).
- Es besteht ein Unterschied zwischen raw type, `A<Object>` und `A<? extends Object>`. Im Falle `A<Object>` sind nur Elemente vom Typ `Object` erlaubt, wohingegen bei letzterem Fall auch Unterklassen von `Object` verwendet werden können.
- Parametertypinformation ist zur Laufzeit nicht bekannt (automatische generierte Casts sind somit unmöglich) (siehe Beipsiel xiii). Überladen von Methoden funktioniert hier nicht: `meth(F<G> f)`, `meth(F<C> f)` – in diesem Falle steht nur die Information `meth(F f)` zur Verfügung.
- Im Falle einer statischen Methode, die einen generischen Parameter `VerketteteListe<T>` verwendet, siehe Beispiel: `static <T extends Vergleichbar<T>> boolean isSorted(VerketteteListe<T> l)`, ist es nötig, diesen Typparameter `T` zu „deklarieren “, was durch das folgende Konstrukt geschieht: `<T extends Vergleichbar<T>>`.

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) cannot convert from A to B

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

(v) `D d = new F();` – erlaubt

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

(vii) `A a = new A<B>();` – erlaubt

(viii) `A<B> a = new A();` – erlaubt (Warnung: unchecked conversion)

(ix) `A a = new G();` – erlaubt

(x) `A<D> a = new G<E>();` – nicht erlaubt: (incompatible types) cannot convert from G<E> to A<D>

(xi) `A<? extends E> a = new G<F<A<B»>();` – erlaubt

(xii) `E e = a.i();` – erlaubt

(xiii) `F f = a.i();` – nicht erlaubt: (incompatible types) cannot convert from E to F

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