Interfaces (Schnittstellen)

Ziel von Interfaces

- Interfaces trennen den Entwurf von der Implementierung
- Interfaces legen Funktionalität fest, ohne auf die Implementierung einzugehen
- Beim Implementieren der Klasse ist die spätere Verwendung nicht von Bedeutung, sondern nur die bereitzustellende Funktionalität
- Ein Anwender (eine andere Klasse) interessiert sich nicht für die Implementierungsdetails, sondern für die Funktionalität

Vorteile von Interfaces für die Zusammenarbeit

Ein Interface zu entwerfen geht schneller, als eine Klasse zu implementieren.

Ist das Interface einer Klasse festgelegt, inklusive ausreichender Dokumentation, so kann man

- Diese Klasse implementieren, ohne wissen zu müssen, wo genau und wie genau sie verwendet wird
- Diese Klasse in weitergehender Implementierung verwenden, auch wenn sie noch nicht ausgeführt werden kann

Damit kann man besser gleichzeitig an verschiedenen Teilen des Projekts arbeiten

Vorteile von Interfaces für die Implementierung

- Ist einmal ein Interface vorhanden, so hat man ein klares und kleineres Ziel
- Es lassen sich bessere Tests schreiben, da die Funktionalität genau festgelegt ist
- Während der Implementierung braucht man nicht darüber nachzudenken, wo es dann verwendet wird

Vorteile von Interfaces für die Verwendung & Wartbarkeit

- Man kann die Klasse verwenden, ohne die Interna der Klasse zu kennen
- Ändert sich die Implementierung einer Klasse bei gleichbleibendem Interface, so muss der Code nicht mitgeändert werden

Verwendungsbeispiel aus der Standardbibliothek

`LinkedList<E>` ist eine Implementierung des Interfaces `List<E>`
Verwendet man

```
List<E> foo = new LinkedList<E>();
```

anstelle von

```
LinkedList<E> foo = new LinkedList<E>();
```

so kann man jederzeit `LinkedList` durch eine andere Implementierung des Interfaces `List` ersetzen

Deklarationsbeispiel in der Standardbibliothek

`java/lang/Comparable.java` (Kommentare gekürzt):

```
/* Copyright 1997-2006 Sun Microsystems, Inc.
   LICENSE: GPL2 */
package java.lang;
import java.util.*;

/**
 * Compares this object with the specified object
 * for order. Returns a negative integer, zero, or
 * a positive integer as this object is less than,
 * equal to, or greater than the specified object.
 */
public interface Comparable<T> {
  public int compareTo(T o);
}
```

Implementationsbeispiel von Comparable

```
public class Person implements Comparable<Person> {
  private double size;
  private String name;

  public Person(double size, String name) {
    this.size = size;
    this.name = name;
  }

  public int compareTo(Person o) {
    if (size < o.size)       return  1;
    else if (size == o.size) return  0;
    else                     return -1;
  }
}
```

Dummy-Implementierung

Zum Testen kann man auch eine Dummy-Implementierung erstellen, welche die formalen Anforderungen erfüllt, aber nicht das macht, was man meint.

```
/**
 * Dummy-Implementierung des Interfaces Comparable
 * TODO: Instanzvariablen einbauen, wirklich vergleichen
 */

public class TestComparable
  implements Comparable<TestComparable> {

  public TestComparable(double size, String name) { }

  public int compareTo(TestComparable o) {
    return 0;
  }
}
```

Mehrere Interfaces, Beispiel zweites Interface

Eine Klasse kann mehrere Interfaces implementieren. Dazu deklarieren wir hier ein eigenes zweites Beispielinterface

```
public interface Growable {
  public void growBy(double x);
}
```

Mehrere Interfaces, Beispiel Implementierung

und erweitern die Klasse Person von vorhin

```
public class Person
  implements Comparable<Person>, Growable {
...Instanzvariablen und Konstruktor von vorhin...

public int compareTo(Person o) {
  if(size < o.size)          return  1;
    else if(size == o.size) return  0;
    else                       return -1;
  }

public void growBy(double x) {
  size = size + x;
  }
}
```

Mehrere Interfaces, Beispiel Verwendung

```
public class Person
  implements Comparable<Person>, Growable {
...Code der Folie vorher...

public static void main(String[] args) {
  Person hans = new Person(1.89, "Hans");
  Comparable<Person> hanscompare = hans;
  Growable hansgrow = hans;
  }
}
```

Ein Interface ist also gewissermaßen ein Aspekt einer Klasse. Wenn man als Variablentyp dieses Interface angibt, kann man auch sicherstellen, dass man nur diesen Aspekt verwendet.

Sichtbarkeit: `public`, `protected`, `private`, `package`

Sichtbarkeit bedeutet, von wo auf etwas zugegriffen werden kann. Sie kann durch die Schlüsselwörter `public`, `protected`, `private` gesteuert werden

- `public`: Man kann von überall darauf zugreifen
- `private`: Man kann nur innerhalb der Klasse, in der die Deklaration steht, darauf zugreifen
- kein Attribut = package-private: Man kann nur aus dem gleichen Paket darauf zugreifen
- `protected`: Wie `private`, aber Klassen, die davon erben, können auch darauf zugreifen

Generell sollte man so viel wie möglich private machen, insbesondere alle Instanzvariablen.
Eine Anforderung des Hauptprojektes wird sein, zu allem, was public ist, eine ordentliche Javadoc-Dokumentation zu schreiben.