



Enums, Interfaces und Generics

Enum Typen – Definition

- komplexer Datentyp, der festen Menge von Konstanten enthält
- Information der Reihenfolge der Konstanten
- erbt implizit von `java.lang.Enum`

```
enum Tag {  
  
    MONTAG, DIENSTAG, MITTWOCH, DONNERSTAG,  
    FREITAG, SAMSTAG, SONNTAG  
  
}
```

Standardmethoden

- Compiler fügt automatisch Methoden hinzu

```
int compareTo(Enum e)
```

```
int ordinal()
```

```
static Enum valueOf(String name)
```

Eigene Funktionen

```
public enum Planet {  
  
    EARTH    (5.976e+24, 6.37814e6),  
    MARS     (6.421e+23, 3.3972e6);  
  
    private final double masse;    // in kilograms  
    private final double radius;  // in meters  
  
    Planet(double masse, double radius) {  
        this.masse = masse;  
        this.radius = radius;  
    }  
    double gravitationAufOberflaeche() {  
        return G * masse / (radius * radius);  
    }  
    double gewichtAufOberflaeche(double otherMass) {  
        return otherMass * surfaceGravity();  
    }  
}
```

Interfaces – Definition

- formale Eckpunkte, die erfüllt werden müssen
- definiert, wie zwei Objekte miteinander kommunizieren
- definiert was implementiert werden soll, nicht wie
- Referenztyp wie Klasse

Implementierung

```
public interface Vergleichbar extends Verwandt {  
  
    public final double umrechnungsfaktor = 2d;  
  
    public boolean istGroesser(Vergleichbar v);  
  
}
```

Implementierung

```
public class Rechteck implements Berechenbar, Vergleichbar {  
  
    ...  
  
    public boolean istGroesser(Vergleichbar v) {  
        ...  
    }  
}
```

Implementierung

```
public class Rechteck implements Berechenbar, Vergleichbar {  
  
    ...  
  
    public boolean istGroesser(Vergleichbar v) {  
        ...  
    }  
}
```

- **Vorsicht** Bei Änderungen in einem Interface müssen auch alle implementierenden Klassen geändert werden!

Generics - Definition

- Typecast verhindern
- Typsicherheit beim Kompilieren überprüfen
- Typbegrenzung für allgemeine Datenstrukturen

Generische Typen

```
public class Box {  
  
    private Object element;  
  
    public void add(Object e) {  
        element = e;  
    }  
    public Object get() {  
        return element;  
    }  
}
```

```
1 public class BoxDemo {  
2     public static void main(String[] args) {  
        ...  
100    Box integerBox = new Box();  
        ...  
250    integerBox.add("10");  
        ...  
550    Integer someInteger =  
            (Integer)integerBox.get();  
    }  
}
```

Generische Typen

```
public class Box {  
  
    private Object element;  
  
    public void add(Object e) {  
        element = e;  
    }  
    public Object get() {  
        return element;  
    }  
}
```

```
1 public class BoxDemo {  
2     public static void main(String[] args) {  
        ...  
100    Box integerBox = new Box();  
        ...  
250    integerBox.add("10");  
        ...  
550    Integer someInteger =  
            (Integer) integerBox.get();  
    }  
}
```

- `java.lang.ClassCastException`

Generische Typen

```
public class Box<T> {  
  
    private T element;  
  
    public void add(T e) {  
        element = e;  
    }  
    public T get() {  
        return element;  
    }  
}
```

```
1 public class BoxDemo {  
2     public static void main(String[] args) {  
        ...  
100    Box<Integer> integerBox = new Box<Integer>();  
        ...  
250    integerBox.add("10");  
        ...  
550    Integer someInteger = integerBox.get();  
        }  
}
```

Generische Typen

```
public class Box<T> {  
  
    private T element;  
  
    public void add(T e) {  
        element = e;  
    }  
    public T get() {  
        return element;  
    }  
}
```

```
1 public class BoxDemo {  
2 public static void main(String[] args){  
    ...  
100 Box<Integer> integerBox = new Box<Integer>();  
    ...  
250 integerBox.add("10");  
    ...  
550 Integer someInteger = integerBox.get();  
    }  
}
```

- Compile Error
- Kein Typecast mehr

Generische Methoden

```
public static <T> void fillBoxes(T arg, List<Box<T>> listOfBoxes){  
    for ( Box<T> box : boxlist)  
        box.add(arg);  
}
```

```
Box.<Integer>fillBoxes(new Integer(10), listOfBoxes);
```

```
Box.fillBoxes(new Integer(10), listOfBoxes);
```

Generische Methoden

```
public static <T> void copy(List<T> dest, List<? extends T> src)
```

```
public static <T, S extends T> void copy(List<T> dest, List<S> src)
```

Unterschiedliche Typparameter

- Mehrfache Typisierung
 - `<A, B, C, ... >`
- Wildcards
 - Beliebiger Typ `<? >`
 - Upper Bounds `<? extends X >`
 - Lower Bounds `<? super X >`
 - `<? extends X & Y >`

Beispiel Typenhierarchie

- **interface** Rechteck **extends** GeoObjekt
- `Liste<Rechteck> recList, Liste<GeoObjekt> objList`
- `objList = recList; // Compiler Error`
- `recList = objList; // Compiler Error`