Aufzälungstypen

Java Foundation Track by Carsten Bokeloh

Aufzählungstypen

- Deklaration eines Aufzählungstyps
- Instanzmethoden der enum-Objekte
- Aufzählungstypen erweitern

Aufzählungen

- enum ist ein neues Schlüsselwort in Java.
- enum ist eine spezielle Form einer Klasse.
- Jedes selbstgeschriebene enum ist eine Klasse, die der Compiler automatisch von der Klasse java.lang.Enum ableitet.
- Die Klasse Enum selbst implementieret Serializable und Comparable und leitet sich direkt von Object ab.
- Die enum-Klasse wird vom Compiler immer als final angelegt.
- Eine selbstgeschriebene enum-Klasse kann beliebige Interfaces implementieren.

Aufzählungen

- Ein eigenständiges Ableiten einer enum-Klasse ist in keiner Form möglich und wird vom Compiler unterbunden.
  - enum MyEnum extends Enum {} // Compilermeldung: '{' expected
  - class MyEnum extends Enum {} // Compilermeldung: classes cannot directly extend java.lang.Enum

Aufzählungen

Definition

```
public enum Color {
    GREEN, YELLOW, RED;
}
```

Benutzung

```
private static void schreibeFarbe(Color typ) {
    switch(typ) {
        case RED:
            System.out.println("Rot");
            break;
        case YELLOW:
            System.out.println("Gelb");
            break;
    }

}
```

Aufzählungen

- Aufzählungen mit enum
  - In der Schreibweise für Aufzählungen ersetzt das Schlüsselwort enum das Schlüsselwort class und ist auch ähnlich zu nutzen.
  - Eine enum-Deklaration erlaubt auch die Deklaration von Methoden und Variablen. Somit verhält sich wie eine bekannte Klassendeklaration , nur dass keine Unterklassen und Oberklassen möglich sind.

Aufzählungen

Aufzählungen für Jahreszeiten sind ein gutes Beispiel

```
public enum Season {
    SPRING, SUMMER, AUTUMN, WINTER;
}
```

Compiler output

```
public class Season extends Enum {
    public static final Season SPRING;
    public static final Season SUMMER;
    public static final Season AUTUMN;
    public static final Season WINTER;
    private static final Season VALUES[];

    static
    {
        SPRING = new Season("SPRING", 0);
        SUMMER = new Season("SUMMER", 1);
        AUTUMN = new Season("AUTUMN", 2);
        WINTER = new Season("WINTER", 3);
        VALUES = new Season[] { SPRING, SUMMER, AUTUMN, WINTER } ;
    }

    private Season(String s, int i)
    {
        super(s, i);
    }

    //weitere Methoden

}
```

Compiler output

- Enumkonstanten werden als statische Konstanten angelegt und im static Initializer ins Leben gerufen
- Elternklasse enum verfügt über den protected Konstruktion Enum(String name, int ordinal)
  - Ableitung muss über Konstruktor verfügen, der super aufruft -> Durch Compiler erzeugt.

Aufzählungen

- Jetzt ist es einfach, diese Werte zu nutzen, da sie ja wie jede andere statische Variable angesprochen werden.
  - Season frühling = Season.SPRING;

Aufzählungen

Nutzung beispielsweise innerhalb von switches:

```
Season season = Season.SPRING;
switch (season) {
case SPRING:
    System.out.println("Frühling");
case SUMMER:
    System.out.println("Sommer");
}
```

aber VORSICHT Verweis auf enum Objekt kann null sein!!!!

Standard Methoden der enums

- name() liefert für ein Enum-Objekt den Namen der Konstante(nicht überschreibbar)
- toString ruft name() auf(überschreibbar).
- valueOf(String) liefert ein Enum-Objekt, welches zur name() Repräsentation passt.
- Enum.valueOf(Class class, String s)
  - ermöglicht das Suchen von Enum Objekten zu einem Konstantennamen und einer enum-Klasse.

Alles aufzählen

```
public void aufzaehlen() {
    for(Season season : Season.values()) {
        System.out.println(season.name());
    }
}
```

```
SPRING
SUMMER
AUTUMN
WINTER
```

Ordinalzahl

- Ordinalzahl im Konstruktor als ID.
  - gibt die Position in der Deklaration an und ist auch Ordnungskriterium der compareTo Methode
- lässt sich nicht ändern und repräsentiert immer die Reihenfolge der deklarierten Konstanten

selbstgeschriebener Konstruktor

```
public enum Season {
    SPRING("mäßig warm"), SUMMER("warm"), AUTUMN("mäßig kalt"), WINTER("kalt");

    Season(String s) {
        System.out.println("Season(String s) creates " +this.name()+" und es ist " + s ) ;
    }
}
```

selbstgeschriebene Konstrukten

```
public class Season extends Enum{
    public static final Season SPRING;
    public static final Season SUMMER;
    public static final Season AUTUMN;
    public static final Season WINTER;
    private static final Season VALUES[];

    static
    {
        SPRING = new Season("SPRING", 0, "mäßig warm");
        SUMMER = new Season("SUMMER", 1, "warm");
        AUTUMN = new Season("AUTUMN", 2, "mäßig kalt");
        WINTER = new Season("WINTER", 3, "kalt");
        VALUES = new Season[] { SPRING, SUMMER, AUTUMN, WINTER } ;
    }

    private Season(String s, int i, String s )
    {
        super(s, i);
        //weiteres zur String ausgabe
    }

    //weitere Methoden

}
```

enum und Konstruktiven

- Wenn ein enum keinen Konstruktor enthält, so erzeugt der Compiler einen Konstruktor mit der Signatur "private MyEnum(String s, int i) ".
- Jeder vom Entwickler erstellter Konstruktor erhält vom Compiler den modifier "private". Die modifier "public" und "protected" sind nicht zuläßig.

enum & Konstruktoren

- Ein vom Entwickler erstellter Konstruktor mit der Signatur "MyEnum(Typ1 t1, Typ2 t2)" wird vom Compiler durch einen Konstruktor mit der Signatur "private MyEnum(String s, int i, Typ1 t1, Typ2 t2)" ersetzt. Insbesondere erhält ein selbstentworfener Defaultkonstruktor die Signatur "private MyEnum(String s, int i)“.
- Da alle Konstruktoren private sind können Enumobjekte nicht mit "new" erzeugt werden.

Zusammenfassung zu Enums

- enum ist eine spezielle Form einer Klasse
- Jedes selbstgeschriebene Enum ist eine Klasse, die der Compiler automatisch von der Klasse java.lang.Enum ableitet
- Die Klasse Enum selbst implementieret Serializable und Comparable und leitet sich direkt von Object ab.
- Die enum-Klasse wird vom Compiler immer als final angelegt.
- Ein eigenständiges Ableiten einer enum-Klasse ist in keiner Form möglich und wird vom Compiler unterbunden.
  - enum MyEnum extends Enum {} // Compilermeldung: '{' expected
  - class MyEnum extends Enum {} // Compilermeldung: classes cannot directly extend java.lang.Enum

Übung

- EnumÜbung
- EnumÜbung2