Programmieren I

Dr. Klaus Höppner

Hochschule Darmstadt – Wintersemester 2009/2010

Rückblick

In der letzten Vorlesung wurde in Referenzmodell für Objekte in Java eingeführt.

Das Referenzmodell erlaubt,

- obwohl *formal* beim Aufruf von Methoden in Java das Prinzip *Call by Value* gilt,
- dass der Zustand von „als Parameter übergebenen“ Objekten verändert werden kann,
- dass bei Objekten „als Rückgabewert von Methoden“ Zustandsänderungen nicht auf eine Kopie beschränkt bleiben,
- da hier tatsächlich *Kopien von Objektreferenzen* über- bzw. zurückgegeben werden.

Beispiel: Klasse Name

```
package person;

public class Name {
    private String vorname;
    private String nachname;
    public Name(String vorname, String nachname) {
        this.vorname = vorname;
        this.nachname = nachname;
    }
    public String getNachname() {
        return nachname;
    }
    public void setNachname(String nachname) {
        this.nachname = nachname;
    }
```


Beispiel: Klasse Name (Forts.)

```
    public String getVorname() {
        return vorname;
    }
    public void setVorname(String vorname) {
        this.vorname = vorname;
    }
    @Override
    public String toString() {
        return String.format("%s, %s", nachname, vorname);
    }
}
```


Beispiel: Klasse Person

```
package person;

public class Person {
    private Name name;
    private String geburtsort;
    public Person(Name name, String geburtsort) {
        this.name = name;
        this.geburtsort = geburtsort;
    }
    public String getGeburtsort() {
        return geburtsort;
    }
    public void setGeburtsort(String geburtsort) {
        this.geburtsort = geburtsort;
    }
```

Beispiel: Klasse Person (Forts.)

```
    public Name getName() {
        return name;
    }
    public void setName(Name name) {
        this.name = name;
    }
    @Override
    public String toString() {
        return String.format("%s (geb. in %s)",
                             name, geburtsort);
    }
}
```


Anwendung

Dass (Objekt-)Variablen in Java Referenzen sind, wird an dem folgenden Beispiel deutlich:

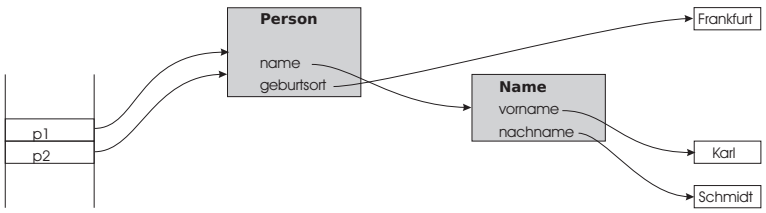
```
import person.*;

public class Anwendung {
    public static void main(String[] args) {
        Person p1 = new Person(new Name("Karl","Schmidt"),
                                                "Frankfurt");
        System.out.println("p1: "+p1);

        Person p2 = p1;
        p2.setGeburtsort("Darmstadt");
        p2.getName().setVorname("Erich");

        System.out.println("p1: "+p1);
        System.out.println("p2: "+p2);
    }
}
```


Schaubild



Klonen von Objekten in Java

Durch die Zuweisung `Person p2 = p1` wird also eine Kopie der *Objektreferenz* erstellt.

Soll hingegen ein *neues Objekt* als Kopie eines anderen erstellt werden, so nennt man das *Clone*.
Zum Klonen muss in Java

- eine Klasse das Interface `Cloneable` implementieren, und
- und die (protected) Methode `void clone()` aus `Object` als public überschrieben werden.

Beispiel: Klasse Name

```
public class Name implements Cloneable {
    private String vorname;
    private String nachname;

    // ...

    @Override
    public Object clone() {
        try {
            return super.clone();
        } catch (CloneNotSupportedException e) {
            return null;
        }
    }
}
```


Anwendung

Hiermit können nun Kopien von Instanzen der Klasse `Name` erzeugt werden:

```
Name n1 = new Name("Karl","Schmidt");
Name n2 = (Name) n1.clone();
n2.setNachname("Meier" );
```

In diesem Beispiel referenzieren `n1` und `n2` verschiedene Objekte, die sich am Ende in ihren Nachnamen unterscheiden.

Beispiel: Klasse Person

```
package person;

public class Person implements Cloneable {
    private Name name;
    private String geburtsort;

    // ...

    public Object clone() {
        try {
            return super.clone();
        } catch (CloneNotSupportedException e) {
            return null;
        }
    }
}
```


Anwendung

Nun betrachten wir folgende Anwendung:

```
import person.*;

public class Anwendung {
    public static void main(String[] args) {
        Person p1 = new Person(new Name("Karl","Schmidt"),
                "Frankfurt");

        System.out.println("p1: "+p1);

        Person p2 = (Person) p1.clone();
        p2.setGeburtsort("Darmstadt");
        p2.getName().setVorname("Erich");

        System.out.println("p1: "+p1);
        System.out.println("p2: "+p2);
    }
}
```

Entspricht die Ausgabe der Erwartung?

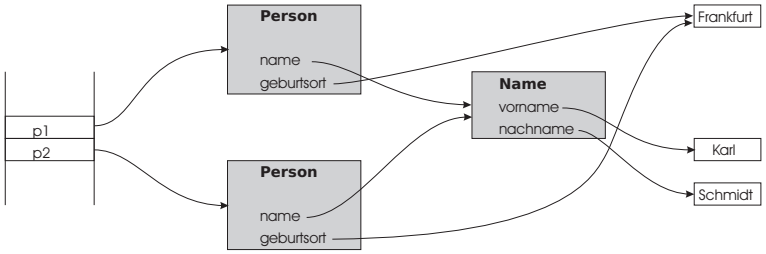
Analyse

Im vorigen Beispiel werden durch `p1.clone()` die Attribute einzeln kopiert (*field-to-field copy*), bei Attributen, die Objektreferenzen sind, findet also ein Kopie der Referenzen statt.

Es existieren nun also zwei Instanzen von `Person`, die denselben Namen und denselben Geburtsort referenzieren. Durch `n2.setGeburtsort(...)` wird ein neuer String als Geburtsort referenziert, hier unterscheiden sich beide Instanzen nun.
Durch `n2.getName().setVorname(...)` wird der Vorname des *gemeinsamen* Namens geändert!

Dies nennt man *Shallow Copy* (flache Kopie).

Schaubild: *Shallow Copy*



Shallow Copy – Deep Copy

Die aus `Object` geerbte Methode `clone` realisiert also eine Shallow Copy, eine flache Kopie.

Das Gegenstück zur Shallow Copy heißt *Deep Copy* (tiefe Kopie). Diese muss selbst implementiert werden.

Java gibt keine Richtlinie vor, ob die Methode `clone` eine Deep oder Shallow Copy implementieren soll. Diese Entscheidung muss der Programmierer selbst treffen.

Hierbei sind Deep Copies im Allgemeinen aufwändiger – sowohl vom Programmieraufwand als auch von der Laufzeit her.

Klasse `Person` mit Deep Copy

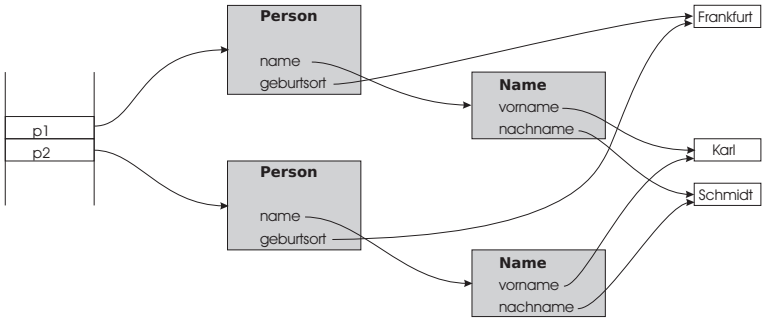
```
package person;

public class Person implements Cloneable {
    private Name name;
    private String geburtsort;

    // ...

    @Override
    public Object clone() {
        return new Person((Name) name.clone(),geburtsort);
    }
}
```

Schaubild: *Deep Copy*



Gleichheit/Identität

Betrachten wir folgenden Quelltext in `main`:

```
Name n1 = new Name("Karl","Schmidt");
Name n2 = n2.clone();

System.out.println(n1==n2 );
```

Dieses Beispiel gibt `false` aus, denn der Operator `==` testet Objekte auf Identität.

Zwei Objektreferenzen sind dabei genau dann identisch, wenn sie dasselbe Objekt referenzieren.

Gleichheit/Identität (Forts.)

In der Klasse `Object` gibt es die Methode
`public boolean equals(Object arg)`
die angibt, ob zwei Objekte gleich sind.

In `Object` ist `equals` identisch zu `==` definiert, aber für eigene Klassen kann man `equals` so überladen, dass Objekte zwar nicht identisch, aber trotzdem gleich sein können.

Im folgenden wird `equals` für die Klasse `Name` überladen, so dass zwei Instanzen von `Name` gleich sind, wenn die Vor- und Nachnamen gleich sind.
Da `equals` zwingend für allgemeine Objekte definiert ist, muss zunächst geprüft werden, ob das Vergleichsobjekt ein Name ist, ansonsten wird `equals` aus `Object` verwendet.

Beispiel: Klasse Name

```
package person;

public class Name implements Cloneable {
    // ...

    @Override
    public boolean equals(Object arg) {
        if (arg instanceof Name) {
            Name other = (Name) arg;
            return nachname==other.nachname &&
                vorname==other.vorname;
        } else {
            return super.equals(arg);
        }
    }
}
```


Fabrikmethoden

Ein häufig anzutreffendes Design in Java sind statische Fabrikmethoden (*factory methods*). Diese „verstecken“ den Konstruktor einer Klasse vor dem Anwender.
Fabrikmethoden sind sinnvoll

- wenn eine neue Instanz nur unter bestimmten Bedingungen verwendet werden soll, und ansonsten ein Nullpointer zurück gegeben wird;
- eine einmal angelegte Instanz immer wiederverwendet werden soll (Singleton).

In der fortgeschrittenen Programmierung werden Fabrikmethoden teilweise zu Fabrik-Klassen erweitert. Diese sind ein wichtiges Element moderner *Entwurfsmuster* (*design patterns*).

Beispiel: Klasse Ort

In diesem Beispiel besitzt `Ort` eine statische Fabrikmethode, die nur Instanzen mit gültiger PLZ anlegen kann:

```
public class Ort {
    public static Ort createOrt(String plz, String ort) {
        if (isPLZgueltig(plz)) {
            return new Ort(plz, ort);
        } else {
            return null;
        }
    }

    private Ort(String plz, String ort) {
        // ...
    }

    public static boolean isPLZgueltig(String plz) {
        // ...
    }
}
```