# Teil II

# Software Entwicklung mit UML (Design) und Java (Implementierung)

# Objekt, Klasse, Instanz

- **Objekt**:
  - Allgemein
  - Oberbegriff

# Objekt, Klasse, Instanz

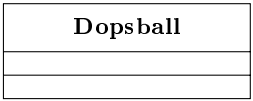
- **Objekt**:
  - Allgemein
  - Oberbegriff
- **Klasse**:
  - abstrakt
  - Gruppe von Objekten
  - Beschreibung der **Typen** der Attribute

# Objekt, Klasse, Instanz

- **Objekt**:
  - Allgemein
  - Oberbegriff
- **Klasse**:
  - abstrakt
  - Gruppe von Objekten
  - Beschreibung der **Typen** der Attribute
- **Instanz**:
  - konkret
  - ein Objekt mit **Werten** für (alle) Attribute (Wertebelegung)

# Modellierung: UML – Java

**Name** einer Klasse:




```
1  package mup1;
2
3  _____ class Dopsball {
4    // Block
5  }
```

# Modellierung: UML – Java

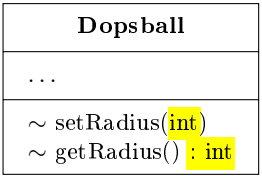
**Attribute** einer Klasse:

| Dopsball |
| --- |
| ∼ radius : int |
| |

```
1     _____ int radius;
```

# Modellierung: UML – Java

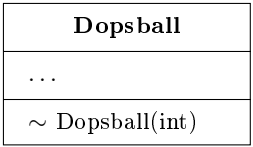
**Methoden** einer Klasse:




```
1  _____ void setRadius(
2    int radius
3  ) {
4    this.radius = radius;
5  }
6
7  _____ int getRadius() {
8    return radius;
9  }
```

# Modellierung: UML – Java

**Konstruktor** einer Klasse:




```
1  _____ Dopsball(
2    int radius
3  ) {
4    this.radius = radius;
5  }
```

# Modellierung: UML – Java

**Sichtbarkeit** einer Klasse:

| UML | Java | Bemerkung |
|---|---|---|
| $+$ | `public` | alle |
| $-$ | `private` | nur innerhalb der Klasse |
| $\#$ | `protected` | beinhaltet *package* |
| $\sim$ | | *package* |
| $/$ | | *derived*, keine Entsprechung in Java |

# Modellierung: UML – Java

**Sichtbarkeit** einer Klasse:

| Dopsball |
| --- |
| - radius : int |
| + Dopsball(int)<br>+ setRadius(int)<br>+ getRadius() : int |

```java
package mup1;

public class Dopsball {

  private int radius;

  public Dopsball(
    int radius
  ) {
    this.radius = radius;
  }

  public void setRadius(
    int radius
  ) {
    this.radius = radius;
  }

  public int getRadius() {
    return radius;
  }
}
```

# Java Class ⇔ Instance

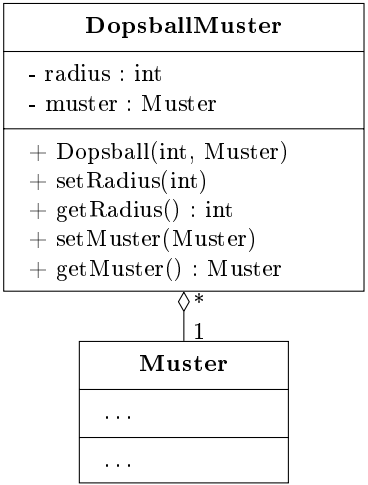
main

```
1  public static void main (String[] args) {
2    ...
3  }
```

new

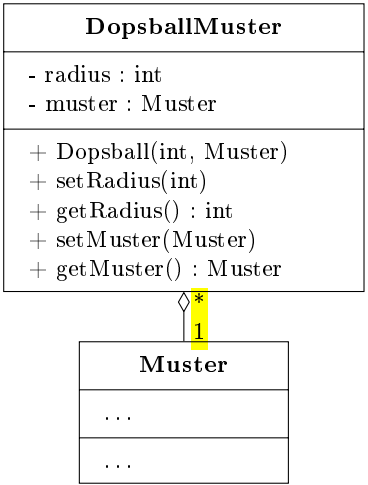
```
1  Dopsball dopsball1 = new Dopsball(1);
2  dopsball1.setRadius(5);
3  int radius1 = dopsball1.getRadius();
4
5  Dopsball dopsball2 = new Dopsball(7);
6  int radius2 = dopsball2.getRadius();
```

# Aggregation



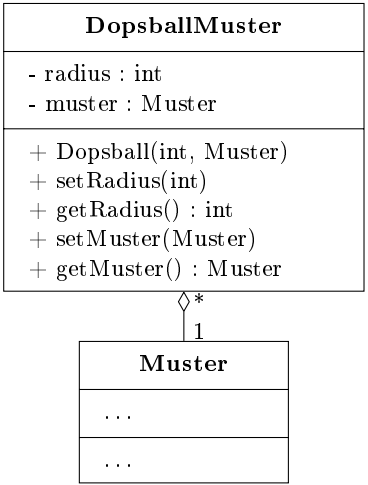

```
1  package mup1;
2
3  public class Muster {
4    ...
5  }
6
```

# Aggregation



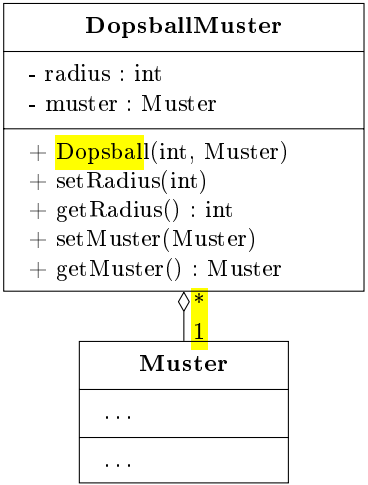

```
1  package mup1;
2
3  public class Dopsball {
4
5    private int radius;
6    private Muster muster;
7
8    public Dopsball(
9      int radius,
10     Muster muster
11   ) {
12     this.radius = radius;
13     this.muster = muster;
14   }
15
16
```

# Aggregation




```
1    public void setRadius(
2      int radius
3    ) {
4      this.radius = radius;
5    }
6
7    public int getRadius()
      {
8      return radius;
9    }
10
```

# Aggregation




```
1     public void setMuster(
2       Muster muster
3     ) {
4       this.muster = muster;
5     }
6
7     public Muster getMuster
        () {
8       return muster;
9     }
10  }
11
```

## Multiplizität

*Multiplizität:* Häufigkeit der Instanzen der beteiligten Klassen.

| UML | Wertebereich | Bedeutung |
|---|---|---|
| $n$ | $n \in \mathbb{N}_0$ | feste Anzahl |
| $*$ | | beliebige Anzahl (inkl. 0) |
| $n..m$ | $n, m \in \mathbb{N}_0,\ n < m$ | mindestens $n$, höchstens $m$ (inkl. $n, m$) |
| $n..*$ | $n \in \mathbb{N}_0$ | mindestens $n$ (inkl. $n$) |

## Multiplizität

*Multiplizität:* Häufigkeit der Instanzen der beteiligten Klassen.

| UML | Wertebereich | Bedeutung |
|---|---|---|
| $n$ | $n \in \mathbb{N}_0$ | feste Anzahl |
| $*$ | | beliebige Anzahl (inkl. 0) |
| $n..m$ | $n, m \in \mathbb{N}_0,\ n < m$ | mindestens $n$, höchstens $m$ (inkl. $n, m$) |
| $n..*$ | $n \in \mathbb{N}_0$ | mindestens $n$ (inkl. $n$) |

Für Aggregationen gilt:

- keine Angabe eines Wertes im Diagramm
  $\rightarrow$ der Wert ist unspezifiziert
  $\rightarrow$ äquivalent zu $*$

# Multiplizität

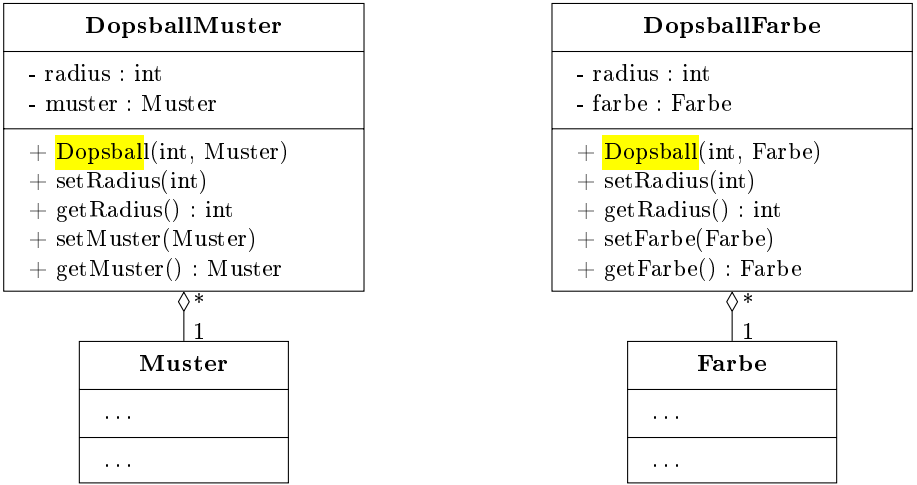
*Multiplizität:* Häufigkeit der Instanzen der beteiligten Klassen.

| UML | Wertebereich | Bedeutung |
|---|---|---|
| $n$ | $n \in \mathbb{N}_0$ | feste Anzahl |
| $*$ | | beliebige Anzahl (inkl. 0) |
| $n..m$ | $n, m \in \mathbb{N}_0,\ n < m$ | mindestens $n$, höchstens $m$ (inkl. $n, m$) |
| $n..*$ | $n \in \mathbb{N}_0$ | mindestens $n$ (inkl. $n$) |

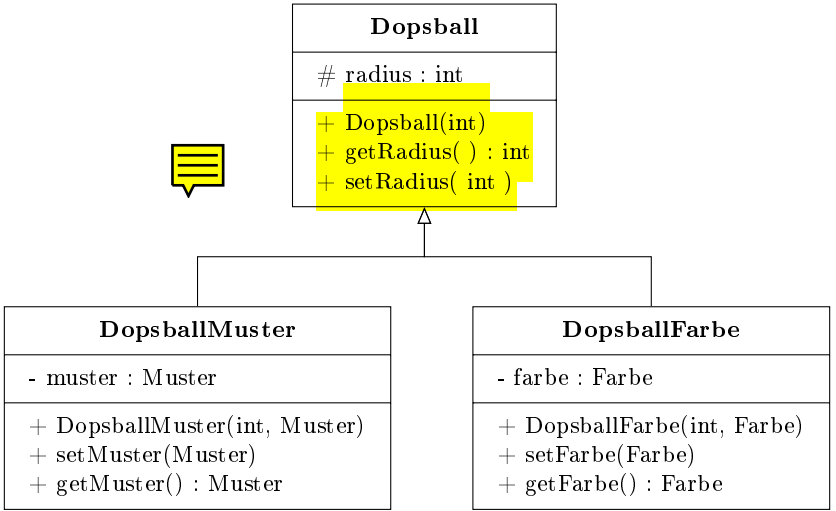
Für Aggregationen gilt:

- keine Angabe eines Wertes im Diagramm
  $\rightarrow$ der Wert ist unspezifiziert
  $\rightarrow$ äquivalent zu $*$
- der Wert muß für ein vollständiges UML festgelegt werden

# Generalisierung – Ableitung



# Generalisierung – Ableitung



# Modellierung: UML – Java

```java
package mup1;

public class Dopsball {

  protected int radius;

  public Dopsball(
    int radius
  ) {
    this.radius = radius;
  }

  public void setRadius (
    int radius
  ) {
    this.radius = radius;
  }

  public int getRadius () {
    return radius;
  }
}
```

# Modellierung: UML – Java

```java
package mup1;

public class DopsballMuster
  extends Dopsball {

  private Muster muster;

  public DopsballMuster (
    int radius,
    Muster muster
  ) {
    super(radius);
    this.muster = muster;
  }

  public void setMuster(
    Muster muster
  ) {
    this.muster = muster;
  }

  public Muster getMuster () {
    return muster;
  }
}
```

```java
package mup1;

public class DopsballFarbe
  extends Dopsball {

  private Farbe farbe;

  public DopsballFarbe (
    int radius,
    Farbe farbe
  ) {
    super(radius);
    this.farbe = farbe;
  }

  public void setFarbe(
    Farbe farbe
  ) {
    this.farbe = farbe;
  }

  public Farbe getFarbe () {
    return farbe;
  }
}
```

# Java: Objektorientierte Programmierung

`java.lang.Object`:

- `int hashCode()`
- `boolean equals(Object obj)`
- `String toString()`
  default: `getClass().getName() + '@' + Integer.toHexString(hashCode())`

# Java: Objektorientierte Programmierung

`java.lang.Object`:

- `int hashCode()`
- `boolean equals(Object obj)`
- `String toString()`
  default: `getClass().getName() + '@' + Integer.toHexString(hashCode())`

- Überladene Methoden: z.B. toString in einer selbstdefinierten Klasse
- `instanceof` (Generalisierung, Ableitung)

## Java: Ausgaben

Ausgaben:

- `System.out.print`:

```
System.out.print("12345");
System.out.print("67890");
```

Ausgabe:

```
1234567890
```

# Java: Ausgaben

Ausgaben:

- `System.out.print`:

```
System.out.print("12345");
System.out.print("67890");
```

Ausgabe:

```
1234567890
```

- `System.out.println`

```
System.out.println("12345");
System.out.println("67890");
```

Ausgabe:

```
12345
67890
```

# Java: Objektorientierte Programmierung

Überladene Methoden: z.B. toString() in einer selbstdefinierten Klasse

```
1  public void toString() {
2    System.out.println("Radius: " + radius);
3  }
```

`instanceof` (Generalisierung, Ableitung)

```
1  DopsballMuster dopsballMuster = new DopsballMuster();
2  boolean isDopsballMuster =
3    dopsballMuster instanceof DopsballMuster;
4  // Ergebnis: true
5  boolean isDopsballMuster =
6    dopsballMuster instanceof DopsballFarbe;
7  // Ergebnis: false
```

# Dokumentation

Einzeiliger Kommentar

```
// Erzeuge eine Variable a und weise ihr den Wert 5 zu
int a = 5;


// Erzeuge eine Variable a
// und weise ihr den Wert 5 zu
int a = 5;
```

# Dokumentation

Einzeiliger Kommentar

```
// Erzeuge eine Variable a und weise ihr den Wert 5 zu
int a = 5;


// Erzeuge eine Variable a
// und weise ihr den Wert 5 zu
int a = 5;
```

Mehrzeiliger Kommentar

```
/* Erzeuge eine Variable a und
   weise ihr den Wert 5 zu
*/
int a = 5;
```

# Dokumentation

Kombination von einzeiligem und mehrzeiligem Kommentar

```
/* Alternativer Ansatz
// Erzeuge eine Variable a und weise ihr den Wert 5 zu
int a;
a = 5;
*/
```

# Javadoc

Javadoc

- Dient zur Dokumentation des Quellcodes
- Dient als Referenz zum Nachlesen (ohne Quellcode zu kennen)

# Javadoc

Javadoc

- Dient zur Dokumentation des Quellcodes
- Dient als Referenz zum Nachlesen (ohne Quellcode zu kennen)

Javadoc einer Methode

```
/** Konstruktor
 * @param farbe Farbe des Objektes
 */
public FarbigesObjekt(
  Farbe farbe
) {
  this.farbe = farbe;
}
```

# Javadoc

Javadoc

- Dient zur Dokumentation des Quellcodes
- Dient als Referenz zum Nachlesen (ohne Quellcode zu kennen)

Javadoc einer Methode

```
/** Getter
 * @return gibt die Farbe des Objektes zurueck
 */
public Farbe getFarbe() {
  return farbe;
}
```