Programmierkurs Java

UE 18 – Dynamisches Binden

Dr.-Ing. Dietrich Boles

Gliederung

- Definition
- Prinzip
- Beispiel Graphik
- Instanz-Methoden
- private Instanz-Methoden
- Klassen-Methoden
- Attribute
- Vorteile
- Beispiel
- Zusammenfassung

Definition

Dynamisches Binden:

- Zuordnung eines Methodenrumpfes zu einem Methodenaufruf erst zur Laufzeit
- beim Aufruf einer Instanz-Methode über eine Objektvariable wird diejenige Methode ausgeführt, die der Klasse des referenzierten Objektes zugeordnet ist (→ **überschriebene Methode)**
- Relevanz: Laufzeit

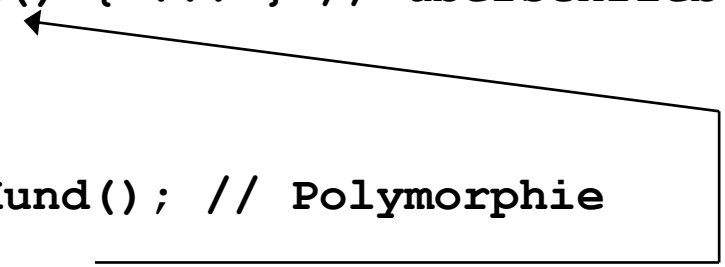
- statisches Binden: Zuordnung eines Methodenrumpfes zu einem Methodenaufruf bereits zur Compilierzeit

Prinzip

```
class Tier {
  void gibLaut() { ... }
}
class Hund extends Tier {
  void gibLaut() { ... } // überschriebene Methode
}

Tier t = new Hund(); // Polymorphie
t.gibLaut();                          // dyn. Binden

static void aergern(Tier tier) {
  tier.gibLaut(); ...
}
aergern(new Tier());  // Aufruf von gibLaut der Klasse Tier
aergern(new Hund());  // Aufruf von gibLaut der Klasse Hund
```



Beispiel Graphik (1)

```
class Graphik {
  void draw() {
    IO.println("Graphik draw");
} }

class Rectangle extends Graphik {
  void draw() {
    IO.println("Rectangle draw");
} }

class Square extends Rectangle {
  void draw() {
    IO.println("Square draw");
} }

class Circle extends Graphik {
  void draw() {
    IO.println("Circle draw");
} }
```

Beispiel Graphik (2)

```
class GraphikSet {
  Graphik[] elemente;
  int next;
  GraphikSet(int groesse) {
    this.elemente = new Graphik[groesse]; this.next = 0;
  }
  void add(Graphik obj) {
    this.elemente[this.next++] = obj;
  }
  void drawAll() {
    for (int i=0; i<this.next; i++)
      this.elemente[i].draw(); // dynamisches Binden!
} }
class GraphikProgramm {
  public static void main(String[] args) {
    GraphikSet menge = new GraphikSet(100);
    menge.add(new Circle());  // Polymorphie!
    menge.add(new Square());
    menge.drawAll(); // Ausgabe: Circle draw
                     //          Square draw
} }
```

Instanz-Methoden

- Instanz-Methoden werden dynamisch gebunden

```
class X {
  void print() {          // Instanzmethode
    IO.println("in X");
  }
  void call() {
    this.print();         // dynamisch gebunden
  }
}

class Y extends X {
  void print() {          // überschrieben
    IO.println("in Y");
  }
  public static void main(String[] args) {
    Y y = new Y();
    y.call();             // Ausgabe: in Y
} }
```

private Instanz-Methoden

- Ausnahme: **private** Instanz-Methoden werden statisch gebunden

```
class X {
  private void print() { // private Instanzmethode
    IO.println("in X");
  }
  void call() {
    this.print();        // statisch gebunden
  }
}

class Y extends X {
  void print() {         // überschrieben
    IO.println("in Y");
  }
  public static void main(String[] args) {
    Y y = new Y();
    y.call();            // Ausgabe: in X
} }
```

Klassen-Methoden

- Klassen-Methoden werden statisch gebunden

```
class X {
  static void print() {    // Klassen-Methode
    IO.println("in X");
  }
  void call() {
    this.print();      // statisch gebunden (X.print();)
  }
}
class Y extends X {
  static void print() {  // überschrieben
    IO.println("in Y");
  }
  public static void main(String[] args) {
    Y y = new Y();
    y.call();              // Ausgabe: in X
} }
```

Attribute

- Dynamisches Binden gilt nur für Methoden nicht für Attribute!

```
class X {
  int i;
  void f() { ... }
}
class Y extends X {
  int i;
  void f() { ... }
}
...
X obj = new Y();
obj.f();            // Aufruf der Methode f der Klasse Y
obj.i = 3;          // Zugriff auf Attribut i der Klasse X

((Y)obj).i = 5;     // Zugriff auf Attribut i der Klasse Y
```

- **Erweiterbarkeit (ohne Quellcode-Änderung):**

  Implementierung von erweiterbaren
  - Funktionen
  - Frameworks

  die auch für Objekte einsetzbar sind, deren Klassen zur Entwicklungszeit der Funktionen bzw. Frameworks noch gar nicht existierten.

  Beispiel: Java-GUI-Framework AWT

Beispiel

```
static boolean nullstelle(Funktion funk, int von, int bis) {
  for (int x = von; x <= bis; x++) {
    if (funk.f(x) == 0) return true;
  }
  return false;
}

class Funktion { int f(int x) { return x; } }

class QuadratFunktion extends Funktion {
  int f(int x) { return x * x; }
}

Funktion id = new Funktion();
Funktion q = new QuadratFunktion();
System.out.println(nullstelle(id, 2, 4));
System.out.println(nullstelle(q, -3, 4));
```

Zusammenfassung

- Polymorphie: Fähigkeit einer Objektvariablen vom Typ T_1, auf Objekte von Klassen eines anderen Typs T_2 verweisen zu können, wobei in Java T_2 Unterklasse von T_1 sein muss
- Statisches Binden: Bereits zur Compilierzeit steht fest, welche Methode ausgeführt wird (Klassenmethode, private Instanzmethoden)
- Dynamisches Binden: Erst zur Laufzeit wird ermittelt, welche Methode ausgeführt wird (nicht-private Instanzmethoden)
- Beim Aufruf einer nicht-private Instanz-Methode über eine Objektvariable wird diejenige Methode ausgeführt, die der Klasse des referenzierten Objektes zugeordnet ist
- Vorteil des dynamischen Bindens: einfache Erweiterbarkeit von Programmen (ohne Quellcode-Änderungen)