CARL VON OSSIETZKY universität OLDENBURG

Teil

Objektorientierte Programmierung

Unterrichtseinheit 32

Dynamisches Binden

Dr. Dietrich Boles

Gliederung

- Definition
- Prinzip
- Beispiel Graphik
- Instanz-Methoden
- private Instanz-Methoden
- Klassen-Methoden
- Attribute
- Vorteile
- Beispiel TicTacToe
- 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 ($\rightarrow$ **überschriebene Methode)**
- Relevanz: Laufzeit

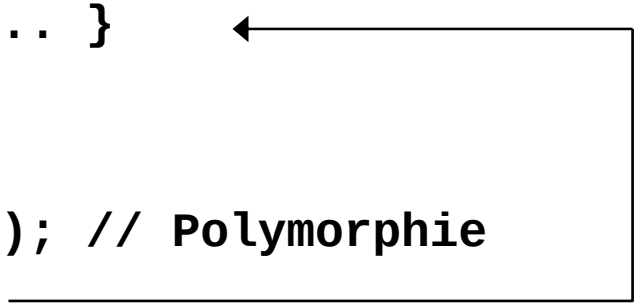
- statisches Binden: Zuordnung eines Methodenrumpfes zu einem Methodenaufruf bereits zur Compilierzeit

Prinzip

```
class A {
  void print() { ... }
}
class B extends A {
  void print() { ... }
}

A object1 = new B(); // Polymorphie
object1.print();                    // dyn. Binden

static void f(A object) {
  object.print();
}
f(new A());  // Aufruf von print der Klasse A
f(new B());  // Aufruf von print der Klasse B
```



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
```

Vorteile

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

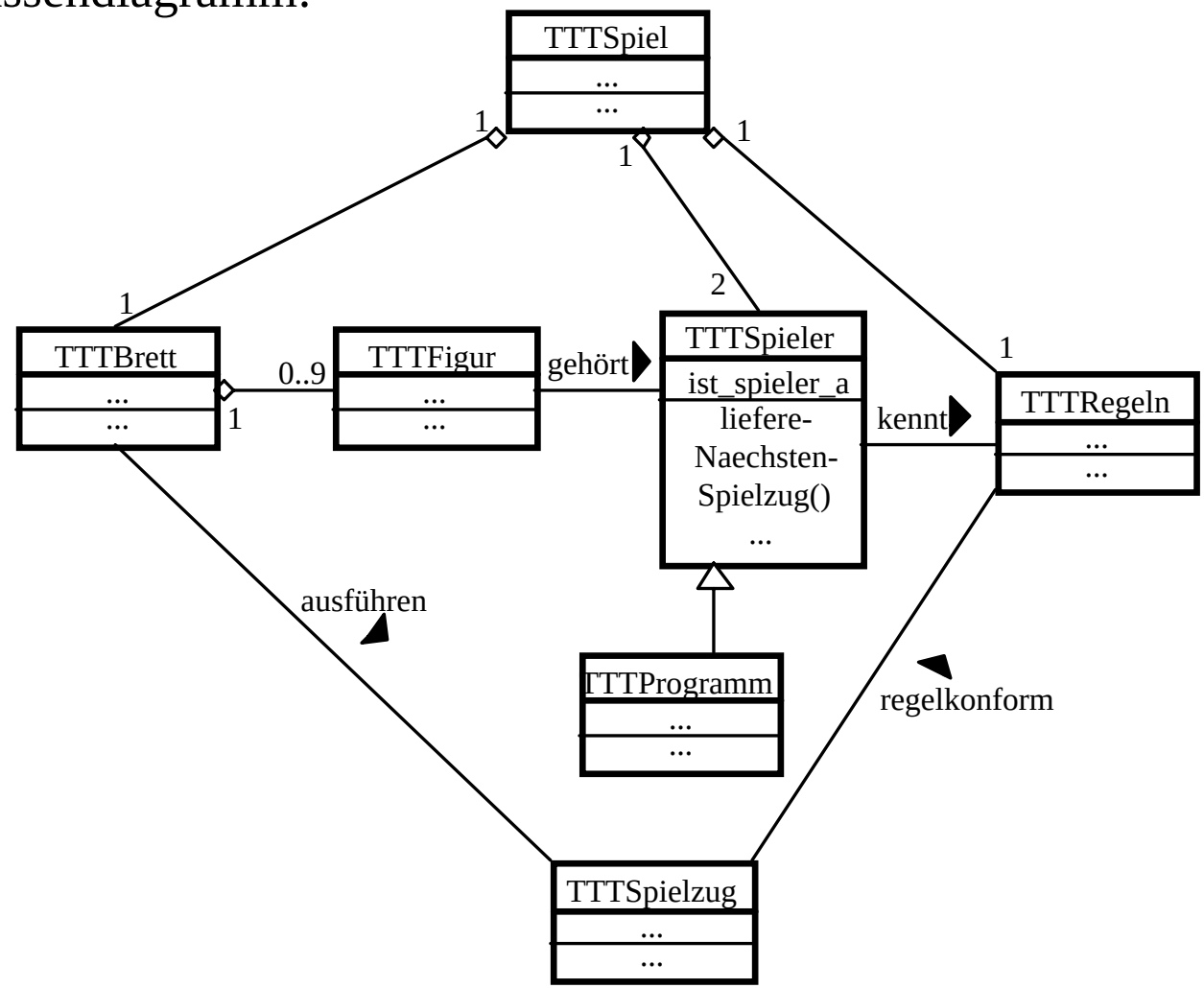
  - Methode `bearbeite` einsetzbar auch für später definierte Graphik-Klassen

```
void bearbeite(Graphik obj) {
  ...
  obj.draw();
  ...
}
```

  - Beispiel: TicTacToe (Folie 12ff; erweitert durch Klasse `TTTProgramm`)

Beispiel TicTacToe (1)

UML-Klassendiagramm:



Beispiel TicTacToe (2)

```
class TTTBrett, TTTFigur, TTTSpielzug, TTTRegeln ...


class TTTSpieler {

  boolean istSpielerA;
  TTTSpieler gegner;

  TTTSpieler(boolean istA) { this.istSpielerA = istA; }

  void setzeGegner(TTTSpieler gegner) { this.gegner = gegner;}

  boolean istSpielerA() { return this.istSpielerA; }

  // erfragt und liefert naechsten Spielzug des Spielers
  TTTSpielzug liefereNaechstenSpielzug(TTTSpielzug gegnerZug) {
    int zeile = IO.readInt("Zeile eingeben: ");
    int spalte = IO.readInt("Spalte eingeben: ");
    return new TTTSpielzug(zeile, spalte);
  }
}
```

Demo

Beispiel TicTacToe (3)

```
class TTTSpiel {
  // Attribute
  TTTSpieler spielerA, spielerB;
  TTTBrett brett;
  TTTRegeln regeln;

  // Konstruktor (Initialisierung eines Spiels)
  TTTSpiel(TTTSpieler a, TTTSpieler b,
           TTTBrett brett, TTTRegeln regeln) {
    this.spielerA = a; this.spielerB = b;
    this.brett = brett; this.regeln = regeln;
  }

  // Durchfuehrung eines Spiels
  void spielen() {
    this.brett.gebeSpielbrettAus();

    // Spieler A beginnt
    TTTSpieler aktSpieler = this.spielerA;
    TTTSpielzug zug = null;
```

Beispiel TicTacToe (4)

```
    do {  // abwechselndes Ziehen bis Spielende erreicht ist
      if (aktSpieler == this.spielerA) IO.println("Spieler A!");
      else IO.println("Spieler B ist am Zug!");
      // legalen Spielzug ermitteln und ausfuehren
      zug = aktSpieler.liefereNaechstenSpielzug(zug);
      if (!this.regeln.spielzugOK(zug)) {
            IO.println("Ungueltiger Spielzug! Verloren!");
            return;
      }
      this.brett.fuehreSpielzugAus(aktSpieler, zug);
      this.brett.gebeSpielbrettAus();
      // der andere Spieler ist nun ab Zug
      if (aktSpieler == spielerA) aktSpieler = this.spielerB;
      else aktSpieler = this.spielerA;
    } while (!this.regeln.spielEnde());
    // Spiel ist zuende
    this.regeln.gebeSiegerBekannt();
} }
```

Beispiel TicTacToe (5)

```
class TicTacToe {

  // hier startet das TicTacToe-Programm
  public static void main(String[] args) {

    TTTSpieler a = new TTTSpieler(true);
    TTTSpieler b = new TTTSpieler(false);

    a.setzeGegner(b);
    b.setzeGegner(a);
    TTTBrett brett = new TTTBrett();
    TTTRegeln regeln = new TTTRegeln(brett);
    TTTSpiel tictactoe = new TTTSpiel(a, b, brett, regeln);

    tictactoe.spielen();
  }
```

Beispiel TicTacToe (6)

```
class TTTProgramm extends TTTSpieler { // neue Klasse
  TTTBrett brett; TTTRegeln regeln;
  TTTProgramm(boolean istA) {
    super(istA);
    brett = new TTTBrett(); regeln = new TTTRegeln(this.brett);
  }
  TTTSpielzug liefereNaechstenSpielzug(TTTSpielzug gegnerZug) {
    if (gegnerZug != null)  // nicht der Anfangszug
      this.brett.fuehreSpielzugAus(this.gegner, gegnerZug);
    // eigenen Spielzug ermitteln (sehr einfache Strategie!)
    TTTSpielzug eigenerZug = null;
    hauptschleife:
    for (int i=0; i<TTTBrett.getGroesse(); i++) {
      for (int j=0; j<TTTBrett.getGroesse(); j++) {
        eigenerZug = new TTTSpielzug(i, j);
        if (regeln.spielzugOK(eigenerZug)) {
          break hauptschleife;
    } } }
    // eigenenSpielzug ausfuehren
    this.brett.fuehreSpielzugAus(this, eigenerZug);
    return eigenerZug;
} }
```

überschreiben

Beispiel TicTacToe (7)

```
class TicTacToe {

  // hier startet das TicTacToe-Programm
  public static void main(String[] args) {

    TTTSpieler a = new TTTSpieler(true);
    TTTSpieler b = new TTTProgramm(false); // Polymorphie
    // einzige Stelle, an der das alte Programm geaendert werden muss!
    // spaeter: Factory-Pattern

    a.setzeGegner(b);
    b.setzeGegner(a);
    TTTBrett brett = new TTTBrett();
    TTTRegeln regeln = new TTTRegeln(brett);
    TTTSpiel tictactoe = new TTTSpiel(a, b, brett, regeln);

    tictactoe.spielen();
  }
```

Beispiel TicTacToe (8)

```
    do {  // abwechselndes Ziehen bis Spielende erreicht ist
      if (aktSpieler == this.spielerA) IO.println("Spieler A!");
      else IO.println("Spieler B ist am Zug!");
      // legalen Spielzug ermitteln und ausfuehren
      zug = aktSpieler.liefereNaechstenSpielzug(zug);
      if (!this.regeln.spielzugOK(zug)) {
            IO.println("Ungueltiger Spielzug! Verloren!");
            return;
      }
      this.brett.fuehreSpielzugAus(aktSpieler, zug);
      this.brett.gebeSpielbrettAus();
      // der andere Spieler ist nun ab Zug
      if (aktSpieler == spielerA) aktSpieler = this.spielerB;
      else aktSpieler = this.spielerA;
    } while (!this.regeln.spielEnde());
    // Spiel ist zuende
    this.regeln.gebeSiegerBekannt();
} }
```

dyn. Binden

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)