CARL VON OSSIETZKY universität OLDENBURG

Teil

Objektorientierte Programmierung

Unterrichtseinheit 26

Objektorientierte Softwareentwicklung

Dr. Dietrich Boles

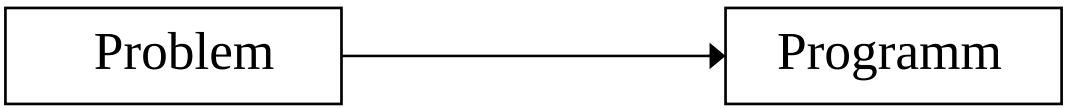
Gliederung

- Softwareentwicklung
- OO-Softwareentwicklung
  - OO-Analyse
  - OO-Entwurf
  - OO-Implementierung
  - OO-Test
  - OO-Dokumentation
- Beispiel Gewicht
  - Prozedurale Zerlegung
  - OO-Softwareentwicklung
- Beispiel TicTacToe
  - Prozedurale Zerlegung
  - OO-Softwareentwicklung
- Beispiel Pruefung
  - OO-Softwareentwicklung
- Zusammenfassung

Softwareentwicklung

Ziel der Softwareentwicklung:

- Erstellung eines Softwaresystems zur Lösung eines Problems



bisherige Vorgehensweise:

- Analyse (Konkretisierung der Problembeschreibung)
- Entwurf (Algorithmus, prozedurale Zerlegung, Top-Down-Entwurf)
- Implementierung (Prozeduren, Anweisungen, ...)
- Test (Überprüfung der Korrektheit und Vollständigkeit)
- Dokumentation (Kommentare, Handbücher, ...)
- Randbedingung: kleine Probleme ("Programmieren im Kleinen")

OO-Softwareentwicklung (1)

Vorgehensweise bei der OO-Softwareentwicklung:

- OO-Analyse:
  - Analyse des Anwendungsgebietes
  - Modellierung des Anwendungsgebietes
- OO-Entwurf / OO-Design:
  - Transformation in ein Modell des Lösungsraumes
- OO-Implementierung:
  - Transformation in ein Programm (= Abstraktion des Anwendungsgebietes)
- OO-Test:
  - Test der Methoden, Klassen und des Zusammenspiels der Klassen
- OO-Dokumentation:
  - UML-Diagramme

OO-Softwareentwicklung (2)

- Kennzeichen der OO-Softwareentwicklung
  - durchgängiger Prozess ohne Strukturbruch
  - Evolutionäres Vorgehen: iterativ und inkrementell
  - geeignet für große Probleme ("Programmieren im Großen")
  - Mittelpunkt der OO-Softwareentwicklung: Klassen und Objekte
  - "Bottom-Up-Entwurf"

- Vorteile:
  - einfache Erweiterbarkeit existierender Programme
  - Wiederverwendbarkeit von Programmen
  - Modularisierung / Datenkapselung
  - natürlichere Modellierung von Problemen

OO-Analyse

Aufgabe:

- Untersuchung des Problem- bzw. Anwendungsbereiches

Ziel:

- Erstellung eines Modells (OOA-Modell) als Abbild des statischen Aufbaus des Anwendungsgebietes sowie der dynamischen Abläufe im Anwendungsgebiet

Aktivitäten:

- Identifikation von Objekten und Klassen
- Identifikation von Beziehungen zwischen Objekten
- Identifikation von Attributen (Eigenschaften) u. Methoden (Verhaltensweisen)
- Identifikation von (Arbeits-)Abläufen

Methoden:

- grammatikalische Untersuchung der Problembeschreibung:
  - Substantive → Klassen/Objekte
  - Adjektive → Attribute
  - Verben → Methoden
  - Sätze mit mehreren Substantiven → Beziehungen

OO-Entwurf/OO-Design

Aufgabe:

- Abbildung des Modells des Anwendungsgebietes (OOA-Modells) auf ein Modell des zu entwickelnden Systems (OOD-Modell)

Ziel:

- konkrete Vorbereitung der Implementierungsphase

Aktivitäten:

- Ergänzung des OOA-Modells um weitere Klassen, Objekte, Attribute,Methoden, die im Problembereich nicht auftreten, bei der Implementierung aber unerlässlich sind (Speicherung, Darstellung, ...)
- Entwurf der Softwarearchitektur
- Berücksichtigung von Betriebssystem, Programmiersprache, Datenverwaltung, ...

Methoden:

- Untersuchung von möglichen Anwendungsszenarien des Systems
- Verwendung von Entwurfsmustern (Design Pattern)

OO-Implementierung

Aufgabe:

- Umsetzung des OOD-Modells in eine konkrete Programmiersprache

Ziel:

- Erstellung eines Programmsystems

Aktivitäten:

- Implementierung der Klassen des OOD-Modells
- Aufbau von Klassenbibliotheken

Methoden:

- relativ mechanische Überführung des OOD-Modells in ein Programm
- Nutzung von Klassenbibliotheken

Anmerkungen:

- auch Nutzung nicht-objektorientierter Sprachen möglich

OO-Test

Aufgabe:

- Überprüfung der Korrektheit des Programmsystems

Ziel:

- Erstellung eines korrekten, vollständigen Programmsystems

Aktivitäten:

- Test der Korrektheit der Methoden der einzelnen Klassen
- Test der Korrektheit der einzelnen Klassen
- Test der Korrektheit des Zusammenspiels der einzelnen Klassen

Methoden:

- übliche Testmethoden

Anmerkungen:

- Vorteile objektorientierter Software (Datenkapselung, Erweiterbarkeit, Wiederverwendbarkeit) kommen zur Geltung
- OO-Modellierungsnotationen sorgen für gute Dokumentation und Verständlichkeit

OO-Dokumentation

Unified Modeling Language:

- UML
- OO-Modellierungsnotation
- Festhalten von Ergebnissen
- Kommunikationsgrundlage
- **keine Entwicklungsmethode!**

- Strukturdiagramme
  - Klassendiagramme
  - Paketdiagramme
- Verhaltensdiagramme
  - Anwendungsfalldiagramme
  - Interaktionsdiagramme
    - Sequenzdiagramme
    - Kollaborationsdiagramme
  - Zustandsdiagramme
  - Aktivitätsdiagramme
- Implementierungsdiagramme
  - Komponentendiagramme
  - Einsatzdiagramme

Beispiel Gewicht

Problemstellung:

Das so genannte Normalgewicht berechnet sich nach der Formel "Körpergröße (in cm) minus 100". Das Idealgewicht beträgt bei Männern 90% und bei Frauen 85% des Normalgewichts.

Schreiben Sie ein Java-Programm, welches nach Eingabe von Größe, Gewicht und Geschlecht ausgibt, ob ein Mensch zu dick oder zu dünn ist, oder ob er/sie zwischen Ideal- und Normalgewicht liegt.

Demo

Beispiel Gewicht / Prozedurale Zerlegung (1)

```
class Mensch {  // Verbund
  float gewicht;
  boolean maennlich;
  int groesse;
}

class Gewicht {

  public static void main(String[] args) {
    do {
      Mensch mensch = menschEinlesen();
      menschUeberpruefen(mensch);
    } while (weitereUeberpruefung());
  }

  static Mensch menschEinlesen() {}
  static void menschUeberpruefen(Mensch m) {}
  static boolean weitereUeberpruefung() {}
}
```

Beispiel Gewicht / Prozedurale Zerlegung (2)

```
static Mensch menschEinlesen() {
  Mensch person = new Mensch();
  person.groesse = IO.readInt("Groesse (cm) eingeben: ");
  person.gewicht = IO.readFloat("Gewicht (kg) eingeben: ");
  person.maennlich = IO.readChar("Maennlich (m/w)") == 'm';
  return person;
}

static boolean weitereUeberpruefung() {
  char weiter = IO.readChar("Weitere Ueberpruefung (j/n)?");
  return weiter == 'j';
}

static void menschUeberpruefen(Mensch m) {}
```

Beispiel Gewicht / Prozedurale Zerlegung (3)

```
static void menschUeberpruefen(Mensch m) {
  float normalgewicht = normalgewichtBerechnen(m.groesse);
  float idealgewicht =
    idealgewichtBerechnen(normalgewicht, m.maennlich);

  if (menschHatUebergewicht(m.gewicht, normalgewicht)) {
    IO.println("Alter Fettsack!");
  } else if (menschHatUntergewicht(m.gewicht, idealgewicht)) {
    IO.println("Na, du Hungerhaken!");
  } else {
    IO.println("Idealer Gewichtsbereich!");
  }
}

static float normalgewichtBerechnen(int cm) {}
static float idealgewichtBerechnen(float norm, boolean mann) {}
static boolean menschHatUebergewicht(float gew, float normal) {}
static boolean menschHatUntergewicht(float gew, float ideal) {}
```

Beispiel Gewicht / Prozedurale Zerlegung (4)

```
static float normalgewichtBerechnen(int cm) {
  return cm - 100;
}

static float idealgewichtBerechnen(float normalgewicht,
                                   boolean maennlich) {
  if (maennlich) {
    return normalgewicht / 100.0f * 90.0f;
  } else {
    return normalgewicht / 100.0f * 85.0f;
  }
}

static boolean menschHatUebergewicht(float gew, float normal) {
  return gew > normal;
}

static boolean menschHatUntergewicht(float gewicht, float ideal) {
  return gewicht < ideal;
}
```

Beispiel Gewicht / OO-Analyse

Objekte: Menschen ⇨

```
class Mensch {
```

Eigenschaften von Menschen: ⇨

- Geschlecht
- Größe
- Gewicht

```
boolean maennlich;
int  groesse;    // cm
float gewicht;   // kg
```

Funktionalitäten von Menschen: ⇨

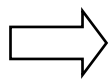
- hat Übergewicht
- hat Untergewicht
- hat ordentliches Gewicht

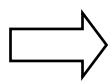
```
boolean hatUebergewicht()
boolean hatUntergewicht()
boolean hatOrdentlichesGewicht()

}
```

Beispiel Gewicht / OO-Entwurf

weitere benötigte Funktionalitäten:

- set/get-Methoden
  - istMaennlich
  - getGroesse
  - getGewicht
- Operationen
  - berechneNormalgewicht
  - berechneIdealgewicht





```
class Mensch {

boolean maennlich;
int  groesse;    // cm
float gewicht;   // kg
boolean hatUebergewicht()
boolean hatUntergewicht()
boolean hatOrdentlichesGewicht()

boolean istMaennlich()
int getGroesse()
float getGewicht()

float berechneNormalgewicht()
float berechneIdealgewicht()
}
```

Beispiel Gewicht / OO-Implementierung (1)

```
class Mensch {
  // Attribute
  float gewicht;   // in kg
  boolean maennlich;
  int groesse;     // in cm

  // Konstruktor
  Mensch(float gew, boolean mann, int groe) {
    this.gewicht = gew;
    this.maennlich = mann;
    this.groesse = groe;
  }

  // set/get-Methoden
  float getGewicht() { return this.gewicht; }
  float getGroesse() { return this.groesse; }
  boolean istMaennlich() { return this.maennlich; }
```

Beispiel Gewicht / OO-Implementierung (2)

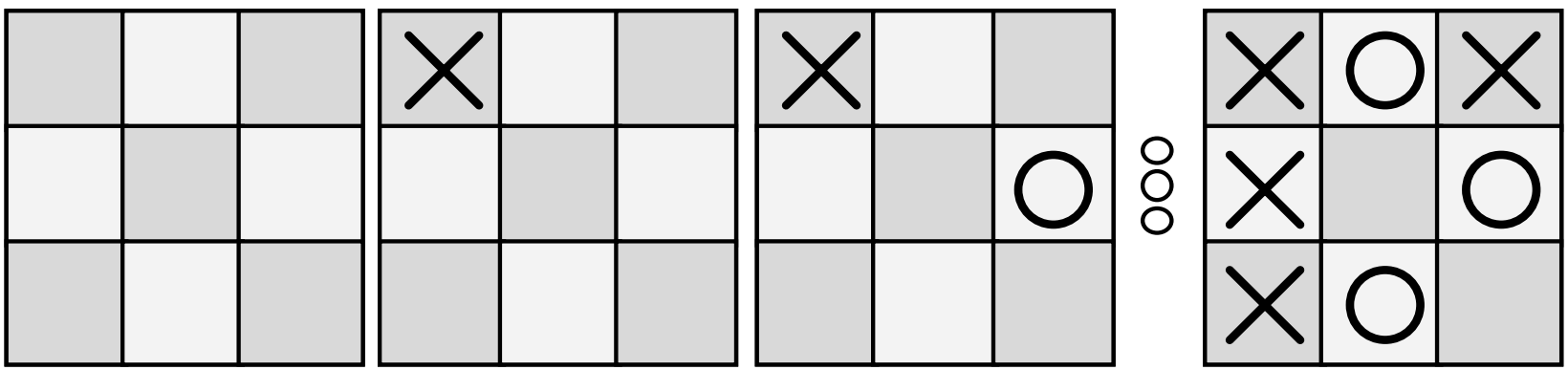
```
  float berechneNormalgewicht() { return this.groesse - 100; }
  float berechneIdealgewicht() {
    float normalgewicht = this.groesse - 100;
    if (this.maennlich)
      return normalgewicht / 100.0f * 90.0f;
    return normalgewicht / 100.0f * 85.0f;
  }
  boolean hatUebergewicht() {
    return this.gewicht > this.berechneNormalgewicht();
  }
  boolean hatUntergewicht() {
    return this.gewicht < this.berechneIdealgewicht();
  }
  boolean hatOrdentlichesGewicht() {
    return this.gewicht <= this.berechneNormalgewicht() &&
           this.gewicht >= this.berechneIdealgewicht();
  }
}
```

Beispiel Gewicht / OO-Implementierung (3)

```
class OOGewicht {
  public static void main(String[] args) {
    do {
      Mensch person =
        new Mensch(IO.readFloat("Gewicht (kg) eingeben: "),
                   IO.readChar("Maennlich (m/w)") == 'm',
                   IO.readInt("Groesse (cm) eingeben: "));

      if (person.hatUebergewicht()) {
        IO.println("Alter Fettsack!");
      } else if (person.hatUntergewicht()) {
        IO.println("Na, du Hungerhaken!");
      } else {
        IO.println("Idealer Gewichtsbereich!");
      }
    } while (IO.readChar("Weitere Ueberpruefung (j/n)?") == 'j');
  }
}
```

Beispiel TicTacToe

Beim TicTacToe-Spiel versuchen zwei Personen das Spiel zu gewinnen, indem sie abwechselnd Kreuz-Figuren (Spieler A) und Kreis-Figuren (Spieler B) auf einem 3x3-Brett so plazieren, dass ihre Figuren in einer Horizontalen, Vertikalen oder Diagonalen eine 3er-Reihe bilden. Anfangs ist das Brett leer. Das Spiel endet mit einem Unentschieden, wenn alle Felder des Brettes besetzt sind und kein Spieler eine 3er-Reihe bilden konnte.



leeres Spielbrett | Zug von Spieler A | Zug von Spieler B | Sieger: Spieler A

Demo

Beispiel TicTacToe / Prozedurale Zerlegung (1)

```
class Spielzug {
  int zeile;   // 0-2
  int spalte;  // 0-2
}

class TicTacToe {

  public static void main(String[] args) {
    int[][] brett = spielbrettInitialisieren();
    spielbrettAusgeben(brett);

    // abwechselndes Ziehen bis Spielende erreicht ist
    boolean aAmZug = true;
    while (true) {
      int ergebnis = spielzugDurchfuehren(brett, aAmZug);
      if (ergebnis >= 0) { // Ende
          ergebnisAusgeben(ergebnis);
          break;
      }
      aAmZug = !aAmZug; // Spielerwechsel
    }
  }
```

Beispiel TicTacToe / Prozedurale Zerlegung (2)

```
static int[][] spielbrettInitialisieren() {
  int[][] brett = new int[3][3];
  for (int zeile=0; zeile<3; zeile++) {
    for (int spalte=0; spalte<3; spalte++) {
      brett[zeile][spalte] = 0;
      // 0 = leeres Feld; 1 = Spieler A; 2 = Spieler B
    }
  }
  return brett;
}

static void spielbrettAusgeben(int[][] brett) {
  IO.print("  ");
  for (int j=0; j<3; j++) {
    IO.print(j); IO.print(" ");
  }
  IO.println();

  ...
}
```

Beispiel TicTacToe / Prozedurale Zerlegung (3)

```
static void ergebnisAusgeben(int ergebnis) {
  if (ergebnis == 1) {
    IO.println("Spieler A hat gewonnen!");
  } else if (ergebnis == 2) {
    IO.println("Spieler B hat gewonnen!");
  } else if (ergebnis == 0) {
    IO.println("Unentschieden!");
  }
}


static int spielzugDurchfuehren(int[][] brett,
                                boolean aAmZug) {
  spielerAusgeben(aAmZug);
  Spielzug zug = naechstenSpielzugErfragen(brett);
  spielzugAusfuehren(brett, zug, aAmZug);
  spielbrettAusgeben(brett);
  return spielEnde(brett);
}
```

Beispiel TicTacToe / Prozedurale Zerlegung (4)

```
static void spielerAusgeben(boolean aAmZug) {
  if (aAmZug)
    IO.println("Spieler A ist am Zug!");
  else
    IO.println("Spieler B ist am Zug!");
}

static Spielzug naechstenSpielzugErfragen(int[][] brett) {
  Spielzug zug = new Spielzug();
  zug.zeile = IO.readInt("Zeile (0-2): ");
  zug.spalte = IO.readInt("Spalte (0-2): ");
  while (!spielzugOK(brett, zug)) {
    IO.println("Ungueltig; neue Eingabe!");
    zug.zeile = IO.readInt("Zeile (0-2): ");
    zug.spalte = IO.readInt("Spalte (0-2): ");
  }
  return zug;
}
```

Beispiel TicTacToe / Prozedurale Zerlegung (5)

```
static boolean spielzugOK(int[][] brett, Spielzug zug) {
  // Koordinaten ok ?
  if (zug.zeile < 0 || zug.zeile > 2 || zug.spalte < 0 ||
      zug.spalte > 2)
    return false;

  // schon besetzt ?
  if (brett[zug.zeile][zug.spalte] != 0)
    return false;

  return true;
}

static void spielzugAusfuehren(int[][] brett,
                               Spielzug zug,
                               boolean spielerA) {
  if (spielerA)
    brett[zug.zeile][zug.spalte] = 1;
  else
    brett[zug.zeile][zug.spalte] = 2;
}
```

Beispiel TicTacToe / Prozedurale Zerlegung (6)

```
static int spielEnde(int[][] brett) {
  if (horizontale(brett, 1) == 1 || vertikale(brett, 1) == 1 ||
      diagonale(brett, 1) == 1) {
    return 1; // Sieger ist Spieler A
  }
  if (horizontale(brett, 2) == 2 || vertikale(brett, 2) == 2 ||
      diagonale(brett, 2) == 2) {
    return 2; // Sieger ist Spieler B
  }
  if (allesBelegt(brett))
    return 0; // unentschieden
  return -1; // noch kein Spielende
}

static boolean allesBelegt(int[][] brett) {
  for (int zeile=0; zeile<brett.length; zeile++)
    for (int spalte=0; spalte<brett.length; spalte++)
      if (brett[zeile][spalte] == 0)
        return false;
  return true;
}
```

Beispiel TicTacToe / Prozedurale Zerlegung (7)

```
static int vertikale(int[][] brett, int spieler) {
  for (int r=0; r<3; r++) {
    int anzahl = 0;
    for (int s=0; s<3; s++) {
      if (brett[r][s] == spieler) anzahl++;
    }
    if (anzahl == 3) return spieler; // Sieger !
  }
  return -1;
}

static int horizontale(int[][] brett, int spieler) {
  for (int s=0; s<3; s++) {
    int anzahl = 0;
    for (int r=0; r<3; r++) {
      if (brett[r][s] == spieler) anzahl++;
    }
    if (anzahl == 3) return spieler; // Sieger !
  }
  return -1;
}
```

Beispiel TicTacToe / Prozedurale Zerlegung (8)

```
static int diagonale(int[][] brett, int spieler) {
  // links nach rechts
  int anzahl = 0;
  for (int f=0; f<3; f++) {
    if (brett[f][f] == spieler) anzahl++;
  }
  if (anzahl == 3) return spieler; // Sieger !

  // rechts nach links
  anzahl = 0;
  for (int f=0; f<3; f++) {
    if (brett[f][2-f] == spieler) anzahl++;
  }
  if (anzahl == 3) return spieler; // Sieger !

  return -1;
}
```

Beispiel TicTacToe / OO-Analyse (1)

Identifikation von Objekten / Klassen:

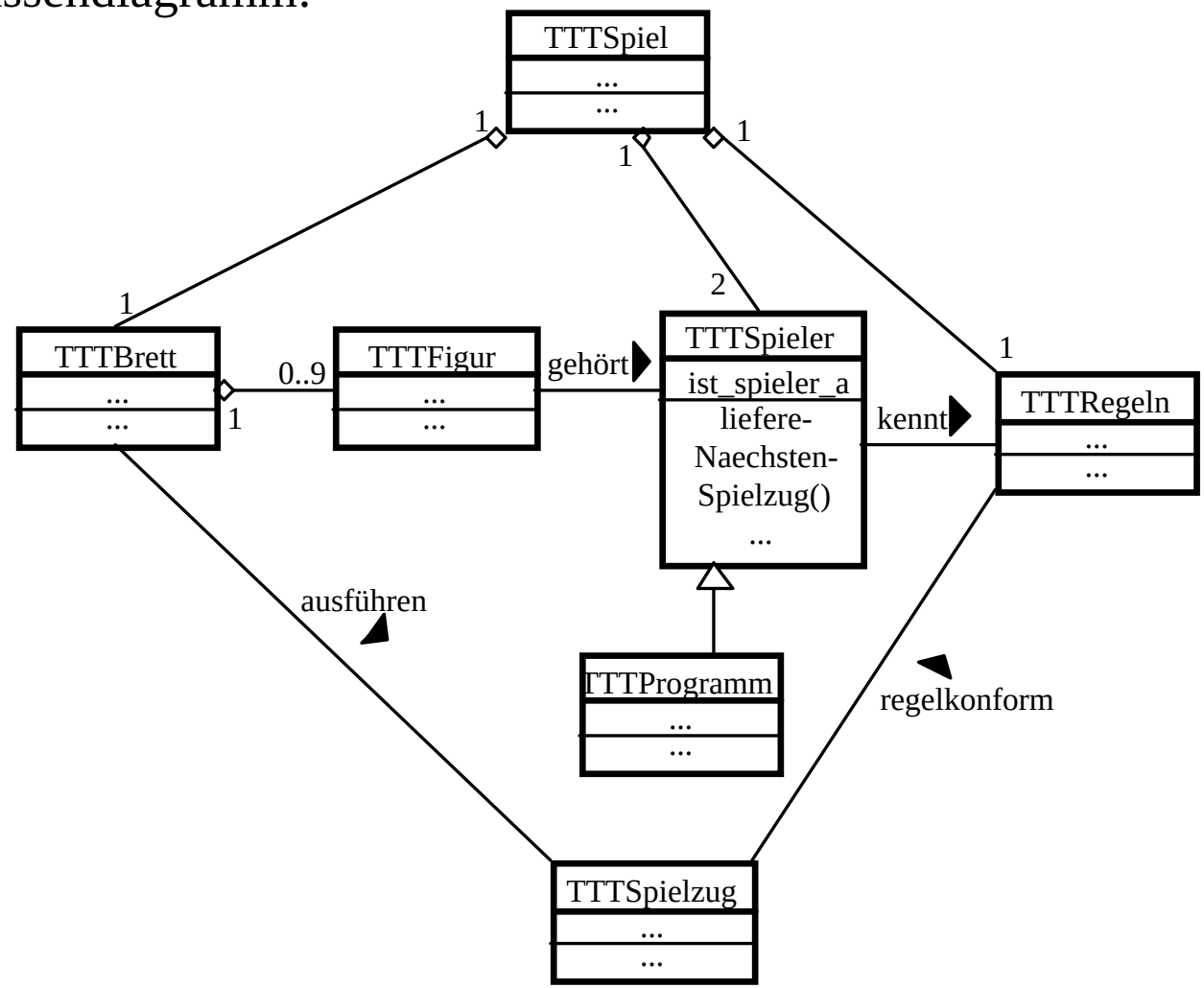
- Spiel (class TTTSpiel)
- Spieler (class TTTSpieler)
- Spielbrett (class TTTBrett)
- Spielfiguren (class TTTFigur)
- Spielregeln (class TTTRegeln)
- Spielzüge (class TTTSpielzug)

Identifikation von Beziehungen zwischen Objekten:

- ein Spiel besteht aus einem Spielbrett
- zu einem Spiel gehören Spielregeln
- zu einem Spiel gehören zwei Spieler
- auf dem Spielbrett können Spielfiguren plaziert werden
- auf dem Spielbrett werden Spielzüge ausgeführt
- jede Spielfigur gehört einem Spieler
- jeder Spieler kennt die Spielregeln
- Spielzüge müssen regelkonform zu den Spielregeln sein
- ...

Beispiel TicTacToe / OO-Analyse (2)

UML-Klassendiagramm:



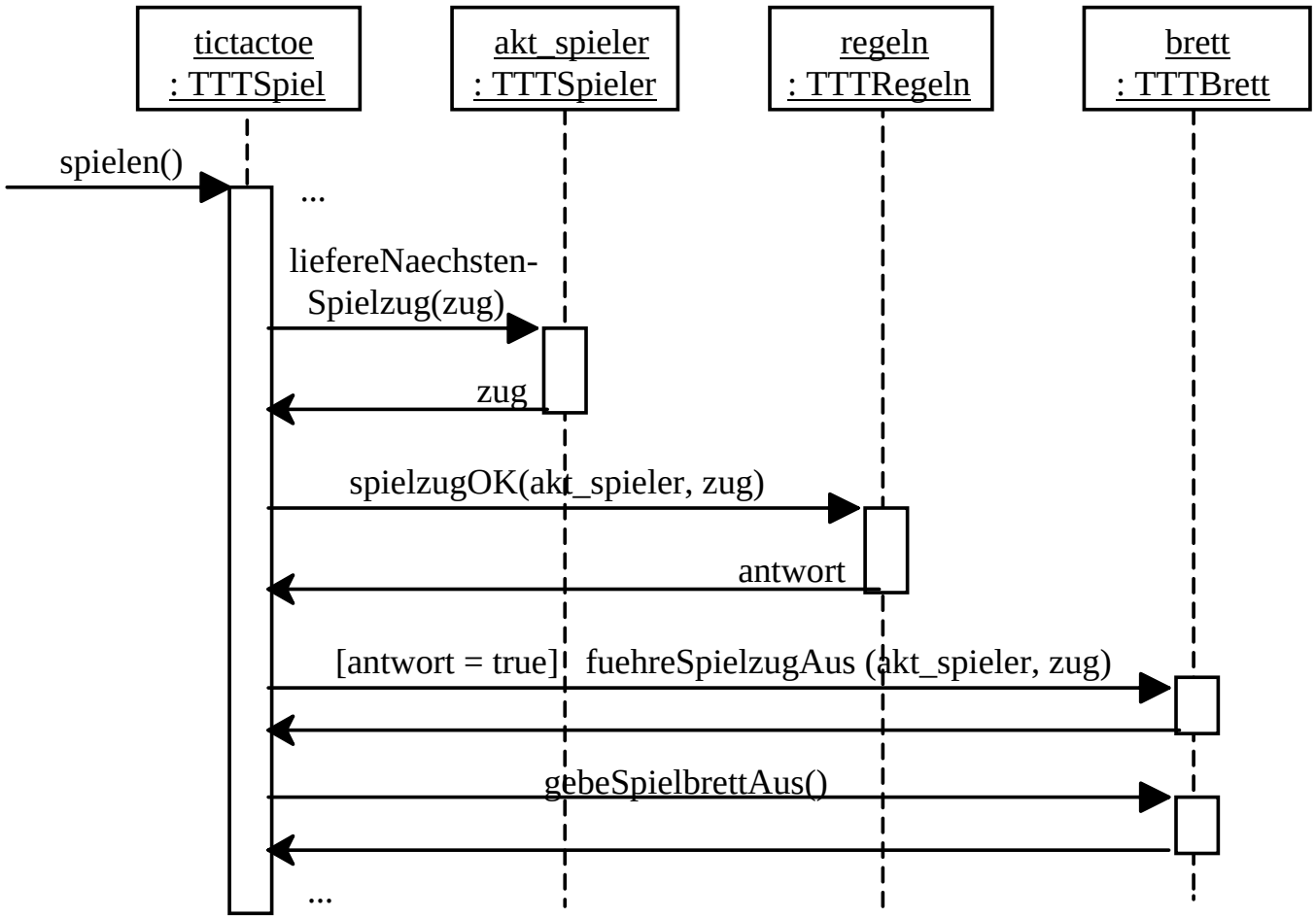
Beispiel TicTacToe / OO-Analyse (3)

Identifikation von Attributen / Methoden:

```
class TTTSpiel:
  void spielen();

class TTTBrett:
  void fuehreZugAus(TTTSpieler spieler, TTTSpielzug zug);
  void gebeSpielbrettAus();

class TTTFigur:
  boolean aOderB;
  int zeile, spalte;

class TTTRegeln:
  boolean spielzugOK(TTTSpielzug zug);
  boolean spielEnde();
  void gebeSiegerBekannt();

class TTTSpieler:
   boolean istSpielerA;
   TTTSpielzug liefereNaechstenZug(TTTSpielzug geg_zug);
```

Beispiel TicTacToe / OO-Analyse (4)

Identifikation von Arbeitsabläufen (UML-Sequenzdiagramm):



Beispiel TicTacToe / OO-Implementierung (1)

```
class TTTSpielzug {

  int zeile, spalte;

  // Konstruktor (Initialisierung eines Spielzugs)
  TTTSpielzug(int z, int s) {
    this.zeile = z;
    this.spalte = s;
  }

  int getZeile() {
    return this.zeile;
  }

  int getSpalte() {
    return this.spalte;
  }
}
```

Beispiel TicTacToe / OO-Implementierung (2)

```
class TTTSpieler {

  boolean istSpielerA;

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

  boolean istSpielerA() { return this.istSpielerA; }
  boolean istSpielerB() { return !this.istSpielerA(); }

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

Beispiel TicTacToe / OO-Implementierung (3)

```
class TTTFigur {
  final static char KREUZ = 'X';  final static char KREIS = 'O';

  boolean aOderB;    // a == true
  int zeile, spalte;

  TTTFigur(boolean aOderB, int zeile, int spalte) {
    this.aOderB = aOderB; this.zeile = zeile; this.spalte = spalte;
  }

  boolean istAFigur() { return this.aOderB; }
  boolean istBFigur() { return !this.istAFigur(); }
  int getZeile() { return this.zeile; }
  int getSpalte() { return this.spalte; }

  static char getAFigur() { return KREUZ; }
  static char getBFigur() { return KREIS;  }
}
```

Beispiel TicTacToe / OO-Implementierung (4)

```
class TTTBrett {
  final static int GROESSE = 3;  // Brettgroesse
  final static TTTFigur LEER = null; // leeres Feld

  TTTFigur[][] brett;

  TTTBrett() { // anfangs ist das Spielbrett leer
    this.brett = new TTTFigur[GROESSE][GROESSE];
    for (int i=0; i<GROESSE; i++)
      for (int j=0; j<GROESSE; j++)
        this.brett[i][j] = LEER;
  }
  // Ausfuehrung eines Spielzugs auf dem Brett
  void fuehreSpielzugAus(TTTSpieler sp, TTTSpielzug zug) {
    // abhaengig vom aktuellen Spieler, wird eine A_Figur oder eine
    // B_Figur an die Stelle gesetzt, die der Spielzug angibt
    int z = zug.getZeile(); int s = zug.getSpalte();
    this.brett[z][s] = new TTTFigur(sp.istSpielerA(), z, s);
  }
```

Beispiel TicTacToe / OO-Implementierung (5)

```
  // Ausgabe des aktuellen Spielbretts
  void gebeSpielbrettAus() {
    IO.print("  ");
    for (int j=0; j<GROESSE; j++) {
      IO.print(j); IO.print(" ");
    }
    IO.println();
    ...
  }

  // Lieferung der Figur an der Stelle z/s
  TTTFigur liefereFigur(int z, int s) {
    return this.brett[z][s];
  }

  // liefere die Brettgroesse
  static int getGroesse() { return GROESSE; }
}
```

Beispiel TicTacToe / OO-Implementierung (6)

```
class TTTRegeln {

  // Konstanten
  final static boolean SPIELER_A = true;
  final static boolean SPIELER_B = !SPIELER_A;

  final static int SIEGER_A = 1;
  final static int SIEGER_B = 2;
  final static int UNENTSCHIEDEN = 0;
  final static int KEIN_SIEGER = -1;

  // Attribute
  TTTBrett brett;

  // Konstruktor (Initialisierung der Regeln)
  TTTRegeln(TTTBrett brett) {
    this.brett = brett;
  }
```

Beispiel TicTacToe / OO-Implementierung (7)

```
boolean spielzugOK(TTTSpielzug zug) {
  // ok, wenn Positionangaben gueltig u. keine Figur auf Feld ex.
  return (zug.getZeile() >= 0) &&  (zug.getSpalte() >= 0) &&
    (zug.getZeile() <= TTTBrett.getGroesse()-1) &&
    (zug.getSpalte() <= TTTBrett.getGroesse()-1) &&
    (brett.liefereFigur(zug.getZeile(),zug.getSpalte()) == null);
}

boolean spielEnde() {  // Felder besetzt o. Sieger ex.
  int besetzteFelder = 0;
  for (int i=0; i<TTTBrett.getGroesse(); i++)
    for (int j=0; j<TTTBrett.getGroesse(); j++)
      if (this.brett.liefereFigur(i, j) != null)
        besetzteFelder++;
  return (besetzteFelder ==
            (TTTBrett.getGroesse() * TTTBrett.getGroesse()))
         || (this.ermittleSieger() > UNENTSCHIEDEN);
}
```

Beispiel TicTacToe / OO-Implementierung (8)

```
void gebeSiegerBekannt() {
  int sieger = this.ermittleSieger();
  if (sieger == SIEGER_A) IO.println("Sieger ist Spieler A!");
  else if (sieger == SIEGER_B) IO.println("Sieger ist Spieler B!");
  else IO.println("Unentschieden!");
}

int ermittleSieger() {
   if (horizontale(SPIELER_A) ||
       vertikale(SPIELER_A) ||
       diagonale(SPIELER_A)) {
    return SIEGER_A; // Sieger ist Spieler A
  }

  if (horizontale(SPIELER_B) ||
      vertikale(SPIELER_B) ||
      diagonale(SPIELER_B)) {
    return SIEGER_B; // Sieger ist Spieler B
  }
  return KEIN_SIEGER; // noch kein Sieger
}
```

Beispiel TicTacToe / OO-Implementierung (9)

```
boolean vertikale(boolean istA) {
  for (int r=0; r<TTTBrett.getGroesse(); r++) {
    int anzahl = 0;
    for (int s=0; s<TTTBrett.getGroesse(); s++) {
      if (this.brett.liefereFigur(r, s) != TTTBrett.LEER &&
          this.brett.liefereFigur(r, s).istAFigur() == istA) {
        anzahl++;
      }
    }
    if (anzahl == TTTBrett.getGroesse()) {
      return true; // Sieger !
    }
  }
  return false;
}
```

Beispiel TicTacToe / OO-Implementierung (10)

```
boolean horizontale(boolean istA) {
  for (int s=0; s<TTTBrett.getGroesse(); s++) {
    int anzahl = 0;
    for (int r=0; r<TTTBrett.getGroesse(); r++) {
      if (this.brett.liefereFigur(r, s) != TTTBrett.LEER &&
          this.brett.liefereFigur(r, s).istAFigur() == istA) {
        anzahl++;
      }
    }
    if (anzahl == TTTBrett.getGroesse()) {
      return true; // Sieger !
    }
  }
  return false;
}
```

Beispiel TicTacToe / OO-Implementierung (11)

```
  boolean diagonale(boolean istA) {
    int anzahl = 0;
    for (int f=0; f<TTTBrett.getGroesse(); f++)
      if (this.brett.liefereFigur(f, f) != TTTBrett.LEER &&
          this.brett.liefereFigur(f, f).istAFigur() == istA)
        anzahl++;
    if (anzahl == TTTBrett.getGroesse()) return true; // Sieger !
    anzahl = 0;
    for (int f=0; f<TTTBrett.getGroesse(); f++) {
      if (this.brett.liefereFigur(f, TTTBrett.getGroesse()-1-f) !=
          TTTBrett.LEER &&
          this.brett.liefereFigur(f,
            TTTBrett.getGroesse()-1-f).istAFigur() == istA)
        anzahl++;
    if (anzahl == TTTBrett.getGroesse()) return true; // Sieger !
    return false;
  }
}
```

Beispiel TicTacToe / OO-Implementierung (12)

```
class TTTSpiel {

  // hier startet das TicTacToe-Programm
  public static void main(String[] args) {
    TTTSpieler a = new TTTSpieler(true);
    TTTSpieler b = new TTTSpieler(false);
    TTTBrett brett = new TTTBrett();
    TTTRegeln regeln = new TTTRegeln(brett);
    TTTSpiel tictactoe = new TTTSpiel(a, b, brett, regeln);

    tictactoe.spielen();
  }
```

Beispiel TicTacToe / OO-Implementierung (13)

```
// 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;
```

Beispiel TicTacToe / OO-Implementierung (14)

```
    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
      TTTSpielzug zug = aktSpieler.liefereNaechstenSpielzug();
      while (!this.regeln.spielzugOK(zug)) {
        IO.println("Ungueltig; neue Eingabe!");
        zug = aktSpieler.liefereNaechstenSpielzug();
      }
      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 Prüfung

Es soll eine Prüfung durchgeführt werden.
Die Prüfung besteht aus einem Quiz, das eine Menge an Prüflingen zu absolvieren haben. Ein Quiz besteht aus einer Menge an Fragen. Bei den Fragen handelt es sich ausschließlich um Wahr/Falsch-Aussagen, d.h. ein Prüfling muss jeweils angeben, ob die Aussage wahr oder falsch ist. Bei einer richtigen Antwort bekommt er eine der Frage zugeordneten Punktzahl.

Eine Prüfung läuft so ab, dass zunächst das Quiz vorbereitet wird, d.h. es werden eine Menge an Fragen eingeben.
Anschließend wird die Prüfung durchgeführt, d.h. die Prüflinge müssen der Reihe nach die Fragen beantworten.
Danach wird eine Rangliste nach den erreichten Punkten erzeugt und die Ergebnisse der Prüfung werden ausgegeben.

Demo

Beispiel Prüfung / OO-Analyse (1)

- Identifikation von Objekten/Klassen:
  - Fragen
  - Quiz
  - Prüfling
  - Prüfung

- Identifikation von Beziehungen zwischen Objekten:
  - Ein Quiz besteht aus mehreren Fragen
  - Eine Prüfung besteht aus einem Quiz, an dem mehrere Prüflinge teilnehmen

- Identifikation von Eigenschaften und Funktionen von Objekten
  - Frage:
    - Fragetext
    - Wahr/falsch
    - Erzielbare Punkte

Beispiel Prüfung / OO-Analyse (2)

- Quiz:
  - Titel
  - Menge an Fragen
  - Frage hinzufügen
  - Frage stellen

- Prüfling:
  - Name
  - Erreichte Punkte
  - Weitere Punkte erzielt

- Prüfung:
  - Quiz
  - Menge an Prüflingen
  - Vorbereiten
  - Durchführen
  - Ergebnisse bekannt geben

Beispiel Prüfung / OO-Implementierung (1)

```
class Frage {
  String frage;     // Fragetext
  boolean richtig;  // richtig oder falsch
  int punkte;       // zu erreichende Punktzahl

  // Konstruktor
  Frage(String frage, boolean richtig, int punkte) {
    this.frage = frage;
    this.richtig = richtig;
    this.punkte = punkte;
  }

  // set/get-Methoden
  String getFrage() { return this.frage; }
  boolean isRichtig() { return this.richtig; }
  int getPunkte() { return this.punkte; }
}
```

Beispiel Prüfung / OO-Implementierung (2)

```
class Quiz {
  String titel;       // Titel des Quizes
  Frage[] fragen;     // Menge an Fragen
  int aktuellerIndex; // aktueller Index in fragen
  int naechsterIndex; // Hilfsvariable

  Quiz(String titel, int maxFragen) {
    this.titel = titel;
    this.fragen = new Frage[maxFragen];
    for (int i=0; i<this.fragen.length; i++) this.fragen[i] = null;
    this.aktuellerIndex = -1; this.naechsterIndex = -1;
  }

  String getTitel() { return this.titel; }

  void neueFrage(Frage f) {
    if (this.aktuellerIndex < this.fragen.length-1) {
      this.fragen[++this.aktuellerIndex] = f;
    }
  }
```

Beispiel Prüfung / OO-Implementierung (3)

```
  // liefert zyklisch die naechste Frage oder null, falls keine
  // (mehr) vorhanden ist
  Frage liefereNaechsteFrage() {
    if (this.fragen.length == 0) return null;
    Frage f = null;
    if (this.naechsterIndex < this.aktuellerIndex) {
      f = this.fragen[++this.naechsterIndex];
    } else {
      this.naechsterIndex = -1;  // bereite naechsten Zyklus vor
    }
    return f;
  }
}
```

Beispiel Prüfung / OO-Implementierung (4)

```
class Pruefling {
  String name;  // Name der Person
  int punkte;   // bisher erzielte Punkte

  Pruefling(String name) {
    this.name = name;
    this.punkte = 0;
  }

  String getName() { return this.name; }
  int getPunkte() { return this.punkte; }

  void neuePunkte(int anzahl) {
    this.punkte += anzahl;
  }
}
```

Beispiel Prüfung / OO-Implementierung (5)

```
class Pruefung {
  // Hauptprogramm
  public static void main(String[] args) {
    Pruefung klausur = new Pruefung();
    klausur.vorbereiten();
    klausur.durchfuehren();
    klausur.ergebnisseBekanntgeben();
  }

  Quiz pruefung;           // das Quiz
  Pruefling[] studenten;  // Menge an Prueflingen

  Pruefung(Quiz quiz, Pruefling[] prueflinge) {
    this.pruefung = quiz;
    this.studenten = prueflinge;
  }

  Pruefung() {
    this.pruefung = null;
    this.studenten = null;
  }
}
```

Beispiel Prüfung / OO-Implementierung (6)

```
void vorbereiten() {
  IO.println("Fragen eingeben");
  IO.println("---------------");
  String titel = IO.readString("Titel des Quizes: ");
  int anzahl = IO.readInt("Anzahl Fragen: ");
  this.pruefung = new Quiz(titel, anzahl);
  for (int i=0; i<anzahl; i++) {
    String text = IO.readString("Frage " + (i+1) + ": ");
    boolean wahr = IO.readChar("wahr/falsch (w/f)?") == 'w';
    int punkte = IO.readInt("Erreichbare Punkte: ");
    Frage f = new Frage(text, wahr, punkte);
    this.pruefung.neueFrage(f);
  }
  IO.println("Prueflinge eingeben");
  IO.println("------------------");
  anzahl = IO.readInt("Anzahl Prueflinge: ");
  this.studenten = new Pruefling[anzahl];
  for (int i=0; i<anzahl; i++) {
    this.studenten[i] =
      new Pruefling(IO.readString("Pruefling " + (i+1) + ": "));
  }
}
```

Beispiel Prüfung / OO-Implementierung (7)

```
void durchfuehren() {
  IO.println("Pruefung");
  IO.println("--------");
  // alle Prueflinge abfragen
  for (int i=0; i<this.studenten.length; i++) {
    IO.println("\"" + this.studenten[i].getName() +
               "\" ist an der Reihe");
    Frage f = null;
    // alle Fragen der Pruefung stellen
    while ((f = this.pruefung.liefereNaechsteFrage()) != null) {
      IO.println(f.getFrage());
      boolean antwort = IO.readChar("wahr/falsch (w/f)?") == 'w';
      if (f.isRichtig() == antwort) {
        IO.println("Hervorragend: Richtig beantwortet");
        this.studenten[i].neuePunkte(f.getPunkte());
      } else {
        IO.println("Niete: Falsch beantwortet");
} } } }
```

Beispiel Prüfung / OO-Implementierung (8)

```
void ergebnisseBekanntgeben() {

  this.ranglisteErstellen();

  IO.println("Pruefungsergebnisse");
  IO.println("--------------------");
  IO.println("Quiz: " + this.pruefung.getTitel());
  for (int i=0; i<this.studenten.length; i++) {
    IO.println("Platz " + (i+1) + ": " +
               this.studenten[i].getName() + " mit " +
               this.studenten[i].getPunkte() + " Punkten");
  }
}
```

Beispiel Prüfung / OO-Implementierung (9)

```
  void ranglisteErstellen() {
    // Bubblesort nach erreichten Punkten
    boolean veraendert = false;
    do {
      veraendert = false;
      for (int i=0; i<this.studenten.length-1; i++) {
        if (this.studenten[i].getPunkte() <
            this.studenten[i+1].getPunkte()) {
          Pruefling help = this.studenten[i];
          this.studenten[i] = this.studenten[i+1];
          this.studenten[i+1] = help;
          veraendert = true;
        }
      }
    } while (veraendert);
  }
}
```

Zusammenfassung

- Objektorientierte Softwareentwicklung: alle Entwicklungsphasen nutzen objektorientierte Konzepte
- OO-Analyse: Erstellung eines OOA-Modells als Abbild des Anwendungsgebietes
- OO-Entwurf: Abbildung des OOA-Modells auf ein Modell des zu entwickelnden Systems (OOD-Modell)
- OO-Implementierung: Umsetzung des OOD-Modells in eine konkrete Programmiersprache
- OO-Test: Test der Methoden, Klassen und des Zusammenspiels der Klassen