

# Kapitel 15

---

# Systemarchitektur

## Ziele

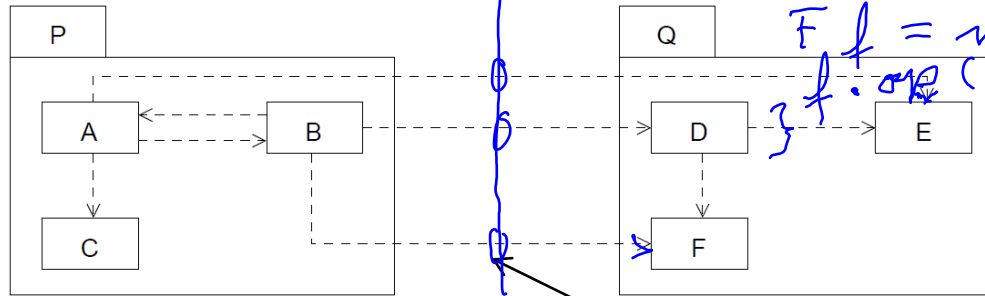
- Grundprinzipien der Systemarchitektur verstehen
- Schichtenarchitekturen kennenlernen
- Modelle und Programme mit Paketen strukturieren  
*UML* *java*
- Eine Architektur für eine einfache Bankanwendung konstruieren

## Grundprinzipien der Systemarchitektur

- Große Softwaresysteme bestehen aus mehr als 50.000 Lines of Code (LoC), (mehr als 10 Personenjahre (PJ) für die Entwicklung).
- Bei mehr als 1 Million LoC spricht man von sehr großen Softwaresystemen, z.B.
  - Windows 95: 10 Mio LoC; Windows XP: 40 Mio LoC; Mac OS X, 2004: 86 Mio LoC
  - SAP NetWeaver (2007): 238 Mio LoC  
(Plattform für Geschäftsanwendungen mit zahlreichen Komponenten)
- Große Softwaresysteme bestehen aus vielen Teilsystemen, Modulen, Komponenten.
- Die Systemarchitektur beschreibt die Struktur eines Softwaresystems durch Angabe seiner Teile und deren Verbindungen (häufig über Schnittstellen).
- Grundregeln bei der Erstellung von Systemarchitekturen sind:
  - *Hohe Kohärenz* (high cohesion):  
Zusammenfassung logisch zusammengehörender Elemente in einem Teilsystem.
  - *Geringe Kopplung* (low coupling):  
Möglichst wenige Abhängigkeiten zwischen den einzelnen Teilsystemen.
  - Vorteil: leichtere Änderbarkeit und Austauschbarkeit von einzelnen Teilen.

# Hohe und geringe Kopplung

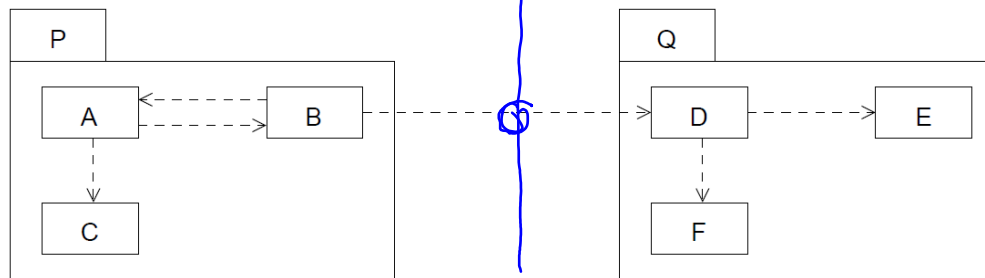
## Teilsysteme mit hoher Kopplung



```
class D {  
    void m() {  
        F f = new F();  
        f.m(...);  
    }  
}
```



## Teilsysteme mit geringer Kopplung

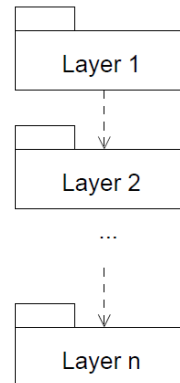


Abhängigkeitsbeziehung:  
„B hängt ab von F“

Wird an einem Teil T etwas geändert, so müssen alle anderen Teile, die eine Abhängigkeitsbeziehung hin zu T haben, auf etwaige nötige Änderungen überprüft werden.

## Schichtenarchitekturen

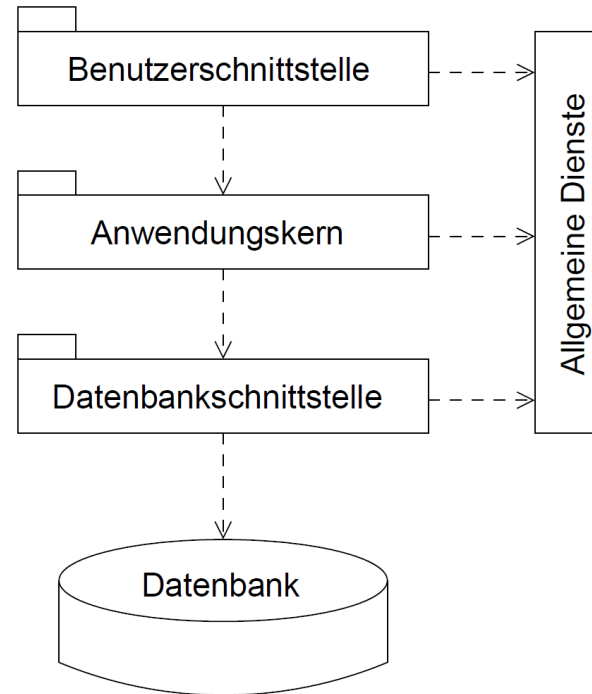
- In vielen Systemen findet man **Schichtenarchitekturen**, wobei jede untere Schicht Dienste für die darüberliegende(n) Schicht(en) bereitstellt, z.B. OSI-Schichtenmodell für Netzwerkprotokolle (7 Schichten), Betriebssystemschichten, 3-Ebenen-Datenbankarchitektur.



*konzept  
interne Schicht*

- Bei **geschlossenen Architekturen** darf eine Schicht nur auf die direkt darunterliegende Schicht zugreifen; ansonsten spricht man von **offenen Architekturen**.
- Sind verschiedene Schichten auf verschiedene Rechner verteilt, dann spricht man von **Client/Server-Systemen**.
- Eine Schicht kann selbst wieder aus verschiedenen Teilsystemen bestehen.

## Drei-Schichten-Architektur für betriebliche Informationssysteme



Bei Client/Server-Architekturen (z.B. Web-Anwendungen) spricht man

- von einem „Thick-Client“, wenn Benutzerschnittstelle und Anwendungskern auf demselben Rechner ausgeführt werden,
- von einem „Thin-Client“, wenn Benutzerschnittstelle und Anwendungskern auf verschiedene Rechner verteilt sind.

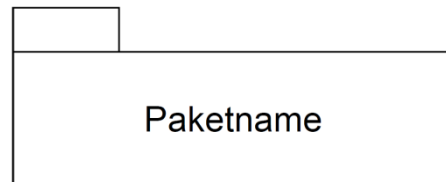
## Die Schichten im Einzelnen

- Benutzerschnittstelle
  - Behandlung von Terminalereignissen (Maus-Klick, Key-Strike, ...)
  - Ein-/Ausgabe von Daten
  - Dialogkontrolle
- Anwendungskern (Fachkonzept)
  - Zuständig für die Anwendungslogik (die eigentlichen Aufgaben des Problembereichs)
- Datenbank-Schnittstelle
  - Sorgt für die Speicherung von und den Zugriff auf persistente Daten der Anwendung.
- Allgemeine Dienste
  - z.B. Kommunikationsdienste, Dateiverwaltung, Bibliotheken (APIs, GUI, math. Funktionen, ...)

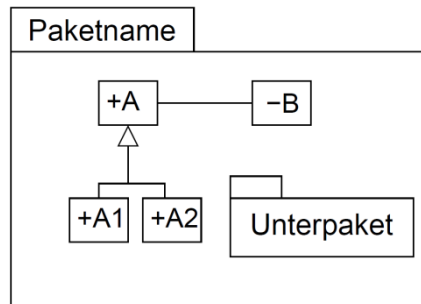
## Pakete

Pakete dienen zur Gruppierung von Elementen größerer Systeme. Sie können sowohl in der Modellierung (UML) als auch in der Programmierung (Java) verwendet werden.

- Paket ohne Anzeigen der Inhalte (UML-Notation):



- Paket mit Anzeigen der Inhalte (UML-Notation):



### Beachte:

- Klassen in Paketen sind öffentlich oder nur im selben Paket sichtbar.
- Pakete können Unterpakete enthalten.

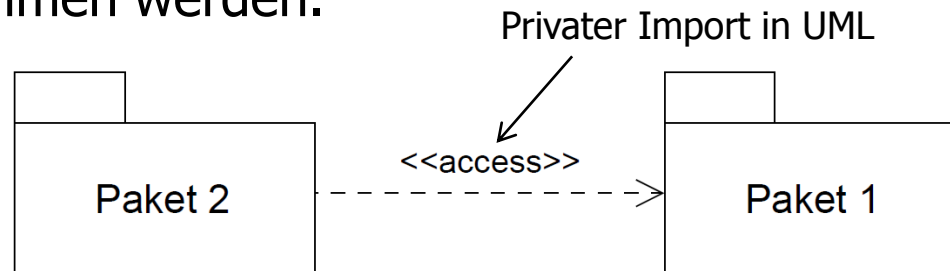
Die öffentlichen Elemente eines Pakets sind außerhalb des Pakets (immer) zugreifbar unter Verwendung ihres qualifizierten Namens,

z.B. **Paketname::A** in UML, **Paketname.A** in Java.

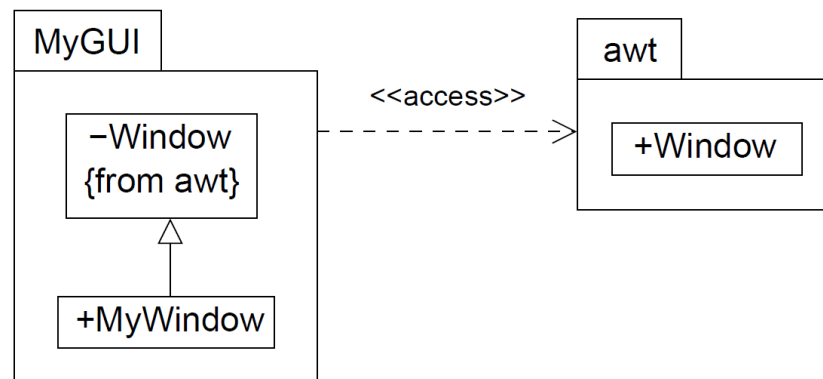


## Importieren von Paketen

- Durch **Importieren** können die Namen von öffentlichen Elementen eines (importierten) Pakets in den Namensraum eines anderen (importierenden) Pakets übernommen werden.

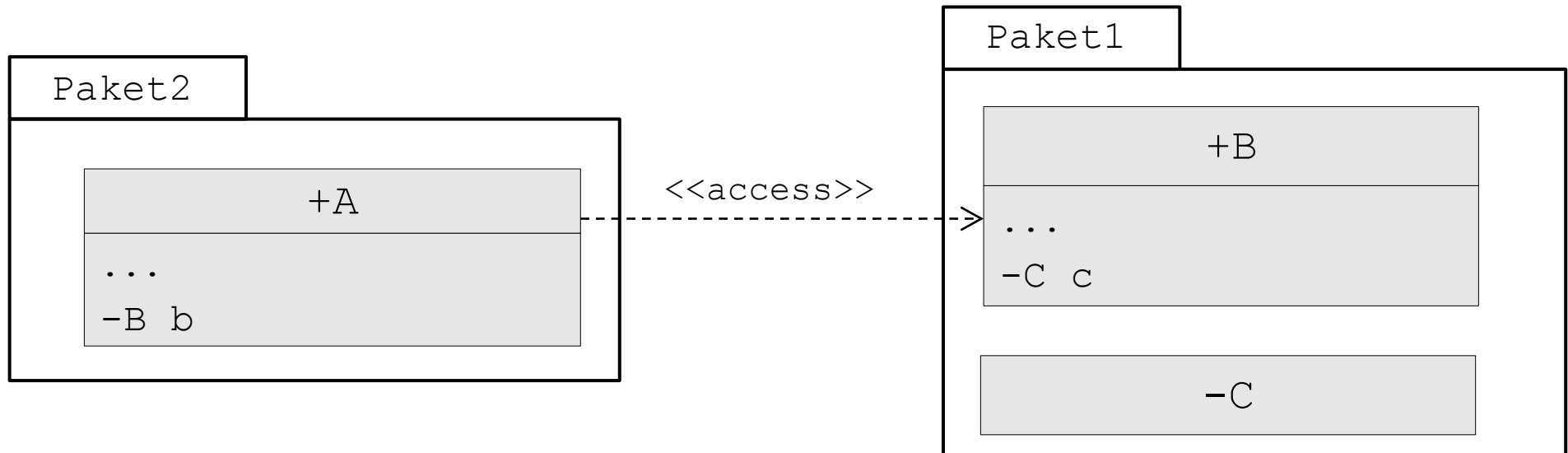


- Privater Import: Importierte Elemente können nicht weitergegeben werden. Ihre Sichtbarkeit wird im importierenden Paket auf „privat“ gesetzt.



## Importieren von Modellelementen

Klassen (und Interfaces) können auch einzeln aus anderen Paketen importiert werden.

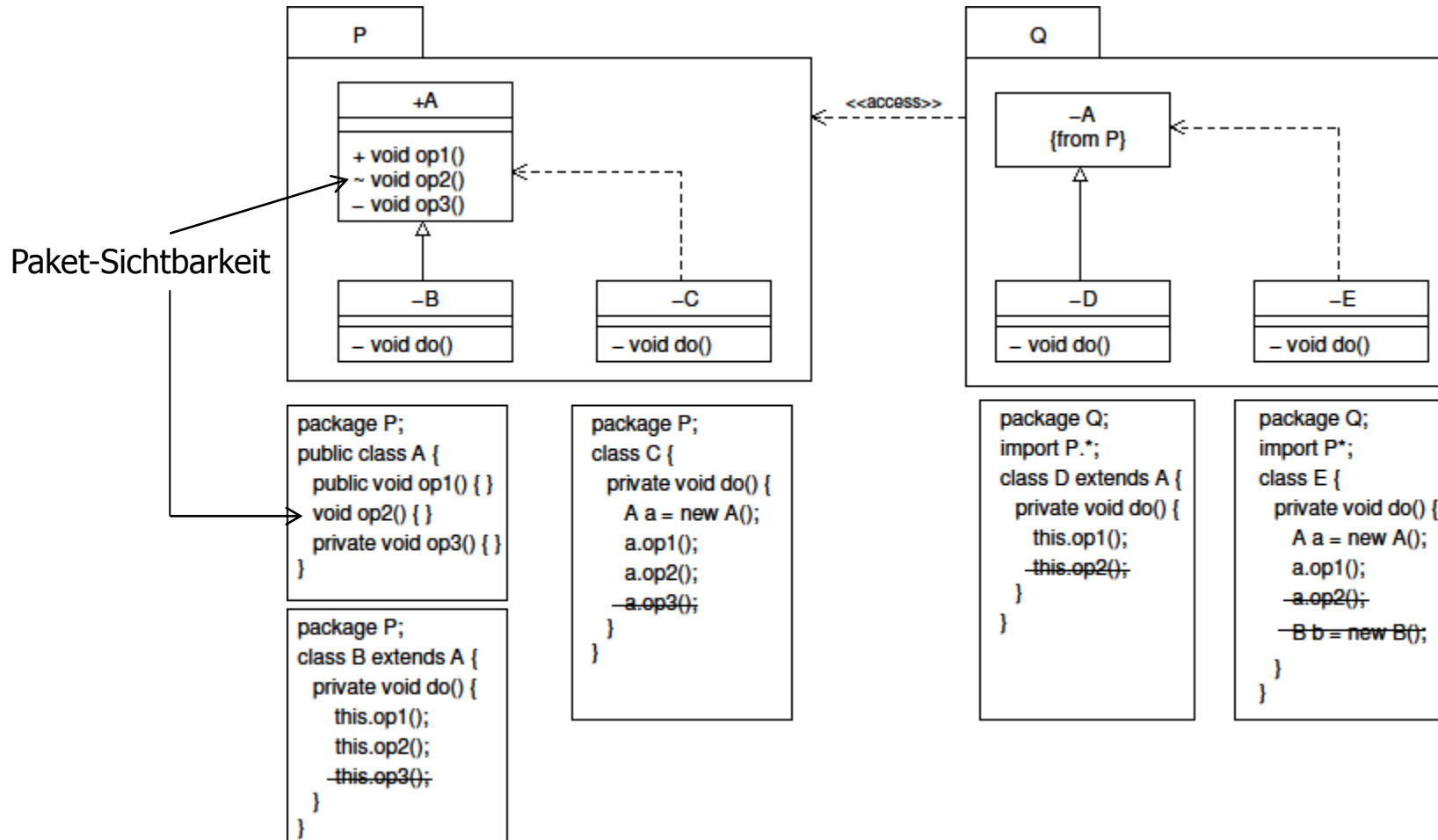


Die öffentliche Klasse B (und ihre öffentlichen Elemente) sind außerhalb des Pakets1 zugreifbar, und zwar direkt mit ihrem Namen (ohne Qualifizierung), wenn die Klasse importiert wird.

## Pakete in Java

- Für jedes Paket wird ein Verzeichnis mit dem Paketnamen erstellt; für Unterpakete werden Unterverzeichnisse eingerichtet.
- Eine Klasse **K**, die zu einem Paket **P** gehört, wird in einer Datei **K.java** in dem Verzeichnis **P** implementiert und abgespeichert.
- Die Datei darf höchstens eine **public class** enthalten.
- Zu Beginn muss in der Datei **K.java** der Paketname angegeben werden, zu dem die Klasse gehört:  
**package P;** bzw.  
**package P.U;** falls die Klasse zu einem Unterverzeichnis **U** von **P** gehört.
- Pakete werden in Java importiert durch:  
**import P.\*;** bzw. **import P.U.\*;**
- Klassen (und analog Interfaces) werden in Java importiert durch:  
**import P.Klassenname;** bzw. **import P.U.Klassenname;**

# Pakete in UML und Java: Beispiel

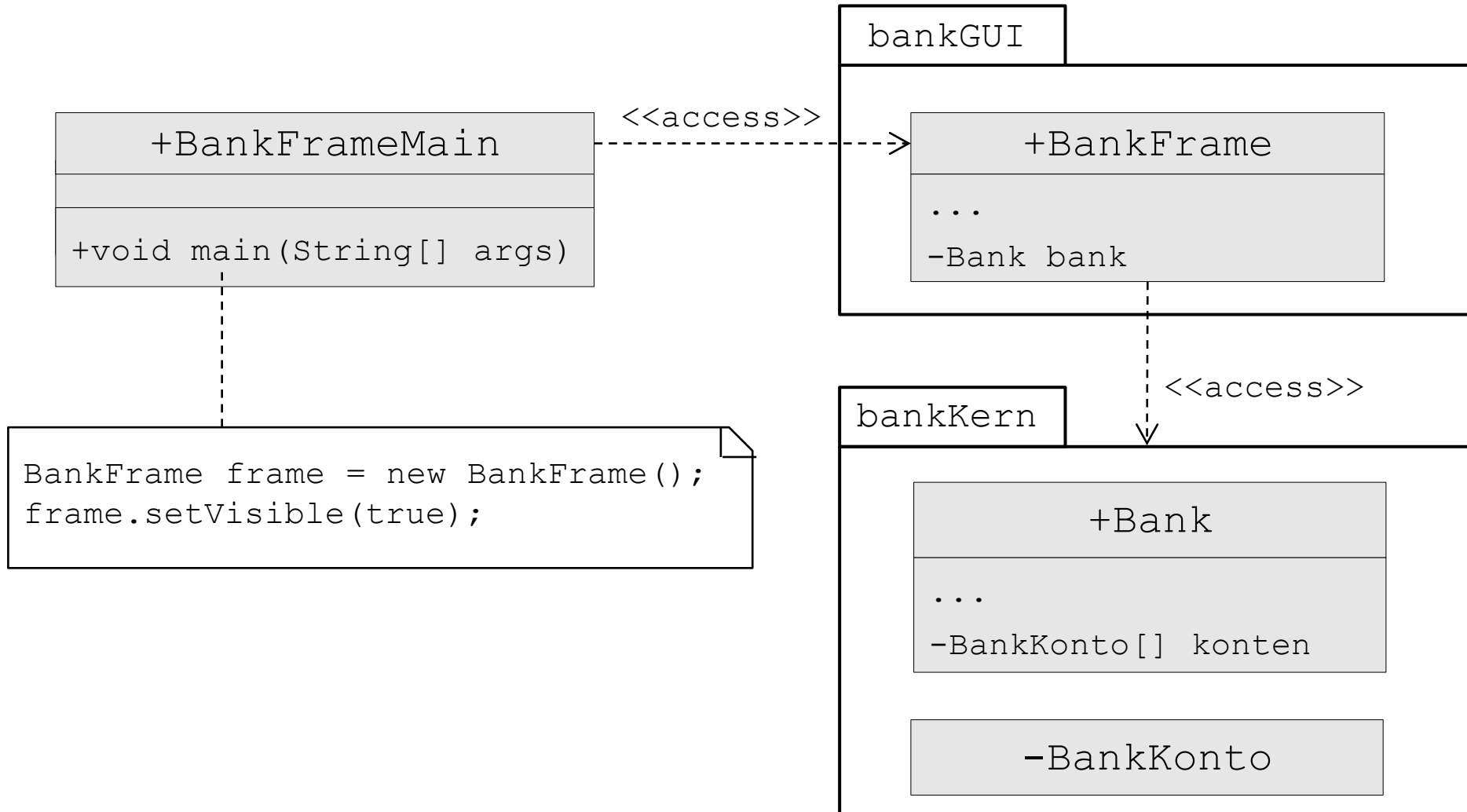


## Beispiel: Einfaches Banksystem



vgl. Zentralübung 10

## 2-Schichtenarchitektur des Banksystems



## Verzeichnisstruktur und Dateien

➤ Hauptverzeichnis enthält:

➤ Datei BankFrameMain.java

```
import bankGUI.BankFrame;  
public class BankFrameMain { ... }
```

➤ Unterverzeichnis bankGUI

➤ Datei BankFrame.java

```
package bankGUI;  
import java.awt.*;  
import javax.swing.*;  
import bankKern.*;  
  
public class BankFrame  
    extends JFrame  
    implements ActionListener  
    { ... }
```

➤ Unterverzeichnis bankKern

➤ Datei Bank.java

```
package bankKern;  
public class Bank { ... }
```

➤ Datei BankKonto.java

```
package bankKern;  
class BankKonto { ... }
```

## Java-Klasse BankFrameMain

```
import bankGUI.BankFrame;
```

```
public class BankFrameMain {
```

```
    /**
```

```
     * Dieses Programmstück startet das Programm.
```

```
     *
```

```
     * @param args
```

```
     */
```

```
    public static void main(String[] args) {
```

```
        BankFrame frame = new BankFrame();
```

```
        frame.setVisible(true);
```

```
    }
```

```
}
```



## Klasse BankFrame

+BankFrame
<ul style="list-style-type: none"><li>- JButton bankEroeffnenButton</li><li>- JButton kontoEroeffnenButton</li><li>- JButton einzahlenButton</li><li>...</li><li>- JTextArea ausgabeBereich</li><li>- Bank bank</li></ul>
<ul style="list-style-type: none"><li>+ BankFrame()</li><li>+ void actionPerformed(ActionEvent e)</li><li>- void bankEroeffnen()</li><li>- void kontoEroeffnen()</li><li>- void einzahlen()</li><li>- void abheben()</li><li>- void kontostandBerechnen()</li><li>- void gesamtSaldoBerechnen()</li></ul>

## Java-Klasse BankFrame (1)

```
package bankGUI;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextArea;

import bankKern.*;
/**
 * Eine grafische Benutzeroberfläche zur Verwaltung von Konten.
 */
public class BankFrame extends JFrame implements ActionListener {

    private JButton bankEroeffnenButton;
    private JButton kontoEroeffnenButton;
    private JButton einzahlenButton;
    private JButton abhebenButton;
    private JButton kontostandButton;
    private JButton gesamtSaldoButton;

    private JTextArea ausgabeBereich;

    private Bank bank;
```

Code von Zentralübung 10,  
hier angepasst an Schichtenarchitektur

## Java-Klasse BankFrame (2)

```
/**
 * In diesem Programmstück wird das Fenster erzeugt.
 */
public BankFrame() {
    this.setTitle("BankFrame");
    this.setSize(700, 350);

    /* Hier werden alle Buttons erzeugt. */
    this.bankEroeffnenButton =
        new JButton("Bank eröffnen");
    this.kontoEroeffnenButton =
        new JButton("Konto eröffnen");
    this.einzahlenButton = new JButton("Einzahlen");
    this.abhebenButton = new JButton("Abheben");
    this.kontostandButton = new JButton(
        "Kontostand eines Kontos ausgeben");
    this.gesamtSaldoButton = new JButton(
        "Gesamtsaldo aller Konten");

    /* Hier wird der Ausgabe-Bereich erzeugt. */
    this.ausgabeBereich = new JTextArea(10, 100);
```

```
/* Hier werden alle Buttons zusammengruppiert. */
JPanel kontenPanel = new JPanel();
kontenPanel.setLayout(new GridLayout(5, 1));
kontenPanel.add(this.kontoEroeffnenButton);
kontenPanel.add(this.einzahlenButton);
kontenPanel.add(this.abhebenButton);
kontenPanel.add(this.kontostandButton);
kontenPanel.add(this.gesamtSaldoButton);

JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(1, 2));
buttonPanel.add(this.bankEroeffnenButton);
buttonPanel.add(kontenPanel);

Container contentPane = this.getContentPane();
contentPane.setLayout(new GridLayout(2, 1));
/* Hier wird die Gruppe von Buttons platziert. */
contentPane.add(buttonPanel);
/* Hier wird der Ausgabebereich platziert. */
contentPane.add(this.ausgabeBereich);
```

## Java-Klasse BankFrame (3)

```
/**
 * Hier wird der Frame als Listener für
 * Knopfdruck Ereignisse bei jedem Button
 * registriert.
 */
this.bankEroeffnenButton.addActionListener(this);
this.kontoEroeffnenButton.addActionListener(this);
this.einzahlenButton.addActionListener(this);
this.abhebenButton.addActionListener(this);
this.kontostandButton.addActionListener(this);
this.gesamtSaldoButton.addActionListener(this);

this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

```
/**
 * Dieses Programmstück wird immer dann ausgeführt,
 * wenn ein Benutzer auf einen Button drückt.
 */
@Override
public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source == this.bankEroeffnenButton) {
        this.bankEroeffnen();
    }
    else if (source == this.kontoEroeffnenButton) {
        this.kontoEroeffnen();
    }
    else if (source == this.einzahlenButton) {
        this.einzahlen();
    }
    else if (source == this.abhebenButton) {
        this.abheben();
    }
    else if (source == this.kontostandButton) {
        this.kontostandBerechnen();
    }
    else if (source == this.gesamtSaldoButton) {
        this.gesamtSaldoBerechnen();
    }
}
```

## Java-Klasse BankFrame (4)

```

/**
 * Diese Methode erzeugt eine neue Bank, falls es
 * momentan keine gibt.
 */
private void bankEroeffnen() {
    if (this.bank != null) {
        this.ausgabeBereich
            .setText("Es wurde bereits eine Bank mit
dem Namen „ + this.bank.getName() + " eröffnet.");
    }
    else {
        String einlesenName = JOptionPane
            .showInputDialog("Name der Bank: ");
        String einlesenMaxKonten = JOptionPane
            .showInputDialog("Maximale Anzahl an Konten: ");
        int maxKonten = Integer.parseInt(einlesenMaxKonten);
        this.bank = new Bank(einlesenName, maxKonten);
        this.ausgabeBereich
            .setText("Folgende Bank wurde eröffnet: "
+ this.bank + " mit dem Namen "+ this.bank.getName());
    }
}

```

```

/**
 * Diese Methode eröffnet ein neues Konto, falls die Bank
 * noch Platz für ein neues Konto hat und es momentan
 * eine Bank gibt.
 */
private void kontoEroeffnen() {
    if (this.bank == null) {
        this.ausgabeBereich
            .setText("Es wurde noch keine Bank eröffnet.");
    }
    else {
        String einlesenKontoNr = JOptionPane
            .showInputDialog("Kontonummer: ");
        int kontoNummer = Integer.parseInt(einlesenKontoNr);

        String einlesenAnfangsbetrag = JOptionPane
            .showInputDialog("Anfangsbetrag: ");
        double anfangsbetrag = Double
            .parseDouble(einlesenAnfangsbetrag);

        boolean eroeffnet = this.bank.kontoEroeffnen(kontoNummer,
            anfangsbetrag);
        if (eroeffnet) {
            this.ausgabeBereich
                .setText("Das Konto mit der Kontonummer " + kontoNummer
+ " und dem Anfangsbetrag „ + anfangsbetrag + " wurde eröffnet.");
        }
        else {
            this.ausgabeBereich
                .setText("Es kann kein neues Konto angelegt werden,"
+ "da diese Bank nicht mehr Konten anbietet.");
        }
    }
}
}

```

## Java-Klasse BankFrame (5)

```
/**
 * Diese Methode zahlt einen gegebenen Betrag auf ein Konto ein, falls
 * dieses existiert.
 */
private void einzahlen() {
    if (this.bank == null) {
        this.ausgabeBereich
            .setText("Es wurde noch keine Bank eröffnet.");
    }
    else {
        String einlesenKontoNr = JOptionPane
            .showInputDialog("Kontonummer: ");
        int kontoNummer = Integer.parseInt(einlesenKontoNr);

        String einlesenBetrag = JOptionPane
            .showInputDialog("Einzuzahlender Betrag: ");
        double betrag = Double.parseDouble(einlesenBetrag);

        boolean eingezahlt = this.bank.einzahlen(kontoNummer,
            betrag);

        if (eingezahlt) {
            this.ausgabeBereich.setText("Es wurden " + betrag
                + " auf das Konto mit der Kontonummer "
                + kontoNummer + " eingezahlt.");
        }
        else {
            this.ausgabeBereich
                .setText("Der Betrag konnte nicht eingezahlt werden, "
                    + "da kein Konto mit der angegebenen Kontonummer existiert.");
        }
    }
}
```

```
/**
 * Diese Methode hebt einen gegebenen Betrag von einem Konto ab, falls
 * dieses existiert.
 */
private void abheben() {
    if (this.bank == null) {
        this.ausgabeBereich
            .setText("Es wurde noch keine Bank eröffnet.");
    }
    else {
        String einlesenKontoNr = JOptionPane
            .showInputDialog("Kontonummer: ");
        int kontoNummer = Integer.parseInt(einlesenKontoNr);

        String einlesenBetrag = JOptionPane
            .showInputDialog("Auszuzahlender Betrag: ");
        double betrag = Double.parseDouble(einlesenBetrag);

        boolean abgehoben = this.bank
            .abheben(kontoNummer, betrag);

        if (abgehoben) {
            this.ausgabeBereich.setText("Es wurden " + betrag
                + " von dem Konto mit der Kontonummer "
                + kontoNummer + " abgehoben.");
        }
        else {
            this.ausgabeBereich
                .setText("Der Betrag konnte nicht abgehoben werden, "
                    + "da kein Konto mit der angegebenen Kontonummer existiert.");
        }
    }
}
```

## Java-Klasse BankFrame (6)

```
/**
 * Diese Methode gibt den Kontostand für ein Konto aus.
 */
private void kontostandBerechnen() {
    if (this.bank == null) {
        this.ausgabeBereich
            .setText("Es wurde noch keine Bank eröffnet.");
    }
    else {
        String einlesenKontoNr = JOptionPane
            .showInputDialog("Kontonummer: ");
        int kontoNummer = Integer.parseInt(einlesenKontoNr);

        double kontoStand = this.bank.kontoStand(kontoNummer);

        this.ausgabeBereich
            .setText("Der Kontostand des Kontos mit der
Kontonummer " + kontoNummer + " ist" + kontoStand);
    }
}
```

```
/**
 * Diese Methode berechnet den Saldo aller Konten.
 */
private void gesamtSaldoBerechnen() {
    if (this.bank == null) {
        this.ausgabeBereich
            .setText("Es wurde noch keine Bank eröffnet.");
    }
    else {
        double gesamtSaldo = this.bank.gesamtSaldo();

        this.ausgabeBereich
            .setText("Das Gesamtsaldo aller Konten dieser Bank ist "
+ gesamtSaldo);
    }
}
```

## Klasse Bank

+Bank
-String name -BankKonto[] konten -int anzahlEroeffneterKonten
+Bank(String name, int maxAnzahlKonten) +String getName() +boolean kontoEroeffnen(int kontoNummer, double anfangsBetrag) -BankKonto sucheBankkonto(int kontoNummer) +boolean einzahlen(int kontoNummer, double betrag) +boolean abheben(int kontoNummer, double betrag) +double kontoStand(int kontoNummer) +double gesamtSaldo()



## Java-Klasse Bank (1)

```
package bankKern;
```

```
/**  
 *  
 * Repräsentation einer Bank mit einem Namen und einer Liste  
 * von eröffneten Konten.  
 */
```

```
public class Bank {
```

```
    private String name;  
    private BankKonto[] konten;  
    private int anzahlEroeffneterKonten;
```

```
/**  
 * Konstruktor  
 *  
 * @param name  
 * @param maxAnzahlKonten  
 */
```

```
public Bank(String name, int maxAnzahlKonten) {  
    this.name = name;  
    this.konten = new BankKonto[maxAnzahlKonten];  
    this.anzahlEroeffneterKonten = 0;  
}
```

```
/**  
 * Diese Methode liefert den Namen der Bank  
 *  
 * @return  
 */
```

```
public String getName() {  
    return this.name;  
}
```

```
/**  
 * Diese Methode eröffnet ein Konto mit der gegebenen  
 * Kontonummer und dem gegebenen Anfangsbetrag. Dazu wird  
 * zunächst ein neues Objekt der Klasse {@link BankKonto}  
 * erzeugt, dieses der Bank hinzugefügt und true  
 * zurückgegeben. Ist die Bank schon voll (d.h. wird die  
 * Maximalanzahl an Konten für diese Bank überschritten),  
 * wird das Konto nicht eröffnet und false zurückgegeben.  
 *  
 * @param kontoNummer  
 * @param anfangsBetrag  
 * @return false falls die Maximalanzahl an Konten  
 * überschritten wurde, true sonst  
 */
```

```
public boolean kontoEroeffnen(int kontoNummer,  
    double anfangsBetrag) {  
    if (this.anzahlEroeffneterKonten < this.konten.length) {  
        this.konten[this.anzahlEroeffneterKonten] =  
            new BankKonto(kontoNummer, anfangsBetrag);  
        this.anzahlEroeffneterKonten++;  
        return true;  
    }  
    return false;  
}
```

## Java-Klasse Bank (2)

```
/**
 * Diese Methode sucht in der Liste der Konten der Bank das Konto mit der
 * gegebenen Kontonummer. Wird ein Konto gefunden, wird dieses
 * zurückgegeben. Falls kein Konto mit dieser Kontonummer existiert, wird
 * null zurückgegeben.
 *
 * @param kontoNummer
 * @return das Objekt der Klasse {@link BankKonto} mit der gegebenen
 *         Kontonummer; null falls kein Konto mit dieser Kontonummer
 *         eröffnet wurde.
 */
private BankKonto sucheBankkonto(int kontoNummer) {
    for (int i = 0; i < this.anzahlEroeffneterKonten; i++) {
        BankKonto aktuellesKonto = this.konten[i];
        if (aktuellesKonto.getKontoNummer() == kontoNummer) {
            return aktuellesKonto;
        }
    }
    return null;
}
```

```
/**
 * Diese Methode zahlt den gegebenen Betrag auf das Konto mit der gegebenen
 * Kontonummer ein. Falls der Betrag eingezahlt werden konnte, wird true
 * zurückgegeben. Falls kein Konto mit dieser Kontonummer existiert, wird
 * false zurückgegeben.
 *
 * @param kontoNummer
 * @param betrag
 * @return false falls kein Konto mit dieser Kontonummer existiert, true
 *         sonst
 */
public boolean einzahlen(int kontoNummer, double betrag) {
    BankKonto aktuellesKonto = this.sucheBankkonto(kontoNummer);
    if (aktuellesKonto != null) {
        aktuellesKonto.einzahlen(betrag);
        return true;
    }
    else {
        return false;
    }
}
```

## Java-Klasse Bank (3)

```
/**
 * Diese Methode hebt den gegebenen Betrag vom Konto mit der gegebenen
 * Kontonummer ab. Falls der Betrag abgehoben werden konnte, wird true
 * zurückgegeben. Falls kein Konto mit dieser Kontonummer existiert, wird
 * false zurückgegeben.
 *
 * @param kontoNummer
 * @param betrag
 * @return false falls kein Konto mit dieser Kontonummer existiert,
 * true sonst
 */
public boolean abheben(int kontoNummer, double betrag) {
    BankKonto aktuellesKonto = this.sucheBankkonto(kontoNummer);
    if (aktuellesKonto != null) {
        aktuellesKonto.abheben(betrag);
        return true;
    }
    else {
        return false;
    }
}
```

```
/**
 * Diese Methode gibt den Kontostand des Kontos mit der gegebenen
 * Kontonummer aus. Falls kein Konto mit dieser Kontonummer existiert, wird
 * {@link Integer#MIN_VALUE} zurückgegeben.
 *
 * @param kontoNummer
 * @return der Kontostand des Kontos oder {@link Integer#MIN_VALUE}, falls
 * kein Konto mit der gegebenen Kontonummer existiert
 */
public double kontoStand(int kontoNummer) {
    BankKonto aktuellesKonto = this.sucheBankkonto(kontoNummer);
    if (aktuellesKonto != null) {
        return aktuellesKonto.getKontoStand();
    }
    else {
        return Integer.MIN_VALUE;
    }
}

/**
 * Diese Methode gibt dem Gesamtsaldo über alle Konten zurück.
 *
 * @return der Gesamtsaldo aller Konten dieser Bank
 */
public double gesamtSaldo() {
    double gesamtSaldo = 0.0;
    for (int i = 0; i < this.anzahlEroeffneterKonten; i++) {
        BankKonto aktuellesKonto = this.konten[i];
        gesamtSaldo = gesamtSaldo
            + aktuellesKonto.getKontoStand();
    }
    return gesamtSaldo;
}
```

## Klasse BankKonto mit Konstruktor und Methoden

-BankKonto
- int kontoNummer - double kontoStand
+ BankKonto(int kontoNummer, double anfangsBetrag) + int getKontoNummer() + double getKontoStand() + void einzahlen(double x) + void abheben(double x)

## Java-Klasse BankKonto

```
package bankKern;  
class BankKonto {  
  
    private int kontoNummer;  
    private double kontoStand;  
  
    public BankKonto(int kontoNummer, double anfangsBetrag) {  
        this.kontoNummer = kontoNummer;  
        this.kontoStand = anfangsBetrag;  
    }  
    public int getKontoNummer() {  
        return this.kontoNummer;  
    }  
    public double getKontoStand() {  
        return this.kontoStand;  
    }  
    public void einzahlen(double x) {  
        this.kontoStand = this.kontoStand + x;  
    }  
    public void abheben(double x) {  
        this.kontoStand = this.kontoStand - x;  
    }  
}
```

## Zentralübung heute:

- Inhaltsverzeichnis der Vorlesung
- Zusammenfassung des Stoffes
- Beantwortung von Fragen