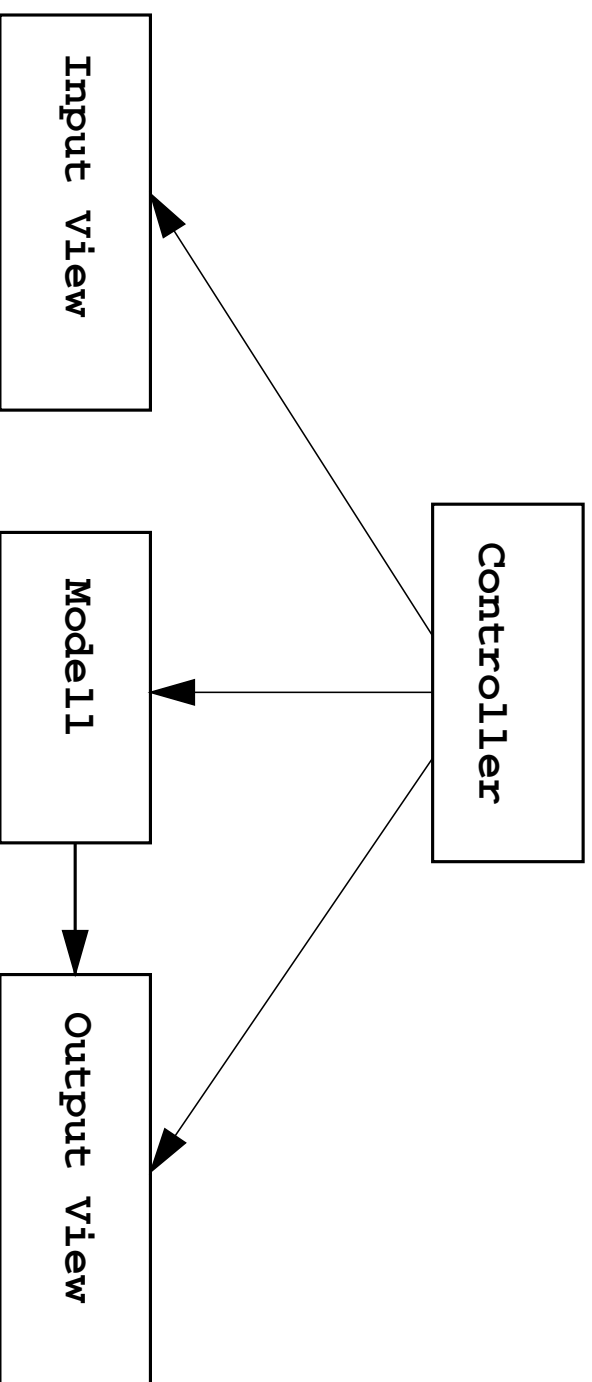


13.7 Modelle

- **Modellobjekte in der Architektur**
- **logische Operatoren**
- **bedingte Anweisungen**

13.7.1 Modellobjekte



Variation einer Standard-Architektur: *Model-View-Controller*

Gründe für diese Aufteilung

- Wiederverwendung von Klassen
 - Standardklassen für Input und Output Views
 - Vorhandene Klassen für Modell (ggf. abändern)
 - Neuer Controller
- kleine (Standard-) Komponenten mit festgelegten Pflichten \Rightarrow besseres Verständnis
- Änderung einfacher

Modell für Temperaturkonversion

Interface für TempCalculator

Methode	Aufgabe
<code>celsiusIntoFahrenheit(double c) : double</code>	wandle Grad Celsius in Grad Fahrenheit um
<code>fahrenheitIntoCelsius(double f) : double</code>	wandle Grad Fahrenheit in Grad Celsius um

```

/** TempCalculator models conversion between Celsius and Fahrenheit */
public class TempCalculator
{
    public TempCalculator() { }

    /** celsiusIntoFahrenheit translates degrees Celsius into Fahrenheit
     * @param c - the degrees in Celsius
     * @return the equivalent degrees in Fahrenheit */
    public double celsiusIntoFahrenheit(double c)
    { return ((9.0/5.0) * c) + 32; }

    /** fahrenheitIntoCelsius translates degrees Fahrenheit into Celsius
     * @param f - the degrees in Fahrenheit
     * @return the equivalent degrees in Celsius */
    public double fahrenheitIntoCelsius(double f)
    { return (f - 32) * (5.0/9.0); }
}

```

```

public class TempConverter
{
    public static void main(String[] args)
    {
        DialogReader reader = new DialogReader();
        TemperatureStreamWriter writer = new TemperatureStreamWriter();
        TempCalculator calc = new TempCalculator();
        int start_temp =
            reader.readInt("Type starting temperature (an integer): ");
        String code =
            reader.readString("Type its temperature scale (C or F): ");
        code = code.trim().toUpperCase();
        char letter = code.charAt(0);

        • Methoden von String:
            trim (), toUpperCase (), charAt (int i)
    }
}

```

```

if ( letter != 'F' && letter != 'C' )
{ System.out.println("TempConverter error: bad scale, " + code); }
else { int fahrenheit;
      int celsius;
      if ( letter == 'F' )
      { fahrenheit = start_temp;
        celsius = (int) calc.fahrenheitIntoCelsius(start_temp);
      }
      else // letter == 'C'
      { celsius = start_temp;
        fahrenheit = (int) calc.celsiusIntoFahrenheit(start_temp);
      }
      writer.displayCelsius(celsius);
      writer.displayFahrenheit(fahrenheit);
    }
}

```

13.7.2 Logische Operatoren

Operator	Operanden	Beschreibung
&&	boolean boolean	logisches Und, Konjunktion
	boolean boolean	logisches Oder, Disjunktion
!	boolean	logisches Nicht, Negation

Ergebnistyp: boolean

Priorität:

- ! hat Priorität vor Infixoperatoren.
- arithmetische Operationen höher als
- Vergleichsoperationen höher als
- && höher als
- ||

Auswertung erfolgt sequentiell

`true && E` \Rightarrow `E`

`false && E` \Rightarrow `false`

`true || E` \Rightarrow `true`

`false || E` \Rightarrow `E`

`! true` \Rightarrow `false`

`! false` \Rightarrow `true`

Nicht alle Operanden werden ausgewertet

\Rightarrow „nicht-strikte Auswertung“.

Beispiel

Gegeben: `int x = 2; double y = 3.5;`

`(x > 1) || (x <= y/0)`

\Rightarrow `true` `||` `(x <= y/0)`

\Rightarrow `true`

13.7.3 Bedingte Anweisung

```
<statement> ::= if ( <expr> ) { <statements> } <else part>  
<else part> ::=  $\epsilon$  | else { <statements> }
```

- Zuerst Auswertung der Bedingung *<expr>*
- Falls Wert *true*, dann werden die *<statements>* ausgeführt
- Falls Wert *false*, dann wird der *<else part>* ausgeführt
 - Falls *<else part>* leer (ϵ), so geschieht nichts
 - Andernfalls werden *<statements>* ausgeführt

Beispiel

```
/** CelsiusToFahrenheit converts an input Celsius value to Fahrenheit.
 *   output: the degrees Fahrenheit, a double */
public class CelsiusToFahrenheit1
{ public static void main(String[] args)
  { int c = 0;
    if (args.length > 0)           // arguments present?
      { c = new Integer (args[0]).intValue (); // yes, set temperature
      }
    double f = ((9.0/5.0) * c) + 32; // an initialization of f
    System.out.println("For Celsius degrees " + c + ",");
    System.out.println("Degrees Fahrenheit = " + f);
  }
}
```

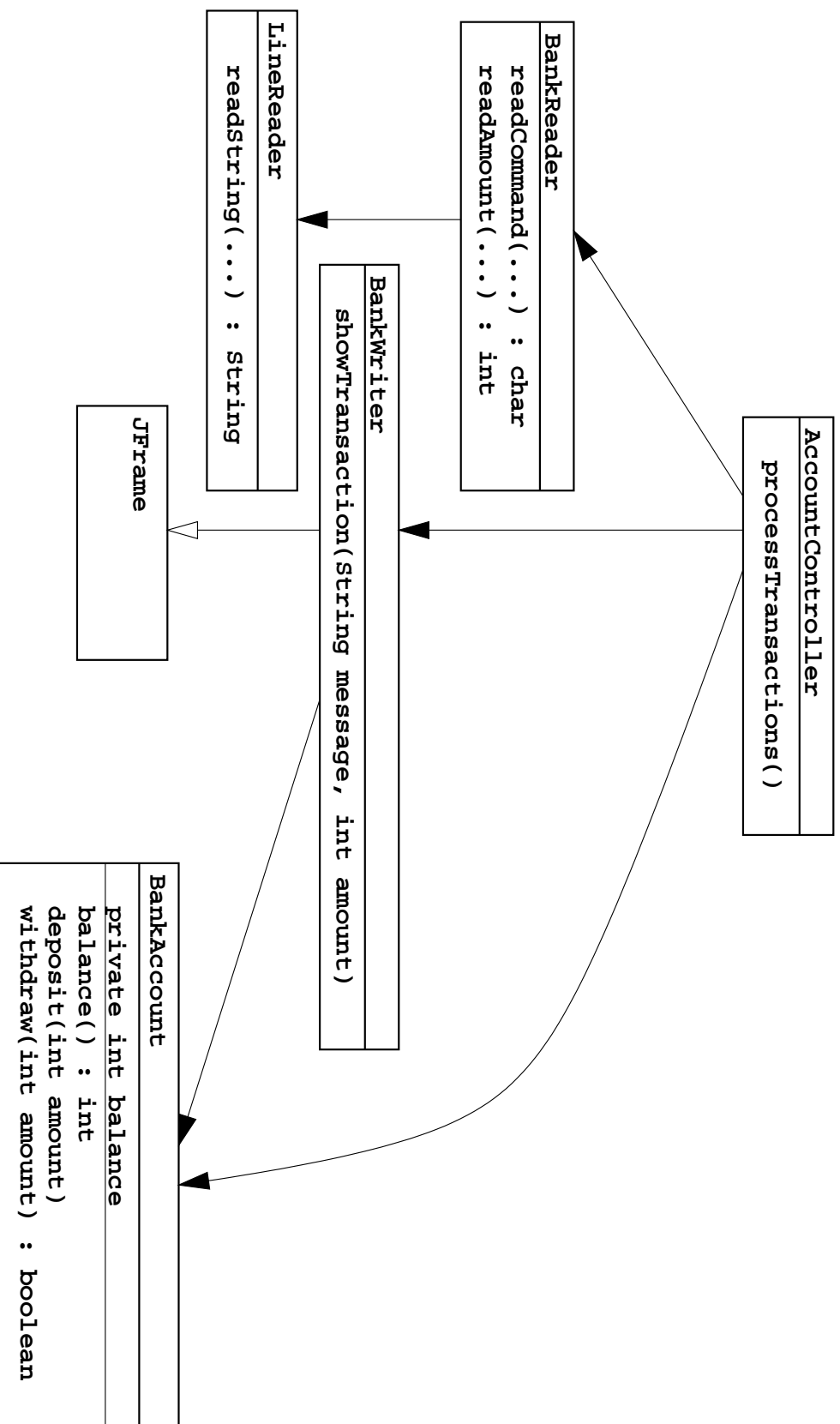
13.7.4 Anwendung: Modell mit Zustand

BankAccount

Attribut	Aufgabe
<code>private int balance</code>	Kontostand; es gilt immer: <code>balance >= 0</code>
Methode	Aufgabe
<code>balance() : int</code>	liefert den Kontostand
<code>deposit(int amount)</code>	zähle amount aufs Konto ein
<code>withdraw(int amount) : boolean</code>	zähle amount vom Konto aus; Ergebnis <code>true</code> , falls genug Geld vorhanden, ansonsten <code>false</code> und es geschieht keine Abhebung

- Zugriff auf `balance` durch *Akzessor-Methode* `balance () : int`
- `deposit` ist *Mutator-Methode*

Architektur für BankAccount



Starterklasse

```
/** AccountManager starts the application that maintains a bank account. */
public class AccountManager
{
    public static void main(String[] args)
    { // create the application's objects:
        BankReader reader = new BankReader();
        BankAccount account = new BankAccount(0);
        BankWriter writer = new BankWriter("BankWriter", account);
        AccountController controller =
            new AccountController(reader, writer, account);
        // start the controller:
        controller.processTransactions();
    }
}
```

Klasse für Konto

```
/** BankAccount models a single account */
public class BankAccount
{ private int balance;

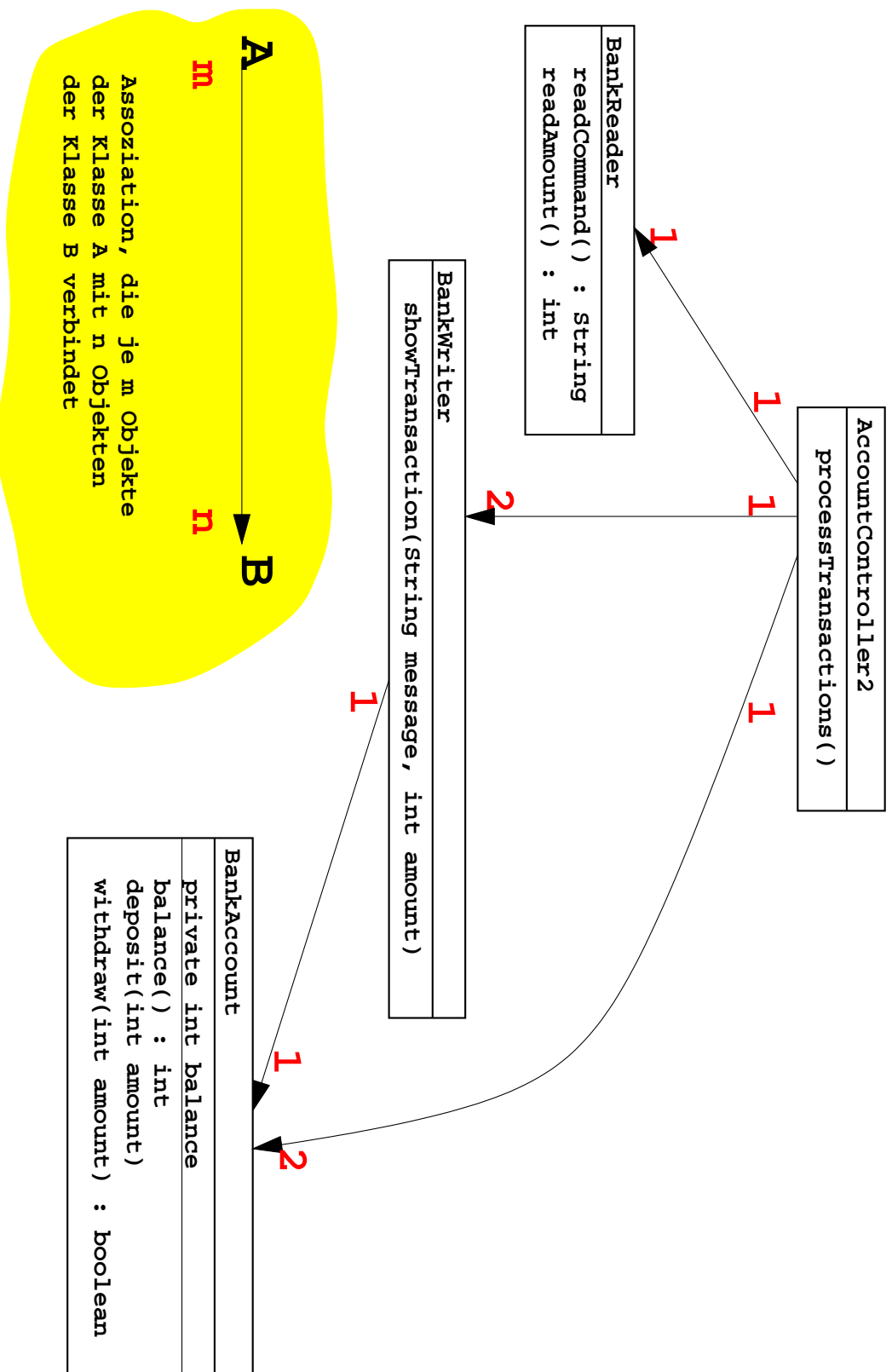
    public BankAccount (int amount)
    { balance = amount; }

    public int balance ()
    { return balance; }

    public void deposit (int amount)
    { balance = balance + amount; }

    public boolean withdraw (int amount)
    { if (amount > balance)
      { return false; }
      else
      { balance = balance - amount;
        return true;
      }
    }
}
```

Verwaltung mehrerer Bankkonten : neuer Controller



13.7.5 Syntax: Ergänzungen

$\langle \text{statement} \rangle ::= \dots \mid$
 $\text{if } (\langle \text{expr} \rangle)$
 $\{ \langle \text{statements} \rangle \} \langle \text{else part} \rangle$
 $\langle \text{else part} \rangle ::= \varepsilon \mid \text{else } \{ \langle \text{statements} \rangle \}$