What walks like a duck,
may have been statically
typed like a duck*!*

Programmierung mit Schnittstellen

OOPM, Ralf Lämmel

Deklarationsform für Datenkapseln
bisher

(+ Konstruktoren)

```
public class Datentypsname {

  private Typ1 Komponente1;
  ...
  private TypN KomponenteN;

  public Result1 Operation1(... Argumente1 ...) { ... };
  ...
  public ResultM OperationM(... ArgumenteM ...) { ... };

}
```

Deklarationsform für Datenkapseln ***unter Verwendung von Schnittstellen***

```
public interface SchnittstellenName {

  Result1 Operation1(... Argumente1 ...);
  ...
  ResultM OperationM(... ArgumenteM ...);

}
```

```
public class ImplementationsName implements SchnittstellenName {

  public Result1 Operation1(... Argumente1 ...) { ... };
  ...
  public ResultM OperationM(... ArgumenteM ...) { ... };

}
```

(+ Attribute, Konstruktoren und weitere Methoden)

Trennung von Schnittstelle und Implementation

- Idee
  - **Getrennte Deklarationsform für Schnittstellen**
    - Beinhaltet die Namen und Typen von Operationen
    - Beinhaltet ***nicht*** die Körper der Methoden
- Gründe für eine solche Trennung
  - “Dokumentation” und “Abstraktion”
  - Es gibt unterschiedliches Verhalten.
  - Es gibt unterschiedliche Repräsentationen.

Wiederkehr des Kontobeispiels

```
public class ZeroCreditAccount {
   private float balance = 0;
   public float getBalance() {
      return balance;
   }
   public void deposit(float amount) {
      if (amount <= 0)
         return;
      balance += amount;
   }
   public float withdraw(float amount) {
      ...
   }
}
```

1 Attribut.
3 Operationen.
1 impliziter Konstruktor.

Ausbau des Bankkontenbeispiels

- Schnittstelle

```
public interface Account {
   float getBalance();
   void deposit(float amount);
   float withdraw(float amount);
}
```

- Unterschiedliche Implementationen
  - Konten ohne Überziehungskredit
  - Konten mit Überziehungskredit
  - ...

Implementation 1: Konten ohne Überziehungskredit

```
public class ZeroCreditAccount implements Account {
   private float balance = 0;
   public float getBalance() {
      return balance;
   }
   public void deposit(float amount) {
      if (amount <= 0)
         return;
      balance += amount;
   }
   public float withdraw(float amount) {
      ...
   }
}
```

Der Compiler würde einen Typfehler berichten, falls wir die Implementation irgendeiner Operation aus der Schnittstelle vergäßen.

Deklarationsform für Datenkapseln *unter Verwendung von Schnittstellen*

```
public interface SchnittstellenName {

  Result1 Operation1(... Argumente1 ...);
  ...
  ResultM OperationM(... ArgumenteM ...);

}

public class ImplementationsName implements SchnittstellenName {

  public Result1 Operation1(... Argumente1 ...) { ... };
  ...
  public ResultM OperationM(... ArgumenteM ...) { ... };

}
```

(+ Attribute, Konstruktoren und weitere Methoden)

Implementation 2: Konten *mit* Überziehungskredit

```
public class SimpleCreditAccount implements Account {
  private float balance = 0;
  private float overdraft = 0;
  public float getBalance() { return balance; }
  public float getOverdraft() { return overdraft; }
  public boolean adjustOverdraft(float amount) {
    if (amount < 0)
      return false;
    if (balance < -amount)
      return false;
    overdraft = amount;
    return true;
  }
  public void deposit(float amount) { ... }
  public float withdraw(float amount) { ... }
}
```

Ein weiteres Attribut

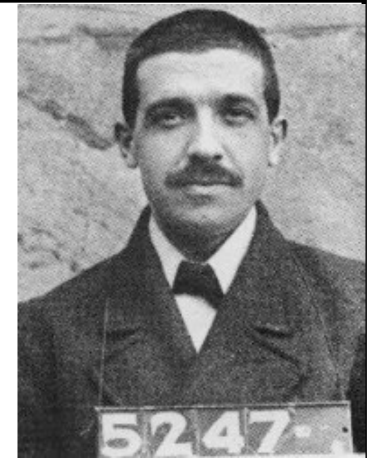
Änderung des Kreditrahmens wird nur gewährt, wenn es zu keinem “inkonsistenten” Kontostand führt.

Unterschiede in den Implementationen

```
// SimpleCreditAccount
public float withdraw(float amount) {
   float result = 0;
   if (amount <= 0)
      return result;
   result = balance - amount < -overdraft ? balance + overdraft : amount;
   balance -= result;
   return result;
}
```

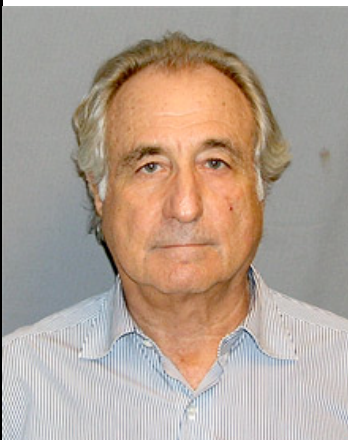
```
// ZeroCreditAccount
public float withdraw(float amount) {
   float result = 0;
   if (amount <= 0)
      return result;
   result = amount > balance ? balance : amount;
   balance -= result;
   return result;
}
```

Implementation 3: Konten mit Betrugsfeature



1910 police mugshot of Charles Ponzi.

- Ponzi-Schema
  - Erträge von Investoren von Einzahlungen bezahlt
  - Verlockung durch hohe (Schein-) Erträge
- Im folgenden: “ertragslose” Ponzi-Konten


2009 police mugshot of Bernard Madoff.

“Ertragslose” Ponzi-Konten

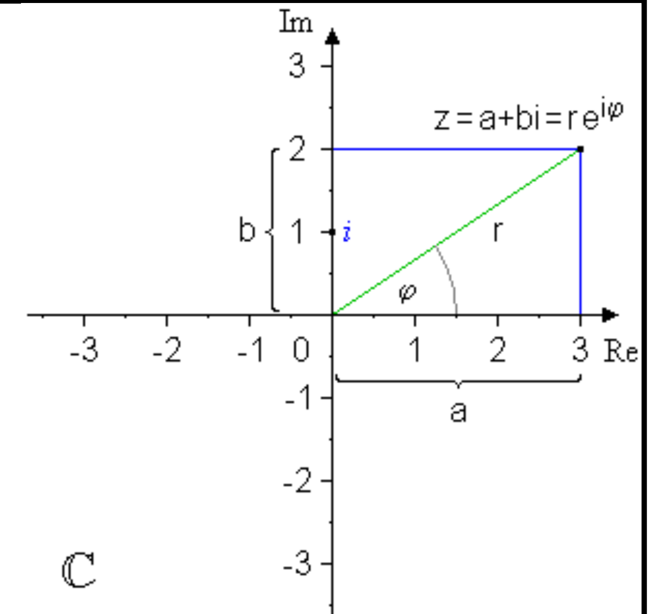
```
public class PonziAccount implements Account {
   private float balance = 0; // The apparent balance
   private Account ponzi = null; // The account of Ponzi
   public PonziAccount(Account ponzi) { this.ponzi = ponzi; }
   public float getBalance() { return balance;  }
   public void deposit(float amount) {
      ponzi.deposit(amount);
      if (amount <= 0)
         return;
      balance += amount;
   }
   public float withdraw(float amount) { ... }
}
```

Auffälligkeit von Ponzi-Konten

```
public float withdraw(float amount) {
   float result = 0;
   if (amount <= 0)
      return result;
   result = amount > balance ? balance : amount;
   result = ponzi.withdraw(result);
   balance -= result;
   return result;
}
```

Die Auszahlung ist durch die Deckung des Betrügers beschränkt.

Ein neues Beispiel: Komplexe Zahlen



http://de.wikipedia.org/wiki/Komplexe_Zahl

* ***Optionen für Repräsentationen***
  - Kartesische Form:

    Real- und Imaginärteil
  - Polarform:

    Betrag und Winkel
* Operationen für Komplexe Zahlen
  - Addition, Substraktion, Multiplikation, Division
  - Wurzel, Exponentation, ...

Operationen für Komplexe Zahlen

* Addition

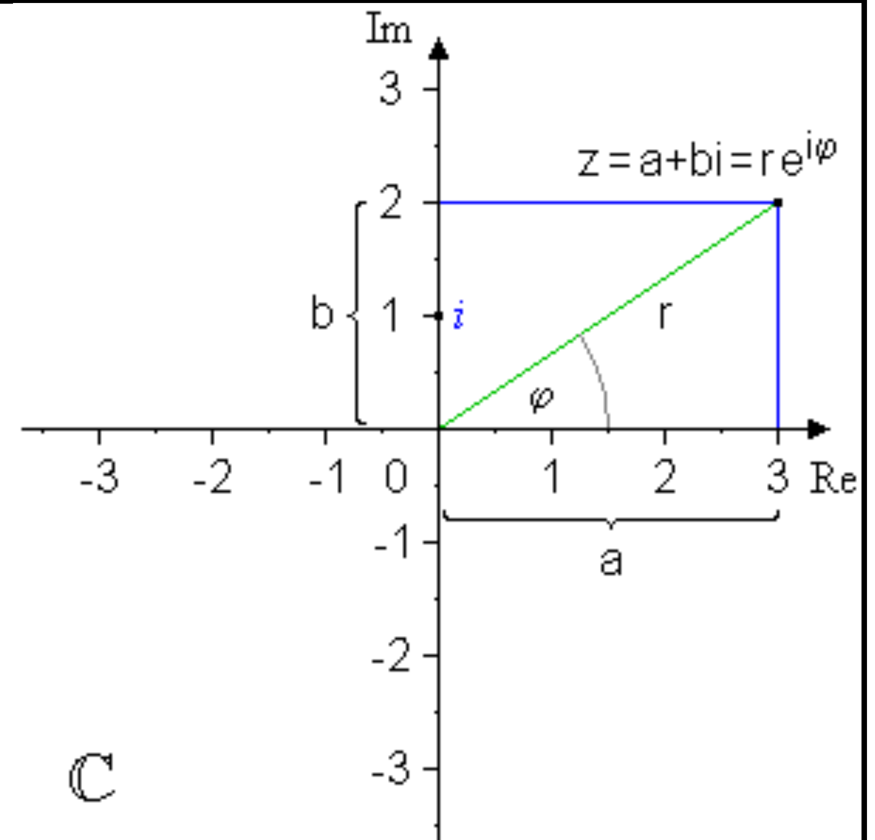
$$(a + b\,\mathrm{i}) + (c + d\,\mathrm{i}) = (a + c) + (b + d)\,\mathrm{i}$$

* Subtraktion

$$(a + b\,\mathrm{i}) - (c + d\,\mathrm{i}) = (a - c) + (b - d)\,\mathrm{i}$$

* ...

Zusammenhänge



http://de.wikipedia.org/wiki/Komplexe_Zahl

$$a = r \cdot \cos\varphi$$

$$b = r \cdot \sin\varphi$$

$$r = |z| = \sqrt{a^2 + b^2}$$

Winkel im Bogenmaß (Radiant) im Bereich -π bis + π

$$\varphi = \arg(z) = \begin{cases} \arctan\frac{b}{a} & \text{für } a > 0, b \,\text{beliebig} \\ \arctan\frac{b}{a} + \pi & \text{für } a < 0, b \geq 0 \\ \arctan\frac{b}{a} - \pi & \text{für } a < 0, b < 0 \\ \pi/2 & \text{für } a = 0, b > 0 \\ -\pi/2 & \text{für } a = 0, b < 0 \\ \text{unbestimmt} & \text{für } a = 0, b = 0 \end{cases}$$

Eine Schnittstelle für Komplexe Zahlen

```
public interface Complex {
   double getRe();
   double getIm();
   double getModulus();
   double getRadianAngle();
   double getAngle();
   Complex addTo(Complex c);
  // Further operations omitted
}
```

Unterstützung zweier Einheiten

Selbstverweis*!*

Das Ergebnis der Operation wird "erzeugt". Vorhandene Zahlen werden nicht ***mutiert***.

```
public class ComplexCartesian implements Complex {
  private double re;
  private double im;
  public ComplexCartesian(double re, double im) {
    this.re = re;
    this.im = im;
  }
  public double getRe() { return re; }
  public double getIm() { return im; }
  public double getModulus() { return Math.hypot(re, im); }
  public double getRadianAngle() { return Math.atan2(im, re); }
  public double getAngle() { return getRadianAngle() * 180 / Math.PI; }
  public Complex addTo(Complex c) {
    return new ComplexCartesian(
      this.getRe() + c.getRe(),
      this.getIm() + c.getIm());
  }
}
```

$$a = r \cdot \cos\varphi$$

$$b = r \cdot \sin\varphi$$

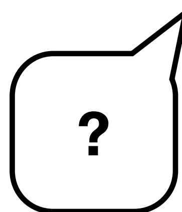
$$r = |z| = \sqrt{a^2 + b^2}$$

$$\varphi = \arg(z) = \begin{cases} \arctan\frac{b}{a} & \text{für } a > 0, b \text{ beliebig} \\ \arctan\frac{b}{a} + \pi & \text{für } a < 0, b \geq 0 \\ \arctan\frac{b}{a} - \pi & \text{für } a < 0, b < 0 \\ \pi/2 & \text{für } a = 0, b > 0 \\ -\pi/2 & \text{für } a = 0, b < 0 \\ \text{unbestimmt} & \text{für } a = 0, b = 0 \end{cases}$$

Übungsaufgabe: mehr Operationen!

```java
public class ComplexPolar implements Complex {
  private double modulus;
  private double angle;
  public ComplexPolar(double modulus, double angle) {
    this.modulus = modulus;
    this.angle = angle;
  }
  public double getRe() { return modulus * Math.cos(angle); }
  public double getIm() { return modulus * Math.sin(angle); }
  public double getModulus() { return modulus; }
  public double getRadianAngle() { return angle; }
  public double getAngle() { return getRadianAngle() * 180 / Math.PI; }
  public Complex addTo(Complex c) { ... }
}
```

Übungsaufgabe: mehr Operationen unter guter Verwendung der Polarform!



Addition in der Kartesischen Form (zum Vergleich)

```
public class ComplexCartesian implements Complex {
   ...
   public Complex addTo(Complex c) {
      return new ComplexCartesian(
         this.getRe() + c.getRe(),
         this.getIm() + c.getIm());
   }
}
```

Addition in der Polarform

```
public class ComplexPolar implements Complex {

   ...

   public Complex addTo(Complex c) {
     double re = this.getRe() + c.getRe();
     double im = this.getIm() + c.getIm();
     return new ComplexPolar(
                Math.hypot(re, im),
                Math.atan2(im, re));
   }
}
```

Umrechnung der Polarform in kartesische Koordinaten

Umrechnung kartesischer Koordinaten in Polarform

Verbesserungsvorschläge willkommen!

Ein neues Beispiel: Feldartige Typen

* Abstraktion der Operationen von Feldern

```
public interface IntArray {
  int getLength();
  int getAt(int i);
  void setAt(int i, int x);
}
```

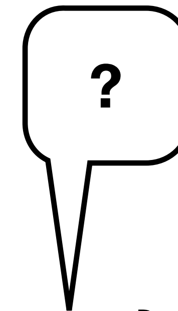
* Implementation per ***Delegation*** an klassische Felder
* Implementation mit ***erweiterbaren*** Feldern
* Implementation mit ***dünnbesiedelten*** Feldern

Implementation 1: Delegation an klassische Felder

```
public class InextensibleIntArray implements IntArray {
  private int[] a = null;
  public InextensibleIntArray(int length) {
    a = new int[length];
  }
  public int getLength() { return a.length; }
  public int getAt(int i) { return a[i]; }
  public void setAt(int i, int x) { a[i] = x; }
}
```

Implementation 2: Erweiterbare Felder

```
public class ExtensibleIntArray implements IntArray {
   private int length = 0;
   private int[] a = new int[length];
   public int getLength() { return length; }
   public int getAt(int i) {
      resizeIfNecessary(i);
      return a[i];
   }
   public void setAt(int i, int x) {
      resizeIfNecessary(i);
      a[i] = x;
   }
   private void resizeIfNecessary(int i) { ... }
}
```



Erweiterbare Felder

```
// We replace the current array by a bigger one if needed.
// We always double size when a new array is needed.
private void resizeIfNecessary(int i) {
   if (i>=length)
      length = i+1;
   if (i>=a.length) {
      int[] b = new int[2*length];
      for (int j=0; j<a.length; j++)
         b[j] = a[j];
      a = b;
   }
}
```

Verdopplung ist oft pragmatisch besser als Erhöhung der Länge auf aktuelle Indexgrenze wegen anzunehmender, nachfolgender Zugriffe.

Implementation 3: Dünnbesiedelte Felder

* Einführung einer extra Redirektionsstufe
  * Feldeinträge bestehen aus Index & Wert.
  * Einträge für 0-Werte werden nicht erfasst.

```
class Entry {
   public int i;
   public int x;
}
```

```
public class SparseIntArray implements IntArray {
   private Entry[] a = new Entry[0];
   public int getLength() { ... }
   public int getAt(int i) { ... }
   public void setAt(int i, int x) { ... }
}
```

Dünnbesiedelte Felder

```
public int getLength() {
   int result = 0;
   for (Entry e : a) {
      if (e==null)
         break;
      if (e.i+1>result)
         result = e.i+1;
   }
   return result;
}
```

Der erste null-Eintrag "beendet" das Feld und definiert damit die Länge (mit dem Spezialfall, dass das Feld komplett besetzt ist).

Wie geht's besser?

Solche lineare Suche ist verboten ineffizient!

Dünnbesiedelte Felder

```
public int getAt(int i) {
   for (Entry e : a) {
      if (e==null)
         break;
      if (e.i==i)
         return e.x;
   }
   return 0;
}
```

Wenn ein Eintrag mit passenden Index gefunden wird, so wird dessen Wert zurückgegeben. Ansonsten wird der Wert als 0 angenommen und zurückgegeben.

Wie geht's schneller?

Dünnbesiedelte Felder

```
public void setAt(int i, int x) {
  int j = 0;
  for (; j<a.length; j++) {
     if (a[j]==null)
        break;
     if (a[j].i==i) {
        a[j].x = x;
        return;
     }
  }
  if (j>=a.length) {
     Entry[] b = new Entry[2*a.length+1];
     for (int k=0; k<a.length; k++)
        b[k] = a[k];
     a = b;
  }
  Entry e = new Entry();
  e.i = i;
  e.x = x;
  a[j] = e;
}
```

In dieser Implementation kann Ersetzen zu Einträgen mit 0 als Wert führen.

Suchen und Ersetzen

Vergrößern

Eintragen

Fortgeschrittene Aspekte der Verwendung von Schnittstellen

* ***Schnittstellenpolymorphismus***: Wie formuliert man Funktionalität über unterschiedlichen Implementationen der gleichen Schnittstelle?
* ***Schnittstellenerweiterung***: Wie verfährt man in einer Situation, in welcher eine Implementation u.U. über eine gegebene Schnittstelle hinausgeht?
* ***Simultane Schnittstellenimplementation***: Wie modelliert man die Tatsache, dass eine Klasse u.U. mehrere Schnittstellen implementiert?

Verwendung von ***Schnittstellenpolymorphismus*** zur Aufsummierung der Kontostände für die Konten einer Bank

```
public class Bank {
   private Account[] accounts = new Account[42];
   public float totalBalance() {
      float result = 0;
      for (Account a : accounts)
         result += a.getBalance();
      return result;
   }
   ...
}
```

Datenkapsel unterschiedlicher Implementationstypen passen in dieses Feld.

“Späte Bindung”:
Der tatsächliche Typ von *a* entscheidet über die anzuwendende Implementation von *getBalance*.

Schnittstellenpolymorphismus

- “Schnittstellentypen sind auch Typen.”
- Sie dürfen wie andere ***Typen*** verwendet werden:
  - für Methodenargumente,
  - für Methodenergebnisse,
  - für lokale Variablen,
  - für Elemente von Feldern,
  - etc.

“***Späte Bindung***”: Methodenaufrufe mit einer Schnittstelle als *statischem Typ des Empfängers* werden i.A. zur Laufzeit auf die jeweilige Implementation abgebildet.

Schnittstellenerweiterung

* Eine Schnittstelle kann eine andere Schnittstelle erweitern.

```
interface A { ... }
interface B extends A { ... }
```

* Mehrere Schnittstellen können eine Schnittstelle erweitern.

```
interface C extends A { ... }
```

* Eine Schnittstelle kann mehrere Schnittstellen erweitern.

```
interface X { ... }
interface Y { ... }
interface Z extends X,Y { ... }
```

Zyklische Erweiterungen sind nicht gestattet.

Bankkonten ***mit Kredit***

```
public interface Account {
   float getBalance();
   void deposit(float amount);
   float withdraw(float amount);
}
public interface CreditAccount extends Account {
   float getOverdraft();
  boolean adjustOverdraft(float amount);
}
public class SimpleCreditAccount implements CreditAccount {
   ...
}
```

Die erweiterte Schnittstelle enthält natürlich auch alle Operationen aus der ursprünglichen Schnittstelle.

Typischerweise wird es nun auch andere Konten mit Kredit geben.

Verwendung von Cast zur Selektion relevanter Konten

Nicht alle Konten sind Konten mit Kredit.

```
public class Bank {
   private Account[] accounts = new Account[42];
   public float totalOverdraft() {
      float result = 0;
      for (Account a : accounts)
         if (a instanceof CreditAccount)
            result += ((CreditAccount)a).getOverdraft();
      return result;
   }
   ...
}
```

Wir machen erst einen Typtest (... instanceof ...) und dann vollziehen wir gegenenfalls den Typcast (... (CreditAccount) ...).

Simultane Schnittstellenimplementation

- Eine Datenkapsel kann verschiedene Funktionen erfüllen.
- Also kann eine Klasse mehrere Schnittstellen implementieren.

```
interface X { ... }
interface Y { ... }
class Z implements X,Y { ... }
```

Simultane Schnittstellenimplementation in einer Bankanwendung

- Schnittstelle zur Zuordnung eines “Wertes”

```
public interface Worth {
  float getWorth();
}
```

- Denkbare Implementationen
  - Konten
  - Immobilien
  - etc.

Immobilien mit Wert

```
public class Property implements Worth {
  private float worth = 0;
  public Property(float worth) {
    this.worth = worth;
  }
  public float getWorth() {
    return worth;
  }
}
```

Ponzi-Konten mit Wert

```
public class PonziAccount implements Account, Worth
{
  ...
  public float getBalance() { ... }
  public void deposit(float amount) { ... }
  public float withdraw(float amount) { ... }
  public float getWorth() { return 0; }
}
```

GAME OVER

* **Zusammenfassung**

  Der Abstraktionsmechanismus der Schnittstelle macht die Schnittstellen (Operationsmengen) von Datentypen explizit und separiert sie von der konkreten Implementation.

* **Ausblick**
  * Programmierung mit Zeigern
  * Abstrakte Datentypen
  * Generizität von Datentypen
  * ...