Script generated by TTT

Title: Seidl: EOE1 (12.01.2012)

Date: Thu Jan 12 12:41:08 CET 2012

Duration: 77:24 min

Pages: 47

- `class A extends B { ... }` deklariert die Klasse A als Unterklasse der Klasse B.
- Alle Members von B stehen damit automatisch auch der Klasse A zur Verfügung.
- Als `protected` klassifizierte Members sind auch in der Unterklasse sichtbar.
- Als `private` deklarierte Members können dagegen in der Unterklasse nicht direkt aufgerufen werden, da sie dort nicht sichtbar sind.
- Wenn ein Konstruktor der Unterklasse A aufgerufen wird, wird implizit zuerst der Konstruktor `B()` der Oberklasse aufgerufen.

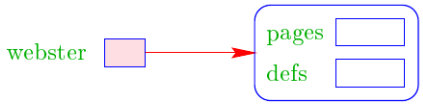
`Dictionary webster = new Dictionary(12400);` liefert:

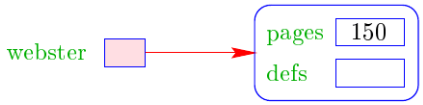
299

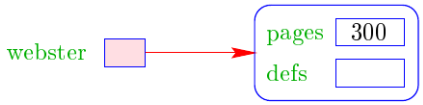
- `class A extends B { ... }` deklariert die Klasse A als Unterklasse der Klasse B.
- Alle Members von B stehen damit automatisch auch der Klasse A zur Verfügung.
- Als `protected` klassifizierte Members sind auch in der Unterklasse sichtbar.
- Als `private` deklarierte Members können dagegen in der Unterklasse nicht direkt aufgerufen werden, da sie dort nicht sichtbar sind.
- Wenn ein Konstruktor der Unterklasse A aufgerufen wird, wird implizit zuerst der Konstruktor `B()` der Oberklasse aufgerufen.

`Dictionary webster = new Dictionary(12400);` liefert:

301








```
public class Words {
   public static void main(String[] args) {
      Dictionary webster = new Dictionary(12400);
      webster.page_message();
      webster.defs_message();
   } // end of main
} // end of class Words
```

- Das neue Objekt `webster` enthält die Attribute `pages` und `defs`, sowie die Objekt-Methoden `page_message()` und `defs_message()`.
- Kommen in der Unterklasse nur weitere Members hinzu, spricht man von einer is_a-Beziehung. (Oft müssen aber Objekt-Methoden der Oberklasse in der Unterklasse umdefiniert werden.)

- Die Programm-Ausführung liefert:

```
Number of pages:        300
Number of defs:         12400
Defs per page:          41
```

11.1 Das Schlüsselwort super

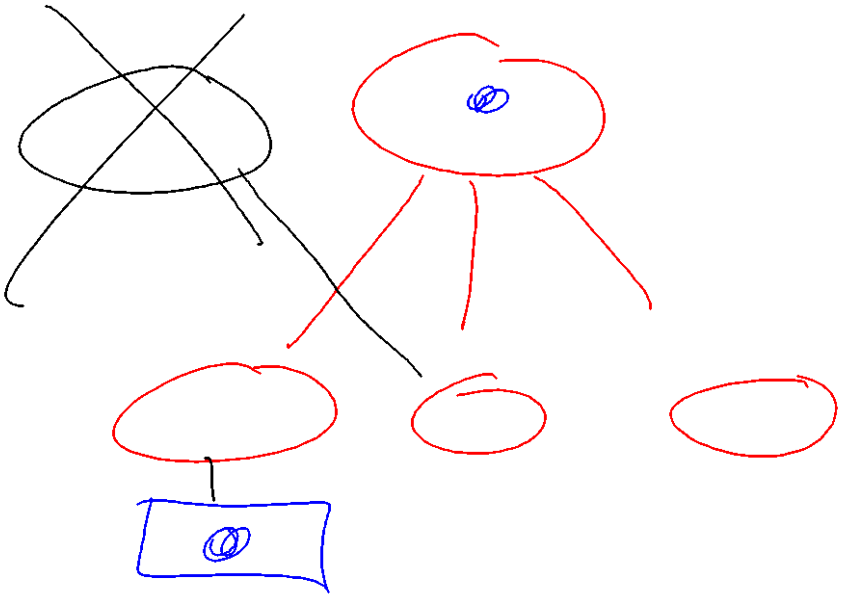
- Manchmal ist es erforderlich, in der Unterklasse explizit die Konstruktoren oder Objekt-Methoden der Oberklasse aufzurufen. Das ist der Fall, wenn
  - Konstruktoren der Oberklasse aufgerufen werden sollen, die Parameter besitzen;
  - Objekt-Methoden oder Attribute der Oberklasse und Unterklasse gleiche Namen haben.
- Zur Unterscheidung der aktuellen Klasse von der Oberklasse dient das Schlüsselwort `super`.

... im Beispiel:

```
public class Book {
   protected int pages;
   public Book(int x) {
      pages = x;
   }
   public void message() {
      System.out.print("Number of pages:\t"+pages+"\n");
   }
} // end of class Book
...
```



```
public class Dictionary extends Book {
   private int defs;
   public Dictionary(int p, int d) {
      super(p);
      defs = d;
   }
   public void message() {
      super.message();
      System.out.print("Number of defs:\t\t"+defs+"\n");
      System.out.print("Defs per page:\t\t"+defs/pages+"\n");
      }
} // end of class Dictionary
```

```
public class Dictionary extends Book {
   private int defs;
   public Dictionary(int p, int d) {
      super(p);
      defs = d;
   }
   public void message() {
      super.message();
      System.out.print("Number of defs:\t\t"+defs+"\n");
      System.out.print("Defs per page:\t\t"+defs/pages+"\n");
      }
} // end of class Dictionary
```


- `super(...);` ruft den entsprechenden Konstruktor der Oberklasse auf.
- Analog gestattet `this(...);` den entsprechenden Konstruktor der eigenen Klasse aufzurufen.
- Ein solcher expliziter Aufruf muss stets ganz am Anfang eines Konstruktors stehen.
- Deklariert eine Klasse A einen Member memb gleichen Namens wie in einer Oberklasse, so ist nur noch der Member memb aus A sichtbar.
- Methoden mit unterschiedlichen Argument-Typen werden als verschieden angesehen.
- `super.`memb greift für das aktuelle Objekt `this` auf Attribute oder Objekt-Methoden memb der Oberklasse zu.
- Eine andere Verwendung von `super` ist nicht gestattet.


```
public class Words {
   public static void main(String[] args) {
      Dictionary webster = new Dictionary(540,36600);
      webster.message();
   } // end of main
} // end of class Words
```

- Das neue Objekt `webster` enthält die Attribute `pages` wie `defs`.
- Der Aufruf `webster.message()` ruft die Objekt-Methode der Klasse `Dictionary` auf.
- Die Programm-Ausführung liefert:

```
Number of pages:        540
Number of defs:         36600
Defs per page:          67
```


311


```
public class Words {
   public static void main(String[] args) {
      Dictionary webster = new Dictionary(540,36600);
      webster.message();
   } // end of main
} // end of class Words
```

- Das neue Objekt `webster` enthält die Attribute `pages` wie `defs`.
- Der Aufruf `webster.message()` ruft die Objekt-Methode der Klasse `Dictionary` auf.
- Die Programm-Ausführung liefert:

```
Number of pages:        540
Number of defs:         36600
Defs per page:          67
```

Was es sonst noch so in Java gibt:

... an nützlichem:	Aufzähldatentypen;
	innere Klassen;
	anonyme Klassen;
... an mysteriösem:	Klassen zur Selbst-Reflektion;
... an komfortablem:	graphische Benutzerinterfaces;
	Malen mit Swing;
... an technischem:	Networking mit Sockets, RMI,
	Jini, Corba, EJB, ...

Was es sonst noch so in Java gibt:

... an nützlichem:	Aufzähldatentypen;
	innere Klassen;
	anonyme Klassen;
... an mysteriösem:	Klassen zur Selbst-Reflektion;
... an komfortablem:	graphische Benutzerinterfaces;
	Malen mit Swing;
... an technischem:	Networking mit Sockets, RMI,
	Jini, Corba, EJB, ...

- Der Ausdruck Const benötigt ein Argument. Dieses wird dem Konstruktor mitgegeben und in einer privaten Variable gespeichert.
- Die Klasse ist als `final` deklariert.
- Zu als final deklarierten Klassen dürfen keine Unterklassen deklariert werden !!!
- Aus Sicherheits- wie Effizienz-Gründen sollten so viele Klassen wie möglich als final deklariert werden ...
- Statt ganzer Klassen können auch einzelne Variablen oder Methoden als `final` deklariert werden.
- Finale Members dürfen nicht in Unterklassen umdefiniert werden.
- Finale Variablen dürfen zusätzlich nur initialisiert, aber nicht modifiziert werden $\Longrightarrow$ Konstanten.


```
public class Eating {
   public static void main (String[] args) {
      Pizza special = new Pizza(275);
      System.out.print("Calories per serving: " +
         special.calories_per_serving());
   } // end of main
} // end of class Eating
```

```
public class Food {
   private int CALORIES_PER_GRAM = 9;
   private int fat, servings;
   public Food (int num_fat_grams, int num_servings) {
      fat = num_fat_grams;
      servings = num_servings;
   }
   private int calories() {
      return fat * CALORIES_PER_GRAM;
   }
   public int calories_per_serving() {
      return (calories() / servings);
   }
} // end of class Food
```



```
public class Pizza extends Food {
   public Pizza (int amount_fat) {
      super (amount_fat,8);
      }
   } // end of class Pizza
```

- Die Unterklasse `Pizza` verfügt über alle Members der Oberklasse `Food` – wenn auch nicht alle direkt zugänglich sind.
- Die Attribute und die Objekt-Methode `calories()` der Klasse `Food` sind privat, und damit für Objekte der Klasse `Pizza` verborgen.
- Trotzdem können sie von der `public` Objekt-Methode `calories_per_serving` benutzt werden.

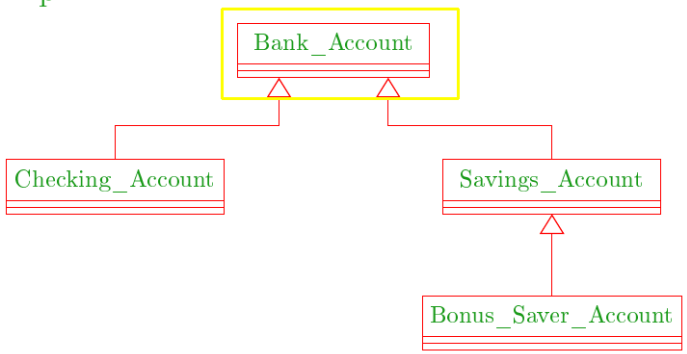

```
public class Pizza extends Food {
   public Pizza (int amount_fat) {
      super (amount_fat,8);
      }
   } // end of class Pizza
```

- Die Unterklasse `Pizza` verfügt über alle Members der Oberklasse `Food` – wenn auch nicht alle direkt zugänglich sind.
- Die Attribute und die Objekt-Methode `calories()` der Klasse `Food` sind privat, und damit für Objekte der Klasse `Pizza` verborgen.
- Trotzdem können sie von der `public` Objekt-Methode `calories_per_serving` benutzt werden.

... Ausgabe des Programms: `Calories per serving: 309`

11.3 Überschreiben von Methoden

Beispiel:



Aufgabe:

- Implementierung von einander abgeleiteter Formen von Bank-Konten.
- Jedes Konto kann eingerichtet werden, erlaubt Einzahlungen und Auszahlungen.
- Verschiedene Konten verhalten sich unterschiedlich in Bezug auf Zinsen und Kosten von Konto-Bewegungen.

Einige Konten:

```
public class Bank {
   public static void main(String[] args) {
      Savings_Account savings =
         new Savings_Account (4321, 5028.45, 0.02);
      Bonus_Saver_Account big_savings =
         new Bonus_Saver_Account (6543, 1475.85, 0.02);
      Checking_Account checking =
         new Checking_Account (9876, 269.93, savings);
      ...
```


Einige Konto-Bewegungen:

```
      savings.deposit (148.04);
      big_savings.deposit (41.52);
      savings.withdraw (725.55);
      big_savings.withdraw (120.38);
      checking.withdraw (320.18);
   } // end of main
} // end of class Bank
```


Einige Konten:

```
public class Bank {
   public static void main(String[] args) {
      Savings_Account savings =
         new Savings_Account (4321, 5028.45, 0.02);
      Bonus_Saver_Account big_savings =
         new Bonus_Saver_Account (6543, 1475.85, 0.02);
      Checking_Account checking =
         new Checking_Account (9876,269.93, savings);
      ...
```

Einige Konto-Bewegungen:

```
        savings.deposit (148.04);
        big_savings.deposit (41.52);
        savings.withdraw (725.55);
        big_savings.withdraw (120.38);
        checking.withdraw (320.18);
    } // end of main
} // end of class Bank
```



```
public class Bank_Account {
   // Attribute aller Konten-Klassen:
   protected int account;
   protected double balance;
   // Konstruktor:
   public Bank_Account (int id, double initial) {
      account = id; balance = initial;
      }
   // Objekt-Methoden:
   public void deposit(double amount) {
      balance = balance+amount;
      System.out.print("Deposit into account "+account+"\n"
                     +"Amount:\t\t"+amount+"\n"
     +"New balance:\t"+balance+"\n\n");
      }
      ...
```



```
   public boolean withdraw(double amount) {
      System.out.print("Withdrawal from account "+ account +"\n"
                     +"Amount:\t\t"+ amount +"\n");
      if (amount > balance) {
          System.out.print("Sorry, insufficient funds...\n\n");
          return false;
      }
      balance = balance-amount;
      System.out.print("New balance:\t"+ balance +"\n\n");
      return true;
   }
} // end of class Bank_Account
```


- Die Objekt-Methode `withdraw()` nimmt eine Auszahlung vor.
- Falls die Auszahlung scheitert, wird eine Mitteilung gemacht.
- Ob die Auszahlung erfolgreich war, teilt der Rückgabewert mit.
- Ein `Checking_Account` verbessert ein normales Konto, indem im Zweifelsfall auf die Rücklage eines Sparkontos zurückgegriffen wird.

Ein Giro-Konto:

```
public class Checking_Account extends Bank_Account {
   private Savings_Account overdraft;
// Konstruktor:
   public Checking_Account(int id, double initial,
                           Savings_Account savings) {
      super (id, initial);
      overdraft = savings;
      }
   ...
```



```
// modifiziertes withdraw():
   public boolean withdraw(double amount) {
      if (!super.withdraw(amount)) {
         System.out.print("Using overdraft...\n");
         if (!overdraft.withdraw(amount-balance)) {
            System.out.print("Overdraft source insufficient.\n\n");
            return false;
         } else {
            balance = 0;
            System.out.print("New balance on account "+ account +": 0\n\n");
      }  }
      return true;
   }
} // end of class Checking_Account
```


Ein Giro-Konto:

```
public class Checking_Account extends Bank_Account {
   private Savings_Account overdraft;
// Konstruktor:
   public Checking_Account(int id, double initial,
                           Savings_Account savings) {
      super (id, initial);
      overdraft = savings;
      }
   ...
```


- Die Objekt-Methode `withdraw` wird neu definiert, die Objekt-Methode `deposit` wird übernommen.
- Der Normalfall des Abhebens erfolgt (als Seiteneffekt) beim Testen der ersten `if`-Bedingung.
- Dazu wird die `withdraw`-Methode der Oberklasse aufgerufen.
- Scheitert das Abheben mangels Geldes, wird der Fehlbetrag vom Rücklagen-Konto abgehoben.
- Scheitert auch das, erfolgt keine Konto-Bewegung, dafür eine Fehlermeldung.
- Andernfalls sinkt der aktuelle Kontostand auf 0 und die Rücklage wird verringert.

```
// modifiziertes withdraw():
   public boolean withdraw(double amount) {
      if (!super.withdraw(amount)) {
         System.out.print("Using overdraft...\n");
         if (!overdraft.withdraw(amount-balance)) {
            System.out.print("Overdraft source insufficient.\n\n");
            return false;
         } else {
            balance = 0;
            System.out.print("New balance on account "+ account +": 0\n\n");
      }  }
      return true;
   }
} // end of class Checking_Account
```


- Die Objekt-Methode `withdraw` wird neu definiert, die Objekt-Methode `deposit` wird übernommen.
- Der Normalfall des Abhebens erfolgt (als Seiteneffekt) beim Testen der ersten `if`-Bedingung.
- Dazu wird die `withdraw`-Methode der Oberklasse aufgerufen.
- Scheitert das Abheben mangels Geldes, wird der Fehlbetrag vom Rücklagen-Konto abgehoben.
- Scheitert auch das, erfolgt keine Konto-Bewegung, dafür eine Fehlermeldung.
- Andernfalls sinkt der aktuelle Kontostand auf 0 und die Rücklage wird verringert.

Ein Sparbuch:

```
public class Savings_Account extends Bank_Account {
   protected double interest_rate;
// Konstruktor:
   public Savings_Account (int id, double init, double rate) {
      super(id,init); interest_rate = rate;
      }
// zusaetzliche Objekt-Methode:
   public void add_interest() {
      balance = balance * (1+interest_rate);
      System.out.print("Interest added to account: "+ account
                  +"\nNew balance:\t"+ balance +"\n\n");
   }
} // end of class Savings_Account
```


- Die Klasse `Savings_Account` erweitert die Klasse `Bank_Account` um das zusätzliche Attribut `double interest_rate` (Zinssatz) und eine Objekt-Methode, die die Zinsen gutschreibt.
- Alle sonstigen Attribute und Objekt-Methoden werden von der Oberklasse geerbt.
- Die Klasse `Bonus_Saver_Account` erhöht zusätzlich den Zinssatz, führt aber Strafkosten fürs Abheben ein.

Ein Bonus-Sparbuch:

```
public class Bonus_Saver_Account extends Savings_Account {
   private int penalty;
   private double bonus;
// Konstruktor:
   public Bonus_Saver_Account(int id, double init, double rate) {
      super(id, init, rate); penalty = 25; bonus = 0.03;
   }
// Modifizierung der Objekt-Methoden:
   public boolean withdraw(double amount) {
      System.out.print("Penalty incurred:\t"+ penalty +"\n");
      return super.withdraw(amount+penalty);
      }
   ...
```



```
   public void add_interest() {
      balance = balance * (1+interest_rate+bonus);
      System.out.print("Interest added to account: "+ account
                  +"\nNew balance:\t" + balance +"\n\n");
   }
} // end of class Bonus_Safer_Account
```

... als Ausgabe erhalten wir dann:


```
Deposit into account 4321
Amount:          148.04
New balance:     5176.49

Deposit into account 6543
Amount:          41.52
New balance:     1517.37

Withdrawal from account 4321
Amount:          725.55
New balance:     4450.94
```



```
Penalty incurred:        25
Withdrawal from account 6543
Amount:          145.38
New balance:     1371.9899999999998

Withdrawal from account 9876
Amount:          320.18
Sorry, insufficient funds...

Using overdraft...
Withdrawal from account 4321
Amount:          50.25
New balance:     4400.69

New balance on account 9876: 0
```

Einige Konto-Bewegungen:

```
      savings.deposit (148.04);
      big_savings.deposit (41.52);
      savings.withdraw (725.55);
      big_savings.withdraw (120.38);
      checking.withdraw (320.18);
   } // end of main
} // end of class Bank
```


- Anlegen eines Kontos `Bank_Account` speichert eine (hoffentlich neue) Konto-Nummer sowie eine Anfangs-Einlage.
- Die zugehörigen Attribute sind `protected`, d.h. können nur von Objekt-Methoden der Klasse bzw. ihrer Unterklassen modifiziert werden.
- die Objekt-Methode `deposit` legt Geld aufs Konto, d.h. modifiziert den Wert von `balance` und teilt die Konto-Bewegung mit.


```
Penalty incurred:       25
Withdrawal from account 6543
Amount:         145.38
New balance:    1371.9899999999998

Withdrawal from account 9876
Amount:         320.18
Sorry, insufficient funds...

Using overdraft...
Withdrawal from account 4321
Amount:         50.25
New balance:    4400.69

New balance on account 9876: 0
```


334

Penalty incurred: 25
Withdrawal from account 6543
Amount: 145.38
New balance: 1371.9899999999998

Withdrawal from account 9876
Amount: 320.18
Sorry, insufficient funds...

Using overdraft...
Withdrawal from account 4321
Amount: 50.25
New balance: 4400.69

New balance on account 9876: 0