Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

✓ Information Hiding
Klassenvariable und Klassenmethoden
Die this-Referenz
Initialisierung von Datenfeldern
Instantiierung von Klassen
Freigabe von Speicher
Die Klasse Object

Information Hiding

- Aus Gründen des Software Engineerings sollte es **keinen direkten Zugriff auf die Daten eines Objektes von anderen Klassen aus** geben. Direkter Zugriff bedeutet, dass über eine Referenz direkt auf ein Datenfeld zugegriffen werden kann.

```
1  // Datei: Person.java
2
3  public class Person
4  {
5      public String name;
6      public String vorname;
7      public int alter;
8
9      public void print()
10     {
11         System.out.println ("Name     : " + name);
12         System.out.println ("Vorname  : " + vorname);
13         System.out.println ("Alter    : " + alter);
14     }
15 }
16
```

Information Hiding

- Beispiel

```
public class TestPerson{
   public static void main (String[] args){
      Person p = new Person();

      // Die Daten der Klasse Person sind nicht geschuetzt, es kann
      // auf sie problemlos aus einer anderen Klasse heraus zuge-
      // griffen werden (sind ja public)!
      p.name = "Mueller";
      p.vorname = "Fritz";
      p.alter = 35;
      p.print();
   }
}
```

```
Name:   Müller
Vorname:        Fritz
Alter:  35
```

Information Hiding

- aus Gründen des Software Engineerings sollte es keinen direkten Zugriff auf die Daten eines Objektes von anderen Klassen aus geben

```
public class Person{
   private String name;     // Der Zugriffsmodifikator private erlaubt
   private String vorname; // das Verbergen
   private int alter;

   public void print(){
      System.out.println ("Name    : " + name);
      System.out.println ("Vorname : " + vorname);
      System.out.println ("Alter   : " + alter);
   }
}
```

- von der Klasse TestPerson hat man nun keinen direkten Zugriff mehr auf die Daten des Objektes der Klasse Person
- Wie könnte man Daten denn verändern lassen, wenn man wollte?

Information Hiding

- Setzen und Holen von Werten erfolgt über getter- und setter-Methoden

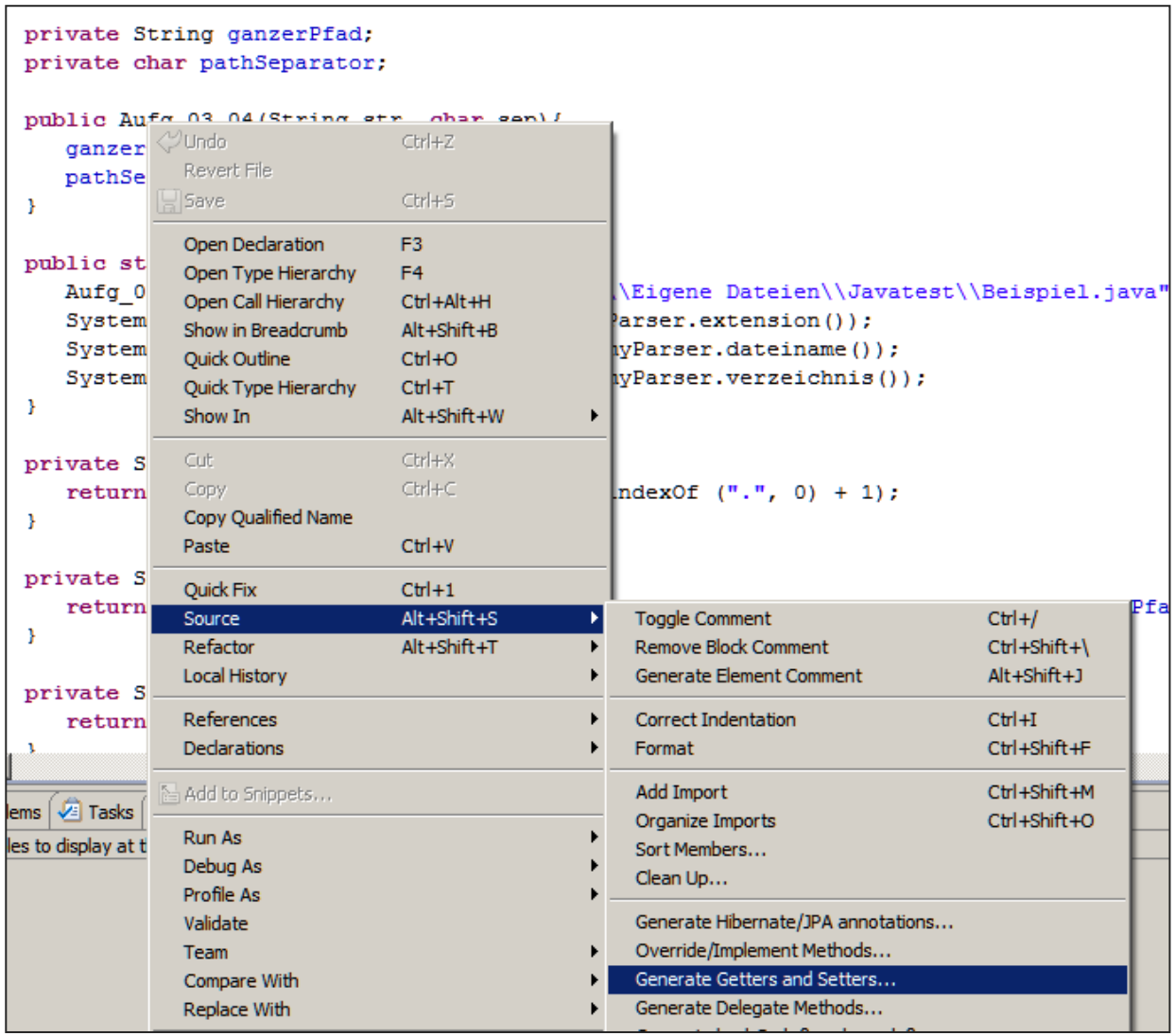
```
public class Person{
   private String name;     // Der Zugriffsmodifikator private erlaubt
   private String vorname; // das Verbergen
   private int alter;

   public void setName(String name){
      this.name = name;
   }
   public String getName(){
       return this.name;
   }

   public void setVorname(String vorname){
      this.vorname = vorname;
   }
   public String getVorname(){
       return this.vorname;
   }
   …
}
```

Information Hiding

- Eclipse kann *getter*- und *setter*-Methoden automatisch generieren



Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Information Hiding
- ✓ Klassenvariable und Klassenmethoden
- Die this-Referenz
- Initialisierung von Datenfeldern
- Instantiierung von Klassen
- Freigabe von Speicher
- Die Klasse Object

Klassenvariable I

- Klassenvariable
  - Klassenvariable stehen für alle Objekte einer Klasse als globale Daten zur Verfügung und werden mit dem Schlüsselwort ***static*** deklariert
  - Beispiel: klassenstärke ist eine Eigenschaft, die keinem individuellen Objekt angehört, sondern übergreifend gilt, daher Verwendung als Klassenvariable

```
public class Schueler {
   // Instanzvariable
   private int nummerDesSchuelers;
   // Klassenvariable
   public static int klassenStaerke = 0;

   public void setzeNummer(){
      nummerDesSchuelers = ++klassenStaerke;
   }

   public void abzaehlen(){
      System.out.println ("Ich bin die Nr.: " + nummerDesSchuelers);
   }
}
```

Klassenvariable II

- Klassenvariable Beispiel

```
public class SchuelerTest{
   public static void main (String[] args){
      System.out.println ("Klassenstaerke vor der Einschulung: "
                          + Schueler.klassenStaerke);
      // Erzeugung eines Arrays fuer Schueler
      Schueler[] schuelerInKlasse = new Schueler [10];

      for (int lv = 0; lv < schuelerInKlasse.length; lv++){
         schuelerInKlasse [lv] = new Schueler();
         schuelerInKlasse [lv].setzeNummer();
      }

      // Ausgabe der Schueler
      for (Schueler element : schuelerInKlasse){
         element.abzaehlen();
      }

      System.out.println ("Klassenstaerke nach der Einschulung: "
                          + Schueler.klassenStaerke);
   }
}
```

Klassenvariable III

- Klassenvariable Zusammenfassung

  - sind nicht Teil von Objekten, sondern werden bei der Klasse geführt (sind deshalb für alle Objekte nur einmal vorhanden)

  - Zugriff ist ohne die Existenz einer Instanz (eines Objektes) einer Klasse möglich (Zugriff erfolgt über den Klassennamen: z.B. Schueler.klassenStaerke)

Klassenmethoden I

- Klassenmethoden
- betrachten wir noch einmal das vorherige Beispiel mit der Schüleranzahl
  - es soll die Methode holeSchuelerAnzahl() hinzukommen
  - diese Methode gibt den Wert der Klassenvariable klassenStaerke zurück

```
public class Schueler2{
   private int nummerDesSchuelers;

   private static int klassenStaerke = 0;

   public void setzeNummer(){
      klassenStaerke++;
      nummerDesSchuelers = klassenStaerke;
   }
   public void abzaehlen(){
      System.out.println ("Ich bin die Nr.: " + nummerDesSchuelers);
   }
   public static int holeSchuelerAnzahl(){
      return klassenStaerke;
   }
}
```

Klassenmethoden II

- Klassenmethoden Fortsetzung Beispiel

```
public class Schueler2Test{
   public static void main (String[] args){
      // Zugriff auf Klassenmethode, ohne dass ein Objekt existiert
      System.out.println ("Klassenstaerke vor der Einschulung: "
                          + Schueler2.holeSchuelerAnzahl());

      Schueler2[] schuelerInKlasse = new Schueler2 [10];

      for (int lv = 0; lv < schuelerInKlasse.length; lv++){
         schuelerInKlasse [lv] = new Schueler2();
         schuelerInKlasse [lv].setzeNummer();
      }

      for (Schueler2 element : schuelerInKlasse){
         element.abzaehlen();
      }

      System.out.println("Klassenstaerke nach der Einschulung: "
                         + Schueler2.holeSchuelerAnzahl());
   }
}
```

Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Information Hiding
- ✓ Klassenvariable und Klassenmethoden
- ✓ Die this-Referenz

Initialisierung von Datenfeldern

Instantiierung von Klassen

Freigabe von Speicher

Die Klasse Object

Die this-Referenz I

- Klassenmethoden können nicht über die *this*-Referenz auf Instanzvariablen bzw. Instanzmethoden zugreifen.
- wird verwendet, um explizit darauf aufmerksam zu machen, dass die Instanzvariable bzw. Instanzmethode verwendet werden soll
- hat ein Datenfeld den gleichen Namen wie eine lokale Variable, hilft die this-Referenz, um auf das verdeckte Datenfeld zuzugreifen

```
public class Person{
   private String name;

   public void setName(String name){
       // durch die Verwendung von this wird klar, dass
       // die Instanzvariable "name" den Wert des Übergabe-
       // parameters "name" erhalten soll
       this.name = name;
   }
   …
}
```

Die this-Referenz II

- this ist eine Referenz auf das aktuelle Objekt
- verlangt eine Methode als Parameter ein Objekt, so kann der Aufruf mit der this-Referenz erfolgen

```
public class DBHandler{
    …
    public void savePerson(Person p){
        // Anweisungen
        …
    }
    …
}
```

```
public class Person{
    …
    dbHandler.savePerson(this);
    …
}
```

Die this-Referenz III

- umgekehrt kann der Rückgabetyp einer Methode eine Referenz auf das eigene Objekt sein

```
public class Person{
   private String name;

   public Person setName(String n){
       this.name = n;
       return this;
   }

   public void print(){
       System.out.println("Name: "+this.name);
}
```

```
public class TestPerson{

    public static void main(String[] args){
       Person p = new Person;
       // weil der Rückgabetyp Person ist, kann direkt print aufgerufen werden
       p.setName("Juergen Klopp").print();

       // ist das Gleiche, wie das
       p = p.setName("Juergen Klopp");
       p.print();
   }
}
```

Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Information Hiding
- ✓ Klassenvariable und Klassenmethoden
- ✓ Die this-Referenz
- ✓ Initialisierung von Datenfeldern

Instantiierung von Klassen

Freigabe von Speicher

Die Klasse Object

Initialisierung von Datenfeldern I

- Default-Initialisierungen von Datenfeldern

Typ	Default-Wert
boolean	false
char	'\u0000'
byte	0
short	0
int	0
long	0
float	0.0f
double	0.0d
Referenztyp	null

- In Java werden Klassenvariablen und Instanzvariablen, d.h. klassen- und objektbezogene Datenfelder, automatisch mit **Default-Werten(Standard-Werten)** initialisiert.
- Lokale Variablen nicht, diese müssen von Hand initialisiert werden

Initialisierung von Datenfeldern II

- Manuelle Initialisierungen von Datenfeldern
  - Klassenvariablen werden beim Laden der Klasse initialisiert.
  - Instanzvariablen werden beim Erzeugen eines Objektes initialisiert.

```
public class Punkt2 {
   // Manuelle Initialisierung von anzahl koennte entfallen, da der
   // Default-Wert auch 0 ist.
   private static int anzahl = 0;
   private int x;                  // Der Default-Wert ist 0
   private int y = 1;              // Manuelle Initialisierung

   public void print(){
     System.out.println ("Die Koordinaten des Punktes sind:");
     System.out.println ("x = " + x + ", y = " + y);
   }

   public static void main (String[] args){
      System.out.println ("Anzahl der Punkte: " + anzahl);
      Punkt2 p1 = new Punkt2();   // Anlegen eines Punkt-Objektes
      p1.print();
      anzahl++;    // Eine bessere Loesung wird spaeter gezeigt
      System.out.println ("Anzahl der Punkte: " + anzahl);
   }
}
```

Initialisierung von Datenfeldern III

Initialisierung mit einem Initialisierungsblock

```
1  // Datei: StaticBlockTest.java
2
3  class StaticBlock
4  {
5     // Zaehlt die erzeugten Objekte von der Klasse.
6     public static int anzahl = 0;
7     // Anzahl der Aufrufe des statischen Initialisierungsblocks.
8     public static int anzahlAufrufeStaticBlock = 0;
9
10    static
11    {
12       System.out.println ("* Betreten des statischen Blocks *");
13       anzahlAufrufeStaticBlock++;
14    }
15 }
16
17 public class StaticBlockTest
18 {
19    public static void main (String[] args)
20    {
21       StaticBlock objekt1 = new StaticBlock();
22       StaticBlock.anzahl++;
23       System.out.println ("Anzahl erzeugter Objekte: "
24                           + StaticBlock.anzahl);
25       System.out.println ("Aufrufe stat. Initialisierungsblock: "
26                           + StaticBlock.anzahlAufrufeStaticBlock);
27
28       StaticBlock objekt2 = new StaticBlock();
29       StaticBlock.anzahl++;
30       System.out.println ("Anzahl erzeugter Objekte: "
31                           + StaticBlock.anzahl);
32       System.out.println ("Aufrufe stat. Initialisierungsblock: "
33                           + StaticBlock.anzahlAufrufeStaticBlock);
34    }
35 }
36
```

Initialisierung von Datenfeldern IV

Konstruktoren zur Initialisierung

- Da ein Konstruktor automatisch nach der Allokierung des Speicherplatzes für ein Objekt aufgerufen wird, wird
  - in der Regel die Initialisierung im Konstruktor durchgeführt
  - Und die Anzahl der erzeugten Objekte am besten auch im Konstruktor hochgezählt.

- Ein Konstruktor
  - Dient zum Initialisieren eines Objektes,
  - Unterscheidet sich von einer normalen Methode unter anderem dadurch, dass der Konstruktor ***ohne Rückgabewerte*** (auch nicht *void*) deklariert wird
  - Unterscheidet sich weiterhin von einer normalen Methode auch dadurch, dass er nicht an eine abgeleitete Klasse vererbt wird,
  - Wird vom Compiler dadurch erkannt, dass er den gleichen Namen trägt, wie die Klasse selbst

Initialisierung von Datenfeldern V

Beispiel zur Initialisierungsreihenfolge

```java
// Datei: Person5.java

public class Person5
{
   private String vorname = "Rainer";
   private String name = "Brang";
   private int alter = 25;

   // Dies ist ein selbst geschriebener Default-Konstruktor
   public Person5()
   {
      System.out.print ("Felder beim Eintritt in den ");
      System.out.println ("Konstruktor:");
      print();
      vorname = "Franz";
      name = "Mueller";
      alter = 35;
   }

   public void print()
   {
      System.out.println ("Vorname:  " + vorname);
      System.out.println ("Name:     " + name);
      System.out.println ("Alter:    " + alter);
   }
}
```

```java
// Datei: TestPerson5.java

public class TestPerson5
{
   public static void main (String[] args)
   {
      Person5 p1 = new Person5();
      System.out.println ("Felder nach dem Konstruktoraufruf: ");
      p1.print();
   }
}
```

Initialisierung von Datenfeldern VI

Aufruf eines Konstruktors im Konstruktor

- Mit Hilfe von *this (parameterliste)* kann aus einem Konstruktor ein anderer Konstruktor der gleichen Klasse aufgerufen werden. Diese Anweisung muss allerdings die erste Anweisung im Rumpf des Konstruktors sein.

Arbeitsteilung zwischen new-Operator und Konstruktor bei einer Aggregation

- Mit dem *new-**Operator*** werden für ein Objekt einer aggregierenden Klasse nur die **Referenzen auf die aggregierten Objekte angelegt.** Im **Konstruktor** werden dann die **aggregierten Objekte erzeugt** und die Referenzen auf die erzeugten aggregierten Objekte den Referenzen des aggregierenden Objektes zugewiesen.

Initialisierung von Datenfeldern VII

- Konstruktor mit Parametern
  - wie Methoden können Konstruktoren mit formalen Parametern versehen werden
  - können wie Methoden überladen werden

```
public class Bruch {

   private int zaehlerfeld;
   private int nennerFeld;

   public Bruch(int zaehler, int nenner){

      zaehlerFeld = zaehler;
      nennerFeld = nenner;
   }

   public void print(){
     …
   }
}
```

```
Bruch refBruch = new Bruch(2, 4);
```

Initialisierung von Datenfeldern VIII

- Defaultkonstruktor
  - wird automatisch für jede Klasse vom Compiler zur Verfügung gestellt (wenn kein Konstruktor im Quellcode steht, kommt der Default-Konstruktor zum Einsatz)
  - parameterloser Konstruktor
  - sobald ein selbst geschriebener Konstruktor verwendet wird, steht der Default-Konstruktor nicht mehr zur Verfügung

```
public class Bruch {

   // hier gibt es keinen selbst geschriebenen Konstruktor!
   private int zaehlerfeld = 2;
   private int nennerFeld = 4;

   public void print(){
      …
   }
}
```

```
// durch new wird der Default-Konstruktor verwendet
Bruch refBruch = new Bruch();
```

Initialisierung von Datenfeldern IX

- Beispiel: Konstruktoren werden hierbei überladen

```
public class Person
   private String vorname;
   private String name;

   public Person (String v, String n){
      System.out.println ("Im Konstruktor mit Parametern!");
      System.out.println ("   Name:     " + n);
      System.out.println ("   Vorname: " + v);
      vorname = v;
      name = n;
   }

   public Person(){
      // dieser Aufruf MUSS die erste Anweisung im Konstruktor sein
      this ("Vorname unbekannt", "Nachname unbekannt");
      System.out.println ("Im parameterlosen Konstruktor!");
   }

   public void print(){
      System.out.println ("Ausgabe der print()-Methode");
      System.out.println ("   Name:     " + name);
      System.out.println ("   Vorname: " + vorname);
   }
}
```

Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Information Hiding
- ✓ Klassenvariable und Klassenmethoden
- ✓ Die this-Referenz
- ✓ Initialisierung von Datenfeldern
- ✓ Instantiierung von Klassen

Freigabe von Speicher

Die Klasse Object

Instantiierung von Klassen I

Ablauf der Instantiierung

Person p1 = new Person();

Ablauf:

- In **Schritt 1** wird die Referenzvariable *p1* vom Typ *Person* **angelegt** und mit *null* initialisiert.
- In **Schritt 2** wird durch *new person* der *new-Operator* aufgerufen und die **Klasse** *Person* **instantiiert,** mit anderen Worten, es wird ein Objekt der Klasse *Person* auf dem Heap erzeugt.
- Schließlich erfolgt **Schritt 3** die **Initialisierung des Objektes.** Es werden Default-Initialisierungen der Instanzvariablen durchgeführt und dann eventuell angegebene manuelle Initialisierungen und Initialisierungen durch einen nicht statischen Initialisierungsblock. Anschließen wird der Konstruktor aufgerufen.
- In **Schritt 4** gibt der *new-Operator* eine **Referenz** auf das neu im Heap erzeugte Objekt zurück, welche **Der Referenzvariablen** *p1* **zugewiesen wird.**

Instantiierung von Klassen II

- Singleton Entwurfsmuster Verhindern der Instantiierung einer Klasse

```
1  // Datei: Test.java
2
3  class Singleton
4  {
5     private static Singleton instance;
6
7     private Singleton()
8     {
9        System.out.println ("Bin im Konstruktor");
10    }
11
12    public static Singleton getInstance()
13    {
14       if (instance == null)
15       {
16          instance = new Singleton();
17       }
18       return instance;
19    }
20 }
21
22 public class Test
23 {
24    public static void main (String[] args)
25    {
26       // Singleton s = new Singleton(); gibt Fehler
27       Singleton s2 = Singleton.getInstance();  // new wird
28                                                // aufgerufen
29       Singleton s3 = Singleton.getInstance();  // new wird nicht
30                                                // mehr aufgerufen
31    }
32 }
33
```

Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Information Hiding
- ✓ Klassenvariable und Klassenmethoden
- ✓ Die this-Referenz
- ✓ Initialisierung von Datenfeldern
- ✓ Instantiierung von Klassen
- ✓ Freigabe von Speicher

Die Klasse Object

Freigabe von Speicher

- Bei einer Speicherbereinigung werden die nicht referenzierten Objekte auf dem Heap entfernt. Mit anderen Worten, ihr Platz wird zum Überschreiben freigegeben.

Lässt man im Beispiel

Person p1 = new Person ();

Durch

p1 = null;

die Referenz *p1* nicht länger auf das mit *new* geschaffene Objekt, sondern auf *null* zeigen, so wird damit vom Programmierer explizit das Objekt im Heap zur Speicherbereinigung freigegeben - vorausgesetzt, es existiert keine weitere Referenz auf dieses Objekt.

Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Information Hiding
- ✓ Klassenvariable und Klassenmethoden
- ✓ Die this-Referenz
- ✓ Initialisierung von Datenfeldern
- ✓ Instantiierung von Klassen
- ✓ Freigabe von Speicher
- ✓ Die Klasse Object

Die Klasse Object

Die Methoden der Klasse *Object* lassen sich in zwei Kategorien einteilen:

- In Methoden, die Threads unterstützen
- Und in allgemeine Utility-Methoden.

➢ Hier werden nur die Utility-Methoden aufgeführt:

- *Public String toString()*
- *Public boolean equals (Object ojb)*
- *Protected Object clone() throws CloneNotSupportedException*
- *Protected void finalize() throws Throwable*

Aufgaben

Aufgabe 07.01 Konstruktor und this

Schreiben Sie eine Klasse Film mit den drei Instanzvariablen *private String titel, private String figur und private int jahr*. Implementieren Sie zwei Konstruktoren:

1) Konstruktor mit zwei String-Parametern namens titel und figur und einem int-Parameter namens jahr zum Initialisieren der drei Instanzvariablen
2) parameterloser Konstruktor, der den anderen Konstruktor mit den drei Werten „kein Titel“, „keine Figur“ und 0 aufruft.

Implemetieren Sie eine Testklasse TestFilm, die zwei Film-Objekte mit den Parametern („Herr der Ringe“, „Gandalf“, 2001) und („Star Trek“, „Mr. Spock“, 1966) erzeugt. Ein drittes Film-Objekt soll ohne Parameter erzeugt werden.

Rufen Sie für jedes erzeugte Film-Objekt dessen print()-Methode auf, um die Werte der Instanzvariablen auszugeben.

Aufgaben

Aufgabe 07.02 Kontovergleich

Importieren Sie die beiden Codevorlagen Konto.java und KontoTest.java aus dem Online-Campus in ein Eclipse-Projekt. Ergänzen Sie diese Programmfragmente. Fehlende Stellen sind durch . . . . . markiert.

Es soll eine Klasse Konto entwickelt werden. Ein Konto beinhaltet den Kontostand und wird einer Person zugeordnet. Um eine Person zu beschreiben, wird die Klasse Person mit den Instanzvariablen name und vorname verwendet. Die Klasse KontoTest wird zum Testen der Klasse Konto verwendet. Sie beinhaltet neben der main()-Methode eine weitere Klassenmethode mit der Bezeichnung kontoVergleich(). Dieser Methode werden zwei Objekte der Klasse Konto übergeben. Die übergebenen Konten werden dann auf die Höhe des Kontostands hin verglichen und das Ergebnis auf dem Bildschirm ausgegeben.

Aufgaben

Aufgabe 07.03 (Teil 1) Klassenvariable und Klassenmethoden für Kinos

Ein Kinobesitzer möchte seine Kinosäle in einem Informationssystem halten können. Hierzu sind die Klassen Kinosaal und TestKinosaal zu entwickeln. Die Klasse Kinosaal besitzt folgende Eigenschaften:

- einen parameterlosen Konstruktor,
- die beiden Instanzvariablen saalNummer und anzahlSitzplaetzeSaal,
- die beiden Klassenvariablen anzahlSitzplaetzeKino und anzahlKinosaele,
- eine get()- und set()-Methode, um die Anzahl der Sitzplätze eines Saals auszulesen bzw. festzulegen,
- die beiden Klassenmethoden getAnzahlSitzplaetzeKino() und getAnzahlKinosaele().

Die Klasse TestKinosaal ist eine Wrapper-Klasse für die Methode main(). In dieser Methode soll die Klasse Kinosaal getestet werden. Hierzu sollen zwei Kinosäle mit 50 bzw. 100 Sitzplätzen angelegt werden. Alle Variablen sollen vom Typ int und private sein.

Aufgaben

Aufgabe 07.03 (Teil 2) Klassenvariable und Klassenmethoden für Kinos

a) Schreiben Sie die Klasse Kinosaal. Bei jedem Erzeugen eines Kinosaals soll der Wert der Variablen anzahlKinosaele um 1 erhöht werden. Jeder Kinosaal soll beim Erzeugen eine eindeutige Nummer saalNummer erhalten, die direkt aus der Anzahl der Kinosäle abgeleitet wird. Mit der Methode public void setAnzahlSitzplaetzeSaal (int anzahlSitzplaetzeSaal) soll für einen neu erzeugten Kinosaal die anzahlSitzplaetzeSaal gesetzt werden. Dabei soll die anzahlSitzplaetzeKino um den Wert anzahlSitzplaetzeSaal erhöht werden.

b) Schreiben Sie die Methode setAnzahlSitzplaetzeSaal() aus Teilaufgabe a) so um, dass die Anzahl der Sitzplätze eines Kinosaals nachträglich geändert werden kann und die Anzahl der Sitzplätze des Kinos entsprechend angepasst wird.

Aufgaben

Aufgabe 07.04 Bundesligatabelle und Deutscher Meister

Recherchieren Sie die aktuelle deutsche Fußball-Bundesligatabelle und legen Sie für die ersten fünf Mannschaften jeweils ein Objekt der Klasse Team an. Die Klasse Team enthält eine statische Klassenvariable punkteGesamt (int) und die Instanzvariablen verein (String), punkte (int) und tordifferenz (int). Die Instanzvariablen sollen über einen Konstruktor initialisiert werden. Beim Anlegen eines Team-Objektes muss die statische Variable punkteGesamt um die jeweilige Punktezahl erhöht werden.

Verwalten Sie die fünf Team-Objekte in einem Array und ermitteln Sie den momentanen deutschen Meister (hat die meisten Punkte, bei Punktgleichheit entscheidet die bessere Tordifferenz). Geben Sie den Vereinsnamen, die Punkte und die Tordifferenz des deutschen Meisters über die Systemausgabe aus.

Aufgaben

Aufgabe 07.05

Schreiben Sie eine Bier-Verwaltungssoftware für einen Brauereimeister. Legen Sie dazu folgende Klasse Bierfass an:

- private Instanzvariablen: String sorte, int liter, int fassId
- private Klassenvariablen: int anzahlFaesser, int literGesamt
- Konstruktor:

  Der Konstruktor erhält die Sorte und die Liter als Parameter übergeben und initialisiert die gleichnamigen Instanzvariablen. Die Klassenvariable anzahlFaesser ist zu inkrementieren, der neue Wert wird als fassId für das jeweilige Bierfassobjekt verwendet.

Implementieren Sie außerdem eine Methode namens zapfen(int liter), welche die übergebene Anzahl von Litern von den Variablen liter und literGesamt abzieht und eine print-Methode, die alle Informationen des jeweiligen Bierfasses ausgibt.

Verwenden Sie zum Testen der Bierfass-Klasse die Datei BierTest.java aus dem OC.