Programmierkurs Anwendungsentwicklung

Der Beginn ...

Prof. Dr. Martin Hirsch

25. September 2023

Dozent

Prof. Dr. Martin Hirsch

- Professor für Softwaretechnik an der FH Dortmund
- Module: PK-A, SWT-A, SWT-B
- Kontakt
  - Email: martin.hirsch@fh-dortmund.de
  - Büro: C.3.49, Online
  - Sprechstunde: Individuell, Termine bitte per Email vereinbaren

Unterlagen

ILIAS

- `https://www.ilias.fh-dortmund.de/ilias/goto_ilias-fhdo_crs_206138.html`
- Unterlagen sind vor einer Veranstaltung im ILIAS. Ggf . werden die Unterlagen nach einer Veranstaltung aktualisiert (Annotationen im PDF)

Hinweis

Alle Materialien des Kurses dürfen nur mit der schriftlichen Erlaubnis des Dozenten an Dritte weitergegeben werden! Missachtung dieses Hinweises ist eine Verletzung des Urheberrechts.

Farblegende:

Normal

Hinweis

Beispiele

Zeiten und Räume

Vorlesung (ab 25.09.2023)

- Zeit: Montags 12:00 Uhr - 13:30 Uhr
- Raum: A.E.02

Praktikum (ab 25.09.2023)

- Gruppe 1: Montags 14:15 Uhr - 15:45 Uhr, Raum: C.1.31
- Gruppe 2: Dienstags 14:15 Uhr - 15:45 Uhr, Raum: C.E.32

Bitte Gruppenbuchstabe berücksichtigen

Modulhandbuch

Modulbeschreibung

Pflichtmodul | Programmierkurs

SG | INDB Informatik Dual Bachelor| VR | ST Softwaretechnik
TB | INDB-TB120 Programmierkurs

Kennnummer	Prüfungsnummer	Workload	Credits	Studiensemester	Häufigkeit des Angebots	Dauer	Lehrsprache
INDB-43023	43023	150 h	5.0 LP	5. Sem.	jährlich	1 Sem.	deutsch

Lehrveranstaltungen Vorlesung: 2 SWS
Praktikum: 2 SWS

https://www.fh-dortmund.de/hochschule/informatik/ueber-den-fachbereich/modulhandbuch.php

Worauf baut die Vorlesung auf?

Welche Ausgabe? zuverlässige 50% haben dies in den letzten Jahren in der Modulprüfung immer falsch gemacht...

```
public class Test {
  public static void main(String [] args) {
    B b = new B();
  }
}
public class A {
  public A() {
    System.out.println("Konstruktor von A");
  }
}
public class B extends A {
  public B() {
    System.out.println("Konstruktor von B");
  }
}
```

Worauf baut die Vorlesung auf?

Voraussetzungen

- Semester 1 + 2
- Insbesondere „Einführung in die Programmierung“ und „Algorithmen und Datenstrukturen“
- Gesundem Menschenverstand!

Modulprüfung

Klausur

- Termin: xx.2.2024 (bitte ggf. Änderungen im Prüfungsplan beachten)
- Dauer: 120 min, keine Hilfsmittel

Für die Modulprüfung relevante Inhalte

- Vorlesung UND Praktikum
- Folien + Annotationen
- Beispielprogramme, Aufgabentypen, Abfrage von Definitionen, Verständnisfragen, Korrektur und Modifikation von vorgegebenen Programmen, Schreiben von Programmen

Praktikum

Achtung

- Keine Bonuspunkte
- In der Klausur wird es einen Aufgabenblock geben, der einen Bezug zu Vorlesungsbeispielen und/oder den Praktikumsaufgaben hat
- Es wird keine Musterlösungen zu den Praktikumsaufgaben geben

Inhalte laut Modulhandbuch

Modulhandbuch

- **Einführung in die Programmiersprachen C, Scala; Vertiefung in Java**
- Vergleich prozeduraler, objektorientierter und funktionaler Programmierkonzepte
- Programmstrukturierung
- Variablen, Zeiger und Referenzen
- (Dynamische) Speicherverwaltung
- Typkonvertierung
- Konstruktoren und Destruktoren / Ressourcenverwaltung

Modulhandbuch

- Überladen/Überschreiben von Operatoren
- Ausnahmebehandlung
- Abstrakte Klassen und Schnittstellen
- Mehrfachvererbung
- Generische Programmierung und Templates
- Funktionale Programmierung
- Multithreading, Concurrency und paralleles Programmieren in Java

Geplante Inhalte

Zeitplan (Änderungen vorbehalten)

VL	Thema
25.09.23	Organisation / Motivation / Wiederholung Grundlagen
02.10.23	Generics
09.10.23	frei (Herbstferien)
16.10.23	Exceptions + Lambda-Ausrücke
23.10.23	Collections
30.10.23	Streams
06.11.23	Dateien und Verzeichisse + Serialisierung
13.11.23	Threads + Parallelprogrammierung in Java: Das Concurrent-Paket)
20.11.23	frei (Blockwoche)
27.11.23	JavaFX + Properties + Bad Smells
04.12.23	Einführung Scala
11.12.23	Scala I
18.12.23	Einführung C
25.12.23	Weihnachtsferien
01.01.24	Weihnachtsferien
08.01.24	Grundlagen I C
15.01.24	Zusammenfassung + Fragen

Literatur

Java

[Ind20] Michael Inden. *Der Weg zum Java-Profi - Konzepte und Techniken für die professionelle Java-Entwicklung*. Heidelberg: dpunkt.verlag, 2020. ISBN: 978-3-960-88843-7.

[Kof22] Michael Kofler. *Java - der Grundkurs*. Bonn: Rheinwerk Computing, 2022. ISBN: 978-3-836-28392-2.

[Ora23] Oracle, Hrsg. *Oracle Java documentation: Java Platform, Standard Edition & Java Development Kit - Version 20 API Specification*. 2023. URL: https://docs.oracle.com/en/java/javase/20/docs/api/index.htmll (besucht am 30. 08. 2023).

Literatur

Scala

[Sfr21] Daniela Sfregola. *Get Programming with Scala* -. New York: Simon und Schuster, 2021. ISBN: 978-1-638-35225-9.

C

[KR88] Brian W. Kernighan und Dennis M. Ritchie. *The C Programming Language* -. New York: Prentice-Hall, 1988. ISBN: 978-8-120-30596-0.

[KS10] Robert Klima und Siegfried Selberherr. *Programmieren in C* -. Berlin Heidelberg New York: Springer-Verlag, 2010. ISBN: 978-3-709-10393-7.

Was wird von Ihnen erwartet?

- Die vorliegende Präsentation reicht nicht aus, um den Stoff des Moduls „Programmierkurs Anwendungsentwicklung“ zu erlernen!
- Neben dem Durcharbeiten der Präsentation empfehle ich Ihnen
  1. den aktiven Besuch der Vorlesung
  2. die aktive Teilnahme am Praktikum und Bearbeitung der Praktikumsaufgaben
  3. das Durcharbeiten der Literatur
  4. Programme selber!!! zu schreiben
  5. Punkt 4) immer und immer wieder zu wiederholen!!!
- Fragen, Anregungen und konstruktives Feedback sind immer willkommen!!!

Bitte

- Wer „aussteigen“ will/möchte, wird gebeten, vertraulich Rücksprache mit mir zu halten! $\Rightarrow$ Wir finden eine Lösung!

Einstieg Java - Pitfalls

Programmiersprachen verstehen . . .

Welche Ausgabe?

```
public class Example1 {
  public static void main(String[] args) {
    Integer a127 = 127;
    Integer b127 = 127;
    Integer a128 = 128;
    Integer b128 = 128;
    if (a127 == b127) {
      System.out.println("127 = 127");
    }
    if (a128 == b128) {
      System.out.println("128 = 128");
    }
  }
}
```

APIs und Quellcode lesen

Lösung

256 Integer objects are created in the range of -128 to 127 which are all stored in an Integer array. This caching functionality can be seen by looking at the inner class, IntegerCache, which is found in Integer:

```
private static class IntegerCache
{
  private IntegerCache(){}

  static final Integer cache[] = new Integer[-(-128) + 127 + 1];

  static
  {
    for(int i = 0; i < cache.length; i++)
    cache[i] = new Integer(i - 128);
  }
}

public static Integer valueOf(int i)
{
       final int offset = 128;
       if (i >= -128 && i <= 127) // must cache
       {
           return IntegerCache.cache[i + offset];
       }
       return new Integer(i);
}
```

So when creating an object using Integer.valueOf or directly assigning a value to an Integer within the range of -128 to 127 the same object will be returned. Therefore, consider the following example:

Mit Bedacht programmieren...

Welche Ausgabe?

```
public class Example2 {
  public static void main(String[] args) throws InterruptedException {
    int END = Integer.MAX_VALUE;
    int START = END - 100;
    int count = 0;
    for (int i = START; i <= END; i++) {
      count++;
      System.out.println(i);
      Thread.sleep(100);
    }
    System.out.println(count);
  }
}
```

Mit Bedacht programmieren...

Effizient?

```
public class Example3 {
  public static void main(String[] args) {
    String oneMillionHello = "";
    for (int i = 0; i < 1000000; i++) {
      oneMillionHello = oneMillionHello + "Hello!";
    }
    System.out.println(oneMillionHello.substring(0, 6));
  }
}
```

Mit Bedacht programmieren. . .

Besser . . . mit der richtigen API

```
public class Example3_lsg {
  public static void main(String[] args) {
    StringBuilder oneMillionHelloSB = new StringBuilder();
    for (int i = 0; i < 1000000; i++) {
      oneMillionHelloSB.append("Hello!");
    }
    System.out.println(oneMillionHelloSB.toString().substring(0, 6));
  }
}
```

Wiederholung Java

Check

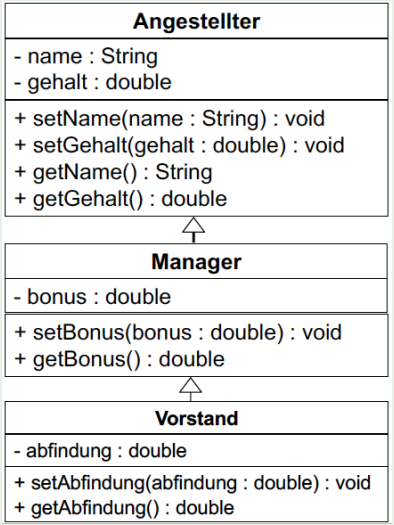
OOP

- Welche Konzepte gibt es?

Grundlagen Java

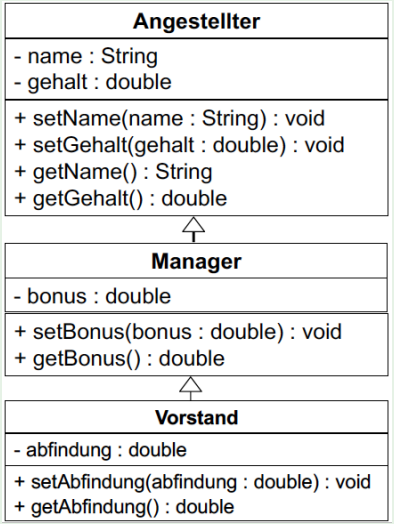
- Klassen (Top-Level-Klasse, Innere Klassen, Anonyme Klassen, Statische geschachtelte Klassen, ...)
- Objekte (Konstruktor, ...)
- Vererbung (Überschreiben, Überladen, Abstract, Final, Polymorphie ...)
- Interfaces
- Speicherverwaltung (Stack, Heap)
- „equals()“()

Beispiel: Personalverwaltung




```
public class Angestellter {
  private String name;
  private double gehalt;
  public Angestellter(String name,
      double gehalt) {
    this.name = name;
    this.gehalt = gehalt;
  }
  public String getName() {
    return name;
  }
  public double getGehalt() {
    return gehalt;
  }
  public void setName(String name) {
    this.name = name;
  }
  public void setGehalt(double gehalt) {
    this.gehalt = gehalt;
  }}
```

Beispiel: Personalverwaltung




```
public class Manager extends
    Angestellter {
  private double bonus;
  public Manager(String name, double
      gehalt, double bonus)
  {
    super(name, gehalt);
    this.bonus = bonus;
  }
  public double getBonus()
  {
    return bonus;
  }
}
```

Beispiel: Personalverwaltung

Main

```
public class TestKlasse {
  public static void main(String[] args) {
    Manager m1 = new Manager("Bill Gates", 100000.00, 50000.00);
    Angestellter a1 = new Angestellter("Meier", 5000.00);
    System.out.println(m1.getName() +" bekommt " + m1.getGehalt() + m1.getBonus());
    System.out.println(a1.getName() + " bekommt " + a1.getGehalt());
  }
}
```

Beispiel: Personalverwaltung

Klasse Angestellter wird um Methode berechneJahreszahlung() erweitert.

```
double berechneJahreszahlung(){
  return 12  * gehalt;
}
```

Problem

Klasse Manager erbt auch diese Methode. Allerdings ist die Berechnung für einen Manager nicht korrekt, da der Bonus nicht berücksichtigt wird.

In Klasse Manager wird Methode berechneJahreszahlung() überschrieben.

```
double berechneJahreszahlung(){
  return super.berechneJahreszahlung() + bonus;
}
```

Beispiel: Personalverwaltung

Es wird automatisch die Methode der zugehörigen Klasse ausgeführt.

```
public class TestKlasse {
  public static void main(String[] args) {
    Manager m1 = new Manager("Bill Gates", 100000.00, 50000.00);
    Angestellter a1 = new Angestellter("Meier", 5000.00);
    System.out.println(m1.getName() + " " + m1.berechneJahreszahlung());
    System.out.println(a1.getName() + " " + a1.berechneJahreszahlung());
  }
}
```

Wichtig

Überschreiben

In einer Vererbungshierachie können Methoden mit gleicher Methodensignatur überschrieben werden

Überladen

In einer Klassen können Methoden mit gleichem Methodennamen durch unterschiedliche Parameterlisten überladen werden

Final

Wird in einer Vererbungshierachie eine Klasse als finale deklariert, so kann von dieser Klasse nicht mehr geerbt werden. Wird eine Methoden als final deklariert, so kann diese Methoden nicht überschrieben werden.

Equals

Referenzielle Gleichheit. Welche Ausgabe?

```
Angestellter a = new Angestellter ("Meier", 5000.00);
Angestellter b = a;
Angestellter c = new Angestellter("Meier", 5000.00);
System.out.println("1. " + (a == b));
System.out.println("2. " + (a == c));
System.out.println("3. " + (a.equals(c)));
```

Equals

fachliche Gleichheit

- Entscheidung, ob zwei Objekte mit den gleichen Attributwerten gleich sind.
- Die von Object geerbte Standardimplementierung für equals nutztdie Gleichheit der Referenzen ($==$) und führt damit nicht zumgewünschten Verhalten
- Für die zugehörige Klasse muss also die equals-Methode überschrieben werden

Equals

Die Java Language Specification fordert für jede equals-Methodedie Einhaltung der folgenden (natürlichen) Eigenschaftent

1. reflexiv: x.equals(x) liefert true
2. symmetrisch: x.equals(y) ist gleich y.equals(x)
3. transitiv: Falls für gegebene Referenzen x, y und z gilt, dass x.equals(y) den Wert true liefert und y.equals(z) den Wert true liefert, dann muss auch x.equals(z) den Wert true liefern
4. konsistent: Falls sich die von x und y referenzierten Objekte nichtändern, dann soll ein wiederholter Aufruf von x.equals(y) zu einem identischen Ergebnis führen
5. Für jede nicht-null Referenz x soll x.equals(null) den Wert false liefern

Equals

Es empfiehlt sich, bei der Implementierung von equals die folgende Reihenfolge der Einzelprüfungen einzuhalten

1. Prüfen, ob die Objektreferenzen identisch sind
2. Prüfen, ob die übergebene Referenz null ist
3. Prüfen, ob die beiden Klassen identisch sind
   - mit der Methode getClass(), falls Unterklassen eine eigene Definition der Gleichheit besitzen
   - mit dem Operator instanceof(), falls die Oberklasse die Gleichheit für alle Unterklassen definiert
4. Typkonvertierung des übergebenen Parameters
5. Vergleich aller relevanten Attribute der beiden Objekte

Polymorphismus

- Polymorphismus bedeutet Vielgestaltigkeit
- Polymorphismus erlaubt eine sehr elegante Programmierung und sorgt für eine gute Erweiterbarkeit der Programme
- Die abgeleitete Klasse (subclass) kann überall da genutzt werden, wo auch die Basisklasse (superclass) erlaubt ist!

Eine Referenz vom Typ Angestellter kann auf ein Objekt der Klasse Angestellter oder auf ein Objekt einer Unterklasse von Angestellter verweisen

```
Angestellter chef = new Manager("Bill Gates", 100000.00, 50000.00);
System.out.println(chef.getName() + "  "+ chef.berechneJahreszahlung());
```

Dynamisches Binden

```
Angestellter[] belegschaft = new Angestellter[2];
belegschaft[0] = new Angestellter("James Gosling", 60000.00);
belegschaft[1] = new Manager("Bill Gates", 100000.00, 50000.00);
for (int i = 0; i < belegschaft.length; i++)
  System.out.println(belegschaft[i].getName() + "  "+ belegschaft[i].berechneJahreszahlung
      ());
```

- Keine Fallunterscheidung erforderlich, es wird automatisch die richtige Methode ausgeführt
- Die Schleife funktioniert unverändert, wenn neue, von Angestellter abgeleitete Klassen in die Belegschaft aufgenommen werden

Dynamisches Binden

Achtung: Die Basisklasse darf nicht für abgeleitete Klassen eingesetzt werden

```
Manager m = new Angestellter("James Gosling", 60000.00);
```

- Für Manager gibt es z.B. die Funktion getBonus(). Ein Objekt vom Typ Angestellter kennt diese Methode aber nicht!
- Da der Compiler nur den Typ der Referenz überprüft, könnte zur Laufzeit m.getBonus() aufgerufen werden
- Das Objekt vom Typ Angestellter kennt diese Methode aber nicht (Laufzeitfehler)

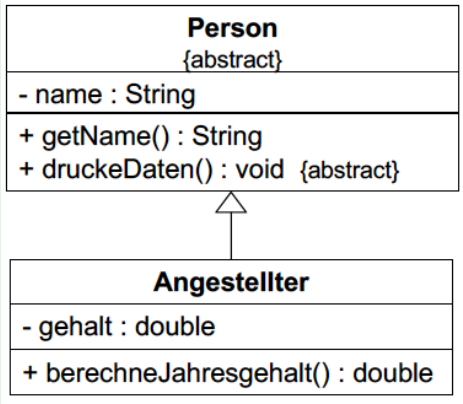
Dynamisches Binden

Zusammenfassung

- Eine Variable vom Typ Angestellter kann eine Referenz auf einen Angestellten, einen Manager oder einen Vorstand enthalten
- Zur Compilezeit kann daher nicht bestimmt werden, welche Methode ausgeführt werden muss
- Die JVM muss daher zur Laufzeit die korrekte Methode suchen
- Sie startet die Suche bei der speziellsten Klasse und geht in Richtungder allgemeineren Klassen, bis sie eine Definition der Methode findet
- Dies erfordert zusätzlichen Aufwand zur Laufzeit
- Dafür sind Programme erweiterbar, ohne dass bereits bestehender Quellcode neu übersetzt werden muss

Beispiel: Personalverwaltung

Abstrakte Klasse



Beispiel: Personalverwaltung

```
public abstract class Person {
  private String name;
  public Person(String name){
    this.name = name;
  }
  public String getName(){
    return name;
  }
  public abstract void druckeDaten();
}
```

```
public class Angestellter extends Person
    {
  private double gehalt;
  public Angestellter(String n, double g
      ){
    super(n);
    gehalt = g;
  }
  public double berechneJahresgehalt(){
    return 12 * gehalt;
  }
  public void druckeDaten(){
    System.out.println(getName() + "
        verdient "+ berechneJahresgehalt
        ());
  }
}
```

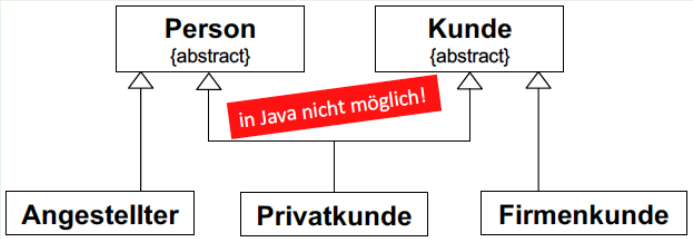
Beispiel: Personalverwaltung

Abstrakte Klasse

Aus einer abstrakten Klasse können keine Objekte erzeugt werden. Eine abstrakte Klasse kann aber durchaus als Referenztyp genutzt werden!

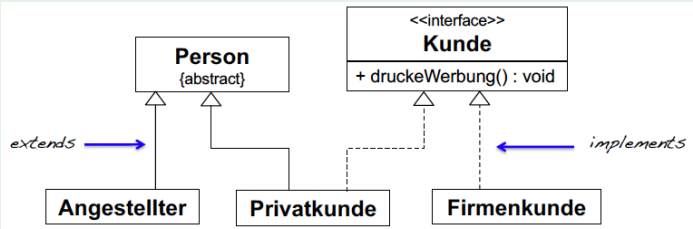
Beispiel: Personalverwaltung

Mehrfachvererbung



Beispiel: Personalverwaltung

Interface



Beispiel: Personalverwaltung

```
public interface Kunde{
  void druckeWerbung();
}
```

Die Methoden einer Schnittstelle sind immer öffentlich und abstrakt

```
public class Privatkunde extends Person
    implements Kunde{
  // Attribute
  // Methoden (Konstruktor und
      druckeDaten von Privatkunde)
  public void druckeWerbung(){
    System.out.println("Lieber Kunde ...
        ");
  }
}
```

Die Methode der Schnittstelle Kunde muss implementiert werden

Dynamisches Binden

Nur Methoden der Schnittstelle können aufgerufen werden

```
Kunde[] k;
k = new Kunde[2];
k[0] = new Firmenkunde();
k[1] = new Privatkunde();
for(int i = 0; i < 2; i++)
  k[i].druckeWerbung();
```

Exkurs: Arten von Klassen

Top-Level-Klasse (TLK)

Klasse definiert auf der höchsten Ebene in einer Java-Datei

```
[modifizierer] class Name[extends K1] [implements I1, I2, I3] {
  ... (Code der Klasse)
}
```

Regeln

- Pro Java-Datei nur eine TLK
- TLK niemals privat oder protected
- extends und implements geben an, von welcher Basisklasse die aktuelle Klasse abgeleitet wird

Geschachtelte Klassen

Allgemein

- Häufig auch member classes, inner classes, nested classes oder lokale Klassen genannt
- Geschachtelte Klassen werden innerhalb einer anderen Klasse definiert
- Auch die Definition innerhalb einer Methode oder sogar innerhalb einer beliebigen {}-Klammernebene ist erlaubt $\rightarrow$ lokale Klasse
- werden in der Regel dazu benutzt, um eng abgrenzbare Funktionen, die nur innerhalb einer TLK oder sogar nur innerhalb einer bestimmten Methode benötigt werden, objektorientiert zu formulieren und zu kapseln

Geschachtelte Klassen

Klasse definiert auf der höchsten Ebene in einer Java-Datei

```
public class TopLevel {
  public class Nested {
    public void m1 (int n){
      if (n<0) {
        class Local { ... }
        int m = 3;
        Local l = new Local ();
}}}}
```

Regeln

- innerhalb der geschachtelten Klasse dürfen keine statischen Variablen oder Methoden definiert werden
- innerhalb der geschachtelten Klasse können Variablen und Methoden der TLK verwenden werden

Anonyme Klassen

Anonyme Klassen sind geschachtelte Klassen ohne Namen

```
Arrays.asList (ar).forEach(new Consumer<String>() {
  public void accept(String t) {
    System.out.print(t);
  }
} );
```

Regeln

- werden eingesetzt, wenn eine geschachtelte Klasse nur einmal benötigt wird
- können an jeder Stelle im Java-Code definiert werden
- Quasi eine „Wegwerf-Klasse“

Statisch geschachtelte Klassen

Regeln

- Statische geschachtelte Klassen können nur direkt innerhalb einer TLK formuliert werden
- Statische geschachtelte Klassen können wie TLK verwendet und instanziiert werden, also auch außerhalb der TLK!
- Statische geschachtelte Klassen können nur auf statische Elemente der TLK zurückgreifen
- Sinn: inhaltlich verwandte, aber formal unterschiedliche Klassen in einer einzigen Code-Datei unterzubringen

Statisch geschachtelte Klassen

Beispiel

```
// Definition einer Top-Level-Klasse
class TopLevel {
  //Definition einer statischen geschachtelten Klasse
  static class Nested {}
  ...
}
}
// Verwendung einer statischen geschachtelten Klasse
TopLevel.Nested var = new TopLevel.Nested();
```

Anwendung

https://hg.openjdk.org/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes/java/awt/geom/Point2D.java