Grundlagen der Informatik für Ingenieure I

5. Klassen, Objekte und Methoden

5.1 Klassen(classes)
- *5.1.1 Definition einer Klasse*
- *5.1.2 Definition von Objektvariablen*
- *5.1.3 Definition von Klassenvariablen*
- *5.1.4 Definition von Methoden*
- *5.1.5 Aufruf von Methoden*
- *5.1.6 Das this - Schlüsselwort*
- *5.1.7 Gültigkeitsbereiche von Variablen*
- *5.1.8 Parameterübergabe*
- *5.1.9 Klassenmethoden*

5.2 Objekte (Instanzen)
- *5.2.1 Erzeugen von Objekten; new-Operator*
- *5.2.2 Zugriff und Zuweisungen auf Klassen-/Objektvariable*
- *5.2.3 Referenzen auf Objekte*
- *5.2.4 Casting und Converting*
- *5.2.5 Relationale Operationen auf Objekte*
- *5.2.6 Zugehörigkeit eines Objekts zu einer Klasse*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Klassen, Objekte und Methoden

- *5.2.7 Java-Klassen-Bibliotheken*
- *5.2.8 Überladen von Methoden (Overloading)*
- *5.2.9 Konstruktoren (Constructors)*
- *5.2.10 Vererbung (Inheritance)*
- *5.2.11 Überschreiben von Methoden (Overriding)*
- *5.2.12 Überschreiben von Konstruktoren*

5.3 Modifiers

5.4 Zusammenfassung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5.1 Klassen (classes)

- Klassen sind die Muster (Baupläne, Templates) der (zur Laufzeit) zu erzeugenden Objekte (Instanzen).
- Das Schreiben eines Programms in Java ist gleichbedeutend mit der Definition einer Menge von Klassen.
- Beim Entwurf eines Programms werden - soweit möglich - bereits vorhandene Teilproblemlösungen als Klassen aus den verschiedenen Klassenbibliotheken benutzt (Wiederverwendbarkeit).
- Bisher haben wir nur eigene Klasse definiert und darüberhinaus andere Klassen (z.B. System, String, Array) mehr oder weniger intuitiv benutzt.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Definition einer Klasse

- Eine Klassendefinition hat folgende Form

```
[modifier] class classname {
        ....
    }
```

  - Ist die neue Klasse außerdem Unterklasse einer anderen Klasse, muss dies mit dem Schlüsselwort ***extends*** deklariert werden:

```
[modifier] class classname extends superclassname {
        ....
    }
```

  - Ein Beispiel war bereits unser “HelloWorld - Applet”:

```
public class HelloWorldApplet extends Applet {
....
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Definition von Objekt-Variablen

◆ Objekt-Variable (Instanz-Variable) sind Variable die objektglobal gültig sind

◆ Die Werte aller Objekt-Variablen bestimmen den Zustand des Objekts.

◆ Aufgrund des Gültigkeitsbereichs werden diese Variablen auf der äußersten Ebene des Klassenrumpfes definiert:

Beispiel:

```
public class DampfLokomotive extends Lokomotive {
    private String baureihe;
    private int leistung;
    private int anzahlAntriebsRaeder;
    private boolean heusingerSteuerung;
     ...
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Definition von Klassenvariablen

◆ Klassenvariable sind **gemeinsame (*shareable*)** Variable aller Objekten einer Klasse.

- **Objektvariable**:
  - Bei der Erzeugung (Instantiierung) eines neuen Objekts erhält jedes Objekt seine “eigene” Kopie der Objektvariablen.
  - Veränderung dieser Kopie haben keinen Effekt auf Objektvariable anderer Objekte.

- **Klassenvariable**:
  - **Im Gegensatz dazu** existieren **Klassenvariable** in der Klasse nur einmal.
  - Alle Objekte dieser Klasse referenzieren die **identische Variable**.
  - Sie ist also für alle Objekte einer Klasse **global**.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Definition von Klassenvariablen

◆ Eine Klassenvariable wird mit dem Schlüsselwort ***static*** vereinbart:

Beispiel:

```
public class KellyFamilyMember {
    static String familyName = "Kelly";    // Klassenvariable
    private String firstName;             // Objektvariable
    private int age;                      // Objektvariable
...
}
/*  Der Familienname ist allen Objekten der Klasse
    KellyFamilyMember gemeinsam */
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Definition von Methoden

◆ Eine **Methodendefinition** setzt sich aus folgenden Teilen zusammen:

- **Signatur** der Methode, bestehend aus:
  - **Methodenname**
  - Eine Liste von **Parametern**
- ***Modifier*** *(public, private, static, ...*- siehe Kapitel 5.3)
- ***Returntype***: Typ des Wertes den die Methode zurückgibt (Objekt oder Prim. Typ)
- ***Exceptions*** *(throw* - siehe Kapitel 10)
- **Rumpf** der Methode

◆ Eine Methodendefinition hat folgende Form:

```
[modifier] returntype methodname( type1 arg1, type2 arg2, ...) {
...
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Definition von Methoden

◆ Warum benötigt man - in Java - den Begriff der **Signatur**?

- In anderen Programmiersprachen ist der Name der
  – Funktion oder
  – Subroutine oder
  – Unterprogramm oder
  – Prozedur

  zur Identifikation ausreichend.

- In Java können verschiedene Methoden den gleichen Namen haben, aber verschiedenen Parameter (Anzahl, Typ) besitzen.

- Zur Identifikation dient die **Signatur** der Methode.

- (Näheres hierzu siehe: 5.2.8 Überladen von Methoden)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Definition von Methoden

◆ Der Typ des Wertes, den eine Methode zurückgibt, wird durch den *"**returntype**"* bestimmt.

◆ Der *"returntype"* ***void*** drückt aus, dass kein Wert zurückgegeben wird.

◆ Die Rückgabe eines Wertes erfolgt mit Hilfe des ***return-Statements.***

```
return [varname];
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Definition von Methoden

- Beispiel:

```
double kontoStand( int ktoNb ) {
    double saldo;
    ...
    // Berechnung saldo
    ...
    return saldo;
}
```

- Diese Methode könnte wie folgt benutzt werden:

```
...
meinKontoStand = meinKonto.kontoStand( meineKontoNb );
...
```

- Am "Ende" einer Methode kann das return-Statement fehlen, falls kein Wert zurückzugeben ist.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Definition von Methoden

◆ Mit dem ***return-Statement*** **ohne Wert** kann man den Rücksprung von einer beliebigen Stelle einer Methode realisieren, wenn dieses aufgund interner Zustandsbedingungen notwendig ist.

- Beispiel:

```
...
if (bedingung == true)
    return;
...
```

oder:

```
...
if (bedingung)
    return;
...
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Aufruf von Methoden

◆ Methoden eines Objektes ruft man üblicherweise mit der sog. ***Dot-Notation*** auf:

```
objectName.methodName( arg1, arg2, ... )
```

Beispiel:

```
meinKonto.getSaldo()
eKlasse.getVorderAchse( produktionsJahr )
```

- Parameter werden in runden Klammern übergeben; ist die Parameterliste leer, erscheinen nur die Klammern.

◆ Bei lokalem Aufruf (innerhalb einer Klasse) ist die Dot-Notation nicht notwendig:

```
methodName( arg1, arg2, ... )
```

Beispiel:

```
getVorderAchse( 1996 )
getVorderAchse()
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Aufruf von Methoden

◆ Ist der Returnparameter einer Methode selbst wiederum ein Objekt, so kann man Methodenaufrufe aneinanderhängen.

Beispiel:

```
eKlasse.getVorderAchse( 1996 ).getArtikelNr()
```

- Es wird die Methode ***getArtikelNr()*** aufgerufen
- ***getArtikelNr()*** ist eine Methode des Objekts, das von der Methode, ***getVorderachse( )*** als Returnparameter zurückgeliefert wird
- ***getVorderachse( )*** ist eine Methode des Objektes ***eKlasse.***

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Aufruf von Methoden

◆ Beispiel (Demo1_TestString):
"Aufruf verschiedener Methoden der Klasse ***String***"

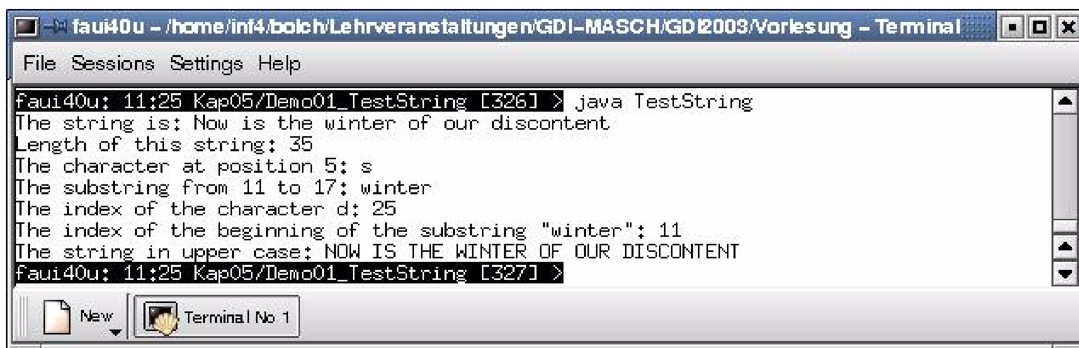
```
public class TestString {

    public static void main( String args[] ) {
        String str = "Now is the winter of our discontent";

        System.out.println( "The string is: " + str );
        System.out.println( "Length of this string: "
                + str.length() );
        System.out.println( "The character at position 5: "
                + str.charAt(5) );
        System.out.println( "The substring from 11 to 17: "
                + str.substring(11, 17) );
        System.out.println( "The index of the character d: "
                + str.indexOf('d') );
        System.out.print( "The index of the beginning of the " );
        System.out.println( "substring \"winter\": "
                + str.indexOf("winter") );
        System.out.println( "The string in upper case: "
                + str.toUpperCase() );
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5.1 Klassen (classes)

5 Aufruf von Methoden

• Ergebnis (Demo1_TestString):
"Aufruf verschiedener Methoden der Klasse ***String***"



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Das *this* - Schlüsselwort

- Das ***this*** - Schlüsselwort
  - Referenzen auf Methoden oder Objektvariablen des *"current objects"* kann mit dem ***this*** - Schlüsselwort in ***Dot-Notation*** erfolgen, was u. a. die Lesbarkeit des Programmcodes steigern kann.

    Beispiele:
    - Der Variablen ***t*** wird die Objektvariable ***x*** **dieses** Objekts zugewiesen:
      ```
      t = this.x;   // entspricht: t = x;
      ```
    - Dieses Objekt **kann auch** als Returnparameter zurückgegeben werden:
      ```
      return this;
      ```
    - Aufruf der Methode *firstmethod*, definiert in **dieser** Klasse;
      als Parameter wird das *"**current object**"* übergeben:
      ```
      this.firstmethod( this ); //entspricht:firstmethod( this );
      ```
  - *this* kann nur in Objektmethoden nicht in Klassenmethoden verwendet werden.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Gültigkeitsbereiche von Variablen

- Der **Gültigkeitsbereich** einer Variablen ist vom **Ort** der Definition *(default)* und von verwendeten ***Modifiern****(siehe 5.3)* abhängig.
  Wir unterscheiden:
  - Klassenvariable (für alle Objekte dieser Klasse global)
  - Objektvariable (für das Objekt global)
  - lokale Variable
    - methodenlokal
    - blocklokal
- In Java (nicht nur) wird nach der Variablendefinition vom Ort der Referenz her gesehen gesucht:
  - zunächst blocklokal,
  - dann methodenlokal,
  - dann auf Objektebene
  - und schließlich auf Klassenebene.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Gültigkeitsbereiche von Variablen

- Wird ein Variablenname mehrfach verwendet (was zulässig ist!), verdeckt (überschreibt, *overrides*) die innere Definition die äußere.

- Will man sowohl auf eine lokale Variable ***x*** als auch auf die Objektvariable ***x*** zugreifen können, so kann man auf die “verdeckte” Objektvariable mit Hilfe des ***this***-Schlüsselworts zugreifen:

```
this.x
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Gültigkeitsbereiche von Variablen

- Beispiel (Demo2_ScopeTest): "Methodenlokale Variable verdeckt Objektvariable... "

```
public class Scope {
    private int test = 10;        // Objektvariable
    static int kVarTest = 30;     // Klassenvariable

    public void printTest() {
        int test = 20;     // methodenlokale Variable
        System.out.println("test = " + test);
        System.out.println("this.test = " + this.test);
        System.out.println("kVarTest = " + kVarTest);
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Gültigkeitsbereiche von Variablen

◆ Beispiel (Demo2_ScopeTest): "Methodenlokale Variable verdeckt Objektvariable... " (cont.)

```
public class ScopeTest {

    public static void main( String args[] ) {
        Scope st = new Scope();
        st.printTest();
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Gültigkeitsbereiche von Variablen

• Ergebnis (Demo2_ScopeTest):

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Parameterübergabe

- ***Pass by Value:***
  - Primitive Typen werden als **Wert** übergeben. Änderungen dieser Größen haben also nur lokalen Effekt
- ***Pass by Reference:***
  - Objekte werden als **Referenz** übergeben; d. h. sämtliche Änderungen erfolgen im Original.
  - Beispiel (Demo3_Pass by Reference): "Objekt als Parameter"

```
class PassByReference {

// ersetzt in einem als Parameter übergebenem Feld
   alle 1 durch 0

      void oneToZero( int vector[] ) {
           for ( int i = 0; i < vector.length; i++ ) {
                if ( vector[i] == 1 ) vector[i] = 0;
           }
      }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Parameterübergabe

```
class TestPassByReference {
     public static void main ( String args[] ) {

          int arr[] = { 1, 3, 4, 5, 1, 1, 7 };

          PassByReference pbr = new PassByReference();

          System.out.print( "Values of the array: [ " );
          for ( int i = 0; i < arr.length; i++ ) {
               System.out.print( arr[i] + "  " ); //ohneVorschub!
          }
          System.out.println( "]" );

          pbr.onetoZero( arr );
          System.out.print( "New values of the array: [ " );
          for ( int i = 0; i < arr.length; i++ ) {
                System.out.print( arr[i] + "  " );
           }
           System.out.println( "]" );
     }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Parameterübergabe

- Ergebnis (Demo3_Pass by Reference):

```
faui40u - /home/inf4/bolch/Lehrveranstaltungen/GDI-MASCH/GDI2003/Vorlesung - Terminal
File  Sessions  Settings  Help
faui40u: 10:56 Kap05/Demo03_PassByReferenceTest [296] > javac PassByReference.java
faui40u: 10:57 Kap05/Demo03_PassByReferenceTest [297] > javac TestPassByReference.java
faui40u: 10:57 Kap05/Demo03_PassByReferenceTest [298] > java TestPassByReference
Values of the array: [ 1 3 4 5 1 1 7 ]
New values of the array: [ 0 3 4 5 0 0 7 ]
faui40u: 10:57 Kap05/Demo03_PassByReferenceTest [299] >
New  Terminal No 1
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Definition von Klassenmethoden

◆ **Klassenmethoden:**

- In Analogie zu den Klassenvariablen gibt es auch Klassenmethoden
- Wie bei Klassenvariablen wird auch hier das Schlüsselwort ***static*** verwendet:

```
static type methodname ( typ1 arg1, typ2 ... )
```

- Eine Klassenmethode ist allen Objekten dieser Klasse zugänglich.
- Andere Objekte können eine Klassenmethode über den Klassennamen in der Dot-Notation aufrufen:

```
classname.methodname
```

- Klassenmethoden können nur auf Klassenvariablen und -parameter zugreifen nicht auf Objektvariablen!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Definition von Klassenmethoden

- Ein Beispiel für die Verwendung der Klassenmethode ist die Klasse ***Math:***
  - Die Klasse ***Math*** stellt eine große Anzahl mathematischer Operationen zur Verfügung.
  - Diese Methoden kann man aufrufen, ohne dass man ein Objekt erzeugen muss.
  - Beispiele:

```
double root = Math.sqrt( 512. );

System.out.print( “Max. von x, y ist:”
                          + Math.max( x, y ) );
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Definition von Klassenmethoden

- Beispiel (Demo4_Classmethod):

```
class PassByReference {
        static void onetoZero( int vector[] ) {      //Klassenmethode
            for ( int i = 0; i < vector.length; i++ ) {
                if ( vector[i] == 1 ) vector[i] = 0;
            }
        }
        public static void main ( String args[] ) {
            int arr[] = { 1, 3, 4, 5, 1, 1, 7 };
            System.out.print(  "Values of the array: [ " );
            for ( int i = 0; i < arr.length; i++ ) {
                    System.out.print( arr[i] + " " );
            }
            System.out.println( "]" );

            onetoZero( arr );         // Aufruf der Klassenmethode

            System.out.print(  "New values of the array: [ " );
            for  ( int i = 0; i < arr.length; i++ ) {
            System.out.print( arr[i] + " " );
            }
            System.out.println( "]" );
        }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Definition von Klassenmethoden

- Die Klassenmethode ***main()***:
  - ◆ Die Signatur der Methode *main()* sieht immer wie folgt aus:

```
public static void main( String args[] )
```

  - *__public__*: Die Methode *main()* ist Klassen bzw. Objekten anderer *packages* zugänglich
  - *__static__*: *main()* ist eine Klassenmethode
  - *__void__*: *main()* gibt keine Parameter zurück

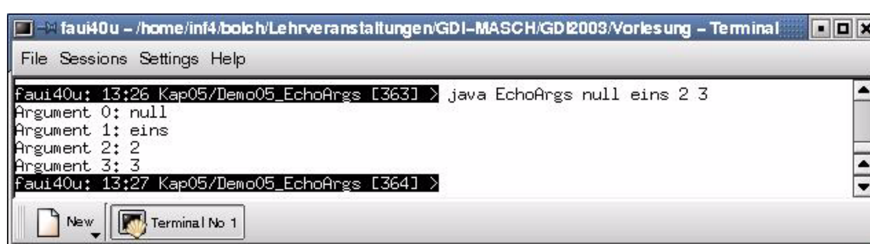
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Definition von Klassenmethoden

- ◆ ***main()*** übernimmt aus der Kommandozeile Parameter (Zeichenketten getrennt durch <b>) und stellt sie im array ***args[]*** zur Verfügung.
  - Beispiel (Demo5_EchoArgs):

```
class EchoArgs {
  public static void main( String args[] ) {
    for ( int i = 0; i < args.length; i++ ) {
        System.out.println( "Argument " + i + ":" + args[i] );
    }
  }
}
```

  - Ergebnis (Demo5_EchoArgs):



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Definition von Klassenmethoden

Parameterübernahme aus der Kommandozeile:

- Mit dem Parameter ***args[]*** in der main-Methode können Parameter aus der Kommandozeile übernommen werden (siehe Kap. 4.8)
- Nur *String*-Variable können aus der Kommandozeile übernommen werden
- Bei anderen Datentypen muss eine Konvertierung vorgenommmen werden.
- Dazu bieten die **Klassen!!** der primitiven Typen geeignete Methoden an (siehe auch Kapitel 4.8).

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Definition von Klassenmethoden

- Eine weiteres Beispiel (Demo6_SumAverage):

```
class SumAverage {
    public static void main ( String args[] ) {
        int sum = 0;

        for ( int i = 0; i < args.length; i++ ) {
            sum += Integer.parseInt( args[i] );
        }

        System.out.println( "Sum is: " + sum );
        System.out.println( "Average is: " +
            (float)sum / args.length );
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Definition von Klassenmethoden

- Ergebnis (Demo6_SumAverage):

```
faui40u - /home/inf4/bolch/Lehrveranstaltungen/GDI-MASCH/GDI2003/Vorlesung - Terminal
File  Sessions  Settings  Help
faui40u: 13:44 Kap05/Demo06_SumAverage [368] > xemacs SumAverage.java
faui40u: 13:45 Kap05/Demo06_SumAverage [369] > javac SumAverage.java
faui40u: 13:45 Kap05/Demo06_SumAverage [370] > java SumAverage 3 7 12 28
Sum is: 50
Average is: 12.5
faui40u: 13:46 Kap05/Demo06_SumAverage [371] >
New   Terminal No 1
```


© Gunter Bolch, Claus-Uwe Linster, Franz-Xaver Wurm • Universität Erlangen-Nürnberg • Informatik 4, 2004kap05_klassen_objekte_methoden.fm 2004-05-
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5.2 Objekte/Instanzen

1 Erzeugung von Objekten, new-Operator

◆ **new-Operator**

```
[classname] varname = new classname( par1, par2,...);
```

- Nach der Erzeugung (Instantiierung) enthält die Variable `varname` die Referenz auf ein Objekt der Klasse `classname`.

- Beispiel1:

```
String str = new String();
```

oder die Definition der Variablen getrennt von der Erzeugung

```
String str;
....
str = new String();
```


© Gunter Bolch, Claus-Uwe Linster, Franz-Xaver Wurm • Universität Erlangen-Nürnberg • Informatik 4, 2004kap05_klassen_objekte_methoden.fm 2004-05-
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Erzeugung von Objekten, new-Operator

- Beispiel 2:

```
Car vw = new Car();

bzw.
Car vw;
...
vw = new Car();
```

◆ Vorbesetzung (Initialisierung) von Objektvariablen:

- Die Argumente in den runden Klammern dienen dazu, Objektvariable auf einen Anfangswert zu setzen.
- Die Anzahl, der Typ und die Zuordnung der Argumente werden in der Klasse definiert.
- Beispiele:

```
Date datum = new Date( 90,4,1,4,30 );
Point punkt = new Point( 1,1 );
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Erzeugen von Objekten, new-Operator

- Ein vollständiges Beispiel (Demo7_CreateDates):

```
import java.util.Date;

class CreateDates {

    public static void main( String args[] ) {
        Date d1, d2, d3;

        d1 = new Date();
        System.out.println( "Date 1: " + d1 );

        d2 = new Date( 71, 7, 1, 7, 30 );
        System.out.println( "Date 2: " + d2 );

        d3 = new Date( "April 3 1993 3:24 PM" );
        System.out.println( "Date 3: " + d3 );
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Erzeugen von Objekten, new-Operator

- Ergebnis (Demo7_CreateDates):

```
faui40u - /home/inf4/bolch/Lehrveranstaltungen/GDI-MASCH/GDI2003/Vorlesung - Terminal
File  Sessions  Settings  Help
faui40u: 14:43 Kap05/Demo07_Date [380] > javac CreateDates.java
Note: CreateDates.java uses or overrides a deprecated API.  Recompile with "-deprecatio
n" for details.
1 warning
faui40u: 14:44 Kap05/Demo07_Date [381] > java CreateDates
Date 1: Mon Mar 17 14:44:09 CET 2003
Date 2: Sun Aug 01 07:30:00 CEST 1971
Date 3: Sat Apr 03 15:24:00 CEST 1993
faui40u: 14:44 Kap05/Demo07_Date [382] >
New   Terminal No 1
```


© Gunter Bolch, Claus-Uwe Linster, Franz-Xaver Wurm • Universität Erlangen-Nürnberg • Informatik 4, 2004kap05_klassen_objekte_methoden.fm 2004-05-
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Erzeugen von Objekten, new-Operator

◆ Was löst der *new-Operator* tatsächlich aus?

- ***new*** erzeugt aus dem *Template* der Klasse ein Objekt (eine Instanz).
- Üblicherweise wird ein Objekt durch eine Variable repräsentiert.
- Dieser Variablen wird durch die Anweisung:

```
var = new classname()
```

  eine Referenz auf das neuerzeugte Objekt zugewiesen.

  new initialisiert das Objekt mit Hilfe eines speziellen in der Klasse definierten Methodentyps. (Hier: `classname()`)

- Dieser Methodentyp wird als ***Konstruktor*** *(siehe Kapitel 5.2.9)* bezeichnet.
- Eine Klasse kann **keinen**, **einen** oder **mehrere** *Konstruktoren* besitzen, mit verschiedener Anzahl und verschiedenen Typen von Argumenten (siehe Beispiel: *Date*).
- Jedes erzeugte Objekt hat seinen eigenen Speicherbereich für die Daten, der Code selbst ist nur einmal vorhanden, da er nur ausgeführt wird.

© Gunter Bolch, Claus-Uwe Linster, Franz-Xaver Wurm • Universität Erlangen-Nürnberg • Informatik 4, 2004kap05_klassen_objekte_methoden.fm 2004-05-
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Zugriff und Zuweisungen auf Klassen- und Objektvariablen

◆ Der lokale Zugriff auf Objekt- bzw. Klassenvariablen unterscheidet sich nicht von dem auf lokale Variablen.

◆ Der nichtlokale Zugriff auf Klassen- und auf Objektvariablen erfolgt in der sogenannten *Dot-Notation*:

```
objectname.variablenname
```

Beispiele:

```
car.vorderAchse
myArray.length
```

◆ **Der nichtlokale Zugriff auf Objektvariable ist eher die Ausnahme!**

© Gunter Bolch, Claus-Uwe Linster, Franz-Xaver Wurm • Universität Erlangen-Nürnberg • Informatik 4, 2004kap05_klassen_objekte_methoden.fm 2004-05-
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Zugriff und Zuweisungen auf Klassen- und Objektvariablen

◆ Wichtig:

- Durch den Zugriff wird ein Wert “zurückgegeben” , es handelt sich also um einen Ausdruck.
- Dadurch ist es möglich auch an Objektvariable "geschachtelter" Objekte, in einfacher Weise heranzukommen.
- Beispiel:

```
vw.vorderAchse.achsSchenkel
```

◆ Zuweisen eines Wertes an eine Objekt- bzw. Klassenvariable:

Beispiel:

```
vw.vorderAchse = vorderAchseGolf;
```


© Gunter Bolch, Claus-Uwe Linster, Franz-Xaver Wurm • Universität Erlangen-Nürnberg • Informatik 4, 2004kap05_klassen_objekte_methoden.fm 2004-05-
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Referenzen auf Objekte

- Durch die Anweisung:

```
var = new Konstruktor
```

  (Konstruktor: `classname( )`)

  wird einer Variablen eine Referenz (ein Verweis) auf das erzeugte Objekt zugewiesen.

- Die Variable selbst enthält nicht das Objekt selbst, sondern es handelt sich um eine Referenz (einen Verweis) auf das Objekt.

- Weisen wir diese Objekte auch anderen Variablen zu oder übergeben wir Objekte als Argumente, so übergeben wir jeweils nur die Referenz auf das Objekt; es finden keine Kopiervorgänge statt!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Referenzen auf Objekte

- Beispiel (Demo8): "Wann referieren Variable identische Objekte?"

```
import java.awt.Point;
class ReferencesTest {
  public static void main ( String args[] ) {
     Point pt1, pt2;
      pt1 = new Point(100, 100);
      pt2 = pt1;    //pt1 und pt2 referenzieren das identische Objekt
      pt1.x = 200;
      pt1.y = 200;
      System.out.println( "Point1: " + pt1.x + ", " + pt1.y );
      System.out.println( "Point2: " + pt2.x + ", " + pt2.y );
  }
}
```

```
faui40u – /home/inf4/bolch/Lehrveranstaltungen/GDI-MASCH/GDI2003/Vorlesung – Terminal
File  Sessions  Settings  Help
faui40u: 15:48 Kap05/Demo08_ReferencesTest [414] > xemacs ReferencesTest.java &
[2] 3936
faui40u: 15:49 Kap05/Demo08_ReferencesTest [415] > javac ReferencesTest.java
faui40u: 15:54 Kap05/Demo08_ReferencesTest [416] > java ReferencesTest
Point1: 200, 200
Point2: 200, 200
faui40u: 15:55 Kap05/Demo08_ReferencesTest [417] >
New   Terminal No 1
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Casting and Converting:

◆ ***Casting*** (Typwandlung) heißt, einen Wert von einem Typ in einen Wert eines anderen Typs umzuwandeln, wobei das Original unverändert bleibt. *Casting* wird angewendet bei primitiven Typen oder bei Objekten.

◆ ***Converting*** heißt z. B., aus einem primitiven Typ ein Objekt mit der Objektvariablen des primitiven Typs zu bilden.

◆ ***Casting*** primitiver Typen:

- Solange man in Richtung eines “größeren” Typs wandelt in dem Sinne, dass keine Information verloren geht, erfolgt das ***Casting*** automatisch.
- Dies ist der Fall bei
  - *byte, char --> int;*
  - *int --> long;*
  - *float --> double;*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Casting and Converting

◆ ***Casting* primitiver Typen** (cont)

- In umgekehrten Fällen muß man explizit ***casten***.
- Gleiches wird bei ganzzahlig nach *float* oder *double* empfohlen.

```
(typname) expression
```

Beispiel:

```
float x, y;
int i;
...
i = (int) (x / y)
```

Da die *“Precedence”* des ***Castings*** höher liegt, als die der arithmetischen Operationen, ist die Klammerung des Ausdrucks wichtig!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Casting and Converting

◆ ***Casting* von Objekten**:

- *Casting* von Objekten ist möglich unter der Voraussetzung, dass ihre Klassen in einer Vererbungsbeziehung zueinander stehen.
- Beispiel:

```
class GreenApple extends Apple {
...
}
...

Apple a = new Apple();
GreenApple agreen = new GreenApple();

a = agreen;             // kein casting, da "upward use"
agreen = (GreenApple) a; // casting da "downward use"
...
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5.2 Objekte/Instanzen

4 Casting and Converting

◆ ***Converting*** primitiver Datentypen zu Objekten und umgekehrt.

- Aus der Überschrift kann man schon entnehmen, daß ein *Casting* nicht möglich ist.
- Primitive Datentypen und Objekte sind etwas grundsätzlich Verschiedenes.
- Man kann jedoch Objekte kreieren, die primitiven Datentypen entsprechen

Beispiel:

```
Integer intObject = new Integer( 35 );
Float floatObject = new Float( 1.2 );
usf.
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Casting and Converting

- Diese Klassen bieten u. a. etliche Methoden **(in der Regel Klassenmethoden)** zur Konvertierung in Objekte anderer (der den primitiven Typen analogen) Klassen an.

- Außerdem gibt es Methoden den “Wert” des Objekts zu extrahieren, was einer Konvertierung in einen primitiven Datentyp entspricht:

Beispiel:

```
int intnumber = intObject.intValue();
float floatnumber = floatObject.floatValue();
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Relationale Operation auf Objekte

◆ Für die primitiven Typen habe wir eine Anzahl relationaler Operatoren kennengelernt (siehe Kapitel 4.6).

◆ Für die Anwendung auf Objekte sind nur zwei davon sinnvoll und erlaubt:

- “ == “
- “ != “

Tatsächlich wird die **Identität** getestet:
Referenzieren verschiedene Variable das identische Objekt?

– Zwei *String* - Objekte mit dem gleichen String als Inhalt ergeben beim “==” - Test den Wert *“false”*!

– Für das Testen auf Gleichheit des Inhalts verschiedener String-Variabler stellt die Klasse *String* geeignete Methoden bereit, wie das Beispiel auf der nächsten Folie demonstriert.

◆ **Hinweis:** Werden zwei String - Variable mit dem gleichen literalen String erzeugt, dann referenzieren beide Variable das identische Objekt!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Relationale Operation auf Objekte

◆ Test auf Gleichheit von Strings am Beispiel (Demo9_EqualsTest):

```
class EqualsTest {
    public static void main( String args[] ) {

        String str1, str2;
        str1 = "she sells sea shells by the sea shore.";
        str2 = "she sells sea shells by the sea shore.";

        System.out.println( "String1: " + str1 );
        System.out.println( "String2: " + str2 );
        System.out.println( "Same object? " + ( str1 == str2 ) );
        System.out.println( "Same value? " + str1.equals( str2 ) );

        str2 = "he sells sea shells by the sea shore.";

        System.out.println( "String1: " + str1 );
        System.out.println( "String2: " + str2 );
        System.out.println( "Same object? " + ( str1 == str2) );
        System.out.println( "Same value? " + str1.equals( str2 ) );

        str2 = new String( str1 );

        System.out.println( "String1: " + str1 );
        System.out.println( "String2: " + str2 );
        System.out.println( "Same object? " + ( str1 == str2 ) );
        System.out.println( "Same value? " + str1.equals( str2 ) );
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Relationale Operation auf Objekte

• Ergebnis (Demo9_EqualsTest):

```
faui40u - /home/inf4/bolch/Lehrveranstaltungen/GDI-MASCH/GDI2003/Vorlesung - Terminal
File  Sessions  Settings  Help
faui40u: 9:15 Kap05/Demo09_EqualsTest [771] > java EqualsTest
String1: she sells sea shells by the sea shore.
String2: she sells sea shells by the sea shore.
Same object? true
Same value? true
String1: she sells sea shells by the sea shore.
String2: he sells sea shells by the sea shore.
Same object? false
Same value? false
String1: she sells sea shells by the sea shore.
String2: she sells sea shells by the sea shore.
Same object? false
Same value? true
faui40u: 9:16 Kap05/Demo09_EqualsTest [772] >
New   Terminal No 1
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Zugehörigkeit eines Objekts zu einer Klasse

◆ Die **Zugehörigkeit eines Objekts zu einer Klasse** kann mit dem *instanceof* - Operator wie folgt ermittelt werden:

- Beispiel1:

```
if ( "Viele Worte" instanceof String )        //true
...
```

- Beispiel2:

```
Point pt = new Point(10,10);
if ( pt instanceof String )                   // false
...
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Java - Klassen - Bibliotheken

◆ Die Java-Klassen-Bibliotheken sind unterteilt in sogenannte ***class packages.*** Die wichtigsten davon sind:

- java.**lang**
  Die Sprache selbst, *Object class, String class, System class*
- java.**util**
  Nützliche Hilfsmittel; z. B. *Date class; Calendar class*
- java.**io**
  Das Java-I/O-System
- java.**net**
  Networking classes, z. B. sockets, url, ...
- java.**awt**
  Abstract Windowing Toolkit (Gestaltungsklassen für grafische Oberflächen (Kapitel 8))
- java.**applet**
  (siehe Kapitel 6)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Java - Klassen - Bibliotheken

◆ Services zur Info-Beschaffung:

- Wir sind nicht in der Lage jede Methode jeder verwendeten Klasse auch nur annähernd zu besprechen, noch können wir Ihnen die Informationen gedruckt zur Verfügung stellen.
- Sie können sich die erforderlichen Informationen wie folgt beschaffen:

`http://www4.informatik.uni-erlangen.de/Services/Doc/Java`

`und dort` **`jdk-1.4.x`**

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Überladen von Methoden *(Overloading Methods)*

◆ Von ***Overloading*** spricht man, wenn man mehrere **Methoden** gleichen Namens, jedoch unterschiedlicher **Signatur** definiert.

◆ Unterschiedliche **Signatur** heißt:

- verschiedene Anzahl von Parametern
- und/oder Parameter verschiedenen Typs.

◆ Der Typ des Returnparameters wird bei der Unterscheidung von Signaturen **nicht** ausgewertet.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Überladen von Methoden *(Overloading Methods)*

◆ Beispiel (Demo10_MyRect):

```
import java.awt.Point;

class MyRect {

    private int x1 = 0;
    private int y1 = 0;
    private int x2 = 0;
    private int y2 = 0;

    void buildRect( int x1, int y1, int x2, int y2 ) {
      this.x1 = x1;
      this.y1 = y1;
      this.x2 = x2;
      this.y2 = y2;
    }
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Überladen von Methoden *(Overloading Methods)*

◆ Beispiel (Demo10_MyRect), cont.:

```
    void buildRect( Point topLeft, Point bottomRight ) {
      x1 = topLeft.x;
      y1 = topLeft.y;
      x2 = bottomRight.x;
      y2 = bottomRight.y;
    }
    void buildRect( Point topLeft, int w, int h ) {
      x1 = topLeft.x;
      y1 = topLeft.y;
      x2 = ( x1 + w );
      y2 = ( y1 + h );
    }
    void printRect(){
      System.out.print( "MyRect: <" + x1 + ", " + y1 );
      System.out.println( ", " + x2 + ", " + y2 + ">" );
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Überladen von Methoden *(Overloading Methods)*

◆ Beispiel (Demo10_MyRect), cont.:

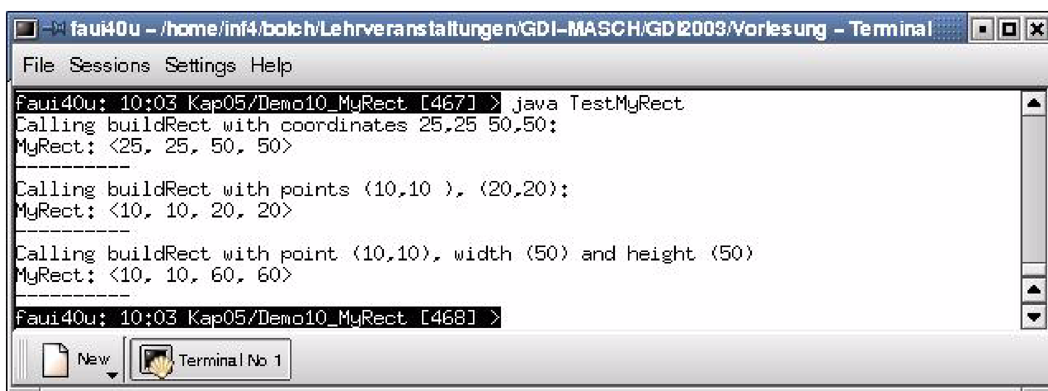
```
class TestMyRect {
    public static void main( String args[] ) {
      MyRect rect = new MyRect();

      System.out.println( "Calling buildRect with coordinates 25, 25, 50, 50:" );
      rect.buildRect( 25, 25, 50, 50 );
      rect.printRect();
      System.out.println( "----------" );

      System.out.println( "Calling buildRect with points ( 10, 10 ), ( 20, 20 ):" );
      rect.buildRect( new Point( 10,10 ), new Point( 20,20 ) );
      rect.printRect();
      System.out.println( "----------" );

      System.out.print( "Calling buildRect with point ( 10, 10 )," );
      System.out.println( " width ( 50 ) and height ( 50 )" );

      rect.buildRect( new Point( 10,10), 50, 50 );
      rect.printRect();
      System.out.println( "----------" );
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5.2 Objekte/Instanzen

8 Überladen von Methoden *(Overloading Methods)*

◆ Ergebnis (Demo10_MyRect):



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Konstruktoren *(Constructor Methods)*

- ***Constructor Methods*** sind die Methoden, die ein Objekt bei der Erzeugung in einen initialen Zustand setzen.
  - Ist kein Konstruktor explizit definiert, kann die Erzeugung (*new*) nur ohne Parameter erfolgen.
  - Die Objektvariablen werden default wie folgt vorbesetzt:
    - Objekte: null
    - boolean: false
    - char: '\0'
    - primitive numerische Typen: 0 bzw. 0.0
  - *Constructor-Methods* haben den Namen der Klasse
  - *Constructors* geben keinen Parameter zurück - trotzdem **nicht** den *Modifier "void"* verwenden.
  - *"**public**"* wird angegeben, wenn der *Constructor* auch von außerhalb eines *packages* (siehe später) erreichbar sein soll.
  - Constructor-Methods können nicht explizit aufgerufen werden.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Konstruktoren *(Constructor Methods)*

- Beispiel (Demo11_Person):

```
class Person {
   private String name;                      // Objektvariable
   private int age;                          // Objektvariable

   Person( String n, int a) {                // Konstruktor
      name = n;
      age = a;
   }
   void printPerson() {                      // Methode
      System.out.print( "Hi, my name is " + name );
      System.out.println( ". I am " + age + " years old." );
   }
}

class TestPerson {                           // Testklasse
   public static void main ( String args[] ) {
      Person p;
      p = new Person( "Laura", 20 );
      p.printPerson();
      System.out.println( "--------" );
      p = new Person( "Tommy", 3 );
      p.printPerson();
      System.out.println( "--------" );

   }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Konstruktoren *(Constructor Methods)*

- Ergebnis (Demo11_Person):

```
faui40u - /home/inf4/bolch/Lehrveranstaltungen/GDI-MASCH/GDI2003/Vorlesung - Terminal
File  Sessions  Settings  Help
faui40u: 10:55 Kap05/Demo11_Person [475] > java TestPerson
Hi, my name is Laura. I am 20 years old.
--------
Hi, my name is Tommy. I am 3 years old.
--------
faui40u: 10:55 Kap05/Demo11_Person [476] >
New    Terminal No 1
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Konstruktoren *(Constructor Methods)*

◆ Das Überladen von Konstruktoren ist analog dem Überladen von Methoden möglich.

- Beispiel (Demo12_MyRect2):

```
import java.awt.Point;

class MyRect2 {
   private int x1 = 0, y1 = 0;
   private int x2 = 0, y2 = 0;
    MyRect2( int x1, int y1, int x2, int y2 ) {
         this.x1 = x1;
         this.y1 = y1;
         this.x2 = x2;
         this.y2 = y2;
    }
    MyRect2( Point topLeft, Point bottomRight ) {
         x1 = topLeft.x;
         y1 = topLeft.y;
         x2 = bottomRight.x;
         y2 = bottomRight.y;
    }
    MyRect2( Point topLeft, int w, int h ) {
         x1 = topLeft.x;
         y1 = topLeft.y;
         x2 = (x1 + w);
         y2 = (y1 + h);
    }
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Konstruktoren *(Constructor Methods)*

- Beispiel (Demo12_MyRect2):

```
void printRect() {
        System.out.print( "MyRect: <" + x1 + ", " + y1 );
        System.out.println( ", " + x2 + ", " + y2 + ">" );
    }
}

class TestMyRect2 {

    public static void main( String args[] ) {
        MyRect2 rect;

        System.out.println( "Calling MyRect2 with coordinates 25,25, 50,50:" );
        rect = new MyRect2( 25, 25, 50,50 );
        rect.printRect();
        System.out.println( "----------" );

        System.out.println( "Calling MyRect2 with points (10,10), (20,20):" );
        rect= new MyRect2( new Point( 10,10 , new Point( 20,20 ) );
        rect.printRect();
        System.out.println( "----------" );

        System.out.print( "Calling MyRect2 with point ( 10,10 ),");
        System.out.println( " width (50) and height (50)" );
        rect = new MyRect2( new Point(10,10), 50, 50 );
        rect.printRect();
        System.out.println( "----------" );
    }
}
```


© Gunter Bolch, Claus-Uwe Linster, Franz-Xaver Wurm • Universität Erlangen-Nürnberg • Informatik 4, 2004kap05_klassen_objekte_methoden.fm 2004-05-
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Konstruktoren *(Constructor Methods)*

- Ergebnis (Demo12_MyRect2):

```
faui40u – /home/inf4/bolch/Lehrveranstaltungen/GDI-MASCH/GDI2003/Vorlesung – Terminal
File  Sessions  Settings  Help
faui40u: 11:21 Kap05/Demo12_MyRect2 [491] > java TestMyRect2
Calling MyRect2 with coordinates 25,25 50,50:
MyRect: <25, 25, 50, 50>
----------
Calling MyRect2 with points (10,10), (20,20):
MyRect: <10, 10, 20, 20>
----------
Calling MyRect2 with point (10,10), width (50) and height (50)
MyRect: <10, 10, 60, 60>
----------
faui40u: 11:21 Kap05/Demo12_MyRect2 [492] >
New  Terminal No 1
```


© Gunter Bolch, Claus-Uwe Linster, Franz-Xaver Wurm • Universität Erlangen-Nürnberg • Informatik 4, 2004kap05_klassen_objekte_methoden.fm 2004-05-
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 Vererbung *(Inheritance)*

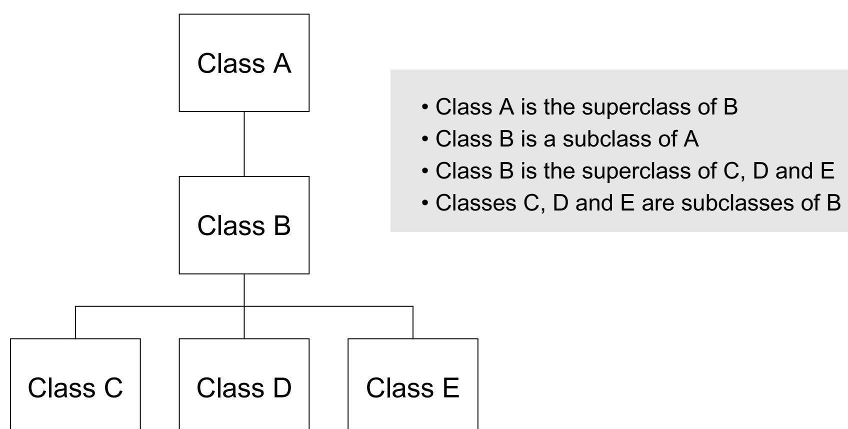
◆ *Inheritance* ist ein grundlegendes Konzept der **O**bjekt-**O**rientierten **P**rogrammierung, das Folgendes beinhaltet:

- Alle Klassen sind hierachisch geordnet. Jede Klasse hat
  - (in Java!) exakt eine **Oberklasse** (*superclass*)
  - und kann **Unterklassen** (*subclasses*) haben.
- Die höchste Klasse in der Hierachie ist eine Klasse mit dem Namen ***Object***; sie ist automatisch Oberklasse aller Klassen.
- Eine Unterklasse ist immer die Erweiterung der Oberklassen; bei der Definition einer neuen Klasse müssen Sie also nur die "Erweiterung" einer Oberklasse definieren, den *"Rest erben Sie".*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 Vererbung *(Inheritance)*

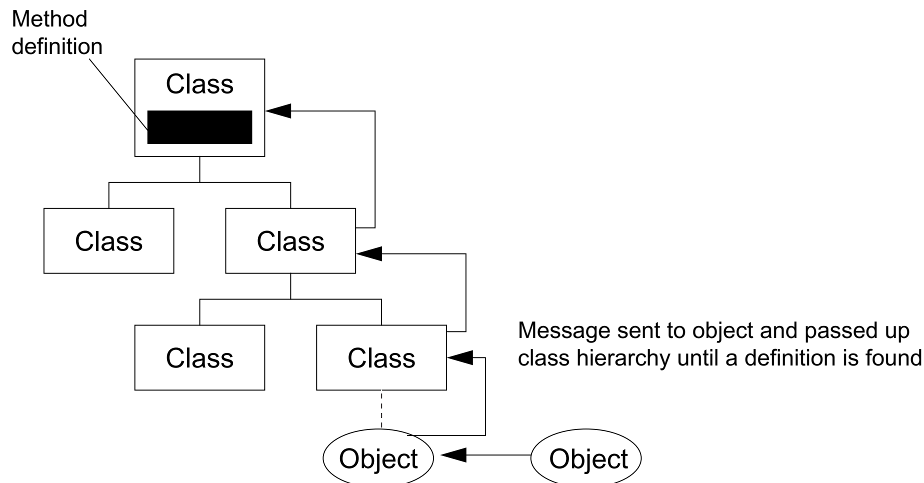
◆ Der erste Schritt beim Entwurf eines größeren Programmsystems ist der Entwurf eines Klassen-Hierachie-Diagramm für die Aufgabenstellung.



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 Vererbung *(Inheritance)*

◆ Wie funktioniert Inheritance?



Bei der Suche einer Methode in einem Objekt wird der Klassenbaum zur Wurzel hin durchlaufen, bis die Definition der Methode gefunden wird.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 Vererbung *(Inheritance)*

◆ Beispiel (Demo13_Temperatur):

- Zur Erweiterung (Vererbung) von Klassen verwendet man das Schlüsselwort **"extends"**

```
class Temperatur {
    double celsius;
    public void setCelsius( double grad ) {
        if ( grad < -273.15 )
                grad = -273.15;
                celsius = grad;
    }
    public double getCelsius() {
        return celsius;
    }
}

class TemperaturFahrenheit extends Temperatur {
    public double getFahrenheit() {
        return 9.0/5.0 * celsius + 32.0;
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 Vererbung *(Inheritance)*

```
public class TemperaturTest {
    public static void  main ( String args[] ) {
        TemperaturFahrenheit temp;
        temp = new TemperaturFahrenheit();
        temp.setCelsius( 49.4 );
        System.out.println( "Temperatur in Grad Celsius: "
                            + temp.getCelsius() );
        System.out.println( "Temperatur in Grad Fahrenheit: "
                            +  temp.getFahrenheit() );
    }
}
```

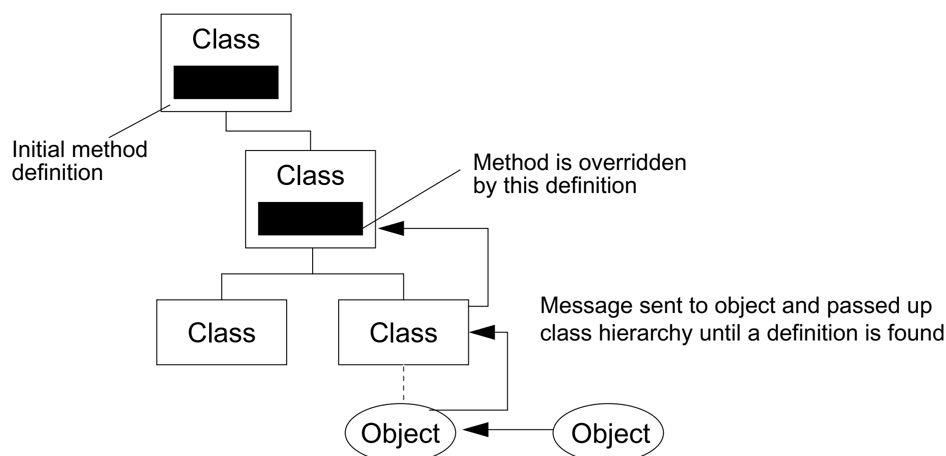
- Ergebnis (Demo13_Temperatur):

```
faui40u: 13:41 Kap05/Demo13_Temperatur [571] > java TemperaturTest
Temperatur in Grad Celsius: 49.4
Temperatur in Grad Fahrenheit: 120.92
faui40u: 13:41 Kap05/Demo13_Temperatur [572] >
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

11 Überschreiben von Methoden *(Overriding Methods)*

◆ Man spricht von Overriding von Methoden dann, wenn bei der Definition einer Methode in einer Subklasse die identische Signatur einer Methode in einer der Superklassen verwendet wird.



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

11 Überschreiben von Methoden *(Overriding Methods)*

◆ Beispiel (Demo14_PrintClass):

```
class PrintClass {
    private int x = 0;
    private int y = 1;

    void printMe() {
        System.out.println( "x is " + x + ", y is " + y );
        System.out.println( "I am an instance of the class "
        +   this.getClass().getName() );
    }
}
class PrintSubClass extends PrintClass {
    protected int z = 3;

    void printMe() {
        System.out.println( "x is " + x + ", y is " + y +  ", z is " + z );
        System.out.println(  "I am an instance of the class " +
                             this.getClass().getName() );
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

11 Überschreiben von Methoden *(Overriding Methods)*

```
public class OverridingTest {
    public static void main( String args[] ) {

        PrintClass obj1 = new PrintClass();

        obj1.printMe();

        PrintSubClass obj2 = new PrintSubClass();

        obj2.printMe();
    }
}
```

- Ergebnis (Demo14_PrintClass):

```
faui40u: 16:11 Kap05/Demo14_PrintClass [585] > java OverridingTest
x is 0, y is 1
I am an instance of the class PrintClass
x is 0, y is 1, z is 3
I am an instance of the class PrintSubClass
faui40u: 16:11 Kap05/Demo14_PrintClass [586] >
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

11 Überschreiben von Methoden *(Overriding Methods)*

- Häufig will man bei Anwendung der ***Overriding*** - Technik nicht die bereits vorhandenen Methoden vollständig ersetzen, sondern nur in ihrer Funktionalität erweitern.

- Ziel ist es also, die Originalmethode in der “*Overriding*-Methode” zu benutzen und nur die zusätzliche Funktionalität zu definieren.

- Ähnlich wie man mit dem ***this - Operator*** das ***“current object”*** referenziert, kann man mit dem ***super - Operator*** dafür Sorge tragen, daß der Methodenname in der Oberklasse referenziert wird.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

11 Überschreiben von Methoden *(Overriding Methods)*

- Beispiel (Demo15_PrintAnotherClass):

```
class PrintAnotherClass {
  private int x = 0;
  private int y = 1;

  void printMe() {
    System.out.println( "I am an instance of the class " +
                        this.getClass().getName() );
    System.out.println( "x is " + x );
    System.out.println( "y is " + y );
  }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

11 Überschreiben von Methoden *(Overriding Methods)*

◆ Beispiel (Demo15_PrintAnotherClass) cont.:

```
class PrintAnotherSubClass extends PrintAnotherClass {
  int z = 3;

  void printMe() {
    super.printMe();
    System.out.println("z is " + z);
  }
  public static void main( String args[] ) {
    PrintAnotherSubClass obj = new PrintAnotherSubClass();
    obj.printMe();
  }
}
```

- Ergebnis (Demo15_PrintAnotherClass):

```
faui40u: 17:34 Kap05/Demo15_PrintAnotherClass [607] > java PrintAnotherSubClass
I am an instance of the class PrintAnotherSubClass
x is 0
y is 1
z is 3
faui40u: 17:35 Kap05/Demo15_PrintAnotherClass [608] >
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

12 Überschreiben von Konstruktoren *(Overriding Constructors)*

◆ *Overriding* von Konstruktoren ist grundsätzlich **NICHT** möglich, da der Konstruktor den Namen der Klasse hat.

◆ Es ist jedoch möglich bei der Initialisierung einer Subklasse den Konstruktor der Superklasse aufzurufen.

- Damit wird sichergestellt, dass die geerbten initialen Eigenschaften des Objekts den “eigenen” Vorstellungen entsprechen.
- Dies geschieht mit

```
super( arg1, arg2, ... );
```

- Dieser Aufruf muss das 1. ausführbare Statement des Konstruktors sein.
- Andernfalls wird automatisch

```
super();
```

ohne Parameter aufgerufen, dh. die *“default”* - Initialisierung ausgeführt.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

12 Überschreiben von Konstruktoren

(Overriding Constructors)

◆ Beispiel (Demo16_NamedPoint) :

```
import java.awt.Point;

class NamedPoint extends Point {
  private String name;

  NamedPoint( int x, int y, String name ) {
    super( x, y );
    this.name = name;
  }
  String getName() {
    return name;
  }
}
```


© Gunter Bolch, Claus-Uwe Linster, Franz-Xaver Wurm • Universität Erlangen-Nürnberg • Informatik 4, 2004kap05_klassen_objekte_methoden.fm 2004-05-
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

12 Überschreiben von Konstruktoren

(Overriding

■ *Constructors)*

◆ Beispiel (Demo16_NamedPoint) cont. :

```
class TestNamedPoint{

  public static void main ( String arg[] ) {
    NamedPoint np = new NamedPoint( 5, 5, "SmallPoint" );
    System.out.println( "x is " + np.x );
    System.out.println( "y is " + np.y );
    System.out.println( "Name is " + np.getName() );
  }
}
```

• Ergebnis (Demo16_NamedPoint) :

```
faui40u: 18:26 Kap05/Demo16_NamedPoint [627] > java TestNamedPoint
 x is 5
 y is 5
 Name is SmallPoint
faui40u: 18:26 Kap05/Demo16_NamedPoint [628] >
```


© Gunter Bolch, Claus-Uwe Linster, Franz-Xaver Wurm • Universität Erlangen-Nürnberg • Informatik 4, 2004kap05_klassen_objekte_methoden.fm 2004-05-
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5.3 Access-, Final- and Static- Modifiers

◆ ***Packages*** sind Zusammenfassungen von Klassen mit eigenem **Namensraum**.

◆ ***Access Modifier*** sind spezielle ***keywords*** mit denen man **Gültigkeits-bereiche** gegenüber den *default* - Spezifikationen einschränken oder erweitern kann.

- Für Variable gilt:
  - *default* : Siehe Kapitel 5.1.7
- Für Objektvariable gilt:
  - *default*: Siehe Kapitel 5.1.7, innerhalb eines *packages*
  - *public*: zugreifbar auch über *packages* hinweg
  - *protected*: zugreifbar wie *“default”* und zusätzlich von Subklassen außerhalb des *packages*
  - *private*: zugreifbar nur für Methoden der Klasse selbst
  - *final*: Namenskonstante
  - *static*: Klassenvariable

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5.3 Access-, Final- and Static- Modifiers

◆ Für Methoden gilt:

- *default*: aufrufbar von allen Klassen innerhalb eines *“packages”* es sei denn, die Methode ist überschrieben worden.
- *public*: aufrufbar auch über *packages* hinweg
- *private*: nur von innerhalb der Klasse selbst aufrufbar
- *final*: nicht überschreibbar
- *static*: Klassenmethode (Klassenmethoden sind **nicht** überschreibbar.)

◆ Für Klassen gilt:

- *final*: keine Subklasse möglich
- *public* auch von Klassen außerhalb des *packages* erreichbar

◆ Weitere Modifier zu behandeln in späteren Kapiteln:

- *abstract*
- *synchronized, volatile*
- *native*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5.4 Zusammenfassung

- Klassen
  - Definition
  - Klassenvariable
  - Klassenmethoden
  - Vererbung
- Objekte / Instanzen
  - Erzeugung (Instantiierung)
    - Parameterübergabe
    - Konstruktoren
    - Initialisierung
  - Objektvariable, Modifier, Gültigkeitsbereiche
  - Zugehörigkeit zu einer Klasse
  - Relationale Operatoren

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5.4 Zusammenfassung

- Objekte / Instanzen (cont.)
  - Methoden, Modifier
    - Signatur
    - Aufruf
    - Parameterübergabe
      - by reference
      - by value
    - lokale Variable, Gültigkeitsbereiche
    - Overloading
    - Overriding
    - Returnparameter
  - Konstruktoren
    - Initialisierung
    - Overloading

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5.4 Zusammenfassung

- Variable
  - Klassenvariable
  - Objektvariable
  - methodenlokale Variable
  - blocklokale Variable
  - Gültigkeitsbereiche
    - Overriding
  - Casting
  - Converting

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Notizen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.