Klassen in Java

- Aller Programmtext steht in Klassen.
- Jede Klasse kann eine Funktion main haben.
- Funktionen werden innerhalb der Klasse programmiert, es gibt kein gesondertes Implementationsfile.
- this ist die Referenz auf das eigene Objekt und wird mit dem .-Selektor verwendet.
- Sichtbarkeit wird über Sichtbarkeitsmodifikatoren für jedes Member separat bestimmt.

```
import java.util.*;

public class BTime
{
  // Instanzdaten
  int h;
  int m;

  public BTime() // Constructor 1
  {
   Calendar c=Calendar.getInstance();
   this.h=c.get(Calendar.HOUR_OF_DAY);
   this.m=c.get(Calendar.MINUTE);
  }
  public BTime(int h, int m) // Constructor 2
  {
   this.h=(h>=0 && h<24)?h:0;
   this.m=(m>=0 && m<60)?m:0;
  }
  public BTime (String s)  // Constructor 3
  {
    String a[]=s.split(":");
    if (a.length==2)
    {
      h=Integer.parseInt(a[0]);
      m=Integer.parseInt(a[1]);
    }else h=m=0;
  }
```

```
public void setH(int h)    // Setter
   {this.h=(h>=0 && h<24)? h:0;}

public void setM(int m)    // Setter
   {this.m=(m>=0 && m<60)? m:0;}


 @Override
 public String toString()
 {
   String s=String.format("%02d:%02d",h,m);
   return s;
 }

 public void show()
 {
   System.out.printf("%02d:%02d",this.h,this.m);
 }
 public void increment()
 {
   m++;
   if (m==60)
     {m = 0; h++;}
   if (h==24)
     {h = m = 0;}
 }
```

```
 public static void main(String args[])
  {
    BTime bt=new BTime();
    System.out.println(bt.toString());
    System.out.println();
    bt.increment();
    System.out.println(bt); // toString wird hier implizit aufgerufen!!
    System.out.println();
    // Verwendung von Kommandozeilenparametern
    // java BTime 9 36
    if (args.length==2)
    {
      BTime bx=new BTime(Integer.parseInt(args[0]), Integer.parseInt(args[1]));
      System.out.println("bx: "+bx+" Uhr");
    }
  }
}
```

Experimentieren Sie mit diesem Beispiel

Main in gesonderter Klasse

```
class BTimeTest
{
    public static void main(String args[])
    {
      BTime t1=BTime.create();
      BTime t2= BTime.create();
      t1.setCurrentTime();
. . .
      System.out.println("");
      System.out.println("via toString. "+t1+" / "+t2);
    }
}
```

- Die Methode main kann auch in einer anderen Klasse programmiert werden.
- Beide Klassen müssen zunächst im selben Verzeichnis liegen

Anmerkungen zu this

- Hinter this verbirgt sich in den Memberfunktionen eine Referenz auf das Objekt, zu dem die Memberfunktion aufgerufen worden ist.
- Man kann this. beim Zugriff auf Member auch weglassen. In manchen Situationen braucht man this aber, z.B. bei Namensgleichheit von Funktionsparameter(n) und Member(n).

```
class abc
{
  int a;
  void setA(int a){this.a=a;}
  . . .
}
```

Klassen und Objekte

- Eine Klasse definiert einen benutzerdefinierten Datentyp.
- In der Regel stellt eine Klasse einen Verbund aus Daten (Membervariablen) und Funktionalität (Memberfunktionen/Methoden) dar.
- Instanzen einer Klasse bezeichnet man als Objekte.
- Objekte sind gekennzeichnet durch:
  - State (Status): Gesamtheit der Werte der Membervariablen
  - Behavior (Verhalten): Bestimmt durch die Gesamtheit der Memberfunktionen

Klassen und Objekte

- Methoden und Membervariablen bilden zusammen die Member einer Klasse.
- Nach außen kann die Sicht in eine Klasse eingeschränkt werden (information hiding).
- Es gibt public, private und protected Member. Ist keine Sichtbarkeit eingestellt, so gilt die Defaultsichtbarkeit /Sichtbar im Package (Verzeichnis) .
- private vor einer Variablen/Funktionsvereinbarung führt dazu, dass dieses Element der Klasse von außerhalb nicht zu sehen ist.
- public vor einer Variablen/Funktionsvereinbarung führt dazu, dass dieses Element von außen ohne Einschränkung sichtbar ist.
- Regel:
  - Daten sollten private sein,
  - Methode (Funktionen) können public sein.

Klassen und Objekte

- Klassen beschreiben Objekte oder können als Bildungsvorschrift für Objekte bezeichnet werden.
- Objekte werden grundsätzlich mit `new <Class_name>(…)` erzeugt.
- Variablen eines Klassendatentyps sind immer nur Referenzvariable (so etwas, wie ein Pointer in c, der zunächst NULL enthält).
- **`Btime bt1;`** ist eine Referenzvariable, vergleichbar mit einer Pointervariablen in c. Es existiert noch kein Objekt.
- Erst mit **`bt1=new Btime();`** wird ein Objekt erzeugt.

Initialisierung

- Zur Initialisierung von Objekten gibt es Constructoren.
- Ein Constructor trägt den Namen der Klasse als Funktionsname und hat keinen Returntyp.
- Constructoren können überladen sein.
- Via this(..) als erste Anweisung eines Constructors, kann ein Constructor einen anderen Constructor aufrufen.
- Die Daten eines Objektes sollen immer valide sein. Dafür haben der u.a. Constructoren zu sorgen.

Constructoren

```
class Btime
{
 int h;
 int m;

 public void setH(int h)    // Setter
    {this.h=(h>=0 && h<24)? h:0;}

 public void setM(int m)    // Setter
    {this.m=(m>=0 && m<60)? m:0;}

 Btime() // Constructor
 {
  Calendar c=Calendar.getInstance();
  this.h=c.get(Calendar.HOUR_OF_DAY);
  this.m=c.get(Calendar.MINUTE);
 }
 Btime(int h, int m) // Constructor
 {
  this.h=(h>=0 && h<24)?h:0;
  this.m=(m>=0 && m<60)?m:0;
 }
  . . .
 public static void main(String args[])
 {
      Btime bt1=new Btime();
      bt1.btimeShow();
 }
}
```

- Wir haben hier zwei überladene Constructoren
  - `Btime()`
  - `Btime(int h, int m)`
- Die set-Funktionen (setter genannt) haben üblicherweise einen Namen, der sich aus ‚set‘, gefolgt vom Variablennamen zusammensetzt.

Memberfunktionen/Methoden

- Funktionen können auch überladen werden.
- Funktionen einer Klasse können sich gegenseitig durch this.func() aufrufen, wobei func hier für die aufzurufende Funktion steht und natürlich Parameter haben kann. Die Angabe von this. kann aber in der Regel auch entfallen.
- Funktionen, die nur den Wert einer Instanzvariablen setzen, heißen setter.
- Funktionen, die nur den Wert einer Instanzvariablen zurückgeben, heißen getter.

Memberfunktionen/Methoden

```
public void setH(int h)
   {this.h=h;}

public void setM(int m)
   {this.m=m;}
```

- Setter sollen dafür sorgen, dass die Instanzvariablen grundsätzlich nur valide Werte enthalten.
- Deshalb ändern wir hier ab!

```
public void setH(int h)
   {this.h=(h>=0 && h<24)? h:0;}

public void setM(int m)
   {this.m=(m>=0 && m<60)? m:0;}
```

Memberfunktionen/Methoden

- Memberfunktionen sollen so implementiert sein, dass die Daten eines Objektes immer valide Werte beinhalten.
- Memberfunktionen kann man funktional in Verwaltungsfunktionen, Implementierungsfunktionen, Hilfsfunktionen und Zugriffsfunktionen einteilen.
- Gültigkeitsbereiche in Memberfunktionen
  - lokaler Block, in dem der Bezeichner verwendet wird,
  - umfassende Blöcke innerhalb der Funktion, in der der Bezeichner verwendet wird,
  - Klasse, in der die Funktion als Memberfunktion deklariert worden ist,
  - wird ein Bezeichner in einem eingeschlossenen Gültigkeitsbereich erneut vereinbart, so verdeckt diese Vereinbarung die ursprüngliche Vereinbarung.

Die Methode toString

```
public String toString()
{
  return String.format("%02d:%02d", h, m);
}
```

- Die Funktion (Methode) toString bildet zu dem Objekt eine Stringrepräsentation – wandelt das Objekt in eine sinnvolle Zeichenkette um.
- Diese Zeichenkette kann dann mit + verkettet oder ausgegeben werden.
- Die Methode show() kann nun entfallen.
- In Ausgaben wird toString automatisch aufgerufen.

Nicht primitive Datentypen

- Die Datentypen boolean, char, die int- und Gleitpunktdatentypen nennt man in Java primitive Datentypen.
- Klassen und Arrays sind “nicht primitive“ Datentypen.
- Variablen von nicht primitiven Datentypen sind immer lediglich Referenzvariablen.
- Das eigentliche Datenobjekt wird gesondert erzeugt (vergl. in main, obiges Beispiel).

Referenzvariable / Objekt

- Im Beispiel der Klasse BTime gab es in main die Variable

  ```
  BTime bt;
  ```

- Dies ist eine Referenzvariable, ein BTime Objekt existiert noch nicht.
- Das eigentliche Objekt wird erst mit

  ```
  bt=new BTime();
  ```

  erzeugt.
- Wie im Quelltext zu sehen, kann beides zusammengefasst werden zu:

  ```
  BTime bt=new BTime();
  ```

Arrays - Referenzvariable

- Bei Arrays verhält sich die Sache ganz ähnlich
- Es wird zunächst eine Referenzvariable angelegt:
- `int array[];`
- `int []array;`
- Die Klammern können vor oder nach dem Namen des Arrays stehen, das ist egal.
- Achtung!! Die Klammern bleiben immer leer.

Arrays - length

- Alle Arrays in Java verfügen über eine Membervariable length.
- Sie enthält die Anzahl der Arrayelemente

```
class ArrayLength
{
 public static void main(String args[])throws Exception
  {
   //Array, durch Initialisierung erzeugt - ohne new
    int array[]={2,4,6,8,10,12};
    for (int i=0; i<array.length; i++)
    {
      System.out.printf("array[%d]: %d\n",i, array[i]);
    }
  }
}
```

Array – Erzeugen mit new

- Das eigentliche Array wird mit mit new angelegt.

```
int array[]=new int[20];
```

- Dabei entsteht ein Array von 20 int-Elementen mit noch undefiniertem Inhalt.
- Nun kann man beispielsweise mit einer Schleife das Array füllen:

```
for(int i=0; i<array.length;i++)array[i]=0;
```

Array – Erzeugen per Initialisierung

- Ein Array kann ebenfalls durch eine Initialisierung angelegt werden.

```
int array[]={1,3,5,7,9};
```

- Auch hier bleiben die eckigen Klammern leer.

Mehrdimensionale Arrays

- Erzeugung durch Initialisierung

```
int mda[][]={{1,2},{1,2,3},{1,2,3,4,5}};
```

- Erzeugung mit new-Operator

```
int mda[][];
mda=new int[2][10];
```

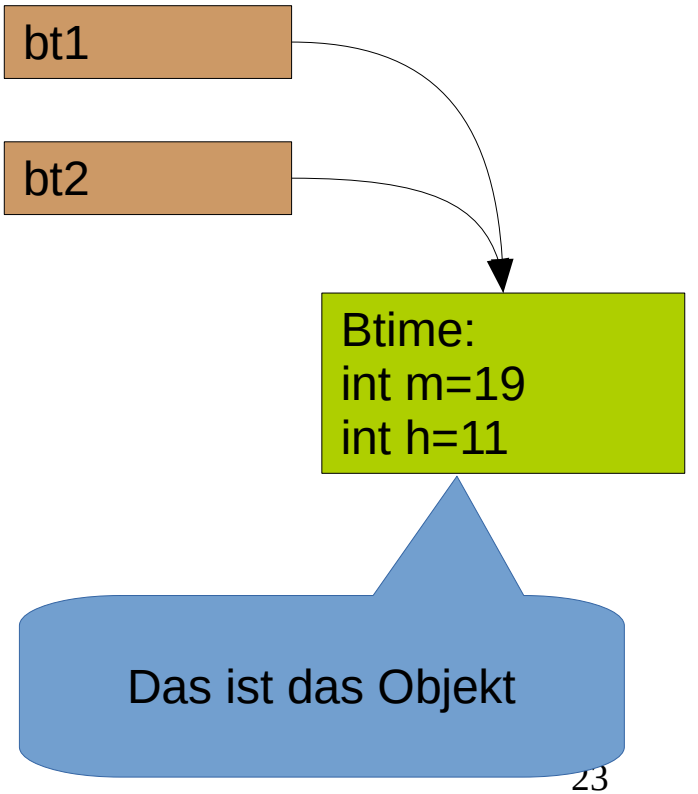
- oder

```
mda=new int[2][];
mda[0]=new int[2];
mda[1]=new int[5];
```

Zuweisungen von Daten nichtprimitiver Datentypen

- Bei der Zuweisung von Daten der nichtprimitiven Datentypen werden nur die Referenzen kopiert, nicht die Objekte.
- Nach Ausführung des Codes ergibt sich die Konstellation auf dem Bild.
- Genauso verhält es sich bei Parameterübergabe an Funktionen.
- Alle Änderungen die am Objekt über bt1 vorgenommen werden, ändern sich auch für bt2, es gibt ja nur das eine Objekt.

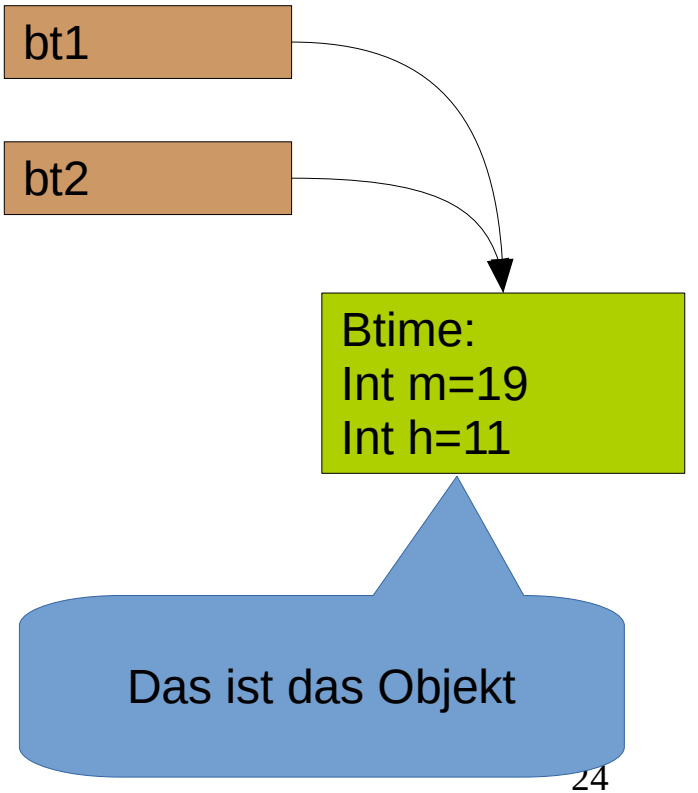
```
BTime bt1=new Btime();
BTime bt2=bt1;
```



Vergleich von Daten nichtprimitiver Datentypen

- Bei dem Vergleich auf Gleichheit (== oder !=) von Daten nichtprimitiver Datentypen wird geprüft, ob es sich um ein und dasselbe Objekt handelt.
- Es wird nicht geprüft, ob zwei Objekte sich gleichen.
- Im Beispiel wird true erzeugt.

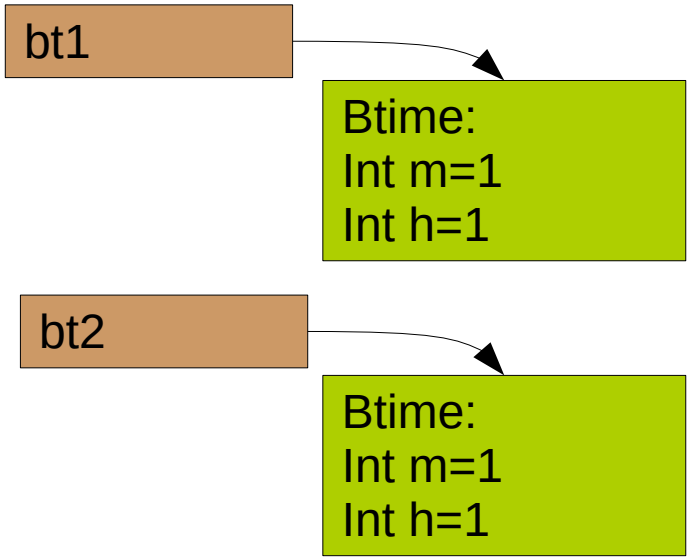
```
if (bt1==bt2)
```



Vergleich von Daten nichtprimitiver Datentypen

- Bei diesem Beispiel gibt es zwei Objekte bt1 und bt1
- Beide Objekte haben die Werte 1 für Minute und Stunde, sie gleichen einander.
- Der Vergleich `if(bt1==bt2)` liefert aber false, weil es zwei Objekte sind.

```
Btime bt1=new BTime(1,1);
Btime bt2=new BTime(1,1);
. . .
if (bt1==bt2)
```



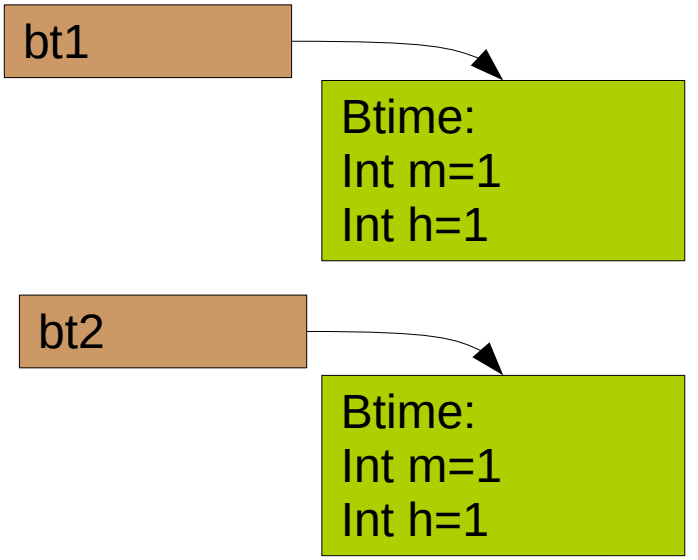
Vergleich von Daten nichtprimitiver Datentypen

- Sollen zwei Objekte auf gleichen Inhalt getestet werden, so ist die Funktion equals zu verwenden.
- Die Anwendung von

  `if(bt1.equals(bt2))`

  liefert true, weil sich die Objekte vollständig gleichen.

```
Btime bt1=new BTime(1,1);
Btime bt2=new BTime(1,1);
. . .
if (bt1.equals(bt2))
```



Vergleich von Daten nichtprimitiver Datentypen

- Die Funktion equals ist in der Klasse Object definiert.
- Diese Klasse bildet implizit die Basisklasse für alle Klassen.
- Somit steht die Funktion equals für den Vergleich von Objekten von der selben Klasse zur Verfügung.

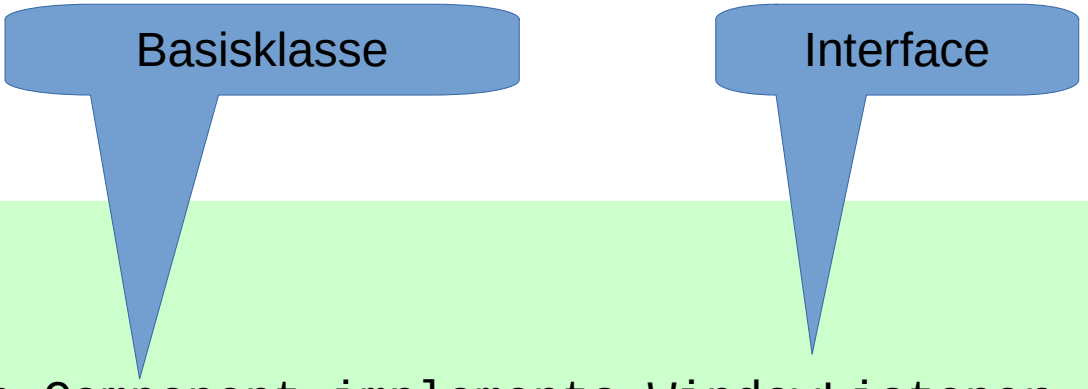
Vererbung

- Vererbung wird mit dem Schlüsselwort **extends** eingeleitet.
- Es gibt immer nur eine Basisklasse (Einfachvererbung)
- Vererbung ist durchsichtig.
- Überschriebene Funktionen in Java verhalten sich wie virtuelle Funktionen in C++.

Interfaces

- Interfaces sind eine Art leere, abstrakte Klassen, die lediglich Funktionsprtotypen beinhalten.
- Eine Klasse kann Interfaces implementieren, dazu wird im Klassenkopf **implements** gefolgt von Interfacenamen angegeben.
- Die Klasse muss nun alle im Interface deklarierten Methoden implementieren.
- Eine Klasse, die Interfaces implementiert, ist vom eigenen Klassentyp, ggf. vom Typ der Basisklassen und vom Typ aller implementierten Interfaces.

Beispiel aus Praktikum einführende Beispiele




```
import java.awt.*;
import java.awt.event.*;

public class MyPanel extends Component implements WindowListener
{
  private String myString;

  public void windowActivated(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowClosing(WindowEvent e){System.exit(1);}
  public void windowDeactivated(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowOpened(WindowEvent e){}
  . . .
```