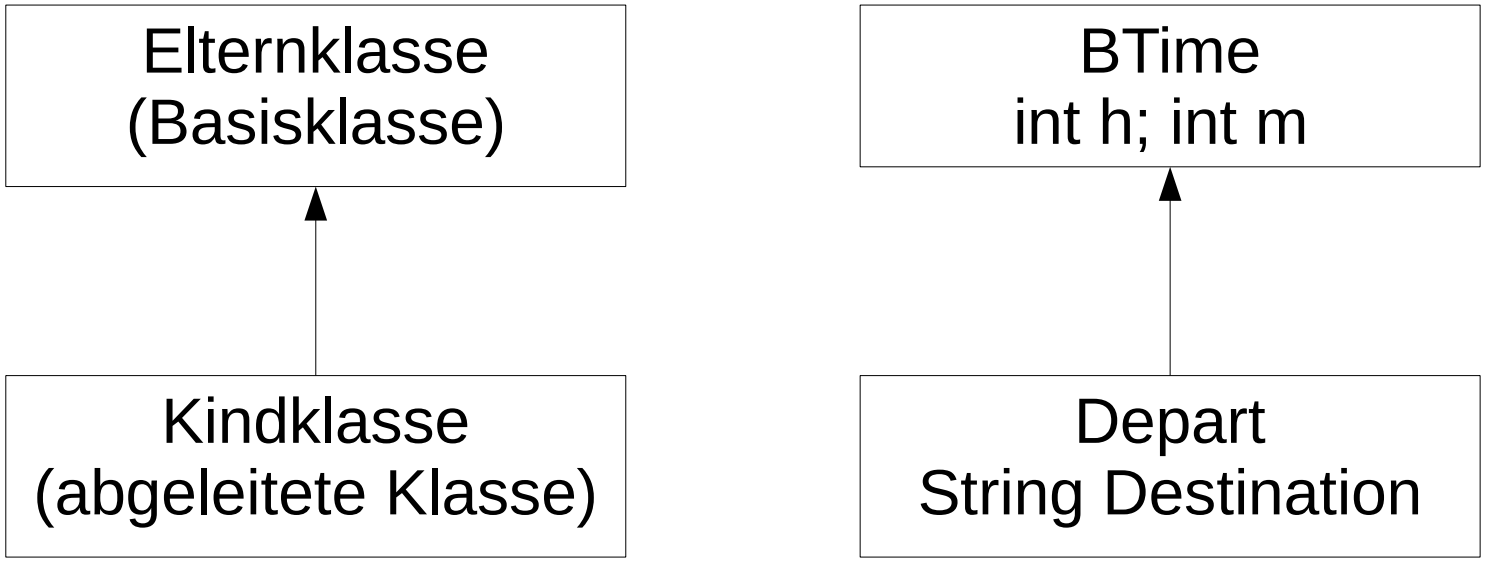
Vererbung

- Spezialisierung einer Klasse
- Eine Kindklasse erbt von einer Elternklasse, man spricht auch von Basis- und abgeleiteter Klasse



Vererbung

- Die Vererbung wird in java durch das Schlüsselwort **extends** eingeleitet.
- Die Kindklasse übernimmt dabei alle Datenmember (Instanzvariablen) und Methoden der Elternklasse.
- Ein Objekt der Klasse Depart enthält somit das Ziel, als auch die Zeit (Stunde, Minute).
- Die private-Member der Elternklasse sind in der Kindklasse ebenso, wie die public Member enthalten. Die private Member sind aber nicht sichtbar in der Kindklasse.
- Die Sichtbarkeit **protected** regelt, dass diese Member auch in abgeleiteten Klassen sichtbar sind, in fremden Klassen jedoch nicht.

Vererbung

- Im Zuge der Vererbung kann es in der Kindklasse Methoden mit der gleichen Signatur, wie in der Elternklasse geben. In diesem Falle überschreibt die Funktion der Kindklasse die Funktion der Elternklasse.
- Man spricht von **überschriebenen Funktionen/Methoden**. Sie verhalten sich **wie virtuelle Funktionen** in C++.
- Überschriebene Methoden können durch die Annotation @Override gekennzeichnet werden, dann prüft der Compiler, ob es wirklich eine zu überschreibende Funktion gibt.
- Gibt es in der Kindklasse eine Instanzvariable mit dem selben Namen, wie in der Elternklasse, spricht man von **Überdeckung**.

Vererbung

```
public class Depart extends BTime
{
   private String Destination;

   Depart(int h, int m, String Dest)
   {
     setM(m);
     setH(h);
     Destination=Dest;
   }

   @Override
   public String toString()
   {
     return super.toString()+" to "+Destination;
   }

   . . . main (nächste Seite)
}
```

Überschriebene Funktion

Aufruf der gleichnamigen Funktion der (unmittelbaren/direkten) Basisklasse

Vererbung

```
public static void main(String args[])
{
  Depart dp[]={new Depart(8,30,"Chemnitz"),
               new Depart(8,43,"Tharandt"),
               new  Depart(8,54,"Wien")};
  for (int i=0; i<dp.length; i++)
        System.out.println(dp[i]);
}
```

```
08:30 to Chemnitz
08:43 to Tharandt
08:54 to Wien
```

Vererbung Constructoren

- Der Constructor einer abgeleiteten (Kind-)Klasse trägt den Namen seiner, der Kindklasse.
- Bei der Objekterzeugung werden die Instanzvariablen der Basisklasse mit dem Defaultconstructor der Basisklasse (Constructor ohne Parameter) initialisiert.
- Über set-Methoden könnten danach Werte zugewiesen werden (siehe setM, setH im Beispiel).

Vererbung Constructoren

```
public class Depart extends BTime
{
   private String Destination;

   // Constructor
   Depart(int h, int m, String Dest)
   {
     Destination=Dest;
     setH(h); // Aufruf der geerbten public setter
     setM(m); // Aufruf der geerbten public setter
   }
  . . .
}
```

Vererbung Constructoren

- Eine elegantere Methode, die Instanzvariablen der Elternklasse zu initialisieren, besteht im Aufruf von **super(.....)**;
- super muss als erste Anweisung des Constructors ausgeführt werden.
- Die übergebenen Parameter im Aufruf von super müssen zu einem Constructor der Elternklasse passen.

Vererbung Constructoren

```
public class Depart extends BTime
{
   private String Destination;

   Depart(int h, int m, String Dest)
   {
     super(h,m);
     Destination=Dest;
   }
  . . .
}
```

Referenzen und Objekte

- Referenzvariable und Objekttyp müssen zueinander passen.
- Bei Objekten abgeleiteter Klassen darf die Referenzvariable auch vom Typ einer Elternklasse sein.
- Ist dies der Fall stehen nur die Methoden/Member der Elternklasse zur Verfügung.

Referenzen und Objekte

- Wird eine überschriebene Methode über eine Referenzvariable der Basisklasse aufgerufen, so wird die Funktion der abgeleiteten Klasse ausgeführt.
- Wir erinnern uns: Überschriebene Methoden sind Memberfunktionen die mit der selben Signatur (Returntyp/Name/Parameterliste) sowohl in einer Basisklasse, als auch in einer abgeleiteten Klasse definiert sind. Hier im Beispiel betrifft das die Funktion to String(). Diese gibt es in Klasse BTime und Depart.
- Beispiel:

```
Depart dp=new Depard(9,5,“Chemnitz“); // Abfahrt 09:05 nach Chemnitz

BTime time=dp;  // erlaubt, upcasting, time referenziert das
                // das Objekt der Klasse Depart
String s=time.toString()
System.out.println(“s: “+s);    // Ausgabe: s: 09:05 to Chemnitz
```

Obwohl time von der Klasse BTime ist, wird die Funktion toString des tatsächlichen Objektes, das von der Klasse Depart ist, ausgeführt.

Referenzen und Objekte

```
// eine main-Funktion zu Depart/Btime

public static void main(String args[])
{
 Depart dp[]={ new Depart(8,30,"Chemnitz"),
               new Depart(8,43,"Tharandt"),
               new  Depart(8,54,"Wien")};
 for (int i=0; i<dp.length; i++)
     System.out.println(dp[i]);


 BTime bt=dp[0];
 System.out.println(bt);
}
```

Referenz der Elternklasse

Aufruf der Funktion Der Kindklasse

Vererbung und Datentyp

- In C war jedes Objekt, jede Variable genau von einem Datentyp.
- In der objektorientierten Programmierung ist jedes Objekt vom Typ seiner eigenen Klasse, aber auch vom Typ aller Elternklassen.
- Alle Klassen erben in Java implizit von der Elternklasse java.lang.Object.
- Ein Objekt vom Typ Depart ist somit auch vom Typ BTime und vom Typ Object.

Interfaces

- Interfaces kann man sich vorstellen, wie ein Headerfile in c, für das es kein zugeordnetes Implementationsfile gibt.
- Ein Interface ist aufgebaut, wie eine Klasse, enthält jedoch keine Instanzvariablen und keine Funktionen, lediglich Funktionsköpfe, gefolgt von ‘;‘.

```
interface Compareable
{
  public boolean lessThan(Compareable other);
}
```

Interfaces

Eine Klasse kann ein Interface implementieren, in dem sie alle Methoden des Interfaces implementiert

```
class BTime implements Compareable
{
  . . .
   public boolean lessThan(Compareable other)
   {
     boolean ret = ((h*60+m)
      <((BTime)other).h*60+((BTime)other).m);
     return ret;
   }
  . . .
}
```

Interfaces

- Ein Objekt einer Klasse, die ein Inteface implementiert, ist dann vom Typ der eigenen Klasse, vom Typ aller Elternklassen und vom Typ des implementierten Interfaces. Eine Klasse kann mehrere Interfaces Implementieren.
- Implementiert eine Klasse ein Interface, so muss sie alle Funktionen des Interfaces implementieren (müssen in der Klasse ausprogrammiert werden).
- Ein Interface stellt sicher, dass die Objekte der Klasse eine bestimmte Funktionalität bereitstellen.

Die komplette Klasse Depart

```
public class Depart extends BTime
{
  private String Destination;

  public Depart(int h, int m, String Dest)
  {
    super(h,m);
    Destination=Dest;
  }

  @Override
  public String toString()
  {
     return super.toString()+" to "+Destination;
  }


   . . . // main-Funktion
}
```

```
public static void main(String args[])
  {
     Depart dp[]={new Depart(9, 5,"Chemnitz"),
                  new Depart(8,43,"Tharandt"),
                  new Depart(8,54,"Wien")};
     for (int i=0; i<dp.length; i++) System.out.println(dp[i]);
     BTime b=dp[0];
     System.out.println(dp[0].Destination);
     String s=b.toString();
     System.out.println("s: "+s);

     if (dp[0].lessThan(dp[1]))
        System.out.println(dp[0]+" eher als "+dp[1]);
     else
        System.out.println(dp[0]+" spaeter  "+dp[1]);
  }
```

Main-Funktion und Vererbung

- In unserem Beispiel gibt es jetzt zwei main-Funktionen.
  - Main in BTime
  - Main ind Depart
- Ein java-Programm, das aus meheren Klassen besteht kann auch mehrere main-Funktionen haben.
- Es wird die main-Funktion der Klasse ausgeführt, die beim Programmstart angegeben worden ist.

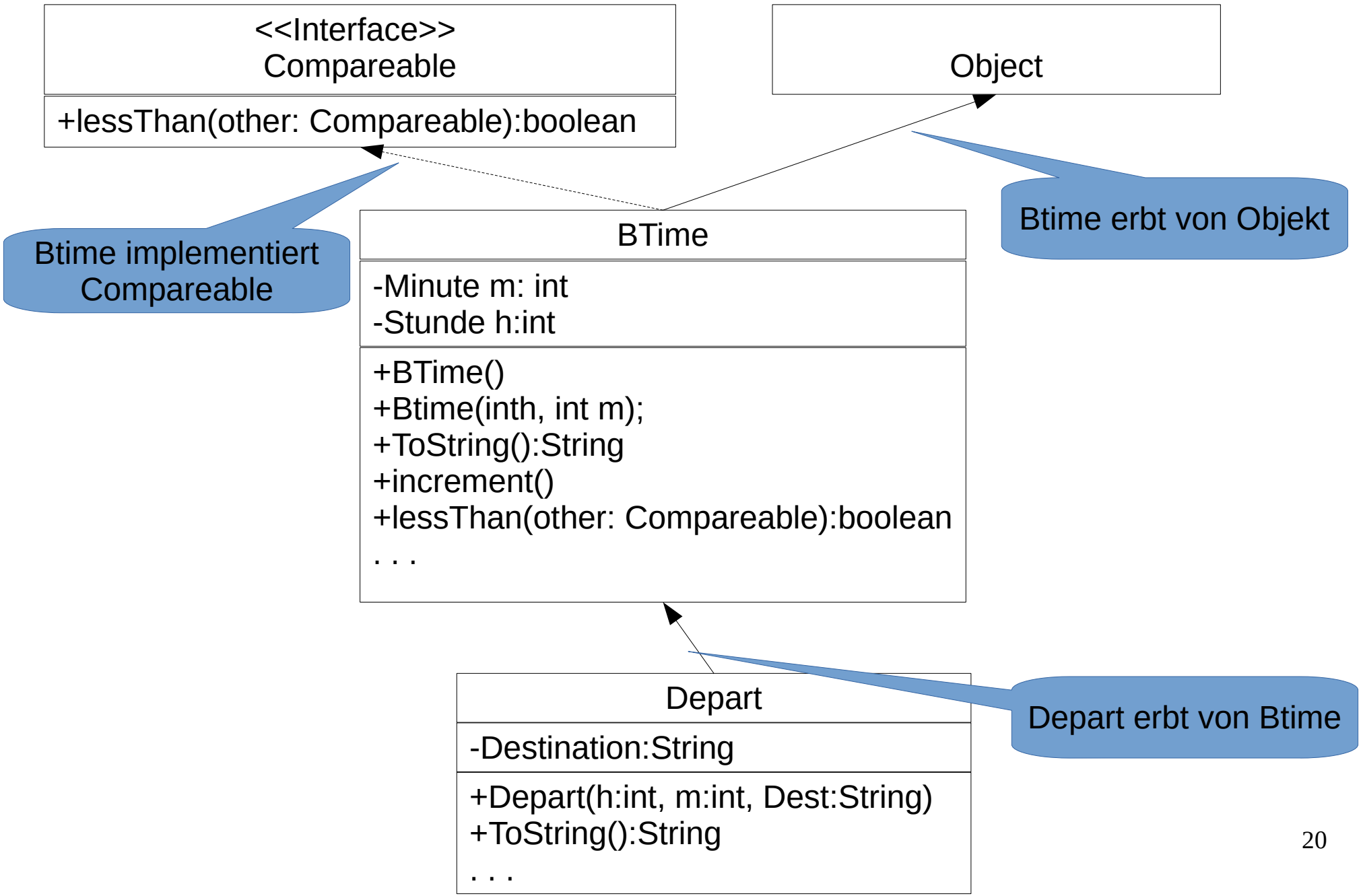
  java depart - es wird die main-Funktion der Klasse Depard ausgeführt

Vererbung/Interface

```
public static void main(String args[])
{
  Depart dp[]={new Depart(9, 5,"Chemnitz"),
               new Depart(8,43,"Tharandt"),
               new Depart(8,54,"Wien")};
  for (int i=0; i<dp.length; i++) System.out.println(dp[i]);
  BTime b=dp[0];
  System.out.println(dp[0].Destination);
  String s=b.toString();
  System.out.println("s: "+s);

  if (dp[0].lessThan(dp[1]))
   System.out.println(dp[0]+" eher als "+dp[1]);
  else
   System.out.println(dp[0]+" spaeter  "+dp[1]);
}
```

Klassendiagramm



Anderes Beispiel Klassendiagramm