Grundsätzliche Struktur von Java-Programmen (Applications)

Allgemein: Objektbeschreibung.java

```
import Benötigte Bibliotheken;

public class Klassenname (=Dateiname) {
 1. Deklaration von Objekten




 public void Methodenname () {

   2. Erzeugen der deklarierten Objekte



   3. Aufträge (und Fragen) an die erzeugten Objekte

   4. Löschen nicht mehr benötigter Objekte (und evtl.
      andere Abschlussoperationen)

 }
}
```

Beispiel Windmuehle.java

```
import stiftUndCo.*; //Bibliothek verwenden

public class Windmuehle { //Neue Objektklasse
  Bildschirm  einFenster; //Deklaration von
  BuntStift   einStift; //Objekten (=Exem-
        //plaren benutzter Objektklassen)


  public void fuehreAus () {
  //Programmieren der Darstell-Methode für
  //die neue Objektklasse Windmuehle
    einFenster  = new Bildschirm(); //Erzeu-
    einStift    = new BuntStift (); //gen

    einStift.normal(); //Nachrichten an be-
    einStift.hoch();   //nutzte Objekte...

    einStift.bewegeBis (200,150);
    einStift.runter();
    einStift.bewegeBis (230,150);
    einStift.bewegeBis (215, 90);
    einStift.bewegeBis (200,150);

  [..]




    einStift.gibFrei();//Vernichten benutzter
      //nicht weiter gebrauchter Objekte
    //nicht: einFenster.gibFrei();
  } //Ende von fuehreAus
} //Ende der Klassendefinition Windmuehle
```

Beispiel NikoHaus.java

```
import stiftUndCo.*;  //Bibliothek einbinden

public class NikoHaus { //Neue Objektklasse
  Bildschirm zeichenFläche; //Deklaration..
  BuntStift meinStift;  //..von Objekten
  Maus meinNagetier;    //..aus der Biblio-
  Hilfe und;            //..thek

  public void zeichne () {
  //zentrale Methode der neuen Klasse
     zeichenFläche = new Bildschirm ();//Erzeu-
     meinStift = new BuntStift (); //gen der
     meinNagetier = new Maus ();  //deklarier-
     und = new Hilfe ();         //ten Objekte

     meinStift.bewegeBis (180,250);//Aufträge..
     meinStift.schreibe ("Nikolaus-Haus");
     meinStift.bewegeBis (200,200);
     meinStift.runter ();
     meinStift.bewegeBis (300,200);
     und.warte (300);
 [..]
     meinStift.hoch();
     while (! meinNagetier.spezialKlick ()) {
        meinStift.bewegeBis (170,270);
        meinStift.schreibe ("Ende mit rechtem
           Mausklick");
     } //Ende von while

     meinNagetier.gibFrei(); //Abbau der Ob-
     meinStift.gibFrei();    //jekte
     zeichenFläche.gibFrei();
  } //Ende von zeichne
} //Ende von NikoHaus
```

Allgemein: StarteObjekt:java

```
public class StartAnwendung (=Dateiname) {
 public static void main (String[] s) {
   1. Deklarieren eines Objekts/Exemplars der oben
      beschriebenen Klasse
   2. Erzeugen des Objekts
   3. Auftrag ans Objekt, sich mit obiger Methode zu
      zeigen
```

Beispiel: StartMuehle.java

```
public class StartMuehle {
  public static void main (String[] s) {
     Windmuehle mühle; //1 Exemplar benennen
     mühle = new Windmuehle (); //Ex.erzeugen
     mühle.fuehreAus(); //Darstell-Auftrag
  }
}
```

Beispiel: StartNiko.java

```
public class StartNiko {
  public static void main (String[] s) {
    NikoHaus meineZeichnung; //Obj. deklarieren
    meineZeichnung = new NikoHaus ();//erzeugen
    meineZeichnung.zeichne (); //Auftrag ans..
  } //gerade erzeugte Objekt, sich darzustellen
}
```

Es ist guter Programmierstil, die grundsätzliche Beschreibung eines Objekts (d.h. die Programmierung der Klasse, sozusagen den Bauplan) und das tatsächliche Erzeugen und Verwenden eines Objekts zu trennen – am besten, wie hier, sogar in zwei verschiedenen Dateien!

Alternative Möglichkeit (Jan. 2008): **Grundsätzliche Struktur von Java-Programmen (Applications)** (verkürzte, modernere Form)

Allgemein: Objektbeschreibung.java

```
import Benötigte Bibliotheken;

public class Klassenname (=Dateiname)  {
  1&2. Deklaration und Erzeugen von Objekten



  public void Methodenname ()
  {
    3. Aufträge (und Fragen) an die erzeugten Objekte









  }
}
```

Beispiel Windmuehle.java

```
import stiftUndCo.*; //Bibliothek verwenden

public class Windmuehle  //Neue Klasse
{
  Bildschirm fenster = new Bildschirm();
  BuntStift  stift   = new BuntStift();

  public void führeAus ()
  {
    stift.normal(); //Aufträge an be-
    stift.hoch();   //nutzte Objekte...

    stift.bewegeBis (200,150);
    stift.runter();
    stift.bewegeBis (230,150);
    stift.bewegeBis (215, 90);
    stift.bewegeBis (200,150);

  [..]

  } //Ende von führeAus
} //Ende der Klassendefinition Windmuehle
```

Beispiel NikoHaus.java

```
import stiftUndCo.*;  //Bibliothek einbinden

public class NikoHaus  //Neue Klasse
{
  Bildschirm blatt = new Bildschirm():
  BuntStift  kuli  = new BuntStift();

  public void zeichne ()
  {
    kuli.bewegeBis (180,250); //Aufträge..
    kuli.schreibe ("Nikolaus-Haus");
    kuli.bewegeBis (200,200);
    kuli.runter ();
    kuli.bewegeBis (300,200);
    Hilfe.warte (300);


    [..]


  } //Ende von zeichne
} //Ende von NikoHaus
```

Allgemein: StarteObjekt:java

```
public class StartAnwendung (=Dateiname)
{
 public static void main (String[] s)
 {
   1. Deklarieren und 2. Erzeugen eines Objekts der
      oben  beschriebenen Klasse
   3. Auftrag ans Objekt, sich mit obiger Methode zu
      zeigen
 }
}
```

Beispiel: StartMuehle.java

```
public class StartMuehle
{
  public static void main (String[] s)
  {
     Windmuehle mühle = new Windmuehle();
     mühle.führeAus(); //Darstell-Auftrag
  }
}
```

Beispiel: StartNiko.java

```
public class StartNiko
{
  public static void main (String[] s)
  {
    NikoHaus hütte = new NikoHaus();
    hütte.zeichne (); //Auftrag an das ..
  }         //gerade erzeugte Objekt, sich..
}           //zu zeichnen
```

Es ist guter Programmierstil, die grundsätzliche Beschreibung eines Objekts (d.h. die Programmierung der Klasse, sozusagen den Bauplan) und das tatsächliche Erzeugen und Verwenden eines Objekts zu trennen – am besten, wie hier, sogar in zwei verschiedenen Dateien!

Natürlich können statt einer Methode auch viele Methoden Aufträge enthalten. So könnte z.B. zeichne() die beiden weiteren Methoden zeichneGrundmauern() und zeichenDach() aufrufen. Gibt es hingegen wie hier nur eine Methode, die Bildschirm und BuntStift braucht, könnten Objekte dieser beiden Klassen auch in der Methode (statt davor) definiert und erzeugt werden.

Ein Vernichten nicht mehr gebrauchter Objekte (Punkt 4. meines Blattes von 2001) ist allgemein in Java nicht üblich. Dies macht i.A. irgendwann der *garbage collector* automatisch.
Die Methoden der Klasse Hilfe können sind *static*, d.h. *warte* kann direkt mit dem Klassennamen aufgerufen werden, ohne dass zuvor ein Objekt der Klasse Hilfe erzeugt werden muss.