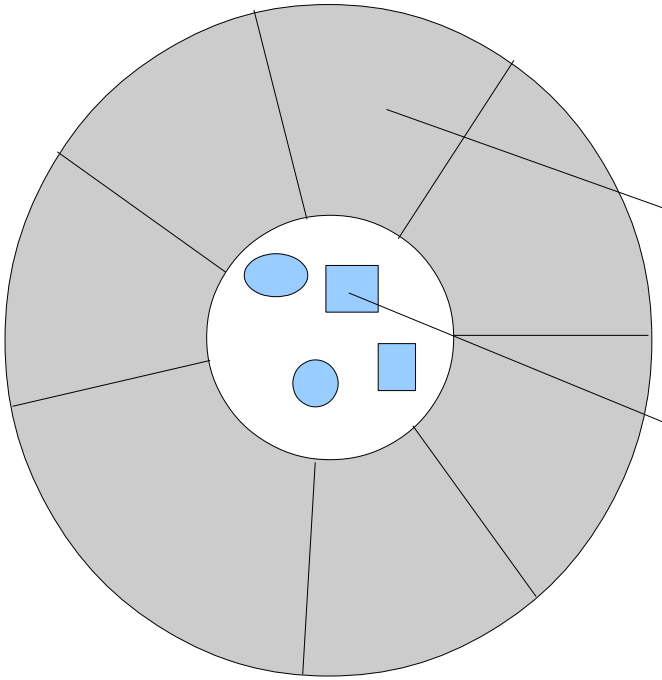
Klassen

- Klassen beschreiben Objekte, die abstrakte oder konkrete Sachverhalte modellieren.
- Objekte sind durch einen Status gekennzeichnet (State). Der Status eines Objektes ergibt sich aus der Summe der Werte ihrer Instanzvariablen.
- Objekte sind durch ein, Ihnen eigenes Verhalten gekennzeichnet (Behavior). Das Verhalten der Objekte einer Klasse wird durch die Methoden der Klasse beschrieben.
- Der Zustand eines Objektes sollte privaten Charakter tragen, er sollte nach außen verborgen sein.

classes – information hiding

Quelle: http://web2.java.sun.com/docs/books/tutorial/java/concepts/class.html

Die Skizze soll ein Objekt stilisieren, bei dem die Daten im Kern, von außen nicht sichtbar angeordnet sind. Der Zgang ist nur über die öffentlichen Methoden möglich. Damit erreicht, dass die Daten nicht unkontrolliert veränert werden können.



Öffentl. Methoden

Private Daten

Klassenkopf

- Der Klassenkopf beinhaltet die der gesamten Klasse zugeordneten Attribute, den Klassennamen und ggf. eine Basisklasse bzw. zu implementierende Interfaces.
- Attribute werden über sog. Modifier festgelegt:
- public: Es handelt sich um eine öffentliche Klasse, fällt public weg, ist die Klasse innerhalb des »Packages« sichtbar. Public ist ein Sichtbarkeitsmodifikator, weitere sind private und protected.
- abstact: Die Klasse wird als abstakte Klasse gekennzeichnet, es können keine Instanzen von ihr erzeugt werden, wohl aber können von ihr weitere Klassen abgeleitet werden (Vererbung). Die Klasse bedarf der Verfeinerung. Auch wenn sie keine abstrakten Methoden enthält, gilt die Klasse damit als abstrakte Klasse.
- final: Es können Instanzen erzeugt werden, aber es können keine Klassen von dieser Klasse abgeleitet werden.

<classdelaration>::
{<class_modifier>}, 'class' <identifier> ['extends' <class_identifier>]
['implements' <interface_identifier>{','<interface_identifier>}]
'{' {<memberdeclaration>} '}'

<class_modifier>::
'public' | 'private' | 'abstract' | 'final'

Syntaxbeschreibung, ähnlich EBNF nach Wirth. Ggf. mal im Netz recherchieren

- extends: erbt von Basisklasse
- implements: implementiert ein Interface
  - Interfaces sind Konstruke, ähnlich Klassen, aber ohne Instanzdaten und ohne implementierte Methoden.
  - Eine Klasse, die ein Interface implementiert, muss alle Methoden des Interfaces implementieren (ausprogrammieren).
  - Ein Objekt ist dann vom eigenen Klassentyp, vom Typ der Basisklasse und wiederum deren Basisklassen in einer Vererbungslinie, und vom Typ der implementierten Interfaces.
  - Eine Klasse kann immer nur von einer Klasse erben, aber beliebig viele Interfaces implementieren.

- Auch Interfaces können voneinander erben,

  wird aber nicht weiter betrachtet.
- Werden mehrere Interfaces implementiert, so werden sie durch Komma voneinander getrennt.
- Es gibt auch Interfaces, die leer sind, man spricht dann von einem „Markerinterface“(zB. Serializable).
- Beispiele folgen später.

```
public class Address extends Object
implements Externalizable, Transferable
{
   . . .
}
```

Klassenkörper

- Der Klassenkörper beschreibt sowohl die Instanzvariablen, deren Werte den Status der Objekte bilden, als auch die Methoden, die das Verhalten beschreiben und die öffentliche Schnittstelle bilden, sowie Konstanten.
- In Ausnahmen kann es auch öffentliche Variablen oder nicht öffentliche Methoden geben.
- Eine Klasse kann auch wieder Klassendeklarationen enthalten, sog. inner classes. Sie spielen beim Eventhandling eine besondere Rolle.

Die Membervereinbarung

(nicht vollständig)

<memberdeclaration>::
<data_member_declaration> | <method_declaration>| <inner_classdecl>

<data_member_declaration>::
{<data_member_modifier>} <type> <identifier> ['=' <initializing>] ';'

<data_member_modifier>::
'public' | 'private' | 'protected' | 'static' | 'final'

<method_declaration>::
{<method_modifier>} [<type>] <identifier> '(' [<parameter_list>] ')'
(';' | <block>)

<method_modifier>::
'public' | 'private' | 'protected' | 'static' | 'final' | 'abstract'

Im Interface

In der Klasse

Anmerkungen zu Datenmembern

- Man nennt sie in Java instance data.
- Instanzdaten ohne Initialisierung werden automatisch mit 0 initialisiert.
- Instanzdaten können in der Klasse initialisiert werden.
- Jedes Member hat seine eigene Sichtbarkeit, es gibt keine Gruppen oder Bereiche, wie in C++.

Die Sichtbarkeit von Member

Situation	public	protected	Package (default)	private
Same class	yes	yes	Yes	Yes
Class in same package	yes	yes	yes	no
Subclass in different pack.	yes	yes	no	no
Non Subclass different pack.	yes	no	no	no

Methoden

[public | protected | private][static][abstract | final][native][syncronized] <returntype> Name <parameterList> [<ausnahmeErzeugung>] <block>

- public, protected, private: regeln die Sichtbarkeit
- static: Die Methode hat Klassenbezug, sie ist mit keinem Objekt verbunden, kann auch nicht auf Instanzvariable zugreifen, sondern nur auf statische oder lokale Variable. Wichtig!!! Static wird am Ende des Dokumentes noch etwas ausführlicher erläutert.
- abstact: Die eigentliche Implementierung der Methode erfolgt in einer erbenden Klasse. Der Mechanismus entspricht einer rein virtuellen Funktion in C++. Enthält eine Klasse abstrakte Methoden, ist die ganze Klasse abstract.
- final: Diese Methoden können in erbenden Klassen nicht überschrieben werden.
- synchronized: Während der Ausführung der Methode kann keine andere Methode, die das Objekt verwendet, ausgeführt werden (bei Threads).

- Mehrere Methoden mit gleichen Namen, aber unterschiedlicher Signatur (Parameterliste, Returntyp) werden als **überladene Funktionen** bezeichnet.
- Die Implementation einer Methode erfolgt in einem, dem Methodenkopf nachgestellten Block (wie in c).
- Innerhalb einer Vererbungslinie können Methoden überschrieben werden, sie verhalten sich dann wie virtuelle Funktionen in c++.
- Regeln über Gültigkeit und Lebensdauer lokaler Variablen entsprechen weitgehend denen von C/C++.
- Lokale Variablen werden nicht automatisch initialisiert, der Compiler prüft aber, ob sie vor ihrer Verwendung einen Wert per Initialisierung oder Wertzuweisung erhalten haben.

Konstruktoren

- Konstruktoren sind spezielle Methoden, die der Initialisierung der Instanzvariablen bei der Objekterzeugung dienen.
- Wie in C++ tragen sie den Namen der umgebenden Klasse und haben keinen Returntyp wohl aber eine, ggf. leere Parameterliste.
- Es kann mehrere, überladene Konstruktoren in einer Klasse geben.
- Aus einem Konstruktor kann man durch einen Aufruf

  this (<parameterliste>); einen anderen Konstruktor der selben Klasse aufrufen, jedoch muss dieser Aufruf die erste ausführbare Anweisung des Konstruktors sein, ähnlich super(….);
- Ein dem Konstruktor sehr ähnliches Konstrukt ist der **Initialisierer**. Er besteht nur aus einem Block und enthält Code zur Initialisierung. Die Übergabe von Parametern ist nicht möglich. Im Gegensatz zu Konstruktoren und Methoden sind in dem Initialisierer Vorwärtsreferenzen auf Klassenebene nicht gestattet. Es kann aber mehrere Initialisierer geben.

Beispiel Initialisierer

```
public class Initialisierer
{

  private String testStr;

  // Das ist der Initialisierer
  // Er hat einen Koerper, aber keinen Funktionskopf
  {
    testStr="Max"+" & "+"Moritz";
  }

  public String getTestStr()
  {
    return testStr;
  }

  public static void main(String args[])
  {
     Initialisierer meinObject=new Initialisierer();
     System.out.println(meinObject.getTestStr());
  }
}
```

Static Initializer

- Einem Initialisierer kann der Modifier static vorangestellt werden, er dient dann der Initialisierung der statischen Variablen der Klasse, also der Variablen mit Klassenbezug.
- Seine Ausführung erfolgt unmittelbar nach dem Laden der Klasse.

```
public final class jdbcMysqlDriver implements Driver
{
/** Self instantiation */
static { new jdbcMysqlDriver();
}
. . .
```

Möglichkeiten der Initialisierung von Instanzdaten

Instanzdaten können initialisiert werden durch

- gewöhnliche Initialisierung,
- einen Konstruktor,
- einen Initialisierer,
- Automatisches Füllen mit mit 0

Konstanten

- Konstanten werden mit den Modifikatoren static final eingerichtet, oft public static final.
- Vor ihrer ersten Verwendung müssen sie mit einem Wert belegt worden sein,
- Sie müssen nicht per Initialisierung mit einem Wert belegt werden.
- Im Sinne eines guten Programmierstils sollten aber doch Konstanten konvetionell initialisiert werden.
- Der Grund für diese ungewöhnliche Tatsache sind inner classes, die später beschrieben werden.

Vererbung

- Vererbung ist eines der zentralen Konzepte der objektorientierten Programmierung.
- Java kennt keine Mehrfachvererbung, eine Klasse kann immer nur von einer Basisklasse erben, aber beliebig viele Interfaces implementieren.
- erbt eine Klasse von einer anderen Klasse, so werden alle Methoden und Member von der Basisklasse in die erbende Klasse übernommen. Ob sie auch sichtbar sind, hängt von den jeweiligen Sichtbarkeitsmodifikatoren ab.
- Die erbende Klasse kann weitere Methoden oder Memberdaten zufügen.
- Die erbende Klasse kann Methoden der Basisklasse überschreiben, dh. Methoden mit der gleichen Signatur, wie es sie bereits in der Basisklasse gibt, definieren. Man spricht von überschriebenen Methoden. Die erbende Klasse stellt in diesem Falle eine Spezialisierung der Basisklasse dar. Überschriebene Methoden verhalten sich in Java in etwa wie virtuelle Methoden in C++, dh. für die Auswahl der tatsächlich auszuführenden Funktion ist der Typ der Referenzvariablen nicht von Bedeutung.

- Instanzvariablen der erbenden Klasse überdecken gleichnamige Instanzvariablen der abgeleiteten Klasse.
- Vererbung wird über das Schlüsselwort extends eingeleitet.
- Wird eine abgeleitete Klasse instanziert, so werden zunächst die Defaultkonstruktoren der Basisklasse(n) gewissermaßen von innen nach außen ausgeführt.
- Ein spezieller Konstruktor der direkten Basisklasse kann über die Anweisung super(<parameterliste>) aufgerufen werden. super(..) muss dabei die erste ausführbare Anweisung im Konstruktor sein.
- Will man von einer überschriebenen Methode die Version der Basisklasse aufrufen, so geht dies über

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

Beispiel zu Implementierung eines Interfaces und Vererbung

```
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){}

  @Override
  public Dimension getPreferredSize()
  {
    return new Dimension(400,200);
  }

  MyPanel(String s) // Constructor
  {
     myString =s;
  }
```

Erbt von java.awt.Component

Implementiert alle Methoden
Von WindowListener

Überschreibt die Methode
GetPreferredSize()
der Klasse java.awt.Component

Statische main-Funktion
Sie hat keinen Zugang zu den Instanzvariablen der Klasse, keine this-Referenz

```
  public static void main(String args[])
  {
    Frame   f=new Frame  (args.length>0?args[0]:"no String - no fun");
    MyPanel p=new MyPanel(args.length>0?args[0]:"no String - no fun");
    System.out.println(p.myString+" in main");
    f.add(p);
    f.addWindowListener(p);
    f.pack(); //setSize(400,200);
    f.setVisible(true);
    f.repaint();
  }

  @Override
  public void paint(Graphics gc)
  {
    //System.out.println(myString+"  in paint");
    gc.drawString(myString,10,20);
  }
}
```

Hier greift die statische main-Funktion über die Referenzvariable p auf die private Variable myString zu. Laert man die main-Funktion in eine andere Klasse aus, geht das nicht.

Überschreibt die Methode
public void paint(Graphics gc)
der Klasse java.awt.Component

```
import java.awt.*;
class MyPanelMain
{
  public static void main(String args[])
  {
    Frame   f=new Frame  (args.length>0?args[0]:"no String - no fun");
    MyPanel p=new MyPanel(args.length>0?args[0]:"no String - no fun");
    System.out.println(p.myString+" in main");
    f.add(p);
    f.addWindowListener(p);
    f.pack(); //setSize(400,200);
    f.setVisible(true);
    f.repaint();
  }
}
```

Compilerfehler:
MyPanelMain.java:8: error: myString has private access in MyPanel
    System.out.println(p.myString+" in main");

Entfernt man die fragliche Zeile, funktioniert das Beispiel wieder. (Aufruf: java MyPanelMain)
Vorausssetzung ist, dass sich beide Klassen (MyPanelMain und MyPanel) im selben Verzeichnis (Package) befinden.
Java sterten mit der main-Funktion der beim Programmaufruf nach java angegebenen Klasse. Dann, zur Laufzeit, werden die darin benötigten Klassen im Namensraum gesucht und nachgeladen. Es ist kein Linker, wie in c nötig. Die virtuelle Maschine „sammelt sich alles nach und nach zusammen“.

Static Member

- Statische Member haben Klassenbezug
- Statische Datenmember existieren nur ein mal für alle Instanzen der Klasse. Sie werden in dem Klassenobjekt, das in Java zu jeder Klasse zur Laufzeit existiert, angelegt.
- Statische Datenmember können über static Initializer oder eine Initialisierung bei der Vereinbarung initialisert werden. Sie dürfen nicht in einem Constructor initialisiert werden. Ein static initializer besteht aus dem Schlüsselwort static, gefolgt von einem Block.

- Statische Methoden können ausgeführt werden, ohne dass es ein Objekt der Klasse, in der sie definiert sind, gibt. Innerhalb der eigenen Klasse werden Sie durch Angabe ihres Namens aufgerufen, ansonsten durch Angabe des Klassennamens . Funktionsname oder Objekt, wenn es ein solches gibt .
- Statische Funktionen können nicht auf die Memberdaten von Objekten der eigenen Klasse zugreifen, sie haben keine this-Referenz. Über eine explizite Referenzvariable geht es natürlich. Siehe Bsp. Vererbung.
- Eine typische statische Funktion ist die main-Funktion.

Objekte und Referenzen

- Wird eine Variable eines Klassentyps oder ein Array vereinbart, so ist dies grundsätzlich nur eine Referenzvariable.
- Das Objekt selbst wird mit new erzeugt.
- Ausnahmen sind beispielsweise initialisierte Strings oder initialisierte Arrays
- String s1="Hans Huckebein";
- Das Objekt existiert, so lange es wenigstens eine Referenzvariable gibt, die dieses Objekt referenziert. Dies muss nicht die Referenzvariable sein, der bei der Erzeugung des Objektes die Referenz zugewiesen wurde.
- Bei der Zuweisung und bei der Übergabe von Objekten als Parameter oder Returnwerte wird ebenfalls immer nur die Referenz übergeben (Ausnahme bildet Remote Method Invocation). Dies ist im Gegensatz zu C/C++ unkritisch, weil die Lebenszeit des Objektes über die Existenz von wenigstens einer Referenzvariablen geregelt ist.
- Ein Objekt kann auch bei der Parameterübergabe an eine Methode erzeugt werden.
- Panel P1=new Panel(new BorderLayout(2,4));