

Abstrakte Klassen und Schnittstellen

Abstrakte Klassen

- Definieren Typen (nicht instanzierbare Klassen)
- Können Methoden- und Attributdefinitionen enthalten
- Methoden können abstrakt definiert werden (Methoden ohne Rumpf)
- Klassen können von abstrakten Klassen erben (Einfachvererbung)

Schnittstellen (Interfaces)

- Definieren Typen
- Enthalten keine Attribute und nur Namen und Signaturen der enthaltenen Methoden
- Können von Klassen implementiert werden (Implementierungsvererbung, mehrfache Vererbung)



Abstrakte Klassen und Schnittstellen

Klassen und abstrakte Klassen

- werden **vererbt** (einfache Vererbung)

Schnittstellen (Interfaces)

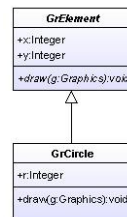
- werden **implementiert** (mehrfache Implementierungsvererbung)



Abstrakte Klassen und Schnittstellen

Abstrakte Klassen

```
public abstract class GrElement {  
    ...  
    public abstract void m ( ... );  
    ...  
}
```



- Definieren Typen und nicht instanzierbare Klassen
- Können Methoden- und Attributdefinitionen enthalten
- Methoden können abstrakt definiert werden (Methoden ohne Rumpf)



Vererbung von abstrakten Methoden

Wenn wir von einer Klasse abstrakte Methoden erben, so haben wir zwei Möglichkeiten:

1. Wir überschreiben alle abstrakten Methoden und implementieren sie. Dann kann die ererbende Klasse instanziiert werden
2. Wir überschreiben die abstrakte Methode nicht, so dass sie normal vererbt wird. Das bedeutet, eine abstrakte Methode bleibt und die Klasse muss wiederum abstrakt sein.

Abstrakte Klassen dürfen abstrakte Methoden enthalten, instanzierbare Klassen nicht.



Abstrakte Klassen (Beispiel)

Abstrakte Klassen

```
public abstract class GrElement {
    int x,y;
    Color fill, outline;
    public abstract void draw (Graphics g);
}

public class GrCircle extends GrElement {
    int r;
    public void draw (Graphics g) {
        g.drawOval(x-r, y-r, 2*r, 2*r);
    }
}

public class GrRectangle extends GrElement {
    int width, height;
    public void draw (Graphics g) {
        g.drawRect(x-width/2, y-height/2, width, height);
    }
}
```

Abstrakte Klasse, nicht
instanzierbar, aber
gemeinsamer Supertyp

Abstrakte Methode

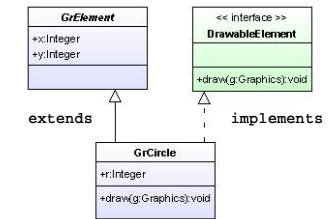
Instanzierbare
Subklassen



Schnittstellen (Interfaces)

Schnittstellen

```
public interface DrawableElement {
    public void draw ( ... );
    ...
}
```



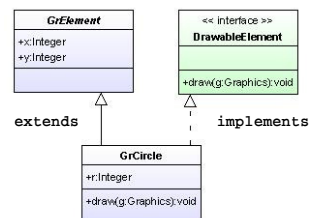
- Definieren Typen
- Enthalten nur Namen und Signaturen von Methoden
- Können von Klassen implementiert werden (**implements**)
- Instanzierbare implementierende Klassen müssen **ALLE** Methoden der Schnittstelle implementieren



Schnittstellen (Interfaces)

Schnittstellen

```
public interface DrawableElement {
    public void draw ( ... );
    ...
}
```



- Definieren **Sichten (Rollen)** von Objekten der implementierenden Klassen
- Objekte können **mehrere** Rollen haben (**mehrere** Schnittstellen implementieren)



Schnittstellen (Beispiel)

Schnittstellen

```
public interface DrawableElement {
    public void draw (Graphics g);
}
```

Interface, keine Instanzen, aber
gemeinsamer Supertyp

Interface-Methode

```
public class GrCircle implements DrawableElement {
    int x, y, r;
    Color fill, outline;
    public void draw (Graphics g) {
        g.drawOval(x-r, y-r, 2*r, 2*r);
    }
}

public class GrRectangle implements DrawableElement {
    int x, y, width, height;
    Color fill, outline;
    public void draw (Graphics g) {
        g.drawRect(x-width/2, y-height/2, width, height);
    }
}
```

Instanzierbare
implementierende
Klassen



Schnittstellen und abstrakte Klassen (Beispiel)

Schnittstellen und abstrakte Klassen

Interface, keine Instanzen, aber gemeinsamer Supertyp

```
public interface DrawableElement {  
    public void draw (Graphics g);  
}
```

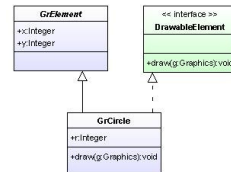
Interface-Methode

```
public abstract class GrElement {  
    int x, y;  
    Color fill, outline  
}
```

Abstrakte Superklasse, nicht instanzierbar

```
public class GrCircle extends GrElement  
    implements DrawableElement {  
    int r;  
    public void draw (Graphics g) {  
        g.drawOval(x-r, y-r, 2*r, 2*r);  
    }  
}
```

Instanzierbare implementierende Klassen



Interessante Systemschnittstellen (aus java.awt.event)

ActionListener
AdjustmentListener
AWTEventListener
ComponentListener
ContainerListener
FocusListener
HierarchyBoundsListener
HierarchyListener
InputMethodListener
ItemListener
KeyListener

MouseListener
MouseMotionListener
MouseWheelListener
TextListener
WindowFocusListener
WindowListener
WindowStateListener



Interessante Systemschnittstellen (z.B. MouseListener)

```
void mouseClicked(MouseEvent e)
```

Invoked when the mouse button has been clicked (pressed and released) on a component.

```
void mousePressed(MouseEvent e)
```

Invoked when a mouse button has been pressed on a component.

```
void mouseReleased(MouseEvent e)
```

Invoked when a mouse button has been released on a component.

```
void mouseEntered(MouseEvent e)
```

Invoked when the mouse enters a component.

```
void mouseExited(MouseEvent e)
```

Invoked when the mouse exits a component.



Interessante Systemschnittstellen (z.B. MouseMotionListener)

```
void mouseDragged(MouseEvent e)
```

Invoked when a mouse button is pressed on a component and then dragged.

```
void mouseMoved(MouseEvent e)
```

Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

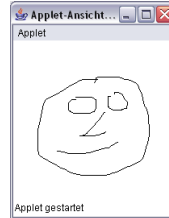


Beispiel: Scribble

```
public class Scribble extends Applet
implements MouseListener, MouseMotionListener {

private int last_x, last_y;

public void mouseClicked(MouseEvent ev) {}
public void mouseReleased(MouseEvent ev) {}
public void mouseEntered(MouseEvent ev) {}
public void mouseExited(MouseEvent ev) {}
public void mouseMoved(MouseEvent ev) {}
public void mousePressed(MouseEvent ev) {
last_x = ev.getX();
last_y = ev.getY();
}
public void mouseDragged(MouseEvent ev) {
int x = ev.getX();
int y = ev.getY();
this.getGraphics().drawLine (last_x, last_y, x, y);
last_x = x;
last_y = y;
}
public void start () {
addMouseListener (this);
addMouseMotionListener (this);
}
}
```



Modifikatoren

- Man unterscheidet
 - Modifikatoren für Klassen
 - Modifikatoren für Attribute
 - Modifikatoren für Methoden
- Funktion von Modifikatoren
 - Sichtbarkeit und Zugriffsmöglichkeiten (Klassen, Methoden, Attribute: public, protected, private)
 - Kennzeichnung als abstrakt (abstract)
 - Steuerung der Vererbbarkeit (final)
 - Klassen- vs. Instanzbezug (static)

```
public class K {
static int i;
protected f() {
...
}
}
```



Modifikatoren für Klassen

Modifikator	Bedeutung
keiner (package)	Klasse ist nur aus demselben Package sichtbar und erreichbar (ggf. aus dem Default-Package).
public	Klasse ist von überall erreichbar.
abstract	Klasse ist abstrakt, d.h. kann nur geerbt, aber nicht instanziiert werden.
final	Klasse ist final, d.h. kann nicht vererbt werden. Es dürfen keine Subklassen gebildet werden.



Modifikatoren für Variablen

Modifikator	Bedeutung
keiner (package)	Variable ist nur innerhalb der eigenen Klasse und des Package erreichbar, zu dem sie gehört.
public	Variable ist überall dort erreichbar, wo auch die Klasse erreichbar ist, zu der sie gehört.
private	Variable ist nur innerhalb der eigenen Klasse erreichbar.
protected	Variable ist nur innerhalb der eigenen Klasse und des Package erreichbar, zu dem sie gehört. Unterklassen können ebenfalls zugreifen.
final	Wert der Variable ist nicht veränderbar (Konstante).
transient	Inhalt der Variablen wird bei Serialisierung ignoriert (da sie flüchtig ist).
static	Variable ist eine Klassenvariable, Klassenbezug



Modifikatoren für Methoden

Modifikator	Bedeutung
keiner (package)	Methode ist nur innerhalb der eigenen Klasse und des Package erreichbar, zu dem sie gehört.
public	Methode ist von überall erreichbar.
private	Methode ist nur innerhalb der eigenen Klasse erreichbar.
protected	Methode ist nur innerhalb der eigenen Klasse und des Package erreichbar, zu dem sie gehört. Unterklassen können ebenfalls zugreifen.
abstract	Methode besitzt keinen Rumpf. Dieser muss von einer Unterklasse implementiert werden. Klasse muss ebenfalls als abstrakte Klasse definiert werden.
final	Methode kann von Unterklassen nicht überschrieben werden.
native	Methode ist in einer anderen Programmiersprache realisiert. Wie bei abstract kein Methodenrumpf.
static	Methode ist eine Klassenmethode, kein Instanzbezug

