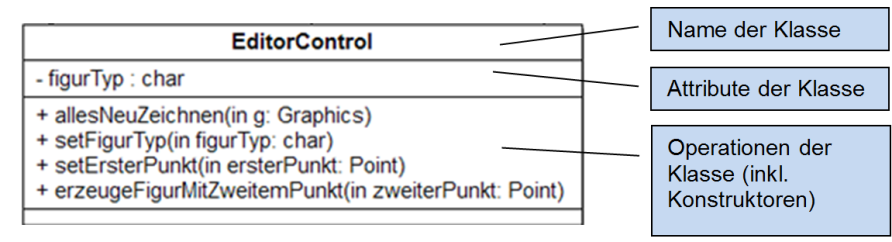
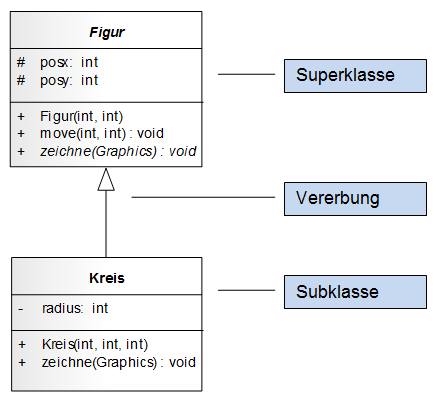
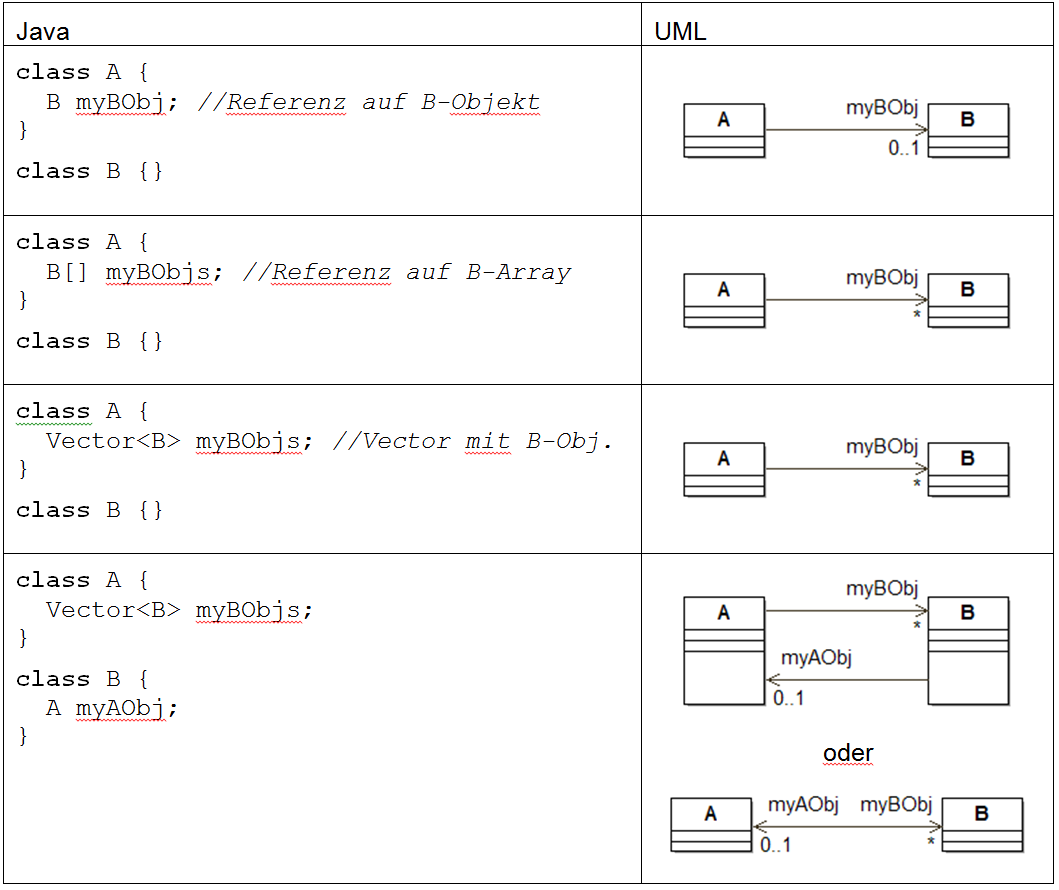
**Klassendiagramm:**

private: -   
public: +  
package: ~  
protected: #

-Statische Operationen (Schlüsselwort **static** bei der Deklaration in Java) werden unterstrichen dargestellt.  
-*Abstrakte* Klassen werden durch eine kursive Schreibweise des Klassennamens dargestellt.  
-Interfaces werden wie Klassen dargestellt. Dem Klassennamen wird ein <<interface>> vorangestellt. Die Darstellung von Attributen und Operationen erfolgt gleich wie bei der Darstellung von Klassen.  
**Assiozation: Vererbung:**  **Polymorphismus und Delegation:  
Beispiel anhand Grafikeditor:**  
public void zeichneFiguren(Graphics g) {  
 for (Figur f : figuren) {  
 f.zeichne(g);  
 }  
}  
**Klasse Figur:**public abstract class Figur {  
 …  
 public abstract void zeichne(Graphics g);  
}  
**Klasse von Rechteck, welche von Figur erbt:**public class Rechteck extends Figur {  
 …  
 public void zeichne(Graphics g) {  
 g.drawRect(posx, posy, breite, hoehe);  
 }  
}

* Zeichnung **delegiert** das eigentliche Zeichnen der Figuren an die Figurobjekte selber (Aufruf der Methode **zeichne()** bei der Klasse **Figur**)
* In den von der Klasse **Figur** abgeleiteten Klassen wird die Methode **zeichne() überschrieben  
  (Nicht mit Delegierung verwechseln!)**
* Zur **Laufzeit** wird über die Figurreferenz die jeweils passende Methode **zeichne()** ausgeführt  
  🡺 dynamisches Binden  
  🡺 Polymorphismus

**Wartbarkeit:**

* Jede Klasse hat definierte, in sich abgeschlossene Aufgabe.
* Keine Redundanzen im Code
* Einfacher, wenn auch über mehrere Klassen verteilter Code

**Erweiterbarkeit:**

* Hinzufügen von weiteren Subklassen zur Klasse Figur, bestehender Code muss nicht angepasst werden.
* Framework

**Robustheit:**

* Offen für neue Anforderungen