Live Vote

PIN: 8VX3

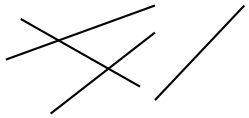
https://ilias.uni-marburg.de/vote/8VX3

10. Klassenerweiterung

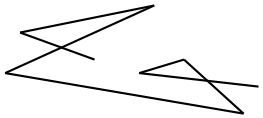
- Klassenerweiterung
- Überschreiben von Methoden, Polymorphie
- Konstruktoren, Attribute
- Abstrakte Klassen

10.1 Motivation

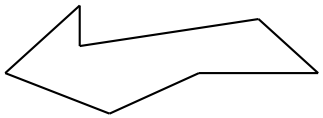
- Bei der Entwicklung von Programmen, die aus mehreren Klassen bestehen, ist oft zu beobachten, dass Klassen sehr ähnlich zueinander sind.
- Betrachten wir dazu folgende Aufgabe:
  - In einem Projekt sollen Klassen zur Manipulation von geometrischen Objekten implementiert werden. Folgende Objekttypen sind von Interesse:



Kantenmenge

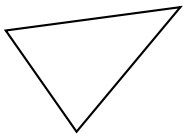


Kantenzug

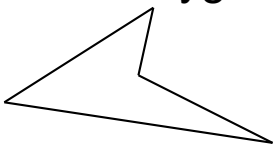


Polygon

orthogonales Rechteck
(O-Rechteck)



Dreieck



Viereck

  - Wir beschließen für jeden Objekttyp eine Klasse zu implementieren.
    - Alle Klassen sollen eine Methode length() haben, um die Gesamtlänge aller Kanten zu berechnen.
    - Für Polygone soll noch eine Methode area() den Flächeninhalt liefern.

Beispiel des Codes (1. Versuch)

```
// Kantenmenge
public class EdgeSet{
    private Edge[] edges;

…

    public double length() {
        double sum = 0.0;
        for (Edge e:edges)
            sum += e.length();
        return sum;
    }
}
```

```
public class Polygon {
    private Edge[] edges;

…

    public double length() {
        double sum = 0.0;
        for (Edge e:edges)
            sum += e.length();
        return sum;
    }

    public double area() {
    …
    }
}
```

Diskussion

- Unsere beiden Klassen sehen sehr ähnlich aus. Die Methode length() ist in beiden Klassen gleich.
  - Dies würde auch bei der Implementierung der anderen Klassen der Fall sein.

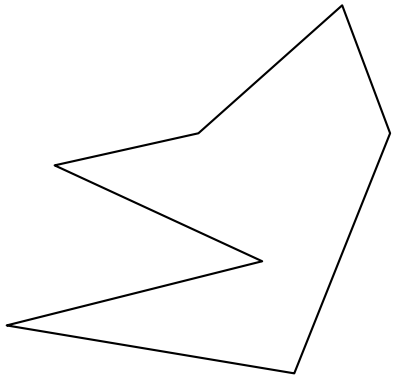
➔ Quellprogramm ist auf Grund der Redundanz nicht wartungsfreundlich.

- Aufgrund der Unabhängigkeit der Klassen kann ein Polygon nicht dort verwendet werden, wo eine Kantenmenge erwartet wird.
  - Aber ein Polygon kann als Spezialfall einer Kantenmenge mit folgenden Zusatzeigenschaften angesehen werden.

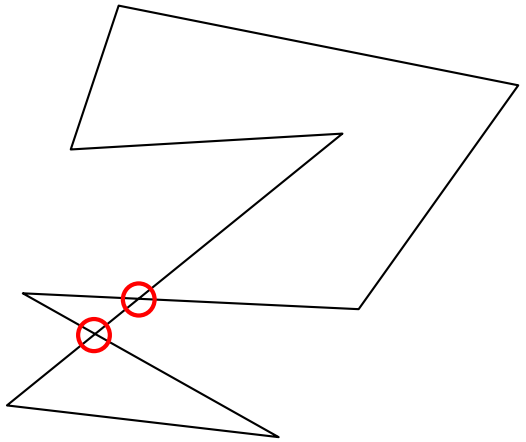
    - Der Anfangspunkt einer Kante ist der Endpunkt der Vorgängerkante.
    - Kanten schneiden sich nicht (abgesehen in den Endpunkten zwei aufeinanderfolgender Kanten).

Einschub: Begriff des Polygons

- Ein Polygon ist ein „**Vieleck**“ mit einer Folge von Kanten.
  - Zwei aufeinanderfolgende Kanten berühren sich in Ihren Endpunkten.
  - Ansonsten gibt es keine Schnitte zwischen Kanten.



ist ein Polygon



ist kein Polygon

Beispiel des Codes (2. Versuch)

```
public class OneClass {
    private Edge[] edges;
    boolean istPolygon;

    public OneClass(Edge[] in, boolean poly) {
        this.edges = new Edge[in.length];
        istPolygon = poly;
        for (int i = 0; i < in.length; i++) {
            this.edges[i] = in[i];
        }
        if (poly)
          // Hier müssen noch einige Überprüfungen erledigt werden.
    }

    public double length() {
        double sum = 0.0;
        for (Edge e:edges)
            sum += e.length();
        return sum;
    }


    public double area() {
         // Wie kann man verhindern, dass die Methode für ein
         // Objekt aufgerufen wird bei dem istPolygon „false“ ist??
    }

}
```

Diskussion

- Lösung mit einer Klasse OneClass
  - Die Methode length() wird nur noch einmal implementiert.
  - Wir können jetzt Objekte, die Polygone sind, auch nutzen, wenn nur ein EdgeSet erwartet wird.
    - Umgekehrt geht es aber leider auch!!

  - Wir müssen jetzt selbst aufpassen, dass gewisse Methoden, wie z. B. die Methode area(), nur für Polygone, aber nicht für Kantenmenge aufgerufen werden.
    - Das kann sehr leicht zu Fehlern in unserem Programm führen!

  - Wir benötigen zusätzlich eine Boolesche Variable, um zu unterscheiden, ob das Objekt ein Polygon oder eine Kantenmenge ist.
    - Dies verkompliziert an vielen Stellen die Programmierung

- **Beide bisherige Lösungen sind also nicht wirklich gut.**
  - Mit dem Konzept der Klassenerweiterung können wir aber diese Nachteile vermeiden.

10.2 Klassenerweiterung in Java

- Zunächst sollen folgende vereinfachende Annahmen gelten:
  - Klassen, Methoden und Datenfelder sind **public**

- Sei K eine Klasse. Dann lässt sich eine neue Klasse IsA_K

```
public class IsA_K extends K { … }
```

erzeugen, wobei IsA_K alle Objekteigenschaften (Datenfelder **ohne** Schlüsselwort static und Objektmethoden) der Klasse K besitzt.

Neben diesen Eigenschaften kann IsA_K weitere Eigenschaften besitzen, die dann wie gewohnt in die Klassendefinition mit aufgenommen werden.

Beispiel des Codes (3. Versuch)

```
public class EdgeSet{
    private Edge[] edges;

…

    public double length() {
        double sum = 0.0;
        for (Edge e:edges)
            sum += e.length();
        return sum;
    }
}
```

```
public class Polygon extends EdgeSet {
    // edges bereits in EdgeSet definiert
    // Methode length wird aus EdgeSet
    // übernommen.

    public Polygon(Edge[] edges) {
        …
        // Details dazu später

    }

    public double area() {
        ...
    }
}
```

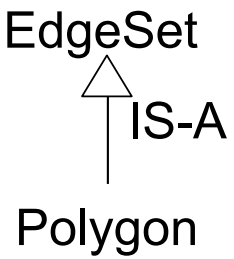
Verwendung der Klassen

- Wir können die Klassen wie bisher verwenden.
  - Erzeugen von Objekten
    - Polygon poly = new Polygon(/* ggf. Parameter einfügen */);
    - EdgeSet edges = new EdgeSet(/* ggf. Parameter einfügen */);
  - Folgende Methodenaufrufe sind möglich:
    - double r1 = edges.length();
    - double r2 = poly.area();
    - double r3 = poly.length();
      - Dies ist möglich, da die Klasse Polygon alles bekommt, was auch in der Klasse EdgeSet vorhanden ist.
    - double r4 = edges.area()
      - Dies ist nicht möglich, da die Methode area nicht in der Klasse EdgeSet existiert. → Fehler fällt bereits dem Compiler auf!

Wichtige Begriffe

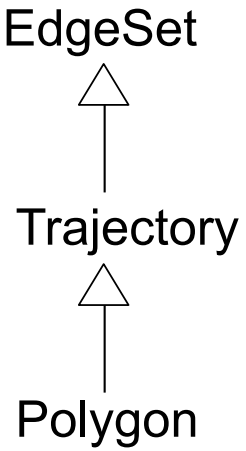
- Die erweiternde Klasse wird als **Unterklasse** (Subklasse) und die erweiterte Klasse als **Oberklasse** (Superklasse) bezeichnet.
- In der Literatur wird statt **Klassenerweiterung** auch der Begriff **Vererbung** verwendet.
  - In dieser Vorlesung werden wir Klassenerweiterung als Begriff benutzen.

- Graphisch können solche Klassenerweiterungen mit einem Pfeil von der Unterklasse zur Oberklasse dargestellt werden.
  - Um zu verdeutlichen, dass die Unterklasse über alle Eigenschaften der Oberklasse verfügt, können wir an der Kante das Label „IS-A“ hinzufügen.
    - Ein Polygon-Objekt ist ein (IS-A) EdgeSet-Objekt.
  - Standard „UML“ Notation: Pfeil mit nicht-gefülltem Dreieck



Klassenerweiterung ist transitiv

- Man kann eine Klasse erweitern, die bereits eine andere Klasse erweitert hat.
  - public class Trajectory extends EdgeSet {… }
  - public class Polygon extends Trajectory {…}
    - Damit bekommt Polygon alles, was in EdgeSet und Trajectory **nicht** mit dem Modifier static definiert wurde.



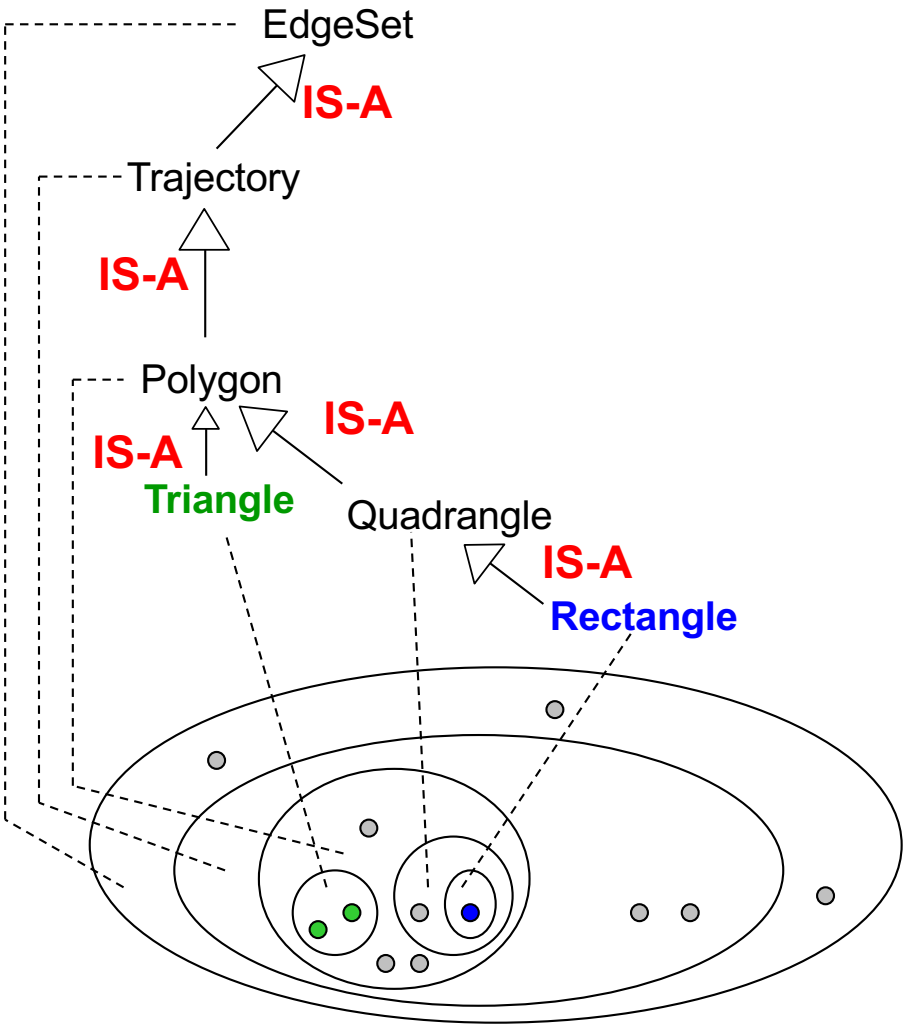
- Beispiel
  - Falls EdgeSet die Methode length() besitzt, dann ist die Methode so auch in Polygon vorhanden.

```
Polygon p = new Polygon(points);
double res = p.length();
```

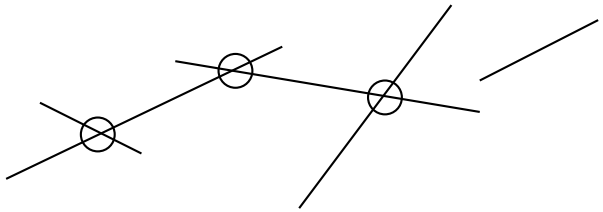
Oberklasse Object

- In Java gibt es eine Klasse Object, von der alle Klassen direkt oder indirekt erben.
  - Die Klasse Object hat keine Oberklasse.
  - Die Klasse Object besitzt z. B. eine Methode toString(), um ein Objekt als String zu repräsentieren.
    - Diese Methode wird bei der Ausgabe des Objekts von den Methoden print und println genutzt.
  - Klassen, die keine extends-Klausel besitzen, erweitern trotzdem die Klasse Object.
  - Wir werden später noch im Detail über diese Klasse sprechen.

Veranschaulichung von IS-A-Beziehungen



- Eigenschaften von EdgeSet
  - Anzahl der Kanten in der Menge
  - Methode length()
  - Methode intersection() zur Berechnung aller Schnittpunkte



Methode intersection()

- Objekte der anderen Klassen können zusätzlich noch weitere Eigenschaften besitzen z.B.
  - Trajectory: maxAngle
  - Polygon: area
  - ….

10.3 Überschreiben

- Die Klassenerweiterung in Java ermöglicht, dass Methoden aus Oberklassen auch in Unterklassen nochmals neu implementiert werden.
  - Ein Grund dafür ist, dass man in den Unterklassen häufig die spezifischen Eigenschaften der Objekte ausnutzen kann, um die Funktionalität schneller zu implementieren.
  - Ein zweiter Grund ist, dass bei einer Methode, die den Zustand des Objekts verändert, man noch die spezifischen Eigenschaften des Objekts der Unterklasse sicherstellen muss.

Beispiel (Schnittpunkte)

- Klasse EdgeSet

der Klasse EdgeSet müssen wir jede Kante mit allen anderen Kanten auf einen Schnitt testen.
Bei n Kanten ist die Anzahl der Vergleiche (n+1)*n/2.

- Klasse Polygon
  - In der Klasse Polygon wissen wir bereits, dass nur die Endpunkte der Kanten sich berühren, aber sonst keine weiteren Schnitte vorliegen.
    - Es genügt also hier linear alle n Kanten zu besuchen und deren Endpunkte auszugeben.
    - Wir bieten deshalb eine Neuimplementierung der Methode in der Klasse Polygon an.

```
public Point[] intersection() {
    Point[] points = new Point[edges.length];
    for (int i=0; i < points.length; i++)
        points[i] = edges[i].getStart();
    return points;
}
```

Beispiel (Einfügen von Kanten)

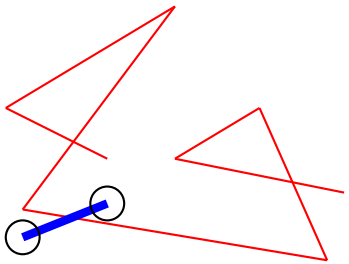
- In der Klasse EdgeSet soll eine Methode angeboten werden, um neue Kanten in die Menge einzufügen.
  - Unsere Lösung ist deshalb in dem Array etwas mehr Platz zu reservieren als notwendig und die Kanten im vorderen Teil des Arrays abzuspeichern.
    - Wir merken uns in einem Datenfeld `size` die aktuell belegten Plätze im Array.
  - Beim Einfügen einer neue Kante wird dann einfach die nächst freie Position benutzt.

```
public class EdgeSet {
    private Edge[] edges;
    private int size;

    ...

    public void insert(Edge newEdge) {
        if (size < edges.length)
            edges[size++] = newEdge;
        else
            System.out.println("Storage Overflow");
    }
}
```

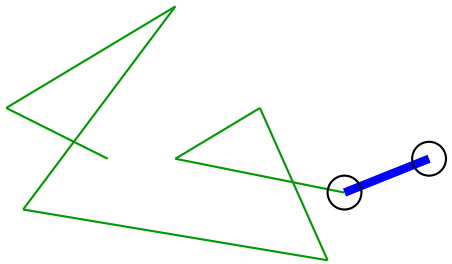
Kommentar: Irgendwann wird das Array zu klein sein. Wir lernen später Datenstrukturen kennen, die wachsen können.

Beispiel (Einfügen von Kanten)

- In der Klasse Trajectory können wir leider die bisherige Methode aus der Klasse EdgeSet nicht benutzen.
  - Es muss sichergestellt werden, dass ein Objekt der Klasse Trajectory aus einer zusammenhängenden Folge von Kanten besteht.
- Diese Eigenschaft kann sichergestellt werden, wenn wir das Einfügen einer neuen Kante nur am Ende oder Anfang des Kantenzugs erlauben.
  - Die neue Kante newEdge muss also einen gemeinsamen Eckpunkt mit der ersten Kane oder der letztden Kante im Kantenzug haben.



Nicht erlaubt!



Erlaubt

- Dadurch wird garantiert, dass das Objekt nach dem Einfügen weiterhin die Eigenschaft eines Kantenzugs besitzt.

Definition (Überschreiben)

- Beim Überschreiben einer Methode m_O aus einer Oberklasse O durch eine Methode m_U in einer Unterklasse U gelten folgende Eigenschaften:
  - Die Rückgabetypen der Methoden m_U und m_O müssen gleich sein.
  - Die Signaturen der Methoden m_U und m_O müssen übereinstimmen.

- Bei einer Änderungsoperation wird sichergestellt, dass die Eigenschaften des Objekts einer Klasse stets erhalten bleiben.
  - Diese Eigenschaften werden auch als ***Klasseninvarianten*** bezeichnet.
    - Alle Objekte der Klasse erfüllen die Klasseninvarianten.
  - Beispiel einer Klasseninvariante
    - Objekte der Klasse Trajectory repräsentieren eine Folge von Kanten, die sich in ihren Endpunkten berühren.

Definition (Überschreiben)

- Beim Überschreiben einer Methode m_O aus einer Oberklasse O durch eine Methode m_U in einer Unterklasse U gelten folgende Eigenschaften:
  - Die Rückgabetypen der Methoden m_U und m_O müssen gleich sein.
  - Die Signaturen der Metho

Java stellt das nicht automatisch sicher, bietet aber Unterstützung, die wir später kennen lernen werden.

- Bei einer Änderungsoperation wird sichergestellt, dass die Eigenschaften des Objekts einer Klasse stets erhalten bleiben.
  - Diese Eigenschaften werden auch als ***Klasseninvarianten*** bezeichnet.
    - Alle Objekte der Klasse erfüllen die Klasseninvarianten.
  - Beispiel einer Klasseninvariante
    - Objekte der Klasse Trajectory repräsentieren eine Folge von Kanten, die sich in ihren Endpunkten berühren.

Verwendung überschriebener Methoden

- Für die Objekte der Unterklasse steht **nicht** mehr die ursprüngliche Implementierung der Methode zur Verfügung.

- Beispiel

```
Edge[] edges = new Edge[10];
…
Trajectory trajectory = new Trajectory(edges);

Point p = new Point(1.0, 2.0), q = new Point(2.0, 3.0);

// Hier wird immer die Methode aus der Klasse Trajectory verwendet.
trajectory.insert(new Edge(p,q));
```

Verbieten von Überschreiben

- Soll das Überschreiben einer Methode m_O aus einer Oberklasse O in einer Unterklasse U verhindert werden, so muss bei der Definition von m_O das Schlüsselwort **final** verwendet werden.
  - Beispiel
    - Die Methode length() aus der Klasse EdgeSet soll in den Unterklasse nicht mehr überschrieben werden.

```
public class EdgeSet{
    private Edge[] edges;
        …
    public final double length() {
        …
    }
}
```

- In einer Unterklasse kann die final-Eigenschaft der Methode nicht mehr verändert werden.

Einmal final immer final.

Unterschiede zum Überladen

- In Java kann es in einer Klasse mehrere Methoden mit dem gleichen Namen geben.
  - Man spricht dann vom **Überladen der Methode**.
- Methoden mit gleichem Namen müssen sich aber in ihrer Signatur unterscheiden.
  - Die Signatur besteht aus dem Namen der Methode und dem Namen der Typen der Parametervariablen.
  - Der Rückgabetyp hat dabei keine Relevanz!

- Beispiel (Einfügen neuer Kanten)
  - Zusätzlich zu der Methode zum Einfügen einer einzelnen Kante, soll noch ein Einfügen eines Array von Kanten unterstützt werden.
  - Wir überladen deshalb die Methode insert und bieten die Methode nochmal mit dem Parametertyp Edge[] an.

Beispiel (Überladen)

- Hinzufügen einer zweiten Methode `insert` zu EdgeSet
  - Zusätzlich zu der Methode zum Einfügen einer einzelnen Kante, soll noch ein Einfügen eines Array von Kanten unterstützt werden.
  - Die Methode `insert` wird überladen.

```
public class EdgeSet {
    private Edge[] edges;
    private int size;

    ...

    public void insert(Edge newEdge) {
        if (size < edges.length)
            edges[size++] = newEdge;
        else
            System.out.println("Storage Overflow");
    }

    public void insert(Edge[] newEdges) {
        if (size + newEdges.length < edges.length)
            for (int i = 0; i < neue.length; i++)
                edges[size++] = newEdges[i];
        else
            System.out.println("Storage Overflow");
    }
}
```

Neudeklaration von Datenfeldern

- Ähnlich wie bei Methoden können auch Datenfelder hinzugefügt werden.
- Es gibt aber einen **fundamentalen Unterschied** zum Überschreiben von Methoden:
  - Es wird dadurch nur ein weiteres Datenfeld gleichen Namens angelegt!
  - Das ursprüngliche Datenfeld existiert weiterhin!

Für Datenfelder gilt:

- Objekte der erweiterten Klasse UK können sowohl auf das neu deklarierte Datenfeld als auch auf das ursprüngliche Datenfeld zugreifen.
- Sei o eine Referenzvariable der Klasse UK und df ein Datenfeld der Oberklasse OK, das in der Klasse UK neu deklariert wurde. Dann gilt:
  - Zugriff auf das neu deklarierte Datenfeld: o.df
  - Zugriff auf das ursprüngliche Datenfeld: ((OK) o).df
- Prinzipiell sollte das Definieren von Datenfeldern mit gleichem Namen wie Datenfelder in der Oberklasse vermieden werden.

Beispiel: Neudeklaration von Datenfelder

```
class OK {
   public float a = 14.0f;

   OK() { }
}
```

```
class UK extends OK {
   public int a = 27;

   UK() { }
}
```

```
public class UKTest {
   public static void main(String[] args) {
      UK k = new UK();
      System.out.println(k.a);
      System.out.println(((OK) k).a);
   }
}
```

Ein Objekt der Klasse UK hat beide Datenfelder!

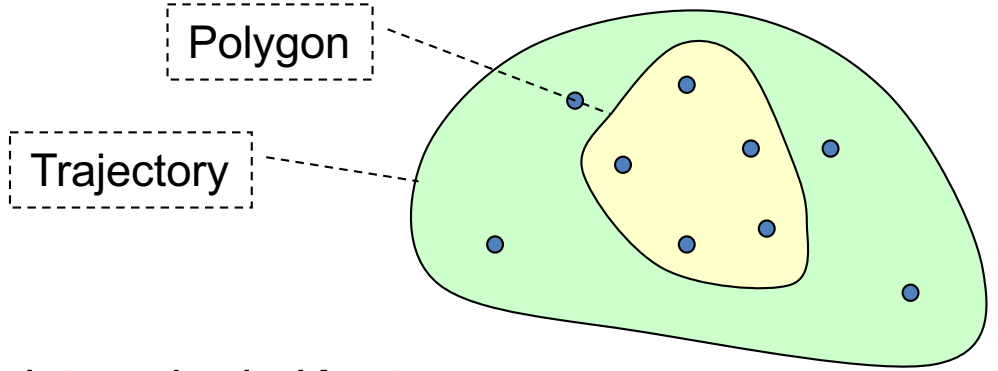
10.4 Polymorphie

- Bereits bekannt:
  - Klassen können als Datentypen von Variablen genutzt werden.

- Bei Ober- und Unterklassen können wir das bei Schnittstellen (Interfaces) bekannte Prinzip verwenden:
  - Eine Referenzvariable vom Typ der Oberklasse darf auf ein Objekt der Klasse K verweisen.

Beispiel

```
Polygon poly;       // kann auf Polygon-Objekte verweisen
Trajectory traj;    // kann auf Polygon- und Kantenzug-Objekte verweisen
```




```
poly = new Polygon();
traj = poly;                    // Polygon ist auch ein Kantenzug
Object o = traj;                // Kantenzug ist ein Objekt
Trajectory[ ] geoArr = {new Polygon(), new Trajectory()};
```

Zentrale Regel der Polymorphie

1. **Beim Methodenaufruf** bestimmt stets die **Klasse des Objekts**, welche Implementierung benutzt wird.
2. **Beim Zugriff auf ein Datenfeld** bestimmt stets **die Klasse der Referenzvariablen**, welche Deklaration gültig ist.

Beispiel 1:

- Bei den folgenden Methodenaufrufen

```
Trajectory[ ] geoArr = {new Polygon(), new Trajectory()};
Point[] points = geoArr[0].intersection();
points = geoArr[1].intersection();
```

werden verschiedene Implementierungen von intersection() angesprochen.

Zentrale Regel der Polymorphie

Beispiel 2:

```
class EdgeSet {
    public final String name = "EdgeSet";
    public String getName() {
        return name;
    }
}


class Trajectory extends EdgeSet{
    public final String name = "Trajectory";
    public String getName() {
        return name;
    }
}
```

```
class Main {
    public static void main(String[] args) {
        EdgeSet es1 = new EdgeSet();
        Trajectory t = new Trajectory();
        EdgeSet es2 = t;
        System.out.println("Feldzugriff:");
        System.out.println(es1.name);
        System.out.println(t.name);
        System.out.println(es2.name);
        System.out.println("\nMethodenaufruf:");
        System.out.println(es1.getName());
        System.out.println(t.getName());
        System.out.println(es2.getName());
    }
}
```

Welche Ausgabe ist korrekt:

a)
```
Feldzugriff:
EdgeSet
Trajectory
EdgeSet

Methodenaufruf:
EdgeSet
Trajectory
EdgeSet
```

b)
```
Feldzugriff:
EdgeSet
Trajectory
EdgeSet

Methodenaufruf:
EdgeSet
Trajectory
Trajectory
```

c)
```
Feldzugriff:
EdgeSet
EdgeSet
Trajectory

Methodenaufruf:
EdgeSet
Trajectory
Trajectory
```

Live Vote

PIN: 1UQN

https://ilias.uni-marburg.de/vote/1UQN

Typkonvertierung von Objekten

- Wurde ein Objekt einer Unterklasse einer Referenzvariablen vom Typ der Oberklasse zugewiesen, so ist es über eine explizite Konvertierung erlaubt, dieses Objekt wieder an eine Referenzvariable der Unterklasse zu übergeben.

- **Beispiele:**

```
Trajectory traj1 = new Polygon();       // lz1 verweist auf Objekt der Klasse Polygon
Polygon poly1 = (Polygon) traj1;        // Konvertierung ist somit erlaubt.
```

```
Trajectory traj2 = new Trajectory();
Polygon poly2= (Polygon) traj2;
// Dies wird zwar ohne Fehlermeldung übersetzt, führt aber bei der
// Programmausführung zu einem Fehler, der eine sogenannte Exception auslöst.
```