Objektorientierte Konzepte

- Objekt
- Klasse
- enthaltender Polymorphismus
- Vererbung

Objekt

- Objekt ist Kapsel mit Variablen und Routinen
- Interaktionen zwischen Objekten durch Senden von Nachrichten und Reagieren auf empfangene Nachrichten
- Eigenschaften jedes Objekts:
  - Identität (identisch = mehrere Namen für ein Objekt)
  - Zustand (gleich $\neq$ identisch)
  - Verhalten (Reaktion auf Nachrichten)
- Simulation der realen Welt

Beispiel eines Objekts

Objekt: einStack

nicht öffentliche (private) Variablen:

`elems`:	`"a"`	`"b"`	`"c"`	`null`	`null`
`size`:	3				

öffentlich aufrufbare Routinen:

`push`:	Implementierung der Routine
`pop`:	Implementierung der Routine

Objektschnittstelle, Datenabstraktion

- Schnittstelle beschreibt, was von außen sichtbar ist
- data hiding (black box, grey box)
- Datenabstraktion = data hiding + Kapselung
- Datenabstraktion wichtig für Wartung
- mehrere unterschiedliche Schnittstellen pro Objekt
  = Datenabstraktionen für unterschiedliche Sichtweisen

Klasse

- jedes Objekt ist Instanz genau einer Klasse
- Klasse beschreibt Struktur ihrer Instanzen
- Konstruktoren zur Erzeugung aller Instanzen
- alle Instanzen einer Klasse haben dieselben Schnittstellen und dasselbe Verhalten
- nicht-identische Instanzen haben unterschiedliche Instanzvariablen, möglicherweise unterschiedliche Zustände

Beispiel einer Klasse (1)

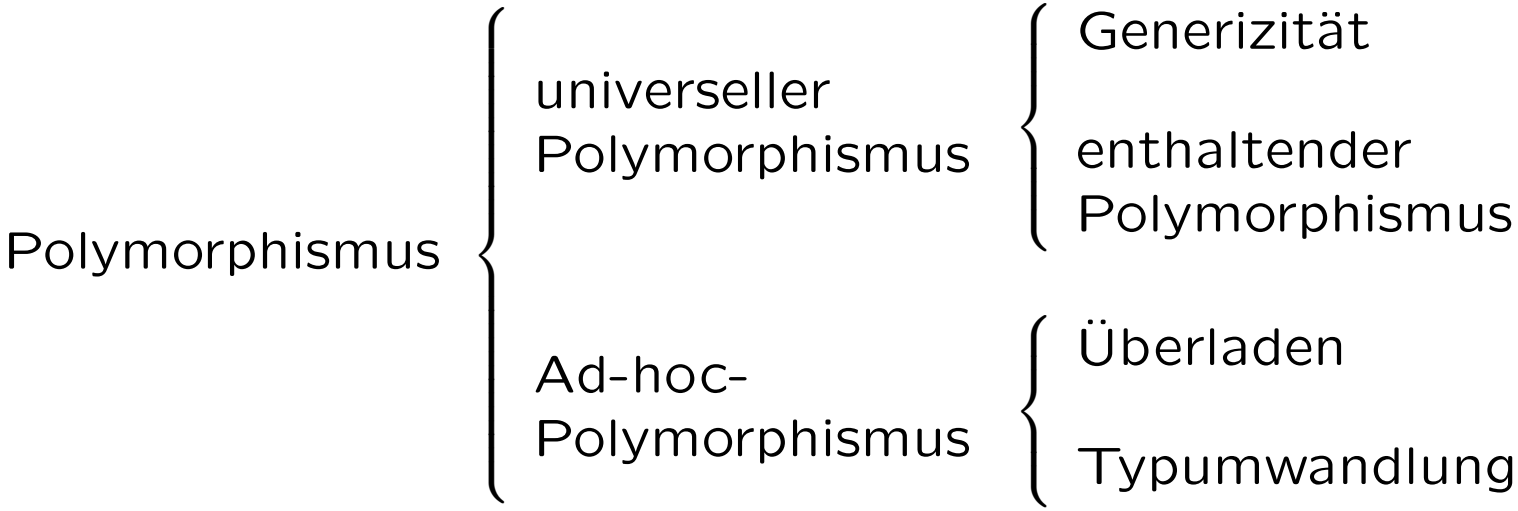
```
class Stack {
    private String[] elems;
    private int size = 0;
    public Stack (int sz) { elems = new String[sz]; }
    public void push (String elem) {
        if (size < elems.length) { elems[size] = elem;
                                   size = size + 1;    }
    }
    public String pop() {
        if (size > 0) { size = size - 1;
                        return elems[size]; }
        else return null;
    }
}
```

Beispiel einer Klasse (2)

```
class StackTest {
    public static void main (String[] args) {
        Stack s = new Stack(5);
        int i = 0;
        while (i < args.length) {
            s.push(args[i]);
            i = i + 1;
        }
        while (i > 0) {
            i = i - 1;
            System.out.println(s.pop());
        }
    }
}
```

Polymorphismus

Objekt hat gleichzeitig mehrere Typen



Typen einer Variable: deklarierter, statischer, dynamischer Typ

Enthaltender Polymorphismus

- Typ entspricht (im Wesentlichen) einer Objektschnittstelle
- Instanzenmenge des Obertyps enthält Instanzenmenge des Untertyps — Beispiel: ein `Student` ist eine `Person`
- Ersetzbarkeitsprinzip:
  U ist Untertyp von T wenn Instanz von U überall verwendbar ist, wo Instanz von T erwartet wird
- dynamischer Typ möglicherweise spezieller als statischer und deklarierter Typ
- dynamisches Binden

Andere Arten des Polymorphismus

- nicht spezifisch für objektorientierte Programmierung
- Generizität (parametrischer Polymorphismus): Programm enthält Typparameter, die durch Typen ersetzt werden
- Überladen: verschiedene Routinen gleichen Namens; Unterscheidung durch deklarierte Typen
- Typumwandlung: semantische Operation; Argument einer Routine in Instanz des Parametertyps umgewandelt

Vererbung

- Ableitung neuer Klassen aus existierenden Klassen
- Änderungsmöglichkeiten:
  - Erweiterung um Methoden und Variablen
  - Überschreiben von Methoden
- Zusammenhang mit enthaltendem Polymorphismus:
  Unterklassenbeziehung entspricht Untertypbeziehung
  (Java, C#, C++, . . . , aber nicht generell)

Beispiel für Vererbung (Oberklasse)

```
class Stack {
    private String[] elems;
    private int size = 0;
    public Stack (int sz) { elems = new String[sz]; }
    public void push (String elem) {
        if (size < elems.length) { elems[size] = elem;
                                   size = size + 1;    }
    }
    public String pop() {
        if (size > 0) { size = size - 1;
                        return elems[size]; }
        else return null;
    }
}
```

Beispiel für Vererbung

```
class CounterStack extends Stack {
    private int counter;
    public CounterStack (int sz, int c) {
        super(sz);
        counter = c;
    }
    public void push (String elem) {
        counter = counter + 1;
        super.push(elem);
    }
    public void count() {
        push (Integer.toString(counter));
    }
}
```