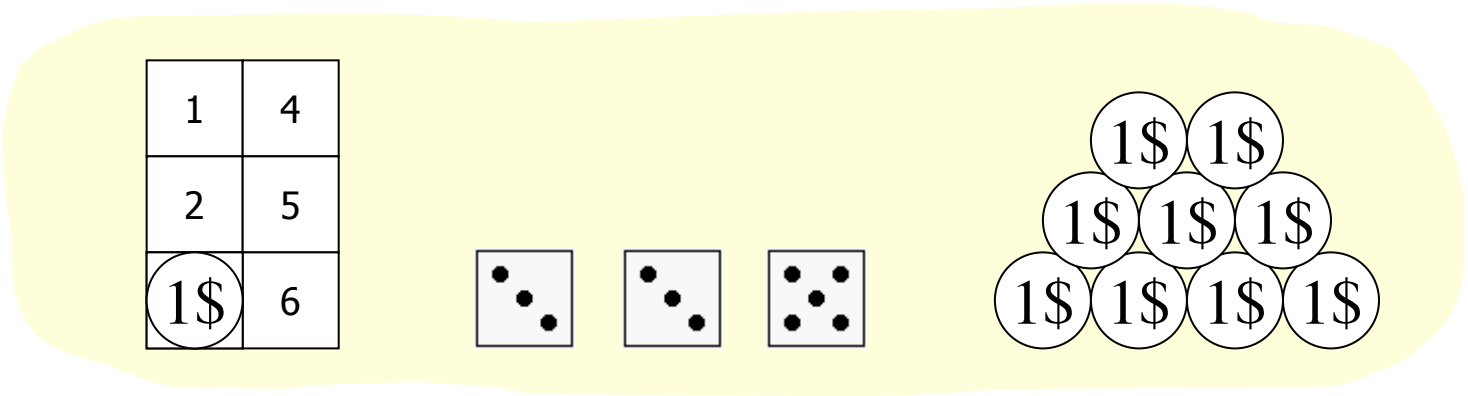
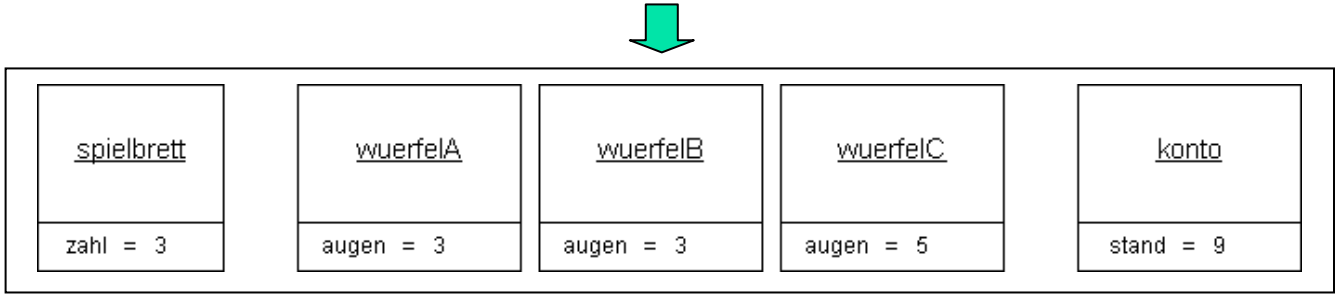
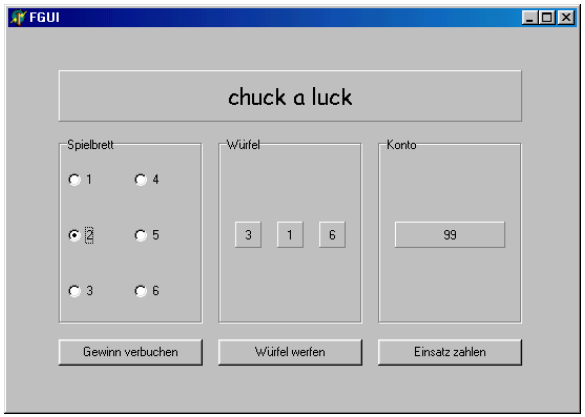
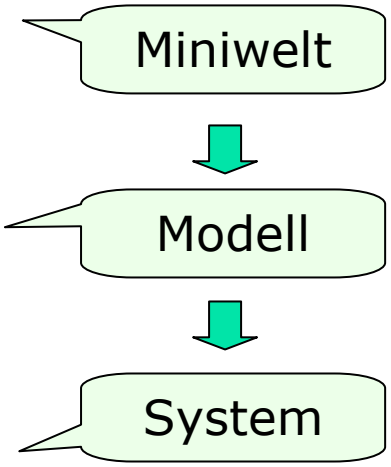
Grundkonzepte der objektorientierten Programmierung Teil 1

Objektorientierte Modellierung









Warum Objektorientierung?

These: Marktreife Software ist sehr schwer zu entwickeln.
(Balzert: Lehrbuch der Software-Technik, S. 27)

„Berühmte" Software-Fehler

• Eine Schweizer Bank überwies einigen Kunden den Betrag, den die Kunden selbst an die Bank hätten überweisen müssen. Der Fehler wurde erst nach 9 Monaten bemerkt, der Bank entstand inzwischen ein Schaden von ca. 200000 DM.

• Beim Bau der Rakete Ariane-5 wurde das bewährte Navigationssystem der Ariane-4 komplett übernommen. Daher konnten die für die Softwaretests vorgesehenen Ausgaben von mehr als 100 Millionen DM drastisch reduziert werden. Eine praxisgerechte Simulation der bei einem Ariane-5-Start anfallenden Datenmengen fand nicht statt. Da die Ariane-5 durch zusätzliche Triebwerke schneller startete als ihr Vorgängermodell, konnten die Computer die Daten nicht schnell genug verarbeiten. Die Ariane-5 musste eine Minute nach dem Start gesprengt werden (Schaden: mehrere Milliarden DM).

• Der als schwarzer Montag in die Börsengeschichte eingegangene 19.10.1987 war zum Großteil durch falsch programmierte Börsenprogramme großer Banken hervorgegangen. Als der Kurs einiger Aktien an diesem Tag besonders stark fiel, verkauften einige Programme automatisch die entsprechenden Aktien. Durch einen Lawineneffekt kam es zu einem weltweit drastischen Absinken der Kurse.

(vgl. Bähnisch: Praktische Informatik 1, S. 193 ff)

Warum Objektorientierung?

These: Marktreife Software ist sehr schwer zu entwickeln.
(Balzert: Lehrbuch der Software-Technik, S. 27)

Statistische Untersuchungen

- Nach einer Untersuchung der Standish Group aus dem Jahr 1995 werden 31% aller Softwareprojekte erfolglos abgebrochen, 53% kosten viel mehr als die ursprünglich vorgesehenen Kosten und nur 16% werden rechtzeitig und ohne Kostenüberschreitung fertiggestellt.

- Normale Software enthält durchschnittlich 25 Fehler pro 1000 Programmzeilen. Bei guter Software kann der Schnitt auf ca. 2 Fehler pro 1000 Programmzeilen reduziert werden. Beim Betriebssystem Windows-95, das aus 10 Millionen Programmzeilen besteht, muss man davon ausgehen, dass es bei der Auslieferung etwa 200 000 Fehler enthält.

(vgl. http://www.standishgroup.com/chaos.html)

Warum Objektorientierung?

These: Objektorientierung ist die derzeitige Antwort auf die gestiegene Komplexität der Softwareentwicklung.
(Oestereich: Objektorientierte Software-Entwicklung, S. 30)

„Hauptprobleme" der Software-Entwicklung

„Eines der Hauptprobleme der Software-Entwicklung ist die Entwicklung zuverlässiger Software. Man bezeichnet ein Programm als zuverlässig, wenn es sich im Betrieb so verhält, wie man es aufgrund der Anforderungen an das Programm erwartet. Je umfangreicher ein Software-Projekt ist, desto unwahrscheinlicher ist es, dass sein Ergebnis jemals fehlerfrei wird. Man muss sogar davon ausgehen, dass es unmöglich ist, umfangreiche Software-Produkte vollständig fehlerfrei zu entwickeln."

„Das andere Hauptproblem der Software-Entwicklung besteht darin, die Software so zu entwickeln, dass sie später problemlos geändert werden kann. Für die Entwickler ist es vorteilhaft, wenn die Entwicklung eines Software-Systems nach seiner Fertigstellung beendet ist. Eine weitergehende Betrachtung zeigt jedoch, dass die Entwicklung eines Software-Systems ein evolutionärer Prozess ist, der oft sehr lange währt, und dessen Ende womöglich nicht abzusehen ist."

(Gumm, S. 661, 662)

Zur Orientierung

Objektorientierung zeigt ihre Vorteile in der Regel erst bei komplexeren Software-Entwicklungsaufgaben.

Komplexe Software-Entwicklungsaufgaben eignen sich jedoch nicht gut, um die Grundkonzepte objektorientierter Programmierung zu erlernen.

Im folgenden sollen diese Grundkonzepte daher zunächst an einfachen und überschaubaren Problemkontexten entwickelt werden.

Danach werden dann umfangreichere Problemstellungen bearbeitet, die den Nutzen der Objektorientierung illustrieren sollen.

Teil 1

Objekte und Klassen

Das Würfelspiel „chuck a luck“

Einsatz zahlen und Zahl tippen

⇩

Würfel werfen

⇩

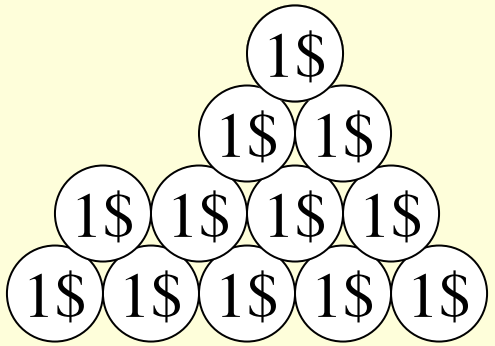
Gewinn auszahlen

Einsatz: 1 $

Gewinn:

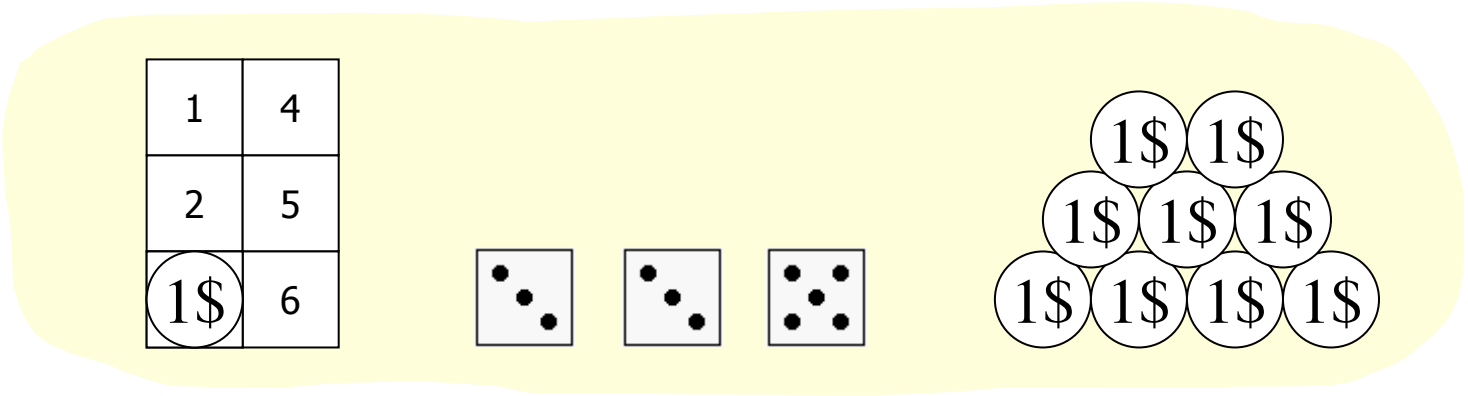
0 Treffer:
1 Treffer: Einsatz + 1 $
2 Treffer: Einsatz + 2 $
3 Treffer: Einsatz + 3 $

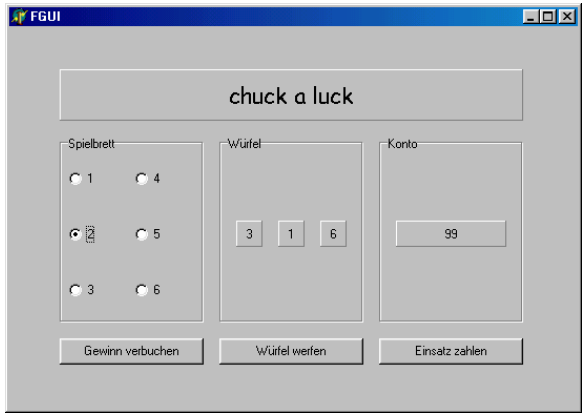
1	4
2	5
3	6

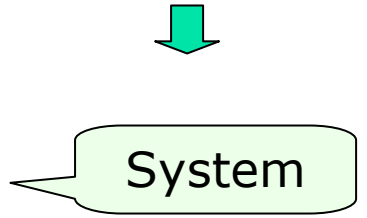


Zielsetzung

Ziel ist es, ein Simulationsprogramm zu entwickeln, mit dem das Würfelspiel „chuck a luck“ am Rechner gespielt werden kann. Am Beispiel dieses einfachen und überschaubaren Systems sollen Grundkonzepte der objektorientierten Programmierung verdeutlicht werden.







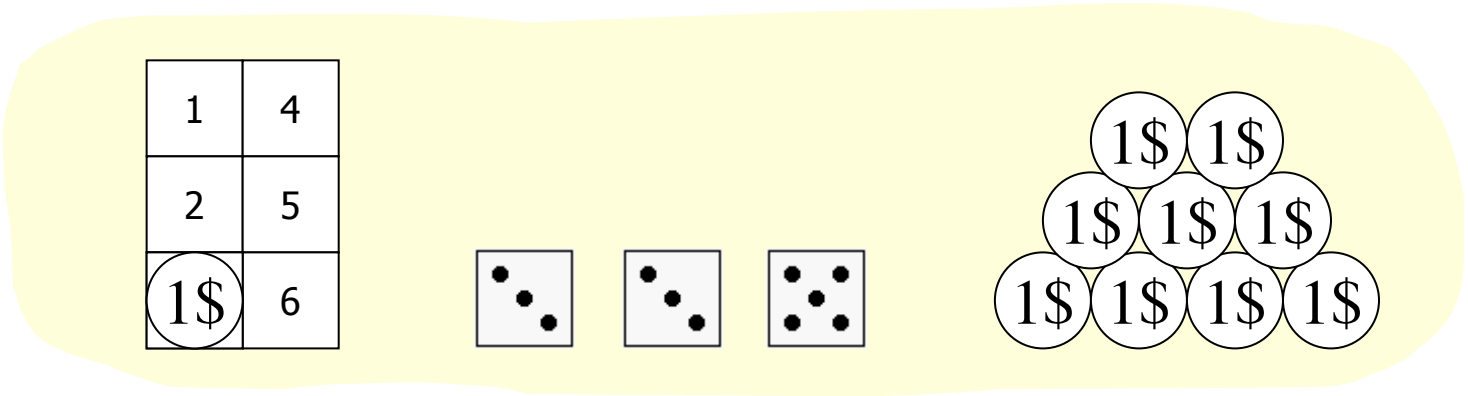
Eine ad-hoc-Lösung mit Hilfe von Delphi

```
type
  TGUI = class(TForm)
    Pueberschrift: TPanel;
    Pspielfeld: TPanel;
    Pwuerfel: TPanel;
    Pkonto: TPanel;
    ...
    BEinsatzZahlen: TButton;
    ...
    procedure BEinsatzZahlenClick
                  (Sender: TObject);
  private
    { Private-Deklarationen }
    spielzahl: integer;
    wuerfel1: integer;
    wuerfel2: integer;
    wuerfel3: integer;
    konto: integer;
  public
    { Public-Deklarationen
  end;
```

Nachteil:
Die Klasse TGUI ist für alles zuständig, neben der Verwaltung der GUI-Objekte auch für die Verwaltung der Spieldaten.
Bei größeren Anwendungen führt ein solcher Programmierstil zu schwer überschaubaren und schlecht wartbaren Programmen.

```
procedure TGUI.BEinsatzZahlenClick
                  (Sender: TObject);
begin
  // Verarbeitung: Einsatz abbuchen
  konto := konto-1;
  // Ausgabe: Aktualisierung der Anzeige
  PKontostand.Caption := IntToStr(konto);
end;
```

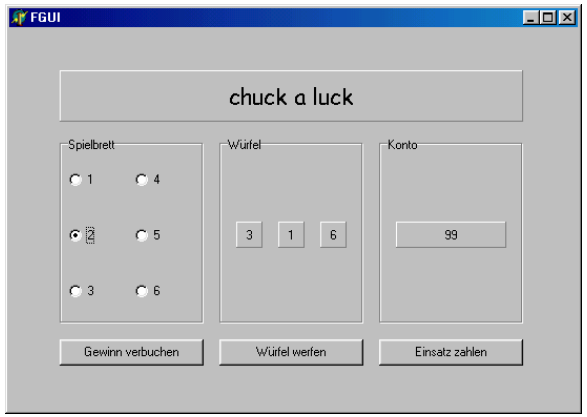
Lösungsansatz mit Modellierung



Modell

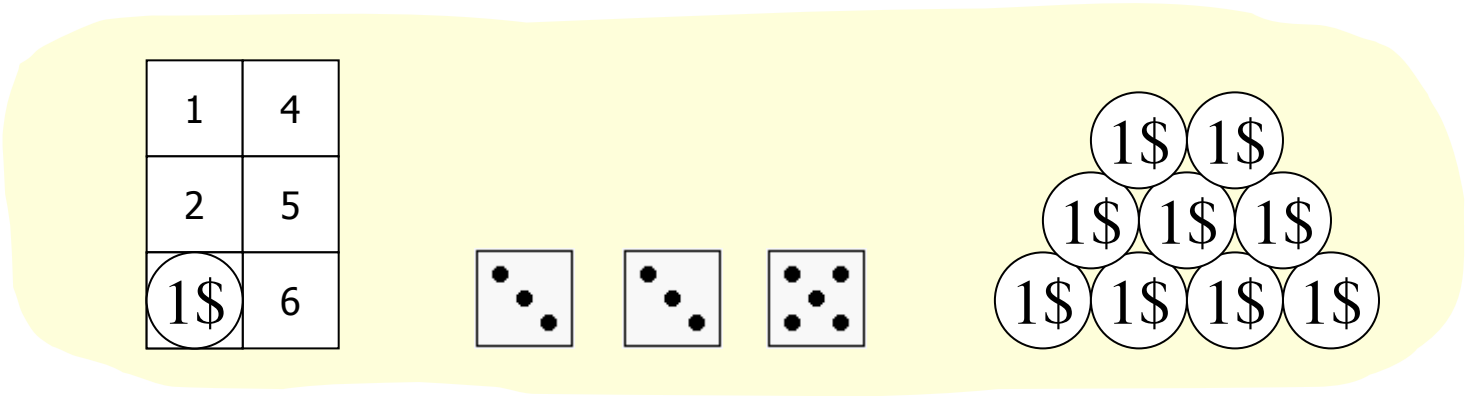
- Abbild der Miniwelt
- Vorlage für das System

System



Ansatz:
Mit Hilfe eines Modells soll die Miniwelt zunächst programmiersprachen-unabhängig beschrieben werden.
Das Modell soll dann helfen, in einem zweiten Schritt das Programm möglichst gut zu strukturieren.

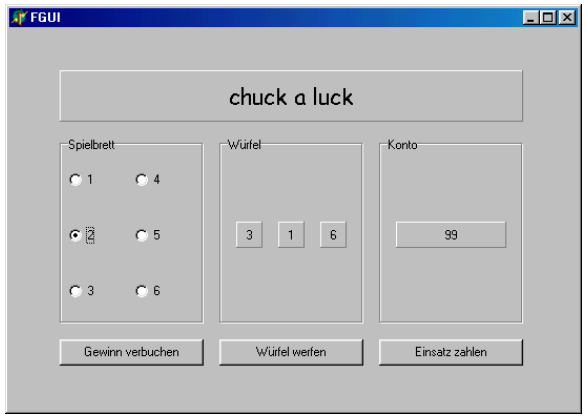
Objektorientierung



Modell

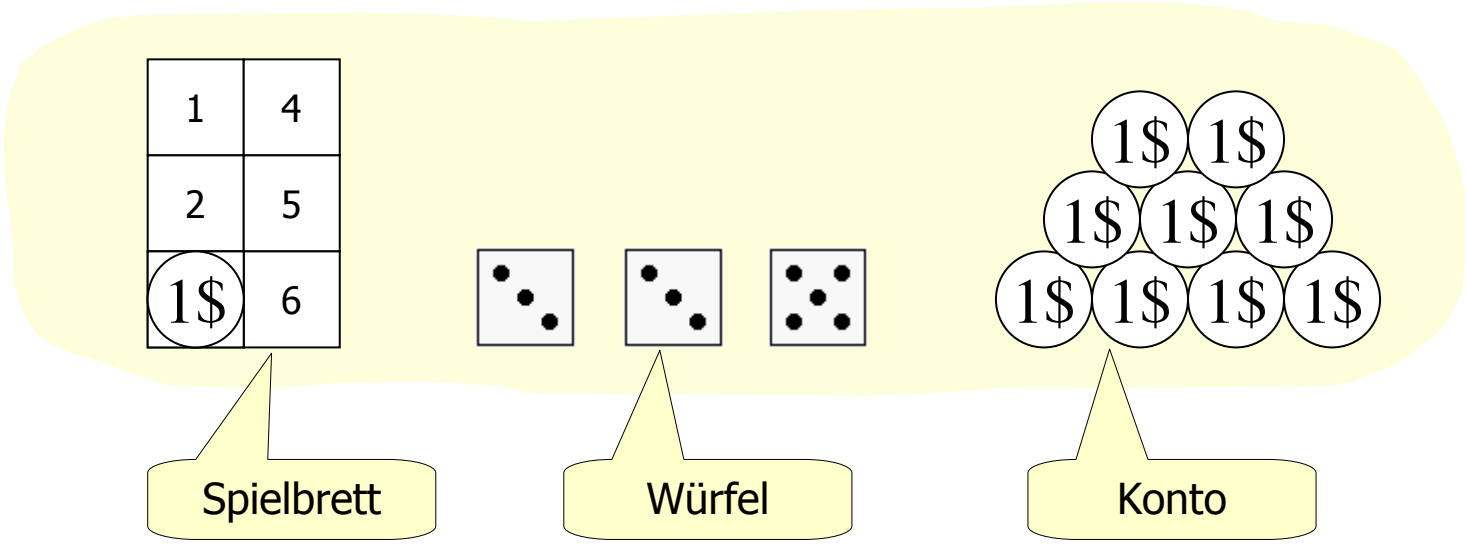
- Abbild der Miniwelt
- Vorlage für das System

System



Idee:
Die Software wird aus unabhängigen „Bausteinen“ zusammengesetzt. Die „Software-Bausteine“ sollen dabei den „Bausteinen der Miniwelt“ entsprechen. Ein Objekt im Sinne der objektorientierten Programmierung ist ein solcher „Software-Baustein“.

Struktur der Miniwelt



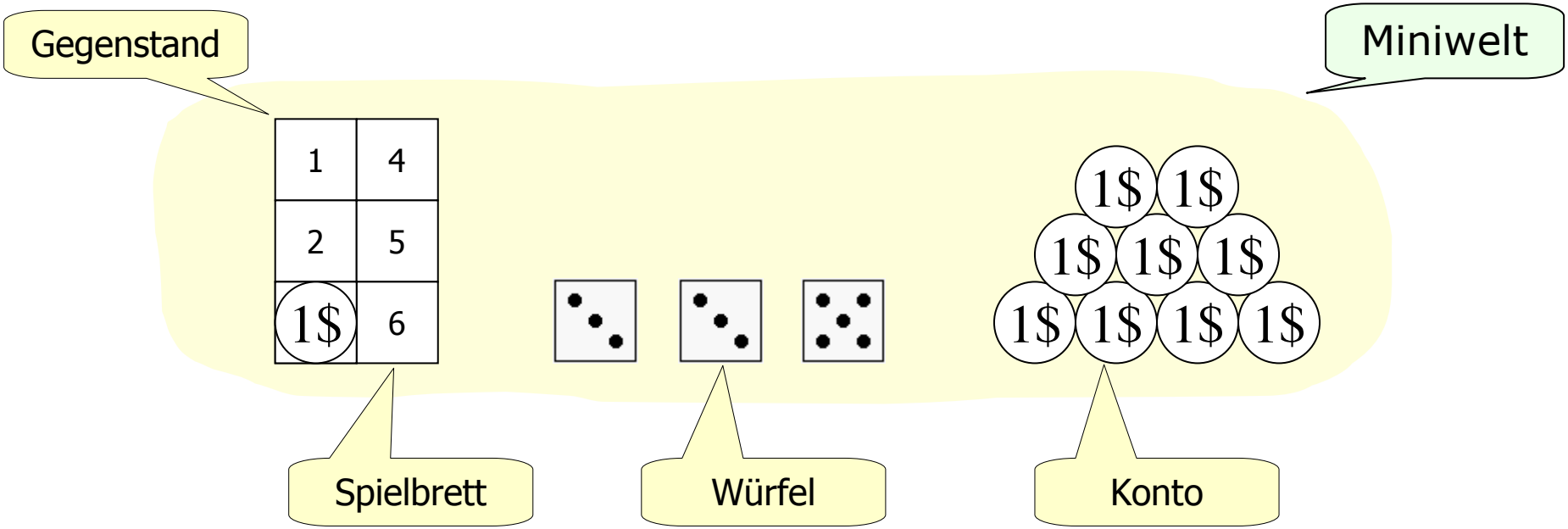
Zuständigkeit:
getippte Zahl merken

Zuständigkeit:
würfeln

Zuständigkeit:
Geldbetrag verwalten

Sichtweise:
Die Miniwelt ist aus Gegenständen aufgebaut. Gegenstände können Personen, Dinge, Sachverhalte, Ereignisse, ... sein. Jeder Gegenstand stellt eine autonome Einheit mit klar begrenzten Zuständigkeiten dar.

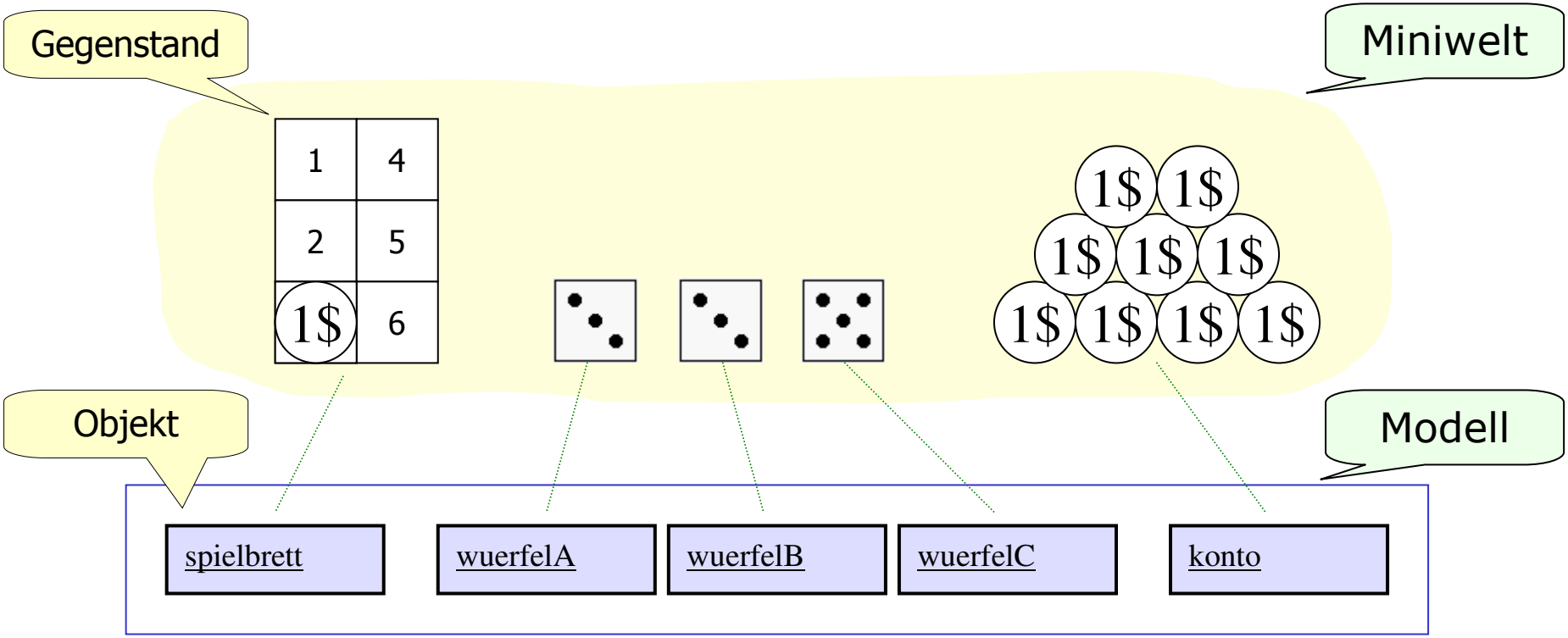
Struktur der Miniwelt



Spielbrett	Würfel	Konto
Eigenschaft: Zahl	Eigenschaft: Augen	Eigenschaft: Stand
Operation: Tipp setzen	Operation: werfen	Operationen: Betrag einzahlen, Betrag abheben

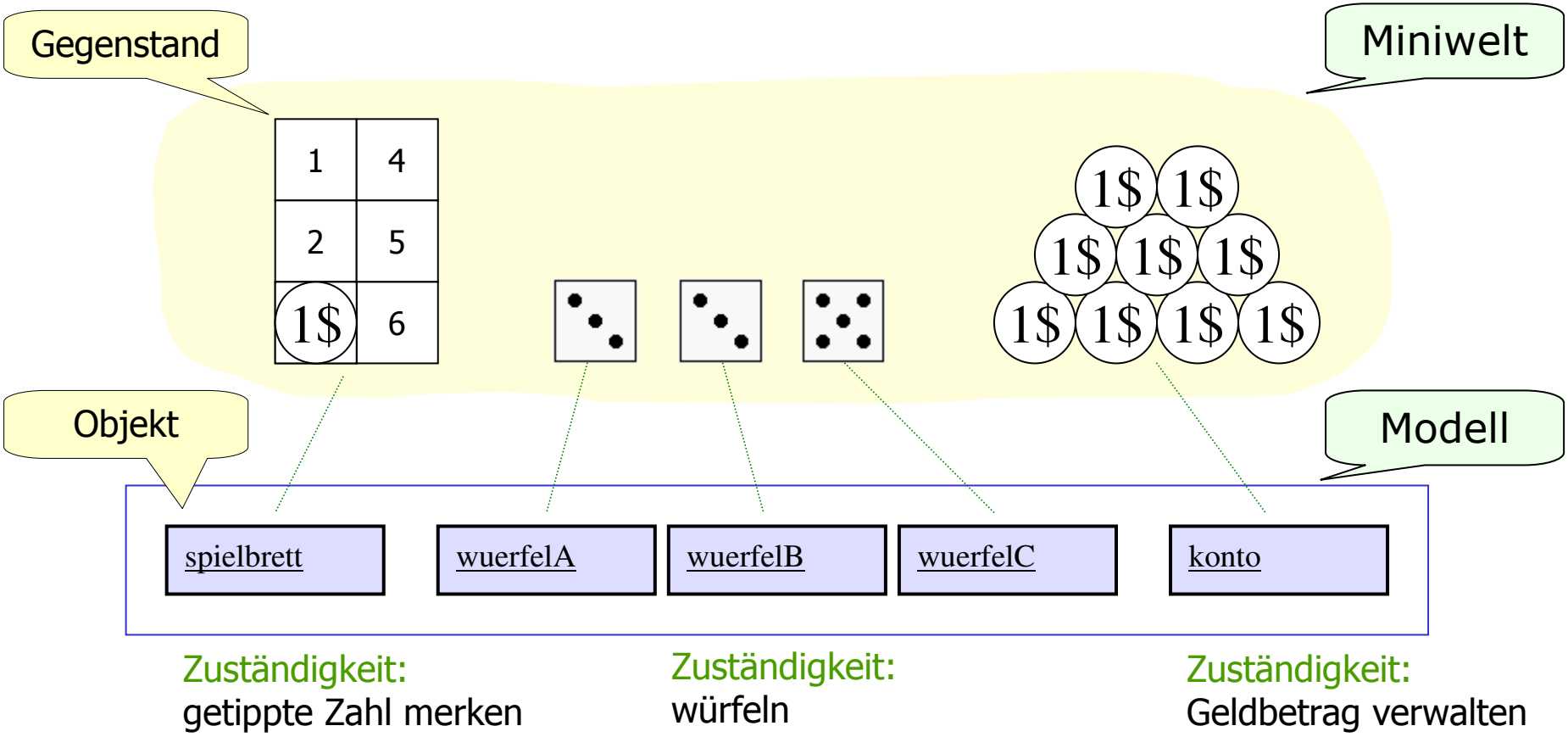
Sichtweise:
Gegenstände der Miniwelt haben charakteristische Eigenschaften. Mit den Gegenständen kann man bestimmte Operationen ausführen.

Grundidee der Objektorientierung



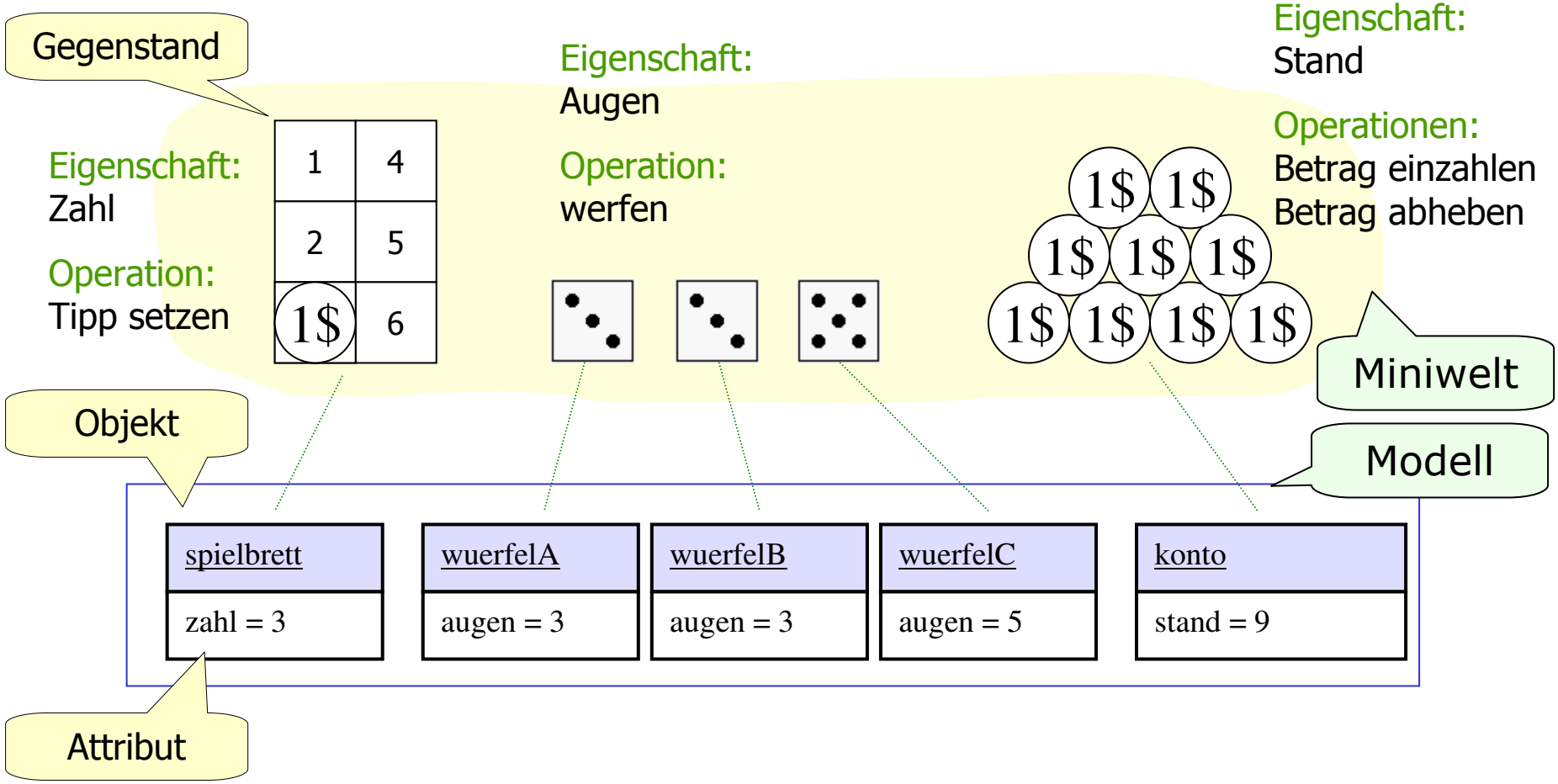
Die Gegenstände der Miniwelt werden mit Hilfe von **Objekten** im Sinne der Informatik beschrieben.

Grundidee der Objektorientierung



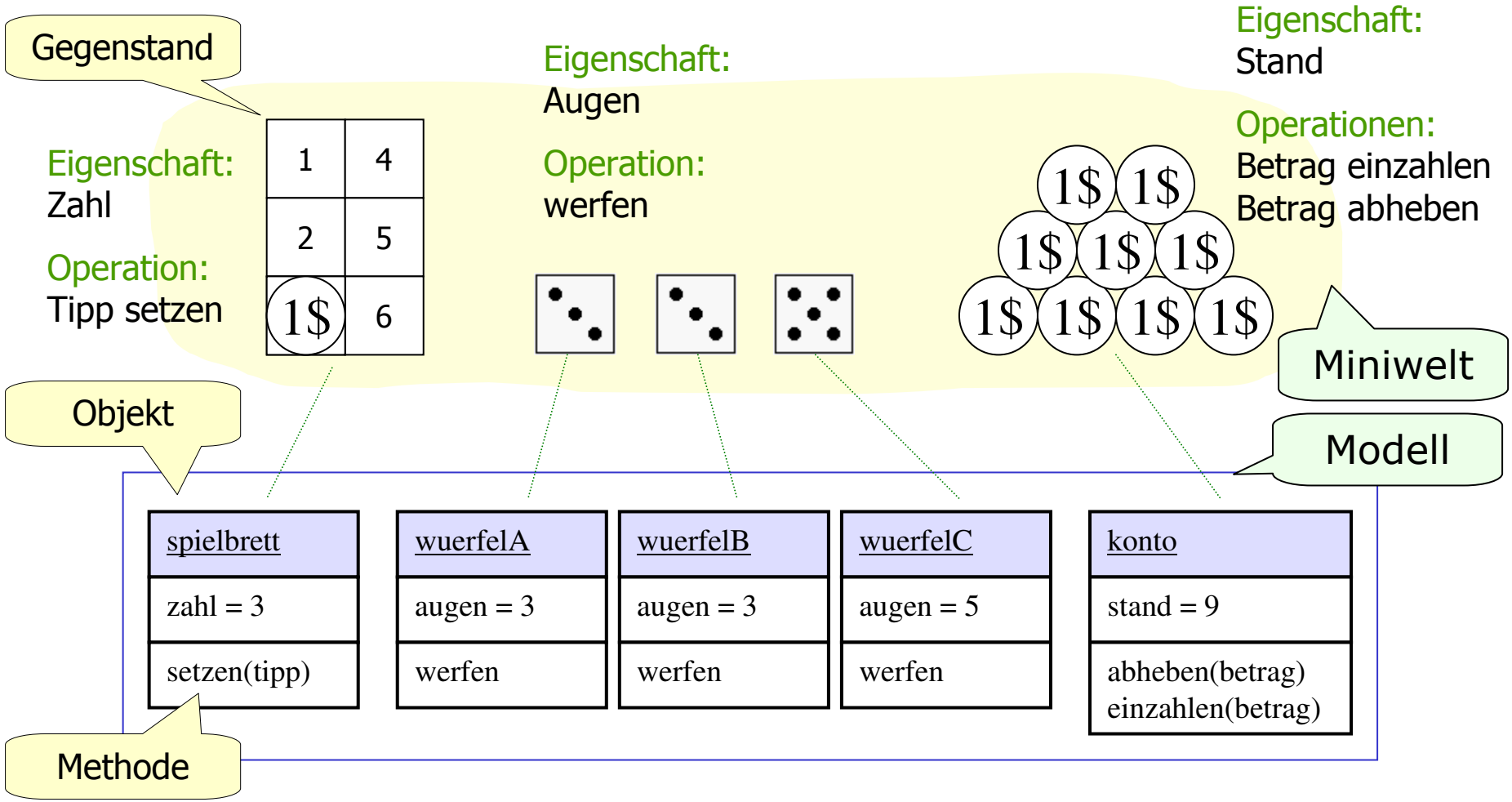
Ein Objekt stellt eine **autonome Einheit** mit klar begrenzten **Zuständigkeiten** dar.

Attribute



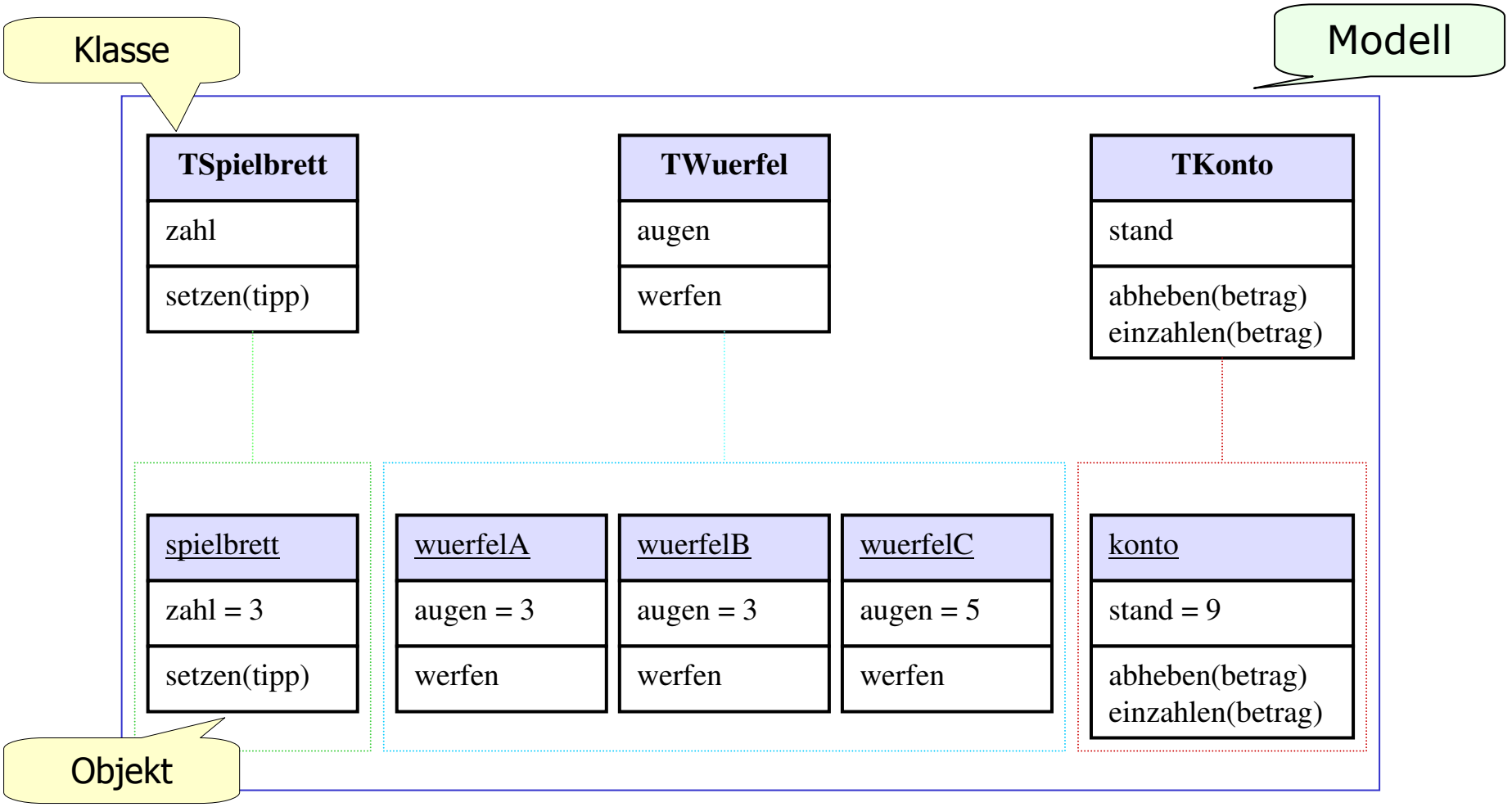
Die charakteristischen Eigenschaften eines Objekts werden mit **Attributen** erfasst. Die Gesamtheit der Attributwerte legt den **Objektzustand** fest.

Methoden



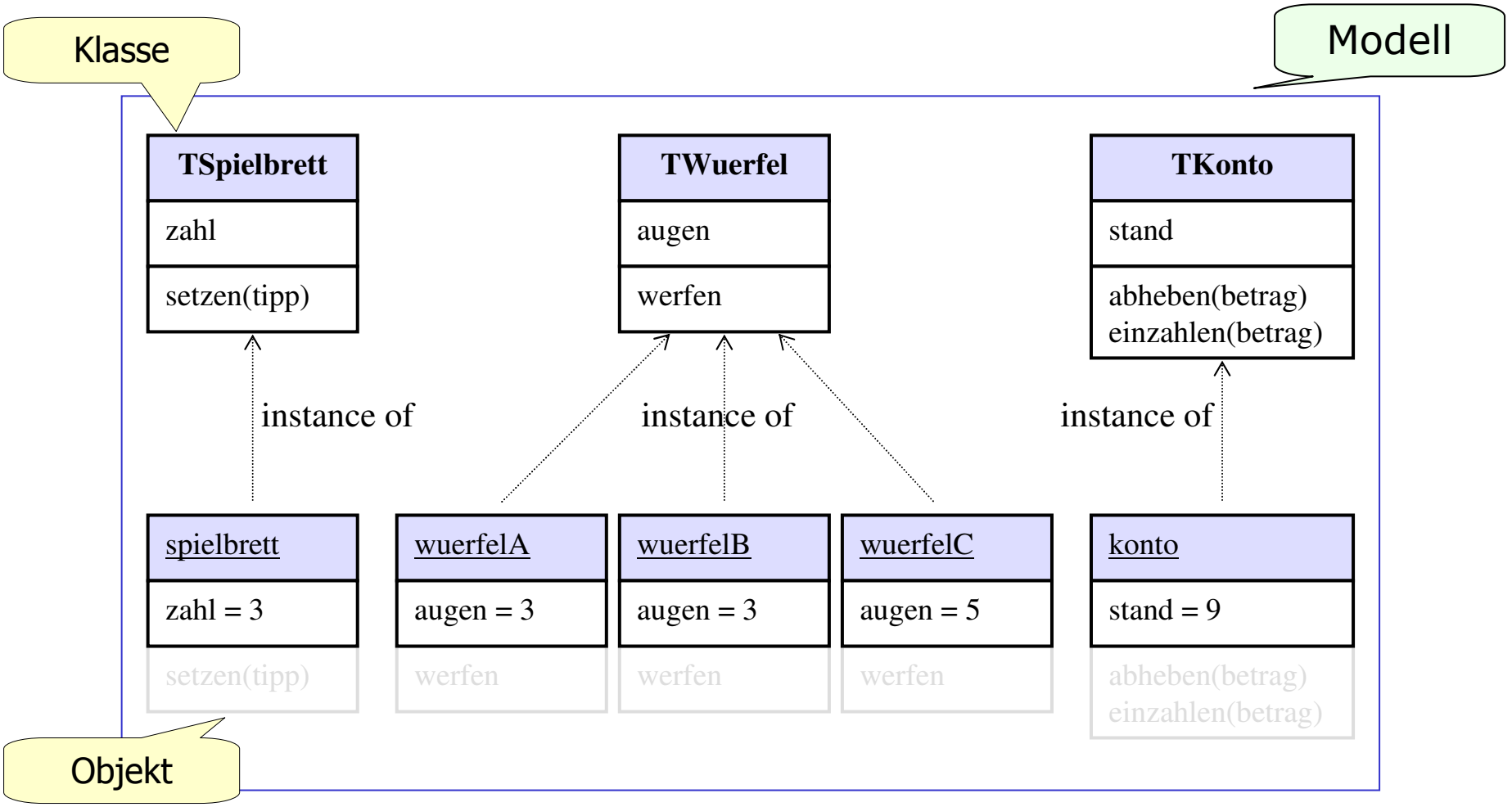
Das Verhalten eines Objekts wird mit Operationen / **Methoden** erfasst.
Diese bestimmen die dynamischen Eigenschaften eines Objekts.

Klassen



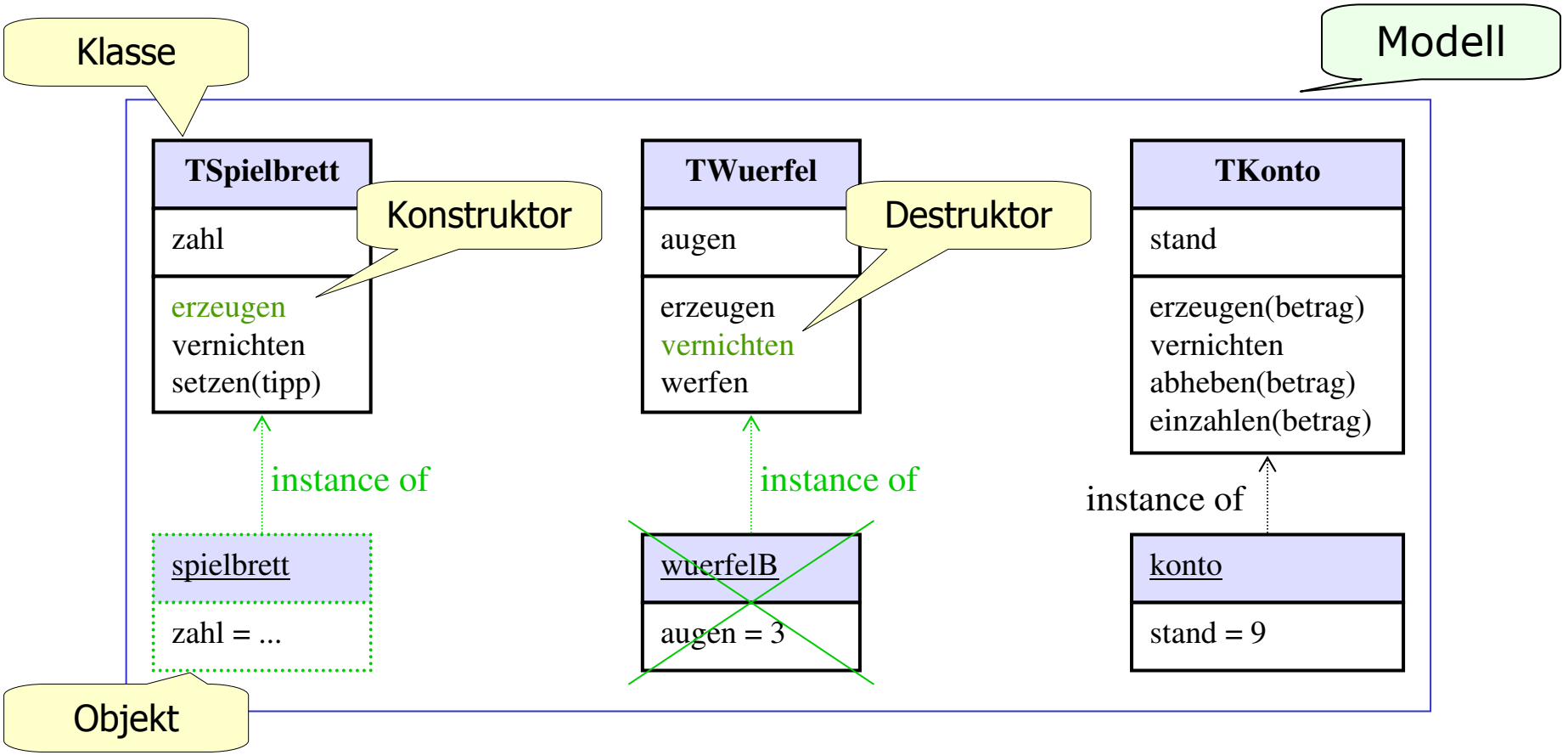
Gleich strukturierte Objekte werden einer **Klasse** zugeordnet.

Klassen



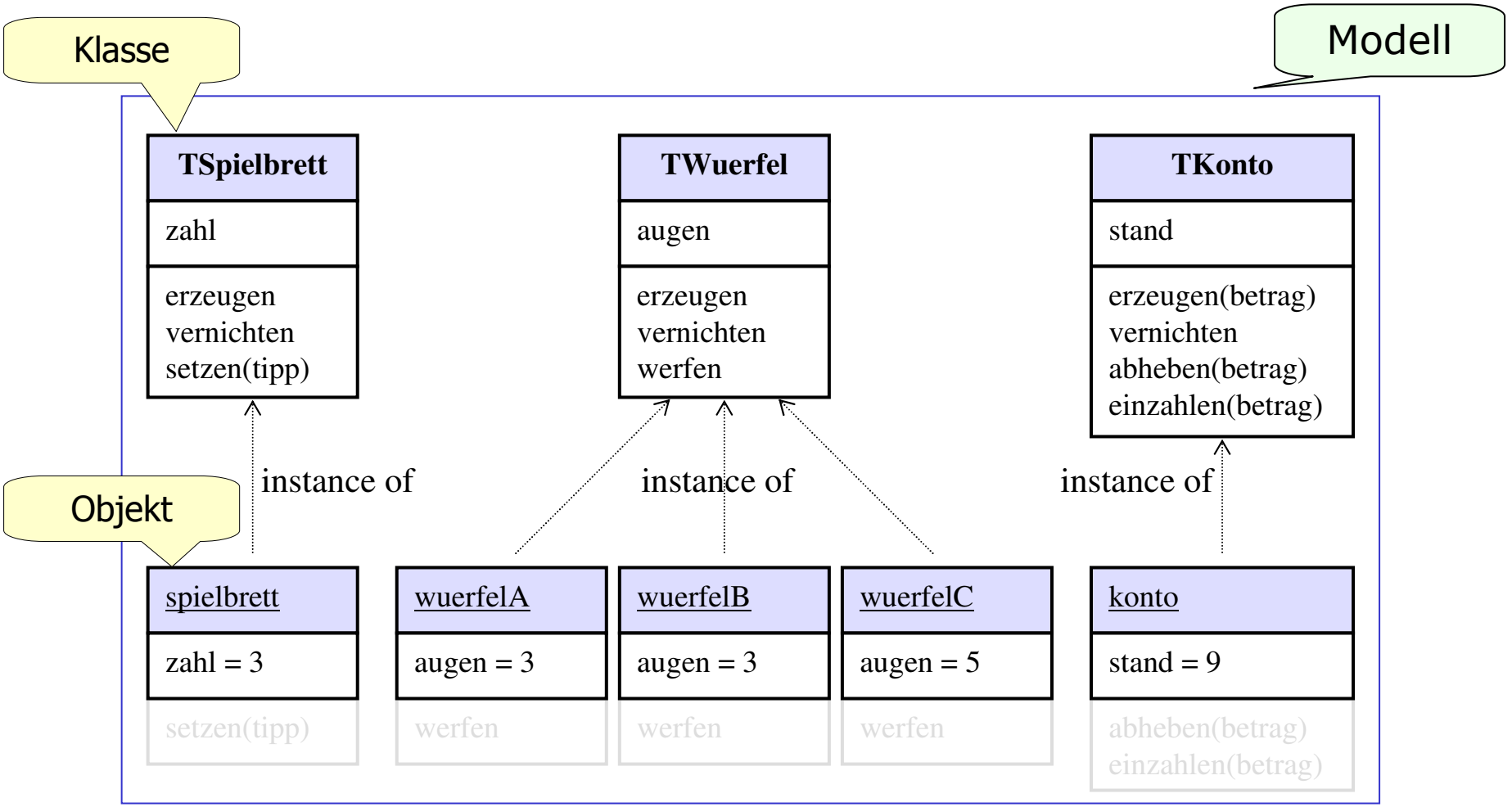
Klassen sind **Baupläne für Objekte**, sie legen den **Typ** dieser Objekte fest. Objekte werden als Exemplare (Instanzen) von Klassen bezeichnet.

Konstruktoren / Destruktoren



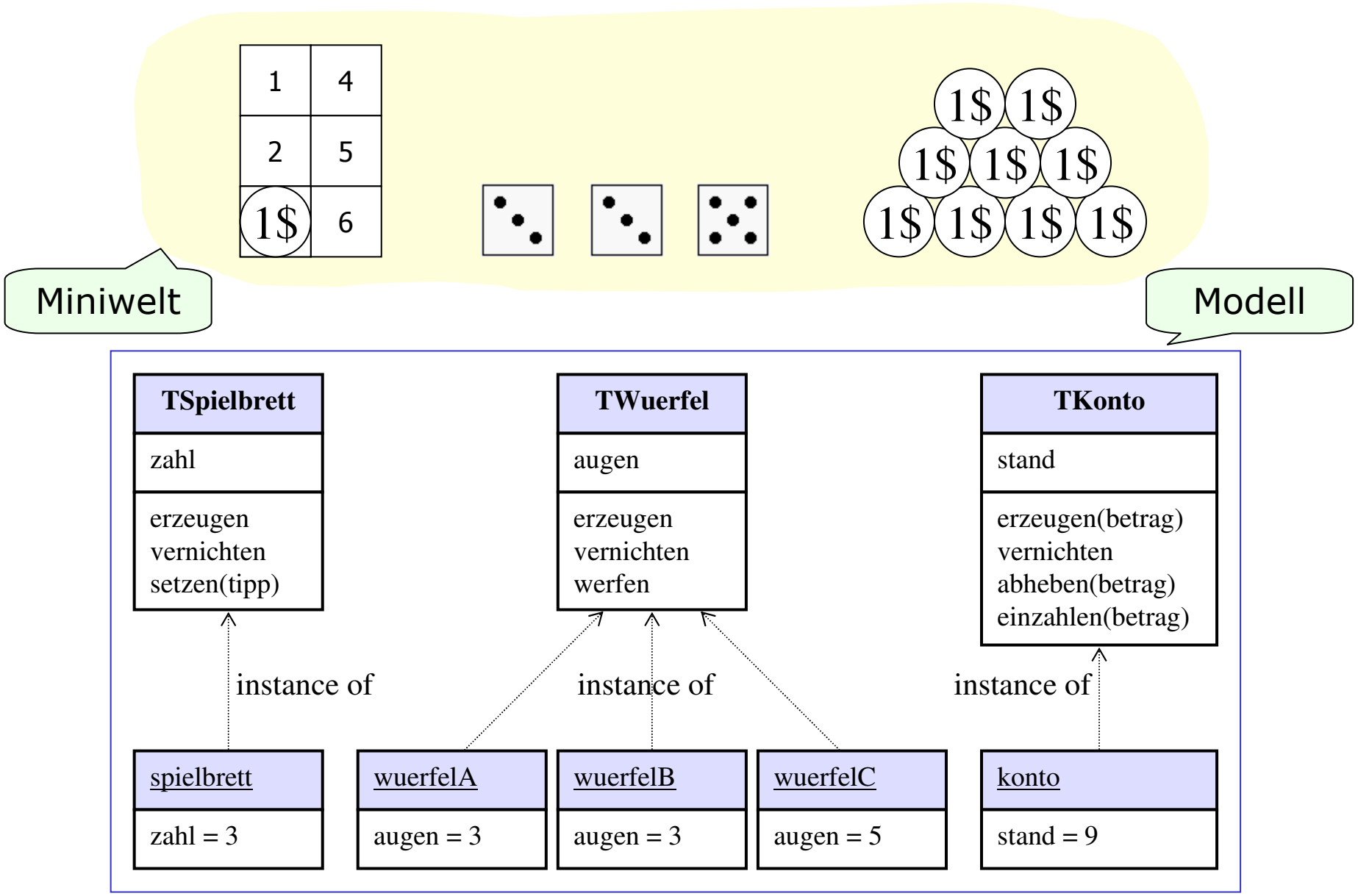
Konstruktoren / Destruktoren sind spezielle Operationen zur Erzeugung bzw. Vernichtung von Objekten.

Klassenmethoden

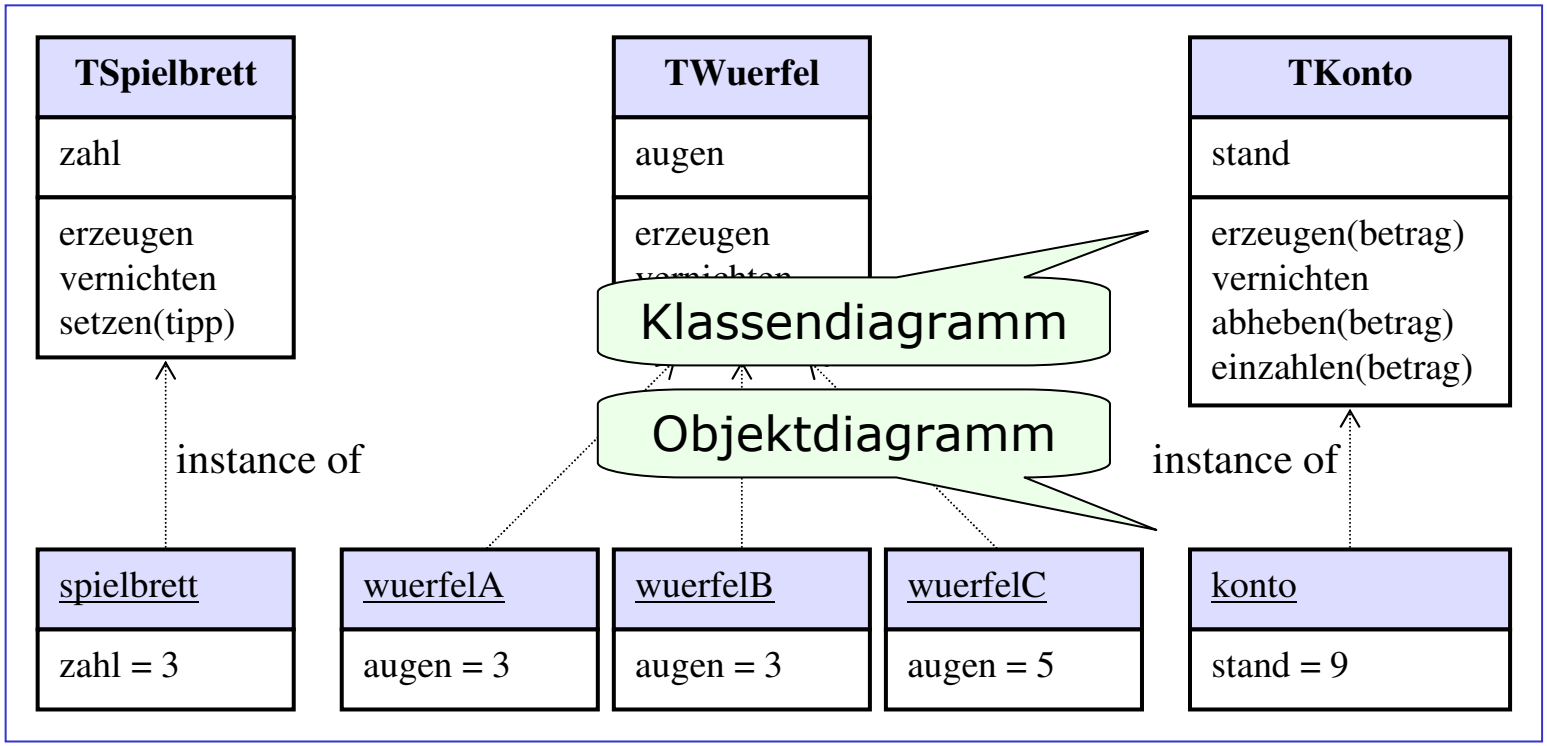


Beachte: Konstruktoren sind **Klassenmethoden**. Objekte verfügen nicht über diese Methoden.

Erstes objektorientiertes Modell



Konventionen



Klassen und Objekte werden hier mit Diagrammen der standardisierten Sprache UML (uniform modeling language) dargestellt. Beachte: Klassenbezeichner beginnen mit Großbuchstaben (hier zusätzlich mit T). Objektbezeichner beginnen jeweils mit Kleinbuchstaben.

unified modeling language

Teil 3

Gekapselte Objekte

Das Geheimnisprinzip

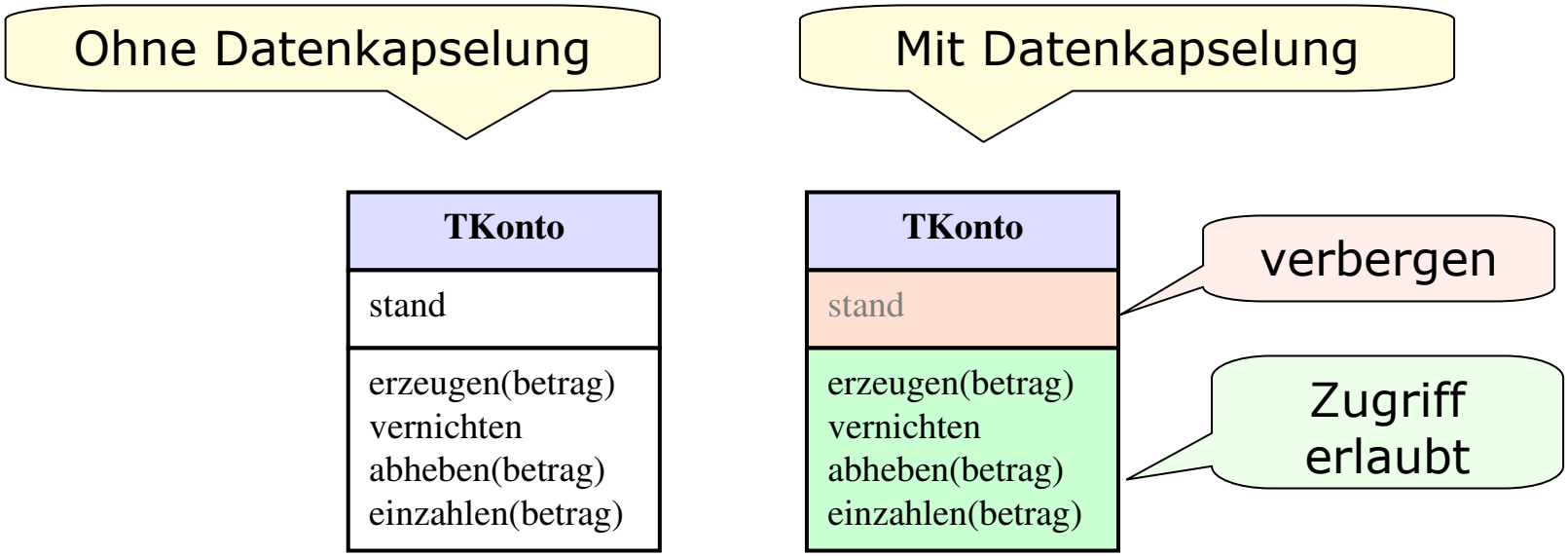
Das **Geheimnisprinzip** besagt, dass Details einer Klasse (insbesondere ihre Implementierung) verborgen werden sollten.

Ein Programmierer, der eine Klasse benutzen möchte, sollte keine Kenntnis über den internen Aufbau dieser Klasse benötigen. Ihm sollte auch nicht erlaubt sein, Interna dieser Klasse zu verwenden.

Ein Vorgehen nach dieser Bedingung ist erforderlich, wenn unabhängige Software-Bausteine entwickelt werden sollen, die ggf. auch wieder ausgetauscht werden müssen.

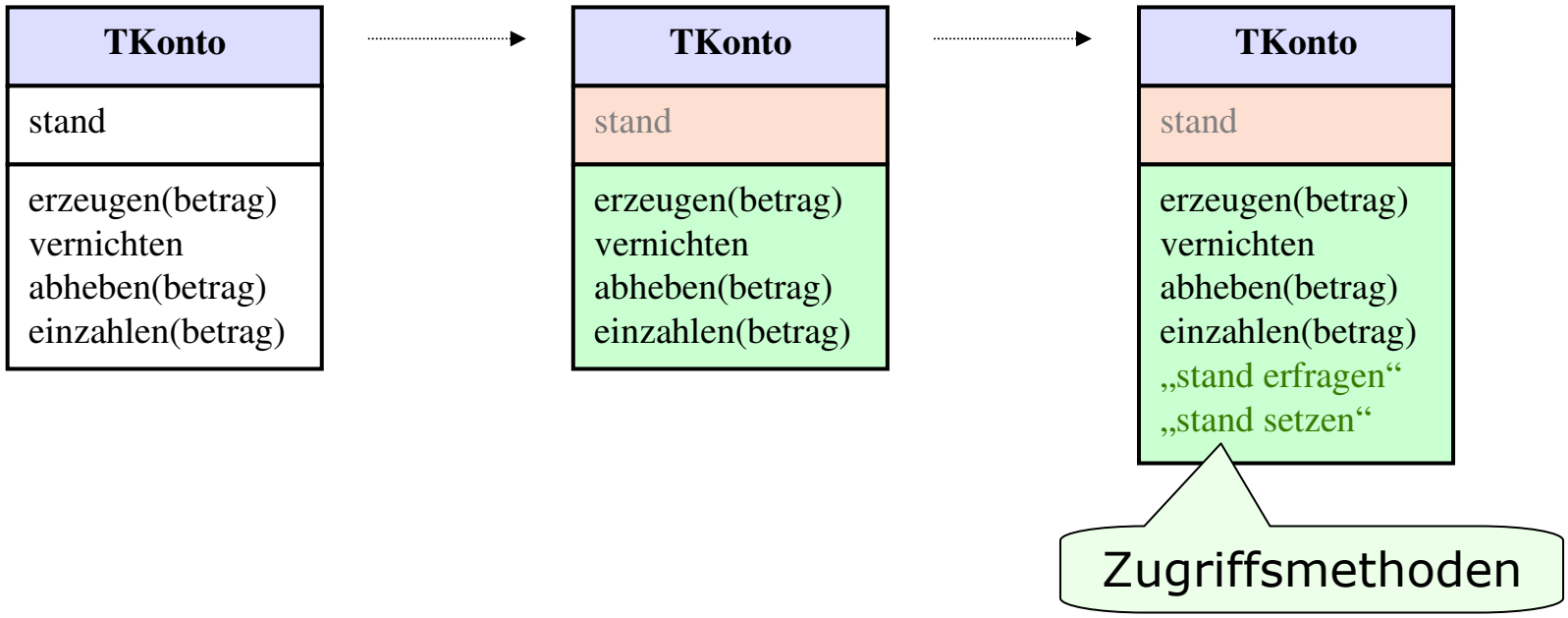
Datenkapselung

Datenkapselung besagt, dass ein Objekt keinen direkten Zugriff auf die Daten ermöglicht, die mit Hilfe von Attributen verwaltet werden.



Zugriffsmethoden

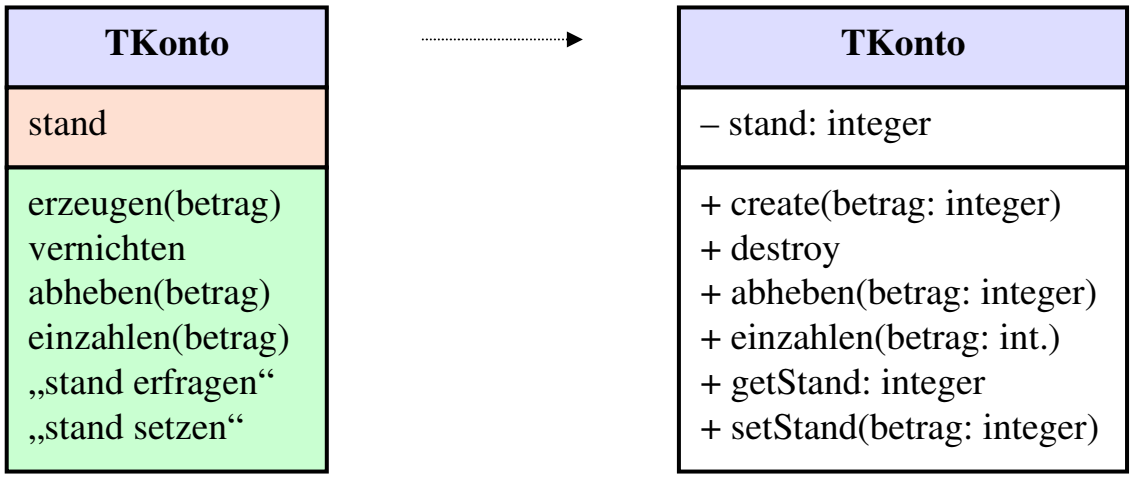
Um weiterhin auf Attributwerte (lesend bzw. schreibend) zugreifen zu können, werden spezielle Zugriffsmethoden benötigt.



Schnittstellenspezifikation

Eine **Schnittstellespezifikation** beschreibt sämtliche Informationen, die man zur Benutzung einer Klasse benötigt.

- Zugriffsrechte auf Attribute und Methoden
- Datentypen der Attribute (und eventuelle Initialisierungswerte)
- Signaturen der Methoden (Parameter u. bei Funktionen der Ergebnistyp)



Teil 4

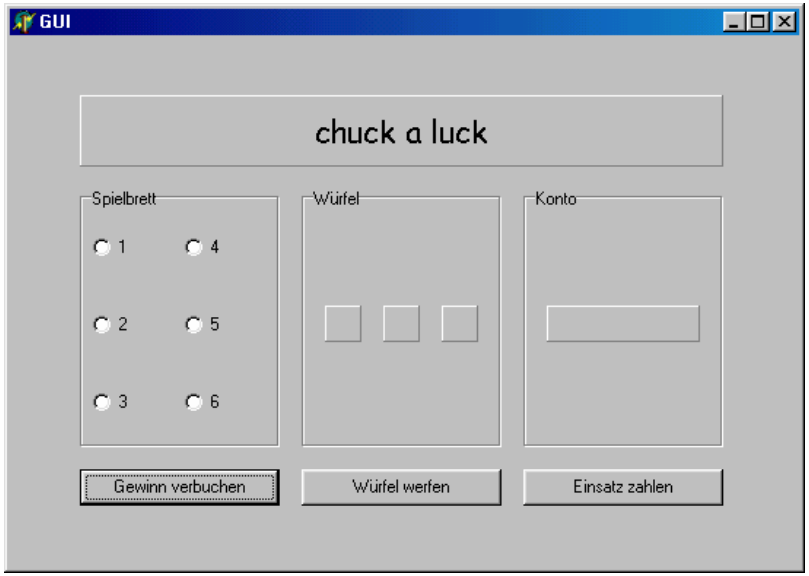
Implementierungen von Objekten und Klassen

Zielsetzung

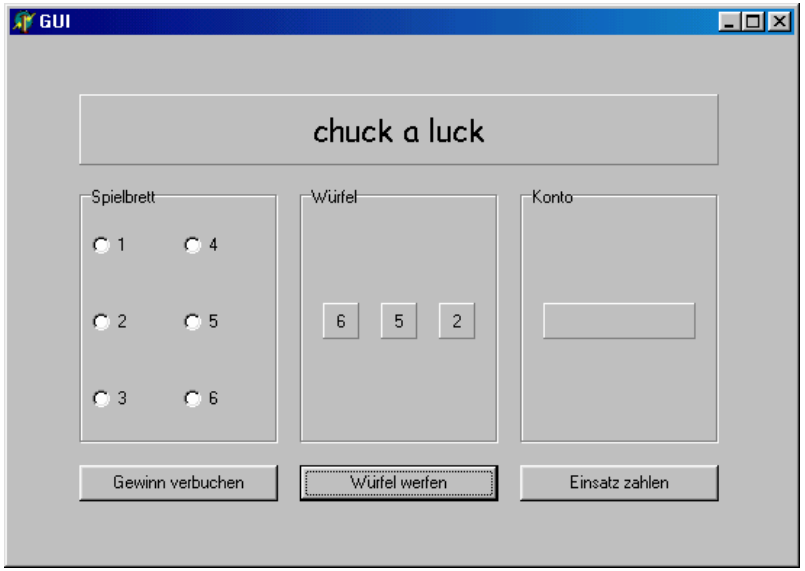
Vorgehensweise

Am Beispiel „Würfel" soll hier gezeigt werden, wie (in Delphi) ein Objekt zu einer „neuen" Klasse erzeugt und benutzt wird.

Wir gehen von einer fertigen Benutzungsoberfläche aus, die aber noch keine Spielaktionen ausführt.



Ziel ist es, die Würfel-Objekte zu implementieren und in das bestehende System zu integrieren.



Vorgehensweise

Schritt 1: Implementierung der Klasse TWuerfel
Hiermit wird der Bauplan für TWuerfel-Objekte festgelegt.

Schritt 2: Erzeugung der TWuerfel-Objekte
Jedes Objekt muss erzeugt werden, bevor es benutzt werden kann.

Schritt 3: Aktivierung der TWuerfel-Objekte
Mit Hilfe der erzeugten Objekte können jetzt die vorgesehenen Aufgaben erledigt werden.

Schritt 4: Vernichtung der TWuerfel-Objekte
Wenn Objekte nicht mehr benötigt werden, sollte der für sie reservierte Speicherplatz wieder freigegeben werden.

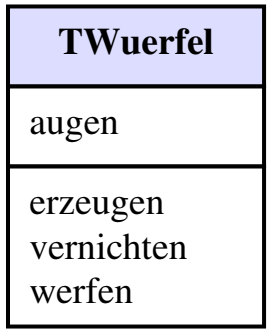
Schritt 1: Implementierung einer Klasse

Klassen werden als Module (Programmeinheiten) implementiert, die in Delphi in zwei Teilen beschrieben werden:

Schnittstellenvereinbarung: Deklaration der Attribute und Methoden

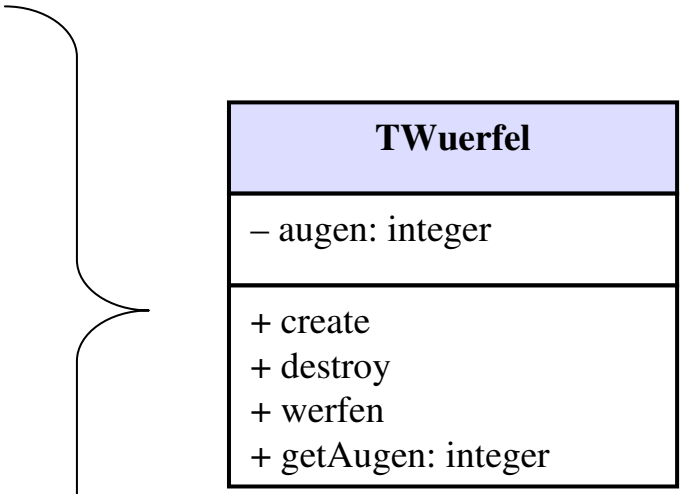
Implementierungsteil: Implementierung der Methoden

```
unit uWuerfel;

interface

  // Deklaration der
  // Attribute und Methoden

implementation

  // Implementierung der
  // Methoden

end.
```



Schritt 1: Implementierung einer Klasse

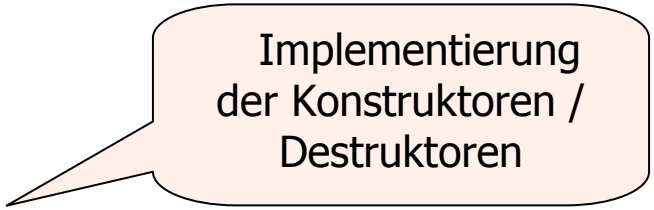
```
unit uWuerfel;

interface

type
  TWuerfel = class
  private
    augen: integer;
  public
    constructor create;
    destructor destroy; override;
    procedure werfen;
    function getAugen: integer;
  end;

...
```



Deklaration der Attribute und Operationen

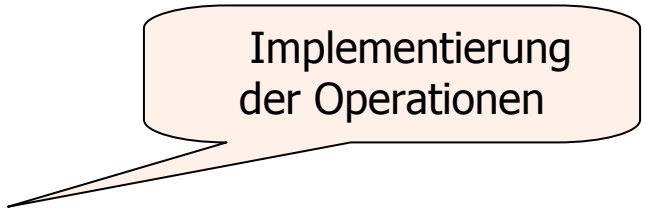
Schritt 1: Implementierung einer Klasse

```
unit uWuerfel;

interface

...

implementation

constructor TWuerfel.create;
begin
  augen := random(6)+1;
end;

destructor TWuerfel.destroy;
begin
end;

...
```



Schritt 1: Implementierung einer Klasse

```
unit uWuerfel;

interface

...

implementation

...

procedure TWuerfel.werfen;
begin
  augen := random(6)+1;
end;

function TWuerfel.getAugen: integer;
begin
  result := augen;
end;

end.
```



Schritt 1: Einbindung einer Modellklasse

```
unit uGUI;

interface

uses

  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls,
  uWuerfel;

type

  TGUI = class(TForm)
    ...
```

Einbindung der Klassen-Unit

Die Unit „uWuerfel“ enthalte die Implementierung der Klasse „TWuerfel“.

Schritt 2: Erzeugung eines Objekts

```
...
type
  TGUI = class(TForm)
    PTitel: TPanel;
    GBWuerfel: TGroupBox;
    GBKonto: TGroupBox;
    PWuerfelA: TPanel;
    PWuerfelB: TPanel;
    PWuerfelC: TPanel;
    PKonto: TPanel;
    ...
    procedure FormCreate(Sender: TObject);
    ...
  private     { Private-Deklarationen }
    wuerfelA: TWuerfel;
  public      { Public-Deklarationen }
  end;
```

Deklaration einer Referenzvariablen für das neue Objekt

Schritt 2: Erzeugung eines Objekts

```
unit uGUI;

interface

...

implementation

{$R *.DFM}

procedure TGUI.FormCreate(Sender: TObject);
begin
  randomize;
  wuerfelA := TWuerfel.create;
  PWuerfelA.Caption  := IntToStr(wuerfelA.getAugen);
end;

...

end.
```

Erzeugung des Objekts durch Aufruf des Konstruktors

Schritt 3: Aktivierung eines Objekts

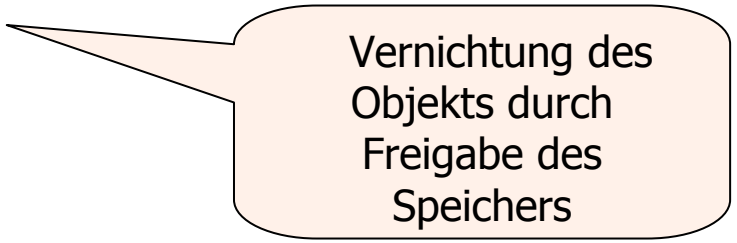
```
procedure TGUI.BWuerfelWerfenClick(Sender: TObject);
begin
  // Aktualisierung der Modell-Objekte
  wuerfelA.werfen;
  // ...
  // Aktualisierung der Anzeige
  PWuerfelA.Caption  := IntToStr(wuerfelA.getAugen);
  // ...
end;
```

Auftrag an das Objekt

Anfrage an das Objekt

Schritt 4: Vernichtung eines Objekts

```
unit uGUI;

interface

...

implementation

{$R *.DFM}

...

procedure TGUI.FormDestroy(Sender: TObject);
begin
  wuerfelA.free;
end;

...

end.
```



Aufgabe

Schauen Sie sich zunächst das bereits implementierte Teilsystem im Verzeichnis „ChuckALuck0-NurWuerfeln“ an.

Erweitern Sie dann das bereits implementierte Teilsystem zu einem funktionsfähigen Gesamtsystem.

Gehen Sie wie folgt vor, wenn Sie eine weitere Klasse implementieren und in das bereits begonnene Projekt integrieren wollen: Datei – Neu – Unit.

Delphi gibt Ihnen ein Programmgerüst vor. Ändern Sie zunächst den Namen der Unit (z. B. uKonto). Ergänzen Sie anschließend die Implementierung der betreffenden Klasse.

Vergessen Sie nicht, auch die Ergänzungen in der Unit „uGUI“ vorzunehmen (Unit einbinden; Objekte erzeugen; ...).