Projekte zur objektorientierten Programmierung in Java (Best. Nr. 4474)

Dieses Modul bietet Anregungen für Informatikprojekte der 10. Klasse. Angeboten werden drei Projekte zur Wahl: die Entwicklung eines Kniffelspiels, die Simulation eines Lagersystems zur Ermittlung eines möglichst effizienten Arbeitsablaufs und eine Simulation der Populationsentwicklung in einem klassischen Räuber-Beute-Szenario. Alle Projekte durchlaufen einen vereinfachten Entwicklungszyklus aus Anforderungsanalyse, Modellierung und Design, Implementierung und Test. Die einzelnen Entwicklungsaufgaben werden anfangs je kurz eingeführt, der Schwerpunkt liegt jedoch auf der selbständigen Arbeit der Schüler. Als Entwicklungsumgebung wird BlueJ verwendet, eine sehr einfache IDE, die speziell für das Erlernen der Java-Programmierung entwickelt wurde. Im Anhang ist eine kurze Einführung in BlueJ beigefügt, falls das Programm noch nicht im Unterricht eingeführt wurde. Neben der Entwicklung des jeweiligen Projekts wird auch auf ein grundlegendes Projektmanagement Wert gelegt.
Bei der Umsetzung der jeweiligen Projekte wurden die verwendeten Techniken auf in der 10. Klasse vorausgesetzte Verfahren beschränkt. Bei der GUI-Entwicklung der Projekte wurde weitgehend auf den Einsatz von Swing verzichtet. Die Beispiele beruhen daher allgemein auf AWT.

Zu allen Projekten ist ein funktionsfähiger Prototyp auf der CD vorhanden.

Autor und Verlag wünschen Ihnen viel Erfolg beim Einsatz dieses Schulbuchmoduls.

Gesamtdatei	
020_Java.ges	Alle Dateien dieser Einheit in Folge
1. Vorwort - Hinführung	
001_Java1.did	Vorwort zu dieser Einheit
002_Java2.hin	Einführung in die Projektarbeit
2. Einführung Projektmanagement	
003_Java3.arb	Einführung Projektmanagement
3. Projekt 1 - Kniffelspiel	
004_Java4.arb	Anforderungen
005_Java5.loe	Beispiel Anforderungen
006_Java6.arb	Modellierung und Design
007_Java7.loe	Beispiel Modellierung und Design
008_Java8.arb	Implementierung und Test
4. Projekt 2 - Räuber-Beute-Simulation: Fuchs und Hase	
009_Java9.arb	Anforderungen
010_Java10.loe	Beispiel Anforderungen
011_Java11.arb	Modellierung und Design
012_Java12.loe	Beispiel Modellierung und Design
013_Java13.arb	Implementierung und Test
5. Projekt 3 - Warenlager-Simulation	
014_Java14.arb	Anforderungen
015_Java15.loe	Beispiel Anforderungen
016_Java16.arb	Modellierung und Design
017_Java17.loe	Beispiel Modellierung und Design
018_Java18.arb	Implementierung und Test
6. Optional - Einführung in BlueJ	
019_Java19.arb	Einführung in BlueJ

Für folgende Dateien ist aus technischen Gründen keine Verlinkung möglich:

Im Ordner 8.1 BlueJ-Projekte\FuchsUndHase:

Fuchs.class
Fuchs.ctxt
Fuchs.java
FuchsUndHase.class
FuchsUndHase.ctxt
FuchsUndHase.java
Hase.class
Hase.ctxt
Hase.java
Oberflaeche$1.class
Oberflaeche$2.class
Oberflaeche$3.class
Oberflaeche$4.class
Oberflaeche.class
Oberflaeche.ctxt
Oberflaeche.java
package.bluej
Position.class
Position.ctxt
Position.java
README.TXT
Taktgeber.class
Taktgeber.ctxt
Taktgeber.java
Tier.class
Tier.ctxt
Tier.java
Welt.class
Welt.ctxt
Welt.java

Im Ordner 8.1 BlueJ-Projekte\Kniffel:

Kniffel.class
Kniffel.ctxt
Kniffel.java
Oberflaeche$1.class
Oberflaeche$2.class
Oberflaeche$3.class
Oberflaeche.class
Oberflaeche.ctxt
Oberflaeche.java
package.bluej
README.TXT
Spieler.class
Spieler.ctxt
Spieler.java
Spielzettel.class
Spielzettel.ctxt
Spielze
Spielze
Spielze
Spielze

SpielzettelPanel.java
Wuerfel.class
Wuerfel.ctxt
Wuerfel.java
WuerfelSymbol$1.class
WuerfelSymbol.class
WuerfelSymbol.ctxt
WuerfelSymbol.java
Wurf.class
Wurf.ctxt
Wurf.java

Im Ordner 8.1 BlueJ-Projekte\Warenlager:

Auftrag.class
Auftrag.ctxt
Auftrag.java
Lager.class
Lager.ctxt
Lager.java
Oberflaeche$1.class
Oberflaeche$2.class
Oberflaeche$3.class
Oberflaeche$4.class
Oberflaeche$5.class
Oberflaeche.class
Oberflaeche.ctxt
Oberflaeche.java
package.bluej
Palette.class
Palette.ctxt
Palette.java
PalettenSymbol.class
PalettenSymbol.ctxt
PalettenSymbol.java
README.TXT
Stapler$1.class
Stapler$Zustand.class
Stapler.class
Stapler.ctxt
Stapler.java
StaplerSymbol.class
StaplerSymbol.ctxt
StaplerSymbol.java
Symbol.class
Symbol.ctxt
Symbol.java
Taktgeber.class
Taktgeber.ctxt
Taktgeber.java
Warenlager.class
Warenlager.ctxt
Warenlager.java

Im Ordner 8.2 JAR-Dateien:

FuchsUndHase.jar
Kniffel.jar
Warenlager.jar

Im Ordner 8. BlueJ-Projekte - Materialien\8.3 Programme:

bluej-311.msi
umlet_10_4.zip

Die dreistelligen Buchstabenkombinationen am Ende der Kurz-Dateinamen bedeuten:

*.hin	Hinführung zum Thema
*.arb	Arbeitsblatt
*.loe	Lösungsblatt
*.did	Hinweise für Lehrer
*.ges	Gesamtdatei

Es folgen einige wenige Beispieldateien, die im Original veränderbare Word-Dateien sind.

Projekt 1: Entwicklung eines Kniffelspiels

2. Modellierung und Design

1. Erste Überlegungen zu Klassen

1. Die Auftragsbeschreibung auf Substantive analysiert

In einem ersten Schritt betrachten wir die Auftragsbeschreibung näher und sehen uns an, mit welchen „Dingen“ das geplante System umgehen soll. Folgendes wird zu den Regeln des Spiels gesagt (Substantive sind hervorgehoben – ohne substantivierte Verben):

Kniffel wird mit fünf Würfeln gespielt. Es wird reihum gewürfelt. Jeder Spieler hat einen Spielzettel auf dem er seine Punkte eintragen muss. Für jedes Feld auf dem Spielzettel gilt dabei eine bestimmte Bedingung, nach der die gewürfelte Konstellation mit Punkten bewertet wird. Jeder Spieler darf pro Runde bis zu drei Mal würfeln. Dabei darf er passende Würfel zur Seite legen und nur mit den verbliebenen Würfeln weiterwürfeln. Spätestens nach dem dritten Würfeln, muss ein Feld für die Eintragung gewählt werden. Kann der aktuelle Wurf keine Bedingung eines Feldes erfüllen, muss ein Feld gestrichen bzw. die Punktzahl 0 vergeben werden. Ziel des Spiels ist es, durch geschicktes Kombinieren der gewürfelten Augen und entsprechendes Auswählen der passenden Felder eine möglichst hohe Punktzahl zu erzielen. Sind alle Felder des Spielzettels ausgefüllt, ist das Spiel beendet. Der Spieler mit der höchsten Punktzahl hat gewonnen.

Im nächsten Schritt betrachten wir die identifizierten Substantive näher und überlegen, ob sie potentielle Klassen in dem geplanten System sein könnten.

Kniffel: bezeichnet das gesamte System, muss daher nicht einzeln modelliert werden, kann als Name für die Main-Klasse verwendet werden, die das System gleichzeitig koordiniert.

Würfel: zentrale Klasse des Systems

Spieler: zentrale Klasse des Systems

Spielzettel: zentrale Klasse des Systems

Punkte/Augen: keine Klassen, Punktstände gehören als Attribut zum Spielzettel, Augen gehören als Attribut zu Würfeln

Feld: keine Klasse, die Felder sind Attribute des Spielzettels, die Punkte halten

Bedingung/Konstellation: keine Klassen, die Bedingungen der Felder müssen in Methoden des Spielzettels überprüft werden, um Mogeln zu verhindern

Runde: keine Klasse, eine Runde ist einfach mehrmaliges Würfeln

Eintragung, Ziel, Spiel: keine Klassen

Wurf: Klasse des Systems, ein Wurf umfasst fünf Würfel mit bestimmten Punkten

2. Identifizierte Klassen

Im obigen Schritt wurden folgende Klassen identifiziert:

`Kniffel`: Main-Klasse und Koordination des Systems

`Wuerfel`: repräsentiert einen Würfel

`Spieler`: repräsentiert einen Spieler

`Spielzettel`: repräsentiert den Spielzettel eines Spielers

`Wurf`: repräsentiert einen Wurf, also die fünf Würfel mit ihren gewürfelten Augen

3. Weitere für die Implementierung wichtige Klassen

Im nächsten Schritt denken wir darüber nach, welche Klassen für eine Implementierung des Systems noch wichtig sind. Dies betrifft vor allem die Oberflächenelemente.

`Oberflaeche`: repräsentiert die Oberfläche des Systems

`WuerfelSymbol`: repräsentiert die Darstellung eines Würfels auf der Oberfläche

`SpielzettelPanel`: repräsentiert die Darstellung des Spielzettels auf der Oberfläche

4. Erste Überlegungen zu Verbindungen der Klassen

Um eine erste Idee davon zu bekommen, wie die Klassen zusammenhängen, versuchen wir erste Zusammenhänge zu verbalisieren.

Jeder `Spieler` hat einen `Spielzettel`.

Jeder `Spielzettel` ist verbunden mit seinem `SpielzettelPanel`.

Jeder `Spieler` würfelt einen `Wurf`.

Jeder `Wurf` besteht aus fünf `Wuerfel`(n).

Jeder `Wuerfel` ist verbunden mit seinem `WuerfelSymbol`.

5. Erste Überlegungen zu Attributen und Methoden

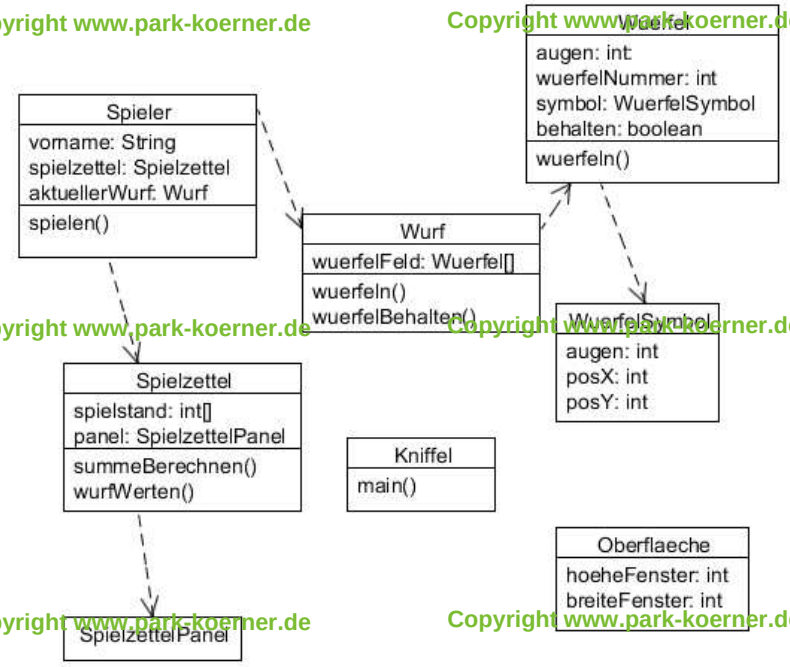
Schließlich versuchen wir unsere bisherigen Erkenntnisse in einer Tabelle festzuhalten. Auch erste Attribute und Methoden können wir bereits angeben.

Klasse	Attribute	Methoden
`Kniffel`		`main()`
`Wuerfel`	`augen`: Typ `int`. Jeder Würfel muss wissen, wie viele Augen gerade gewürfelt wurden `wuerfelNummer`: Typ `int`. Jeder Würfel bekommt eine Nummer zwischen 1 und 5, um ihn eindeutig zu identifizieren. `symbol`: Typ `WuerfelSymbol`. Jeder Würfel braucht eine Referenz auf sein Würfelsymbol. `behalten`: Typ `boolean`. Speichert, ob ein Würfel behalten werden soll.	`wuerfeln()`: Jeder Würfel entscheidet auf Basis einer Zufallsentscheidung wie viele Augen gewürfelt werden.

Spieler	vorname: Typ String. Der Name des Spielers. spielzettel: Typ Spielzettel. Der Spielzettel des Spielers. aktuellerWurf: Typ Wurf. Der aktuell gewürfelte Wurf.	spielen(): Steuert die verschiedenen Aktionen, die während einer Runde, die der Spieler spielt, nötig sind.
Spielzettel	spielstand: Typ int[]. Das Feld repräsentiert die verschiedenen Felder des Spielzettels und hält die Punkte, die jeweils eingetragen sind. panel: Typ SpielzettelPanel. Der Spielzettel braucht eine Referenz auf sein Spielzettelsymbol.	summeBerechnen(): Berechnet die Summen des Spielzettels. wurfWerten(): Steuert, wo ein Wurf auf dem Spielzettel eingetragen werden kann (nicht auf bereits besetzten Feldern) und berechnet entsprechend der Regeln, wie viele Punkte eingetragen werden. Braucht Hilfsmethoden, um die Bedingungen der einzelnen Felder zu überprüfen.
Wurf	wuerfelFeld: Typ Wuerfel[]. Jeder Wurf braucht eine Referenz auf die fünf Würfel.	wuerfeln(): Bei jedem Wurf müssen alle fünf oder die Würfel, die nicht behalten wurden, neu ausgewürfelt werden. wuerfelBehalten(): Steuert, welche Würfel behalten werden und deshalb nicht erneut ausgewürfelt werden.
Oberflaeche	hoeheFenster: Typ int. breiteFenster: Typ int.	
SpielzettelPanel		
WuerfelSymbol	augen: Typ int. Jedes Würfelsymbol muss wissen, wie viele Augen dargestellt werden müssen. posX, posY: Typ int. Jedes Würfelsymbol braucht eine X- und Y-Position auf der Oberfläche.	

6. Erster Versuch eines Klassendiagramms

Auf der bisherigen Basis können wir nun den ersten Versuch eines Klassendiagramms unternehmen.



2. Designentscheidungen im Rahmen der weiteren Modellierung und erste Implementierungsentscheidungen

Mit den bisherigen Informationen haben wir die Basis für unser Projekt gelegt. Für die Implementierung sind aber noch etliche weitere Entscheidungen nötig. Im Folgenden werden einige Entscheidungen dokumentiert.

* Für die Ausgabe von Text an den Spieler wird eine `TextArea` verwendet.

* Für die erwartete Eingabe des Spielers werden Buttons verwendet. Zwei Buttons sind nötig: ein allgemeiner Eingabe-Button, der alle Aktionen bestätigt, zum Beispiel, dass der Name eingegeben wurde, dass die gewünschten Würfel ausgewählt wurden, ... und ein Button „Wurf werten“, der verwendet wird, um einen Wurf vorzeitig zu werten, ohne dreimal zu würfeln.

* Die Oberfläche benötigt verschiedene Methoden, um Abfragen an die Spieler zu senden und zu überprüfen, ob die jeweilige Eingabe vorgenommen wurde.

  - `eingabeAbfragen(String text)`: die Methode gibt einen Text an den Spieler aus und bestätigt, dass die jeweilige erwartete Aktion ausgeführt wurde.
  - `textSetzen(String text)`: die Methode gibt lediglich einen Text an den Spieler aus.
  - `wurfWertenAbfragen(String text)`: fragt ab, ob der Wurf gewertet werden soll und überprüft den Status des Buttons „Wurf werten“.

* Die Oberfläche benötigt die beiden `SpielzettelPanel`, da diese von Anfang an erzeugt werden müssen, noch bevor die Spieler definiert wurden (wegen der Optik der Oberfläche).

* Verschiedene Klassen benötigen einen Zugriff auf die Oberfläche, um sich der Oberfläche hinzuzufügen (`WuerfelSymbol`) oder die Abfragen an den Spieler zu koordinieren. Außerdem soll

stets nur eine Instanz der Oberfläche möglich sein. Deshalb ist es sinnvoll, die Oberfläche als sogenanntes Singleton zu implementieren, das heißt, dass der Konstruktor der Klasse `Oberflaeche` privat ist, die Klasse eine statische private Instanzvariable vom Typ `Oberflaeche` erhält und eine statische Methode `oberflaecheLiefern()`, die die einzige Instanz der Oberfläche erzeugt und diese Variable zurückgibt.

* Die Entscheidung, ob ein Würfel behalten wird oder nicht, wird durch Anklicken des jeweiligen Würfelsymbols getroffen. Deshalb muss für das Würfelsymbol ein `MouseListener` implementiert werden, der diese Entscheidung ermöglicht. Die „behaltenen“ Würfel werden farblich hervorgehoben.

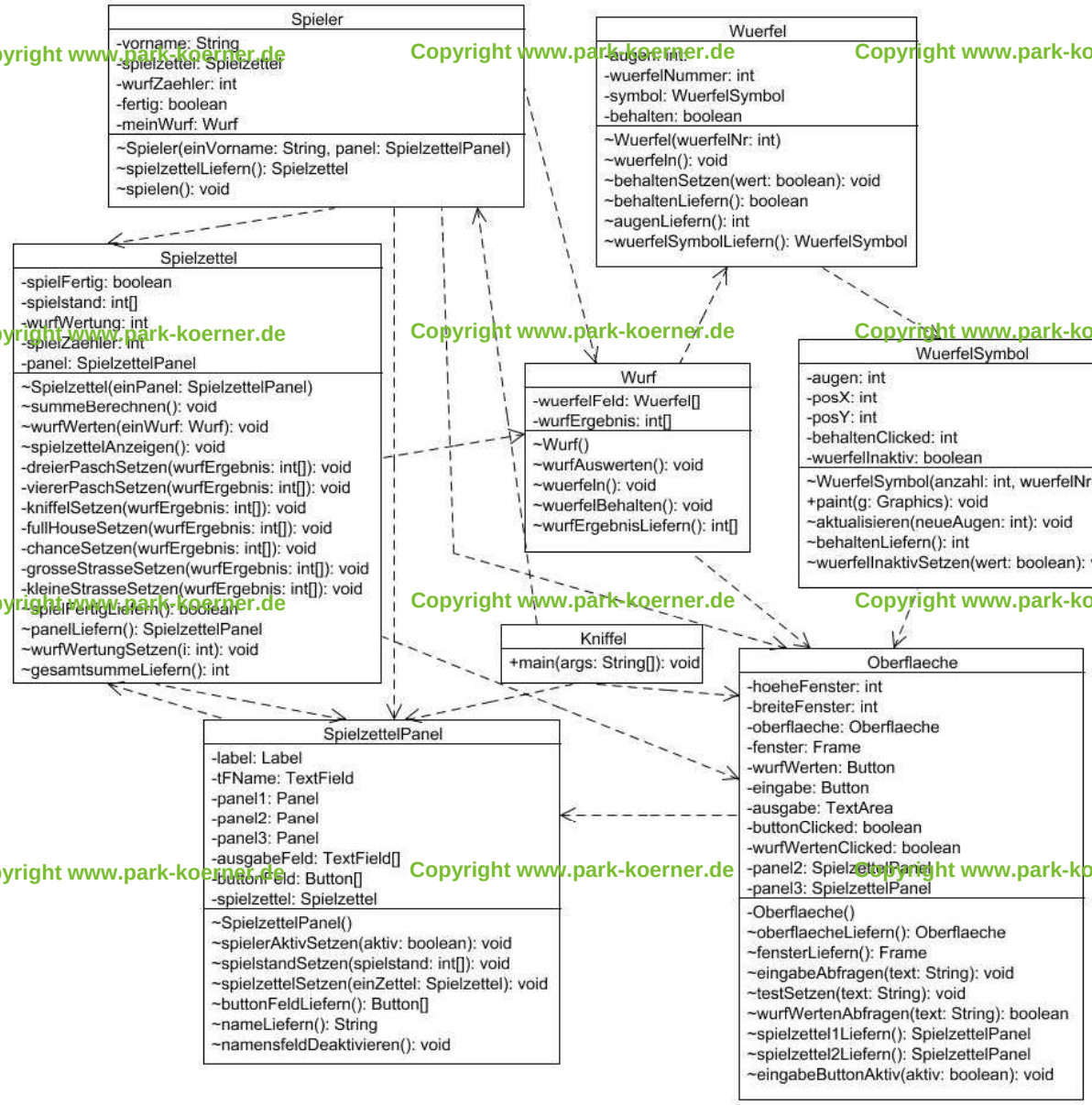
* Die Klasse `Wurf` bekommt ein weiteres Feld vom Typ `int[]` namens `wurfErgebnis` mit der Länge 6. Dieses wird genutzt, um einen Wurf auszuwerten, sprich Einer, Zweier, Dreier etc. zu zählen. Dies hilft bei der späteren Überprüfung, ob ein Wurf eine bestimmte Bedingung erfüllt. Die Auswertung übernimmt eine Methode `wurfAuswerten()`.

* Die Klasse `SpielzettelPanel` ist relativ komplex. Sie benötigt Label als Beschriftungen und Felder, um den Spielstand auszugeben. Für die Felder werden Textfelder (Klasse `TextField`) verwendet. Während der Name als einzelnes Textfeld betrachtet werden kann, bietet sich für die einzelnen Spielstände ein Feld `TextField[] ausgabeFeld` der Länge 17 an. Dabei steht jede Position im Feld für eine bestimmte Kategorie (entsprechend dem Feld `spielstand` der Klasse `Spielzettel`).

* Weiterhin muss entschieden werden, wie der Spieler angeben kann, wo er ein Würfelergebnis auf dem Spielzettel eintragen möchte. Hierzu scheinen zwei Varianten plausibel: Variante 1: Der Spieler gibt das Ergebnis direkt in seinem Spielzettel, also dem `SpielzettelPanel`, ein. Das System überprüft, ob die Eintragung der Bedingung des Feldes entspricht und gibt andernfalls eine Fehlermeldung aus mit der Aufforderung die Eintragung zu korrigieren. Variante 2: Für jedes Feld wird ein Auswahl-Element zur Verfügung gestellt (ein Button oder ein Auswahlfeld ...), das angeklickt werden kann. Das System überprüft, ob die Bedingung des Feldes erfüllt ist und trägt automatisch den richtigen Punktstand ein (die errechnete Punktzahl für das Feld bzw. 0, wenn die Bedingung nicht erfüllt ist). Hier wurde für Variante 2 entschieden. Wenn der Wurf gewertet werden soll, wird neben jedem Ausgabefeld ein kleiner Button eingeblendet, der angeklickt werden kann. Die Buttons sind nur aktiv, wenn das jeweilige Feld noch keinen Eintrag enthält. Sie werden wieder ausgeblendet, sobald die Auswahl getroffen wurde. Dazu ist ein zusätzliches Button-Feld nötig: `Button[] buttonFeld`.

3. Ausführliches Klassendiagramm

Nun können wir ein ausführliches Klassendiagramm erstellen.



4. Beispiel einer Oberfläche

Eine mögliche Oberfläche könnte wie folgt aussehen.

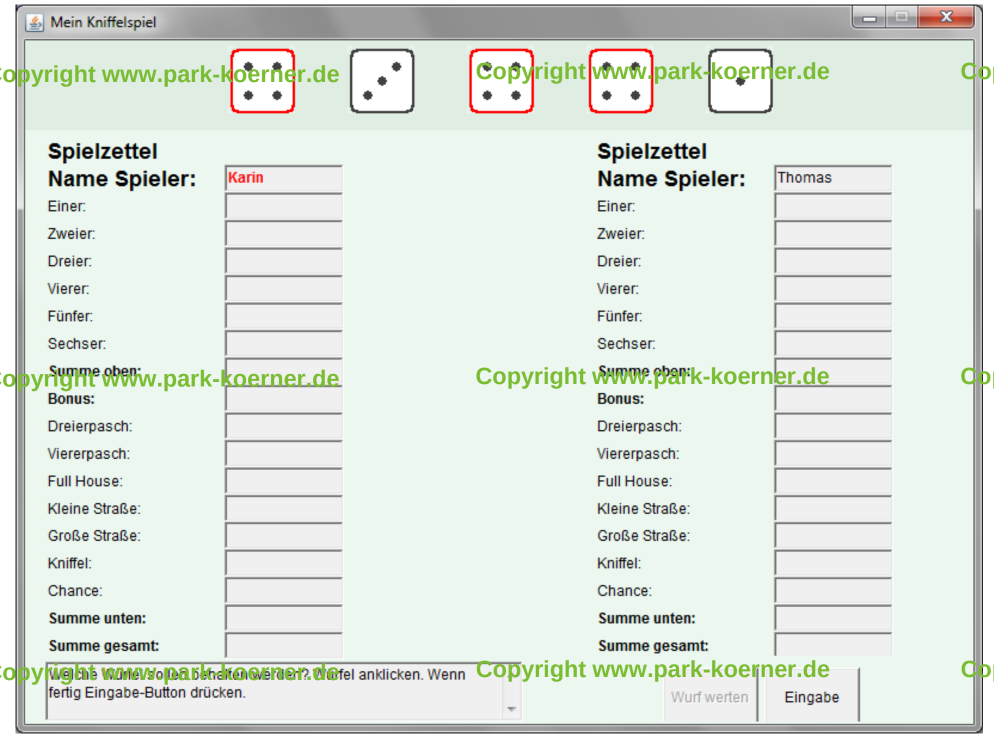


Illustration: http://office.microsoft.com/de-de/ (28.02.14)

Projekt 1: Entwicklung eines Kniffelspiels

3. Implementierung und Test

1. Implementierung

Nachdem wir nun eine klare Vorstellung davon haben, wie das Programm aussehen soll, beginnt das eigentliche Programmieren.

An dieser Stelle solltet ihr euren Zeitplan für das Projektmanagement überprüfen, ihn euren neuen Erkenntnissen anpassen und die Aufgaben so aufteilen, dass mehrere Teams möglichst effizient parallel arbeiten können und die Meilensteine eingehalten werden. Beachtet dabei, dass einige Klassen ziemlich einfach sind, während andere Klassen einen größeren Aufwand für die Implementierung brauchen. In diesem Projekt sind vor allem die Oberflächen-Elemente, also die Oberfläche selbst, die Würfelsymbole und die Spielzettelpanel, und die Bedingungen für die Felder des Spielzettels relativ aufwendig und sollten entsprechend großzügig geplant werden.

2. Einige Tipps und Werkzeuge für die Implementierung verschiedener Klassen

Die Klasse `Oberflaeche`:

In diesem Projekt werden AWT-Klassen für die GUI-Erstellung verwendet. Neben AWT bietet Java auch die fortschrittlicheren, aber auch komplexeren Swing-Klassen für GUI-Aufgaben. Die unten genannten Klassen haben meist auch eine Swing-Entsprechung (erkennbar am „J“ vor dem Namen, also „JButton“, statt „Button“). Natürlich ist auch die Verwendung dieser Klassen möglich. Es sollte aber darauf geachtet werden, AWT- und Swing-Klassen nicht in einem Projekt zu mischen, da dies zu unerwarteten Problemen führen kann.

Die Oberfläche besteht aus verschiedenen Bereichen. Natürlich können Elemente auf der Oberfläche frei positioniert werden. Es ist aber mitunter ziemlich aufwendig, ein optisch ansprechendes Ergebnis zu erzielen. Hilfreich ist es, verschiedene Bereiche als sogenannte Panels zu definieren, die als Container für bestimmte Elemente dienen. Dies erleichtert das Positionieren sehr. Die Klasse `java.awt.Panel` ist dazu einen Blick wert.

Andere wichtige Klassen für die Implementierung der Oberfläche sind: `java.awt.Frame`, `java.awt.Button`, `java.awt.TextArea`.

GUI-Klassen sind grundsätzlich komplex. Es ist nicht nötig, die genannten Klassen komplett zu verstehen, da nur wenige Methoden der jeweiligen Objekte für das Projekt nötig sind. Eine Recherche nach Code-Beispielen im Internet kann helfen, Funktionen besser zu verstehen.

Ein Tipp, der für die Arbeit mit allen Oberflächenelementen hilfreich ist:

Oberflächenelemente sollten erst sichtbar gesetzt werden (`setVisible(true)`), nachdem alle Unterelemente hinzugefügt wurden (der Fenster-Frame zum