

Inhaltsverzeichnis

| | |
|--|-----------|
| 1 Vorwort..... | 4 |
| 2 Unterrichtsentwurf: Erstellung einer Klassenhomepage..... | 5 |
| 2.1 Anmeldevorgang..... | 5 |
| 2.2 Bilder mit eigenem Namen umbenennen..... | 5 |
| 2.3 Eigene Seite aus Vorlage erzeugen..... | 6 |
| 2.4 Präsentation der erstellten Seiten..... | 6 |
| 2.5 Erstellen einer verweissensitiven Grafik des Klassenbildes..... | 6 |
| 2.6 Wohnort auf der Landkarte markieren..... | 7 |
| 2.7 Zeitbedarf..... | 7 |
| 2.8 Veröffentlichung..... | 7 |
| 2.9 Zusammenfassung..... | 8 |
| 3 Einführung und Vertiefung der Zählerbausteine als Blackbox..... | 9 |
| 3.1 Kurze Beschreibung..... | 9 |
| 3.2 Einführungsphase..... | 9 |
| 3.3 Übungs- und Vertiefungsphase..... | 10 |
| 3.4 Einordnung der Einheit in den Unterricht..... | 12 |
| 3.5 Download Profilab..... | 15 |
| 3.6 Ausblick..... | 15 |
| 4 Einführung Zählerbausteine: Seilbahnstation..... | 16 |
| 4.1 Funktion des Drehkreuzes..... | 16 |
| 4.2 Analyse der Schaltung..... | 16 |
| 4.3 Anzeige..... | 17 |
| 4.4 Zähler bis sechs..... | 17 |
| 5 Aufgabenstellungen Vergnügungspark Zählerbaustein..... | 18 |
| 5.1 Karussell – vorwärts zählen..... | 18 |
| 5.2 Hau den Maulwurf..... | 18 |
| 5.3 Süßigkeitenautomat – vorwärts und rückwärts zählen..... | 19 |

| | |
|---|-----------|
| 5.4 Karussell – rückwärts zählen – laden..... | 19 |
| 5.5 Positionslampen Raumschiff – maximaler Zählerstand..... | 20 |
| 5.6 Wasserspiel..... | 20 |
| 5.7 Dosenschießen..... | 21 |
| 5.8 Einarmiger Bandit – Kaskadierung von Zählern..... | 21 |
| 5.9 Aufzug im Geisterhaus – vorwärts und rückwärts zählen..... | 22 |
| 6 Einschätzungsbogen Vergnügungspark Zählerbausteine..... | 23 |
| 7 Lösungen Einführung Zählerbausteine..... | 24 |
| 7.1 Funktion des Drehkreuzes..... | 24 |
| 7.2 Analyse der Schaltung..... | 24 |
| 8 Kooperative Modellierung und Programmierung von Text-Adventure-Spielen..... | 25 |
| 8.1 Einführung..... | 25 |
| 8.2 Teil 1 (Programmierung eines Text-Adventure-Spiels nach Anleitung)..... | 25 |
| 8.3 Teil 2 (Entwicklung und Programmierung eines eigenen Text-Adventure-Spiels)..... | 26 |
| 9 Programmierung eines Text-Adventure-Spiels nach Anleitung..... | 26 |
| 9.1 Beschreibung..... | 26 |
| 9.2 Material..... | 27 |
| 10 Spielen der Text-Adventure-Spiele der anderen Paare..... | 35 |
| 11 Ableitung des UML-Klassendiagramms aus dem programmierten Text-Adventure-Spiel..... | 35 |
| 12 Entwicklung eines Beurteilungsbogens für ein eigenes Text-Adventure-Spiel..... | 35 |
| 12.1 Beschreibung..... | 35 |
| 12.2 Beispiel eines von Schülerinnen und Schülern entwickelten Beurteilungsbogens..... | 36 |
| 13 Entwicklung und Programmierung eines eigenen Text-Adventure-Spiels..... | 37 |
| 13.1 Beschreibung..... | 37 |
| 13.2 Selbsteinschätzung..... | 37 |
| 14 Beurteilung der Text-Adventure-Spiele durch die Schülerinnen und Schüler..... | 39 |
| 15 Bewertung des Text-Adventure-Spiels als Klassenarbeit..... | 39 |

| | |
|--|-----------|
| 16 Anhang Text-Adventure-Spiel-Anleitung 1..... | 40 |
| 17 Anhang Text-Adventure-Spiel-Anleitung 2..... | 44 |
| 18 Anhang Text-Adventure-Spiel-Anleitung 3..... | 48 |

1 Vorwort

In dieser Handreichung sind drei unterschiedlich komplexe Projekte dargestellt, die zu verschiedenen Zeitpunkten im Jahresverlauf der Eingangsklasse angeordnet sind und daher auf unterschiedlichen Vorkenntnissen aufbauen und unterschiedliche Zielsetzungen verfolgen.

- 1) Bereich Software: Erstellung einer Klassenhomepage
(ca. 2 Doppelstunden, erste Unterrichtsstunden)
Seiten 5 bis 8
- 2) Bereich Hardware: Unterrichtssequenz Zähler
(ca. 4 Doppelstunden, Beginn/Mitte 1. HJ)
Seiten 9 bis 24
- 3) Bereich Software: Kooperative Modellierung und Programmierung von Text-Adventure-Spielen (ca. 12 Doppelstunden, Beginn 2. HJ)
ab Seite 25

2 Unterrichtsentwurf: Erstellung einer Klassenhomepage

In den ersten Tagen an einer neuen Schule steht das Kennenlernen im Vordergrund. Die Schülerinnen und Schüler kennen sich nur teilweise untereinander, man muss sich in einem neuen großen Schulzentrum zurechtfinden und auch die Lehrkräfte wollen von ihren Schülerinnen und Schülern möglichst schnell nicht nur den Namen wissen. Eine gemeinsame Homepage, bei der jeder seine Seite hat, aber auch die Klasse als Gesamtheit vertreten ist, soll realisiert werden.

Für den Anfang (erster Tag) empfiehlt es sich, Namensschilder zu tragen. Eine billige einfache Methode ist es, Kreppklebeband mit Filzstift zu beschriften und auf das T-Shirt oder den Pullover zu kleben. Im Rahmen eines kurzen Schulrundgangs bietet es sich an, von jedem Lernenden ein Portrait und ein Klassenfoto anzufertigen. Mit Hilfe des Programms „FastStone Image Viewer“ können die Bilder von der Lehrkraft einfach und schnell nachbearbeitet werden. Es empfiehlt sich, das Klassenbild auf eine Breite von 1000 Pixel zu verkleinern. Die Portraits werden auf das Format 300 x 400 reduziert. Alle Bilder werden im img-Ordner der Vorlage abgelegt. Die Vorlage sollte im Gesamten auf einem Tauschlaufwerk im Schulnetz verfügbar sein.

2.1 Anmeldevorgang

Die Schülerinnen und Schüler bekommen nun den (initialen) Anmeldevorgang erklärt. Als gut hat sich die Methode erwiesen, einen Freiwilligen an den Lehrerrechner nach vorne zu bitten und der Klasse die nötigen Schritte am Beamer zu zeigen. Der Freiwillige bekommt mündliche Instruktionen von der Lehrkraft und wird als eine Art Fernbedienung eingesetzt. Der Vorteil ist, dass die Aktionen nicht zu schnell gezeigt werden und dass eventuelle Fehlbedienungen auch gleich für alle einmal korrigiert werden können.

Es folgt eine kleine Praxisphase, in der die Schülerinnen und Schüler den Anmeldevorgang zur Übung mehrfach durchführen.

2.2 Bilder mit eigenem Namen umbenennen

Der nächste Arbeitsschritt wird von der Lehrkraft präsentiert. Es ist besonders darauf zu achten, dass alle Schülerinnen und Schüler die Aufmerksamkeit zum Beamer richten.

- Start des Windows-Explorers (Windowstaste+E) und Navigation zum img-Ordner auf dem Tauschlaufwerk
- Jeder Lernende möge sein Portrait in seinen Nachnamen (in Kleinbuchstaben) umbenennen. Eventuell vorhandene Umlaute sollen dabei aufgelöst werden. Die Endung .jpg soll erhalten bleiben.

Es folgt die Praxisphase mit der Umsetzung des Gezeigten.

2.3 Eigene Seite aus Vorlage erzeugen

- Man lege sich eine Kopie der Datei „vorlage.html“ an. Diese Kopie wird auf „name.html“ umbenannt, wobei „name“ natürlich der wirkliche Name in Kleinbuchstaben ist.
- Mit Hilfe von NotePad wird die Datei „name.html“ geöffnet und zunächst werden alle Stellen mit „Vorname Name“ durch den eigenen Namen ersetzt.
- Im -Tag wird der Verweis auf das eigene Portrait angepasst.
- Die Daten der eigenen Adresse werden eingefügt.
- Der Lernende kann nun seine Lieblings- und Gar-Nicht-Gernfächer, seine Hobbies, sein Lieblingsessen und seine Lieblingsmusik ergänzen.

2.4 Präsentation der erstellten Seiten

Die Schülerinnen und Schüler stellen sich mit Hilfe ihrer Seite der Klasse vor.

2.5 Erstellen einer verweissensitiven Grafik des Klassenbildes

Wenn man mit der Maus über das Klassenbild fährt, soll der Name des Lernenden angezeigt werden. Außerdem soll ein Klick zur entsprechenden Seite verzweigen. Die Konturen werden mit Hilfe des Hilfsprogramms „ImageMapper“ erstellt und die Koordinaten in die Zwischenablage kopiert. Jeder Lernende erstellt nun neben seiner .html-Seite ein Textdokument mit seinem Namen. Hier fügt er die Koordinaten ein und ergänzt seinen Namen und den Verweis zu seiner .html-Seite. Es ist nun die Aufgabe der Lehrkraft die .txt-Dateien der Schülerinnen und Schüler in die Map der main.html Datei zu integrieren.

Das ImageMapper-Programm in Aktion.

```
<AREA SHAPE="poly" HREF="name.html" COORDS="56,121, 49,124, 44,129, 41,136, 40,143, 40,150, 40,157, 40,164, 40,171, 37,178, 31,183, 26,188, 20,192, 14,196, 10,202, 8,209, 6,216, 5,223, 5,230, 5,237, 5,244, 3,251, 2,258, 1,265, 0,272, 0,279, 0,286, 0,293, 0,301, 0,309, 0,316, 0,323, 0,330, 4,336, 7,343, 11,349, 13,356, 15,363, 15,370, 17,377, 19,384, 22,391, 24,398, 26,405, 27,412, 28,420, 29,427, 29,434, 31,441, 32,449, 32,456, 33,463, 33,470, 34,477, 37,484, 43,488, 50,488, 54,482, 60,478, 66,482, 72,486, 79,487, 85,483, 83,476, 82,469, 81,462, 80,455, 80,448, 80,441, 84,435, 88,429, 92,423, 95,416, 96,409, 96,402, 96,395, 96,388, 96,381, 96,374, 98,367, 100,360, 103,353, 106,346, 108,339, 110,332, 110,325, 111,318, 111,311, 111,304, 111,297, 111,290, 111,283, 111,276, 111,269, 111,262, 112,255, 113,248, 114,241, 114,234, 114,227, 114,220, 114,213, 113,206, 110,199, 106,193, 101,188, 96,183, 90,179, 83,177, 79,171, 79,164, 79,157, 80,150, 80,143, 78,136, 73,131, 68,126, 62,122"
```

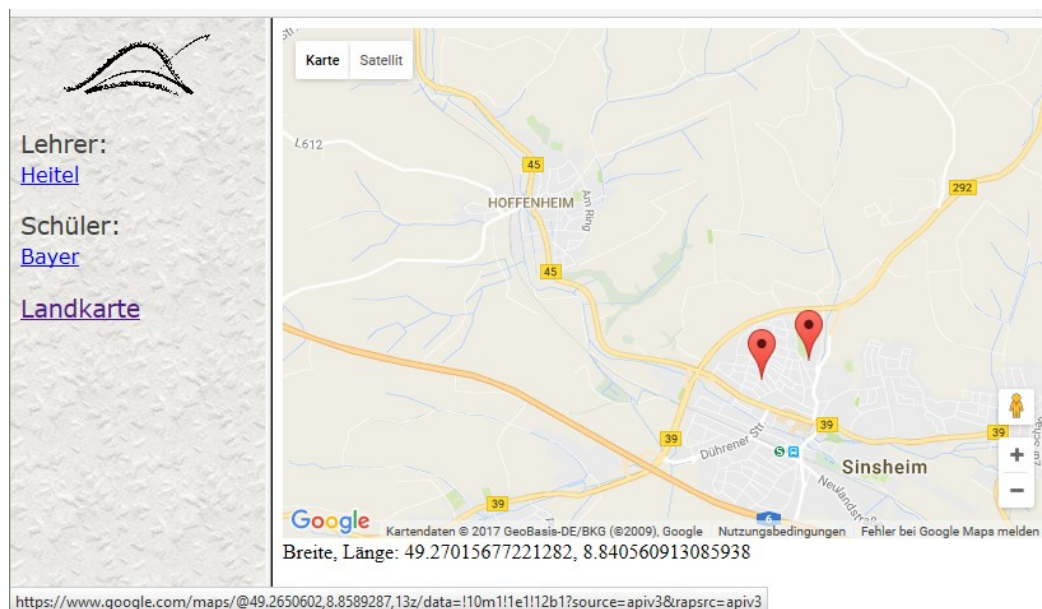
```
ALT="Vorname Nachname" TITLE="Vorname Nachname">
```

2.6 Wohnort auf der Landkarte markieren

Mit Hilfe einer Landkarte soll ein Überblick über die Wohnorte der Schülerinnen und Schüler gewonnen werden. Möglicherweise ist das der Anfang zur Bildung kleiner Lern- und Arbeitsgruppen. Auf der Landkarte navigiert man sich zum eigenen Wohnort. Ein Klick zeigt den zugehörigen Breiten- und Längengrad unterhalb der Landkarte an. Jeder Lernende ändert seine .txt-Datei so ab, dass sie seine Daten anzeigt z.B.:

```
["Friedrich-Hecker-Schule<br>Kelterbuckel 2<br>74889 Sinsheim",  
  49.25822649067473, 8.878369331359863],
```

Die Lehrkraft integriert alle Schüler .txt-Dateien in die tgie2map.html-Datei. Es ist genau auf die Syntax zu achten, da sonst der JavaScript-Interpreter abbricht. Die soeben bestimmten Koordinaten helfen auch, den Link zur Landkarte auf den Seiten der einzelnen Schülerinnen und Schüler anzupassen.



2.7 Zeitbedarf

Bis zur Präsentation der Schülerseiten muss man ca. 90 Minuten Zeit einplanen. Der Rest sollte innerhalb einer weiteren Doppelstunde zu erledigen sein.

2.8 Veröffentlichung

Bei Bedarf kann man die erstellten Seiten auf einem Web-Server publizieren. Da die Inhalte sehr persönlich und nur für den Schulbedarf gedacht sind, sollte die Seite durch ein Kennwort geschützt werden.

2.9 Zusammenfassung

Die Schülerinnen und Schüler haben an diesem Beispiel grundlegende Kenntnisse im Umgang mit den Computern im Schulnetz gewonnen. Sie haben die Zwischenablage verwendet und bereits einige Tastaturkürzel verinnerlicht. Außerdem wurde die Aufmerksamkeit für Präsentationen am Beamer geschärft. Das wichtigste Ziel dieser Unterrichtsstunden ist jedoch das gegenseitige Kennenlernen.

3 Einführung und Vertiefung der Zählerbausteine als Blackbox

3.1 Kurze Beschreibung

Schülerinnen und Schüler haben heute selten Zugang zu den „Innereien“ technischer Geräte. War es vor wenigen Jahren noch möglich, Notebooks und Mobiltelefone zu öffnen um einen Einblick in ihre Komponenten und ihrer Funktion zu erlangen, verkleben viele Hersteller heute ihre Produkte. Damit sind die Innereien unzugänglich, uneinsichtig und versteckt. Was wenn man doch mal einen Wecker öffnet, was findet man heute? Im Zeitalter höchster Integration findet man keine Zahnräder, Federn und Schrauben im Wecker mehr. Eine Batterie, ein angeklebtes IC und ein LC-Display genügen. Einfach langweilig!

So finden sich unter den Schülerinnen und Schüler nur noch wenige Hobby-Bastler, welche Schaltungen von ihrem Inneren her ergründen, Geräte zerlegen und wieder funktionstüchtig zusammenschrauben. Die Vorkenntnisse der meisten Schülerinnen und Schüler sind im Hardwarebereich gering, jedoch ist der Wille, etwas Sinnvolles zu entwerfen und zu bauen, groß.

Daher steht die Anwendung und Konfiguration digitaler Schaltungen und Geräte im Vordergrund bis ausreichend Grundwissen und Motivation vorhanden ist um die Innereien einer Schaltung zu untersuchen. Ausgehend von diesem Gedanken erarbeitet die Unterrichtseinheit den Zählerbaustein anhand seiner Anwendung und konfiguriert einen vorgegebenen Baustein für unterschiedlichste Anwendungen. Das Innere des Bausteins spielt keine Rolle, es bleibt in einer Blackbox verborgen.

Die Einheit besteht aus einer Einführung, Dauer ca. 90 min, und einer offenen Übungs- und Vertiefungsphase. Am Ende der Vertiefungsphase stehen offene, reichhaltige Aufgabenstellungen, welche den Lernenden Raum für eigene Ideen eröffnen. Die unterschiedlichen Anwendungen des Zählerbausteins trainieren die Modellbildungsfähigkeit der Schülerinnen und Schüler.

3.2 Einführungsphase

3.2.1 Vorstellung

Die erste Doppelstunde stellt den Zählerbaustein als Baustein vor und definiert anhand eines Versuchs oder Gedankenspiels in den Blättern „Einführung_Zaehler.pdf“ die folgenden Grundbegriffe der Schaltwerke.

- Steigende und fallende Flanke
- Takt, tastengesteuert
- Reset
- Low aktiv

Diese Phase bietet noch keine Individualisierung der Inhalte, sondern konzentriert sich auf grundlegende Arbeitsweise mit Proflab. Wo befindet sich der Zählerbaustein, welche Bausteine existie-

ren, was seht in der Hilfedatei? Am Ende bauen die Schülerinnen und Schüler eine Schaltung unter Verwendung eines Zählerbausteins auf. Das Grundthema aller Aufgaben bleibt eine Seilbahnstation. Somit ist ein reibungsloser Übergang zwischen den Aufgabenstellungen gewährleistet. Die Sozialform aller Stunden ist die Partnerarbeit mit dem Simulator.

3.2.2 Stundenverlauf

| Dauer | Unterrichtsschritt | Sozialform |
|--------|--|---------------|
| 5 min | Einstieg: Schilderung Problemstellung | Plenum |
| 10 min | Funktion Drehkreuz | Partnerarbeit |
| | Impulsdiagramm Beschriften mit | |
| | steigende Flanke: Person betritt Drehkreuz | |
| | fallende Flanke: Person verlässt Drehkreuz | |
| | statischer Wert: Person im Drehkreuz | |
| 15 min | Untersuchen Schaltung | |
| | Schaltung auf Tauschlaufwerk vorführen | |
| | Funktion der Eingänge und Kodierung der Ausgänge | Partnerarbeit |
| 15 min | Besprechung Ergebnis | Plenum |
| 5 min | Vorstellen Aufgabe 3, 4 | Plenum |
| 30 min | Bearbeitungszeit Aufgabe 3, 4 | Partnerarbeit |
| 20 min | Abschlussbesprechung | |
| | Ende | |

3.3 Übungs- und Vertiefungsphase

Diese Phase setzt einige Aspekte der folgenden Gebiete des Konzepts um:

- Individuelle Lernprozesse gestalten
- Fachliche und überfachliche Kompetenzen systematisch fördern
- Fachliche und überfachliche Kompetenzen feststellen

Die Dauer der Einheit sind ca. 180 Minuten inklusive Besprechung der Aufgaben.

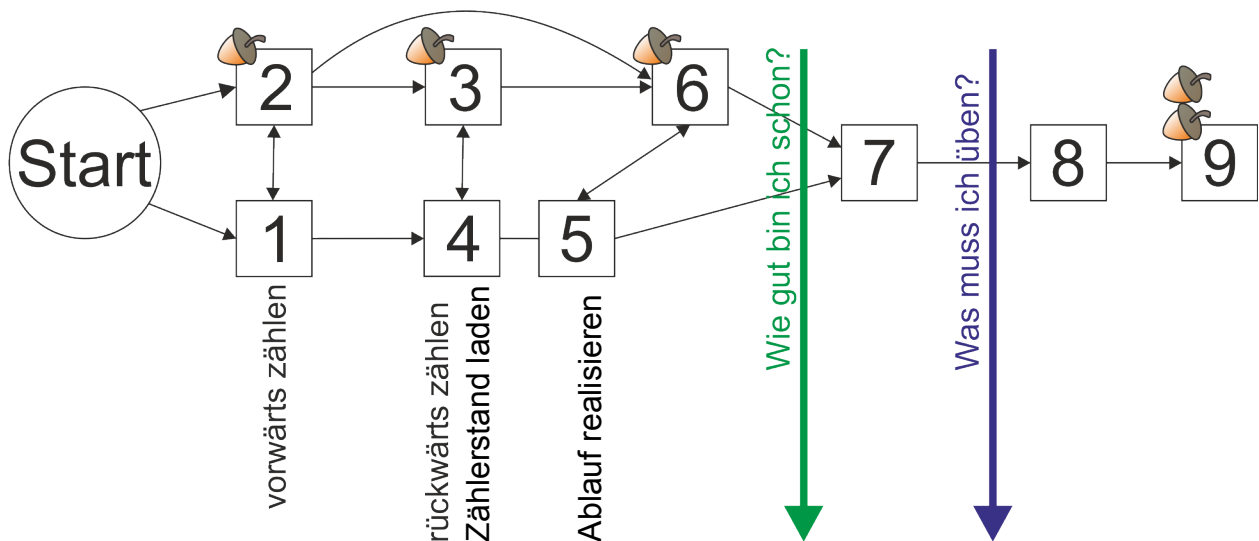


Abbildung 3.1: Bearbeitungsplan der Aufgaben, entnommen aus dem Selbstreflexionsbogen

Vorrangiges Ziel des Unterrichts ist die Selbstständigkeit der Schülerinnen und Schüler sowie das Einarbeiten der ingenieurwissenschaftlichen Arbeitsweise. Schaltungen werden nicht nur auf Papier gezeichnet und theoretisch durchdacht, sondern durch die Schülerinnen und Schüler in Partnerarbeit im Simulator aufgebaut und selbstständig getestet. Wegen der Klassengröße erweist sich die Arbeit am Digiboard oder einzelnen IC als zu unübersichtlich. Viele Schülerinnen und Schüler sind anfangs mit der selbstständigen Kontrolle der Ergebnisse überfordert und verlangen eine Überprüfung ihrer Ergebnisse und eine Bestätigung durch die Lehrkraft. Selbstständiges Arbeiten muss deshalb von der ersten IT-Stunde an geübt werden. Abschlussbesprechungen am Ende der Stunde stellen nicht fehlerhafte und korrekte Lösungen gegenüber, sondern zeigen mehrere unterschiedliche Lösungsansätze für eine Problemstellung. Zusätzlich ergeben sich sehr positive Auswirkungen auf das Klassenklima, denn während Schülerinnen und Schüler ihre Lösungen im Plenum vorstellen, ergibt sich fast von alleine eine wertschätzende Kommunikation. Jeder ist gehalten zuzuhören, denn der Vergleich mit der eigenen Lösung bringt zusätzliches Wissen.

Die Schülerinnen und Schüler erarbeiten selbstständig unter Zuhilfenahme der Unterlagen der vorherigen Einheit weitere Anwendungsmöglichkeiten und Betriebsmodi des Zählerbausteins. Ein Bearbeitungsplan, siehe Abbildung 3.1, ergänzt durch einen Selbstreflexionsbogen, leitet durch die Aufgabenstellungen und ermöglicht den Lernenden die Diagnose seines Lernfortschritts. Haselnüsse demonstrieren die Schwierigkeit der Aufgabenstellungen und teilen außerdem in Fundamentum und Additum ein. Stellen Schülerinnen und Schüler fest, dass sie eine Aufgabe nicht lösen können oder eine entscheidende Idee für die Lösung fehlt, stehen Dateien mit unterschiedlichem Vorfertigungsgrad zur Verfügung. Eine Musterlösung wird den Lernenden nie bereitgestellt, dies widerspricht der Selbstständigkeit. Ein weiteres Augenmerk bei der Gestaltung der Aufgaben ist gezieltes Säen von Vorwissen. Wenn möglich beinhalten Aufgaben bereits weiterführende Elemente, wie das Zustandsdiagramm, und legen so einen Anknüpfungspunkt zur Vernetzung des Wissens. Weiterhin werden mehrere Aufgaben in der Einheit Schieberegister wiederverwendet um unterschiedliche Codierungsarten zu vergleichen, ähnlich der Abiturprüfung.

Aufgabe 7 dient zur Einschätzung der erworbenen Fertigkeiten und zeigt den in der Klassenarbeit zu erwartenden Schwierigkeitsgrad an. Die Selbstreflexion am Ende von Aufgabe 7 verdeutlicht den Schülerinnen und Schülern die noch zu leistende Arbeit für einen guten Lernerfolg. Aufgabe 8, Ein-armiger Bandit, stellt einen offenen Auftrag und eine reichhaltige Aufgabenstellung dar. Nach Erreichen des Fundamentums im ersten Aufgabenteil, bleibt den Lernenden freigestellt, wie sie ihr Spiel erweitern. Über die Jahre sind hierbei sehr unterschiedliche Lösungen entstanden. Zwei harte Nüsse, letzte Aufgabe 9, ist als Additum für herausragend schnelle und leistungsstarke Schülerinnen und Schüler.

3.4 Einordnung der Einheit in den Unterricht

Der gesamte Unterricht folgt im groben einer Bottom-Up-Strategie innerhalb des Gajski-Y-Diagramms in Abbildung 3.2. Im Zentrum des Gajski-Diagramms steht die Schaltungsebene, eine Ebene geringer Abstraktion, eine konkrete Schaltung aus Widerständen, Transistoren, Kondensator und Spule. Dreht man sich auf der Ebene im Kreis ergibt sich eine andere Ansicht der Schaltung, der Abstraktionsgrad ändert sich nicht. Bewegt man sich auf einer Achse nach außen, auf eine höher gelegene Ebene, nimmt die Abstraktion der Schaltung zu. Die Intention des IT-Hardwareunterrichts ist während der Schulzeit auf dem dargestellten Pfad von außen nach innen zu durchschreiten.

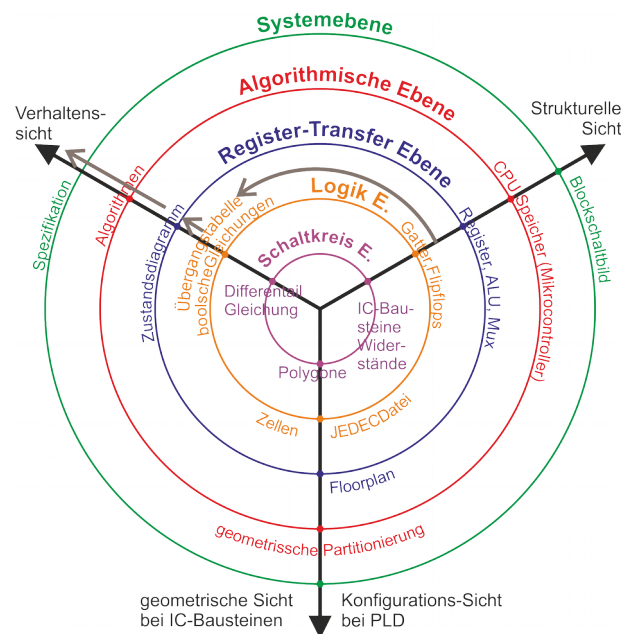


Abbildung 3.2: Gajski-Y-Diagramm

Die Eingangsklasse beginnt mit dem Gedanken der Information, streift kurz die Schaltungsebene, erklimmt die Logik-Ebene und erreicht am Ende die Register-Transfer-Ebene.

Die Schaltungsebene steht nicht im Mittelpunkt des Unterrichts. Trotzdem ist ein kurzer Ausflug auf die Ebene geringster Abstraktion für das Verständnis aller höher gelegenen Ebenen äußerst fruchtbar. Der Mikrocontroller erarbeitet in der Jahrgangsstufe 1 die algorithmische Ebene. Die System-Ebene ist durch die Netzwerktechnik Jahrgangsstufe 2 der letzte Abstraktionsschritt im IT-Unterricht.

Elemente geringer Abstraktion bieten für jeden einen fassbaren Anknüpfungspunkt. Langsam nimmt die Abstraktion zu, der Aufstieg auf die Logik-Ebene wird vollzogen.

Die vorgestellte Einheit ist ein „harter“ Einsprung von der Logik-Ebene zur Register-Transfer-Ebene. Was dramatisch klingt, folgt einem einfachen Prinzip:

Die Vorerfahrung der Schülerinnen und Schüler wirkt sich stark auf den Lernerfolg aus.

Dieser erste Einblick auf die Register-Transfer-Ebene – ohne auch nur ein Flipflop zu kennen – erlaubt auf einfache Weise, komplexe Abläufe in eine Schaltung zu übertragen. Außerdem bietet diese Einheit einen Ausblick auf die Möglichkeiten im weiteren Unterricht.

Die Komplexität bleibt begrenzt, der Zählerbaustein besitzt nur wenig Betriebsmodi. Jeder Betriebsmodus wird konfiguriert.

(a) Grundlagen Elektrotechnik: Schaltkreis-Ebene

- Wichtigster Aspekt ist das Einüben der Partnerarbeit sowie verantwortungsvoller Umgang mit dem Material
- Messung von Stromstärke und Spannung anhand eines Multimeters
- Stromkreis
- Widerstand
- Übungen an vorgegebenen Schaltungen, jedoch keine Vertiefung hinsichtlich Rechengesetze
- Stromkreis, Widerstand

(b) Analog, Digitalwandlung: Zahlensysteme Dezimal, Dual, Hexadezimal

- Versuch am Digiboard

(c) Wahrheitstabellen

- Infotraffic, siehe www.swisseduc.ch/informatik/infotraffic/

(d) Grundverknüpfungen UND, ODER, Inverter

(e) Impulsdigramme

(f) Multiplexer

(g) Komparator / Vergleicher

- Makros in Profilab einbinden
- Taktgeber

Sprung auf die Register-Transfer-Ebene (im Dezember)

(h) Zählerbausteine

- Einführung des Zählerbausteins als Blackbox
- Anwendung des Zählerbausteins auf eine Problemstellung
 - Speicher
 - Grundsätzliche Automatenstruktur
 - Codierung von Zuständen

- Begriffe: Laden, Low-aktiv

- (i) Matrixanzeige Matrixspeicher (Diode)
- (j) EEPROM als einfache programmierbare logische Schaltung

Abstieg auf die Logik-Ebene (im Februar)

- (k) SR-Flipflop
- (l) D-Latch (taktzustandsgesteuert)
- (m) D-Flipflop (taktflankengesteuert)

Aufstieg auf die Register-Transfer-Ebene

- (n) Schieberegister
- (o) Rekapitulation Schieberegister und Zähler
- (p) Medwedew-Automat

- Struktur
- Konstruktion einfacher Zähler
- Zustandskodierung nach Ausgangsgröße

(q) Zustandsübergangsdiagramm des Medwedew-Automaten

(r) AutoEditWorkbook, www.atocc.de: Chickenruns

- Bei den Chickenruns handelt es sich um Übungsaufgaben zu NEA. Dieses Automatenmodell entspricht nicht dem im Abitur geforderten Automatenmodell, bietet aber hervorragende Illustrationen gängiger Fehler bei Moore und Medwedew-Automaten sowie Grenzen von Hardware. Nicht zu vergessen sind die Anknüpfungspunkte des allgemeineren NEA zu Automatenmodellen in der Softwaretechnik, insbesondere der UML.
- Moore-Automat
- Struktur
- Wiederbearbeitung der Aufgaben des Medwedew-Automaten unter dem Aspekt minimale Anzahl Speicher → Gegenüberstellung der Automatenmodelle

(s) Rekapitulation Moore- und Medwedew-Automat und deren Konsequenzen auf das Entwurfsmuster

(t) Rechenwerk und Übergang zum Mikrocontroller

- Addierwerk
- Subtrahierwerk
- Zahlenkreis

- Einfache CPU durch Kombination von EEPROM, Addier-/Subtrahierwerk sowie mehreren Registern

3.5 Download Profilab

Eine Demoversion von Digital Profilab kann unter der folgenden Adresse heruntergeladen werden. Die Demo kann nicht speichern und besitzt eine begrenzte Simulationsdauer.

www.abacom-online.de/html/demoversionen.html

3.6 Ausblick

CPLD oder FPGA sind im aktuellen Lehrplan bereits angesprochen und stellen eine sehr gute Basis für einen handlungsorientierten Unterricht dar. Insbesondere die Verwendung von VHDL als Beschreibungssprache in Kombination einer Modellierung auf Register stellt für Schülerinnen und Schüler einen immensen Gewinn dar. Einerseits erhalten sie einen Einblick in die Entwicklung größerer Studiengangsstrukturen, andererseits bietet insbesondere VHDL einen sehr einfachen Einstieg in die Programmierung paralleler Prozesse. Der Einstieg in VHDL sollte nicht zu spät und nicht ohne konkrete Hardware erfolgen, denn die Arbeit mit Impulsdigrammen der professionellen Simulatoren ist sehr abstrakt und kann sich auf Dauer demotivierend auf die Lernenden auswirken.

Weiterhin bietet VHDL durch Modularität die Möglichkeit, ein Projekt innerhalb der Klasse als großes Gruppenpuzzle umzusetzen. Versagt eine Gruppe oder ist sie zu langsam, können andere Gruppen trotzdem ihre Schaltungen testen, denn Schaltungen können als simulationsfähige aber nicht synthetisierbare Beschreibung an alle Gruppen ausgegeben werden.

4 Einführung Zählerbausteine: Seilbahnstation

In eine Gondelbahn dürfen maximal 15 Personen einsteigen. Das Drehkreuz am Eingang erfasst die einsteigenden Personen. Befinden sich 15 Personen im Eingangsbereich vor der Gondel, wird das Drehkreuz gesperrt.

Eine Schaltung wurde durch einen Kollegen entwickelt. Da der Kollege wegen Krankheit die Schaltung nicht vollenden kann, sind Sie beauftragt, die Eingangssperre fertigzustellen. Außer der Schaltung in Digilab und einem Impulsdiagramm für das Drehkreuz ist keine Dokumentation vorhanden.

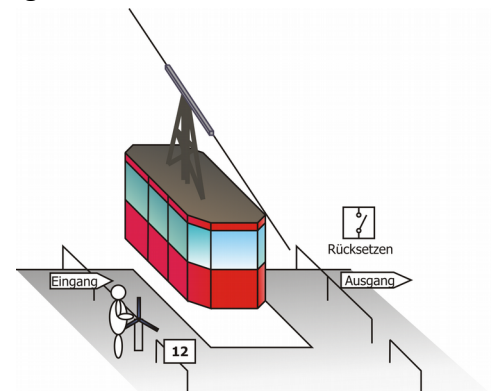
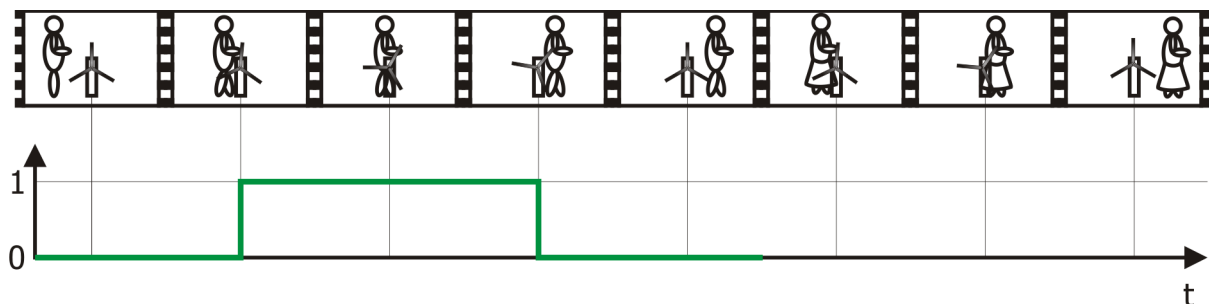


Abbildung 4.1: Seilbahnstation

4.1 Funktion des Drehkreuzes

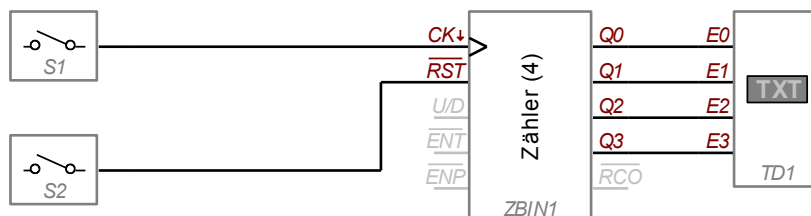
Die folgende Darstellung zeigt das Verhalten des Drehkreuzes. Vervollständigen Sie das Impulsdiagramm.



4.2 Analyse der Schaltung

Kopieren Sie die Schaltung Seilbahn.prj und die Datei Seilbahn.bmp vom Tauschverzeichnis auf den Desktop. Schalten Sie während der Simulation die Schaltfläche und ein.

Untersuchen Sie die Schaltung und beantworten Sie die folgenden Fragen und vervollständigen Sie das Impulsdiagramm. Beschriften Sie die Schalter in Digilab mit ihrer Funktion.



- Welche Funktion hat der Eingang CK?
- Bei welcher Flanke erhöht der Zähler den Zählerstand?
- Welche Funktion hat der Eingang \overline{RST} ?
- Wie wird der Zählerstand dauerhaft auf '0000' gehalten?

4.3 Anzeige

Hinweis:

Erstellen Sie zur Lösung der folgenden Aufgabe jeweils eine Wahrheitstabelle.

- a) Ergänzen Sie die Schaltung um eine rote LED, welche anzeigt, dass die Kabine voll besetzt ist.
- b) In welchem Code stellt der Zählerbaustein den Wert an den Ausgängen Q0 bis Q3 dar?
- c) Ergänzen Sie die Schaltung um eine gelbe LED, welche anzeigt, dass noch höchstens zwei Plätze frei sind. Es leuchtet entweder die rote oder die gelbe LED.

4.4 Zähler bis sechs

Die Schaltung soll nun für eine Kabinenbahn eingesetzt werden. Beschalten Sie den Zähler so, dass er maximal auf sechs zählt und dann wieder auf null springt.

Hinweis: Verwenden Sie den Rücksetzeingang \overline{RST} .

5 Aufgabenstellungen Vergnügungspark Zählerbaustein

Nutzen Sie sowohl das Heft als auch die Hilfe von Profilab.

5.1 Karussell – vorwärts zählen

An einem Karussell wird die Anzahl der Umdrehungen mit einem Zählerbaustein erfasst und auf einem Hex-Display ausgegeben. Der Zählerstand wird bei einer fallenden Flanke erhöht. Am Boden des Karussells ist ein Magnet angebracht, siehe Abbildung 5.1. Passiert der Magnet den Redekontaktschalter T1, so schließt der Kontakt und das Signal S1 nimmt einen High-Pegel an. Ein Reset-Taster T2 am Bedienfeld setzt den Zählerstand auf Null.

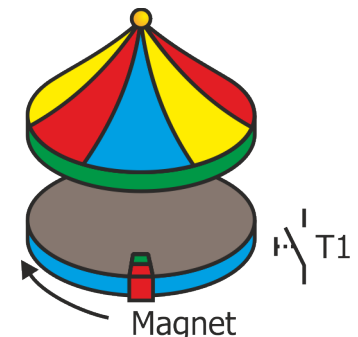


Abbildung 5.1: Anordnung von Magnet und Schalter

- Entwickeln Sie eine Schaltung, welche die Anzahl der Umdrehungen des Karussells anzeigt.
- Eine LED soll ab 5 Umdrehungen rot leuchten.
- Nach 7 Umdrehungen soll das Karussell anhalten.

5.2 Hau den Maulwurf

Beim Maulwurfspiel in Abbildung 5.2 tauchen aus 9 Löchern zufällig Maulwurfsköpfe aus ihren Löchern auf. Der Spieler erhält einen Punkt, wenn er mit dem Hammer den Maulwurf zurück in sein Loch schlägt.

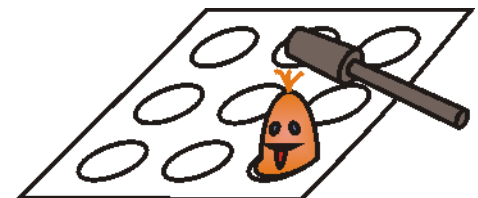


Abbildung 5.2: Spielbrett mit Maulwurf

Das Spielfeld ist als Makro für Profilab verfügbar. Liegt an einem Eingang „Mauw“ ein High-Pegel an, taucht ein Maulwurf auf. Im Kopf jedes Maulwurfs befindet sich ein Taster. Trifft der Spieler diesen Taster mit dem Knüppel, erzeugt er einen Low-Zustand am Ausgang Tref.

Die Aufgabenstellung gliedert sich in unterschiedliche Vorfertigungsstufen.

- Geben Sie die Trefferzahl auf dem Hex-Display aus. Ergänzen Sie hierfür einen Zählerbaustein. Ein Treffer darf nur erfasst werden, wenn der Maulwurf sichtbar ist.
- Haut man dem Maulwurf auf den Kopf, verschwindet er.
- Trifft der Spieler mindestens 10 Maulwürfe, leuchtet eine gelbe LED. Hat der Spieler 14 mal während eines Spiels getroffen, leuchtet die grüne LED.
- Zusatzaufgabe: Der Spieler hat bis zum Spielende nur 15 Schläge zur Verfügung.

5.3 Süßigkeitenautomat – vorwärts und rückwärts zählen

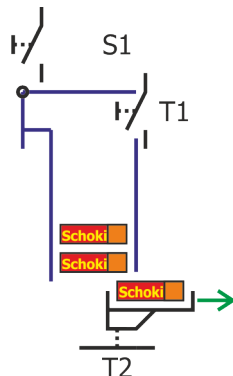


Abbildung 5.3: Schoki entnehmen

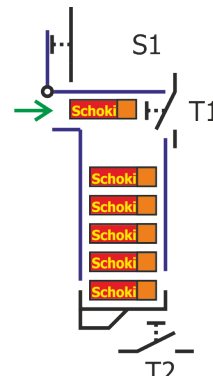


Abbildung 5.4: Schoki befüllen

Im Vergnügungspark verteilt befinden sich mehrere Automaten an denen Besucher Schokoladenriegel erwerben können. Damit das Wartungspersonal entlastet wird, sollen die Automaten einen Indikator für den Füllstand im Kontrollzentrum ansteuern. Eine gelbe LED leuchtet auf, wenn sich weniger als fünf Schokoladenriegel im Automaten befinden. Der Automat muss dann nachgefüllt werden. Eine rote LED zeigt an, dass der Automat leer ist.

Entnimmt man einen Schokoladenriegel, wird T2 geöffnet. An T2 entsteht ein High-Pegel.

Um einen Schokoriegel nachzufüllen, öffnet man die Klappe an der Rückseite des Automaten. Schalter S1 wechselt in einen High-Zustand. Wird ein Schokoladenriegel nachgefüllt, drückt er auf T1, eine steigende Flanke entsteht.

- Erstellen Sie mithilfe eines Zählerbausteins die Überwachungsschaltung. Zeigen Sie die Anzahl Schokoladenriegel auf einem Hex-Display an.
- Zusatzaufgabe schwierig: Sparen Sie S1 ein. Achten Sie auf die Laufzeit der Impulse. Nutzen Sie Inverter um eine kurze Zeitverzögerung zu erreichen.

5.4 Karussell – rückwärts zählen – laden

Das Karussell der ersten Aufgabe soll nun automatisch nach fünf Umdrehungen anhalten. Auf einem Hex-Display soll die Anzahl verbleibender Umdrehungen angezeigt werden.

- a) Die Schaltung setzt den folgenden Ablauf um:
- Starttaster T3 lädt die Anzahl der Umdrehungen in den Zählerbaustein.
 - Redekontaktschalter T1 verringert bei jeder Umdrehung den Zählerstand.
 - Nach der gewünschten Anzahl von Umdrehungen stoppt das Karussell. Eine rote LED zeigt den Stoppzustand an.

Hinweis: Unter Diverses > Plus > Masse > Test finden Sie Symbole für Low- und High-Pegel.

- b) Ergänzen Sie die Schaltung um eine grüne LED, welche anzeigt, dass sich der Motor dreht.

5.5 Positionslampen Raumschiff – maximaler Zählerstand

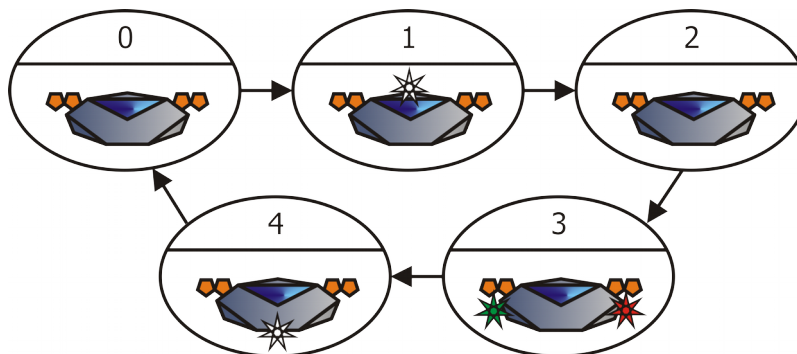


Abbildung 5.5: Zustandsdiagramm: Blinkfolge mittlerer Schiffe

Im Restaurant der Zukunftswelt werden die Besucher durch ein fliegendes Modell eines Planetenverwüsters unterhalten. Am Modell sind Positionslampen angebracht. Die Blinkfolge ist als Zustandsdiagramm in Abbildung 5.5 dargestellt. Entwickeln Sie eine Schaltung für die Blinkfolge. Benutzen Sie einen Taktgenerator, einen Zählerbaustein und Grundverknüpfungen.

5.6 Wasserspiel

Diese Aufgabe besitzt als Hilfestellung unterschiedliche Vorlagen. In der einfachen Vorlage ist einen großen Teil der Schaltung bereits fertiggestellt.

Im Teich gibt es verschiedene Düsen, mit welchen auf Ziele gespritzt werden kann. Diese Ziele tauchen in folgender Reihenfolge auf. Trifft man ein Ziel, erhöht sich die Trefferanzeige.

- 1) Eine Hummel fliegt fünfmal im Kreis. (50 s)
- 2) Ein Boot fährt dreimal im Kreis. (30 s)
- 3) Nessie taucht auf. Trifft man Nessie, spritzt sie zurück oder in die Zuschauer.
Achtung ducken! (10 s)
- 4) Ein Frosch hüpft zweimal auf eine Seerose. (20 s)
- 5) Die Seerose geht unter, Nessie taucht ab. (10 s)
- 6) Die Ente taucht auf. (10 s)
- 7) Die Ente taucht unter. (10 s)
- 8) Der Ablauf beginnt von vorne.

Jeder Treffer ergibt einen Punkt. Trifft man die Ente 10 mal, erhält man eine Riesenstoffente gratis.

- a) Bauen Sie eine Schaltung für den Spielablauf. Ein aktives Tier wird durch eine leuchtende LED dargestellt. Die Bewegung der Tiere ist irrelevant.
- b) Bauen Sie die Trefferanzeige des Spiels auf. Stellen Sie ein Tier durch einen Taster dar. Ein Treffer erzeugt einen High-Pegel.

Die Länge des Spiels entscheidet sich an einer Zielscheibe zu Beginn des Spiels. Trifft man die Mitte der Scheibe, dauert das Spiel 7 Minuten. Trifft man den Rand der Scheibe, dauert das Spiel 5 Minuten. Verfehlt man die Scheibe, kann man 3 Minuten spielen.

- c) Erstellen Sie eine Schaltung, um die Spieldauer zu stoppen. Nutzen Sie einen Zeitgeber, welcher eine Periodendauer von einer Minute erzeugt. Zum Testen können Sie die Periodendauer auf 10 s reduzieren. Die Zielscheibe wird durch Taster umgesetzt. Sie benötigen mehrere Multiplexer.
- d) Argumentieren Sie, weshalb folgender Auftrag nicht mit einem Zählerbaustein gelöst werden kann:

Die Spieldauer wird durch den Gesamtwert der eingeworfenen Münzen bestimmt. Es können mehrere unterschiedliche Münzen eingeworfen werden.

50 Cent → 1 Minute Spieldauer

1 Euro → 3 Minuten Spieldauer

2 Euro → 7 Minuten Spieldauer

5.7 Dosenschießen

Der Aufbau des Spiels ist in Abbildung 21 dargestellt. Fällt eine Dose durch die Lichtschranke, wird die Schranke unterbrochen, LD wechselt auf Low. Die Anzahl der getroffenen Dosen wird auf einem Hex-Display gezeigt. Es gibt drei Bälle.

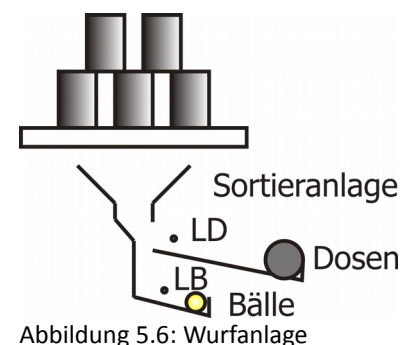


Abbildung 5.6: Wurfanlage

- a) Bauen Sie die Schaltung für die Trefferanzeige auf.
- b) Die verbleibende Anzahl der Bälle wird nun ebenfalls auf einer Anzeige gezeigt.

Hinweise:

- Sie benötigen einen zusätzlichen Zählerbaustein.
 - Unter Diverses > Plus > Masse > Test finden Sie Symbole für Low- und High-Pegel.
- c) Erweitern Sie die Schaltung so, dass mit dem Taster „Spiel beginnen“ ein neues Spiel gestartet wird.

5.8 Einarmiger Bandit – Kaskadierung von Zählern

Ein einarmiger Bandit verfügt über drei Walzen. Die Walzen drehen unterschiedlich schnell. Am schnellsten dreht die rechte Walze. Auf jeder Walze sind die Zahlen 0 bis 9 abgebildet. Der Spieler gewinnt, wenn die drei Walzen identische Zahlen anzeigen. Verwenden Sie den Dezimalzähler.

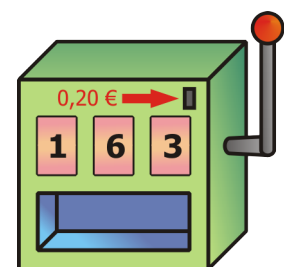


Abbildung 5.7: einarmiger Bandit

- a) Bauen Sie in Profilab eine Schaltung für einen einarmigen Banditen. Bilden Sie die Walzen mit Hilfe von kaskadierten Zählerbausteinen nach. Der Bandit zählt mit hoher Geschwindigkeit vorwärts hoch. Verwenden Sie einen Taktgenerator mit einstellbarer Frequenz f .

Zähler mit größerer Bitzahl kann man durch Kaskadierung von Zählern realisieren. Dabei verbindet man den RCO-Ausgang einer niedrigen Stufe mit dem ENT-Eingang der Folgestufe. Sowohl die Zählrichtungsumschaltung als auch die Taktansteuerung erfolgt für alle Stufen gemeinsam.

- b) Erweitern Sie den Banditen um

- Münzeinwurf,
- Betrugsmodus,
- langsames Auslaufen,
- ...

5.9 Aufzug im Geisterhaus – vorwärts und rückwärts zählen

Im Einstiegsbereich der Geisterbahn dürfen aus Brandschutzgründen nie mehr als 15 Personen sein. Deshalb wird der Besucherandrang durch eine Schleuse begrenzt. Für den Besucher ist die Schleuse als Aufzug getarnt. Befinden sich 12 Personen im Aufzug schließt sich die Türe 1 und der Aufzug setzt sich in Bewegung. Wenn alle Personen den Aufzug verlassen haben, fährt der Aufzug wieder nach oben. Drehkreuz 1 öffnet nur, wenn alle Personen im Aufzug Drehkreuz 2 passieren dürfen. Ein Wagen ist grundsätzlich mit 2 Personen besetzt.

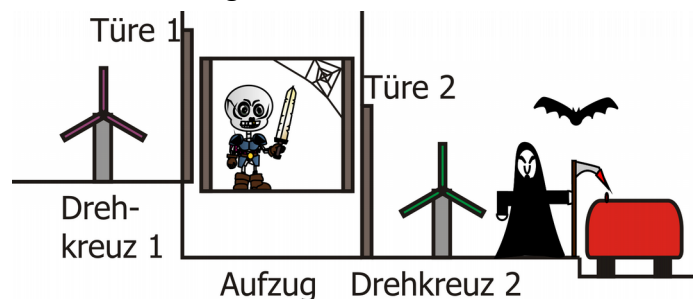


Abbildung 5.8: Position der Schleuse in der Geisterbahn

- a) Erfassen Sie die Anzahl von Personen im Aufzug mithilfe eines Zählerbausteins. Geben Sie die Anzahl von Personen im Aufzug auf einem Hex-Display an.
- b) Erweitern Sie die Schaltung so, dass Drehkreuz 1 bei voll besetzter Aufzugskabine gesperrt ist. Die Sperrung wird durch eine rote LED angezeigt.
- c) Erweitern Sie die Schaltung so, dass eine leere Aufzugskabine durch eine grüne LED angezeigt wird.
- d) Berücksichtigen Sie Drehkreuz 2 und abfahrende Wagen.

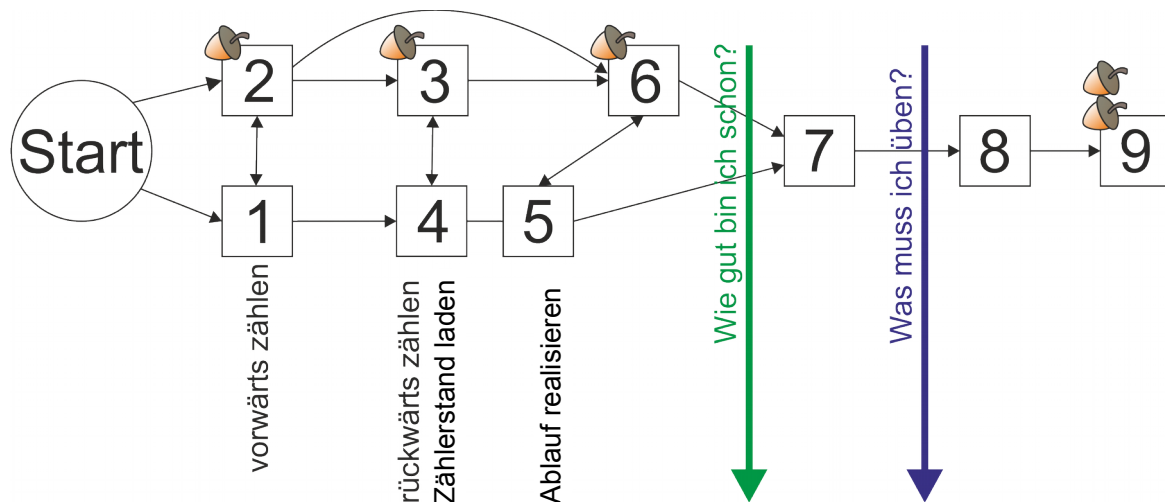
6 Einschätzungsbogen Vergnügungspark Zählerbausteine

Dieser Bogen hilft dir beim Erarbeiten weiterer Eigenschaften von Zählerbausteinen. Beginne bei Start und arbeite dich auf einem Weg durch den Vergnügungspark bis Aufgabe 8 ab. Mit Haselnüssen gekennzeichnete Aufgaben haben einen etwas höheren Schwierigkeitsgrad, sind jedoch interessanter und bringen mehr Wissen und Erfahrung. Wenn du eine Problemstellung nochmals üben möchtest, kannst du die alternative Aufgabe lösen.

Benutze die Hilfe von Profilab und das Heft. Überprüfe dein Ergebnis mit Hilfe einer Simulation selbst.

Wenn du Aufgabe 7 erreichst, dann fülle mit grünem Stift die Einschätzungsfragen aus. Nachdem Aufgabe 7 gelöst ist, beantworte die Einschätzungsfragen ein weiteres Mal. Dann weißt du genau wo du noch üben musst und wo du dich verschätzt hast.

Zeitungsumfang: 140 Minuten



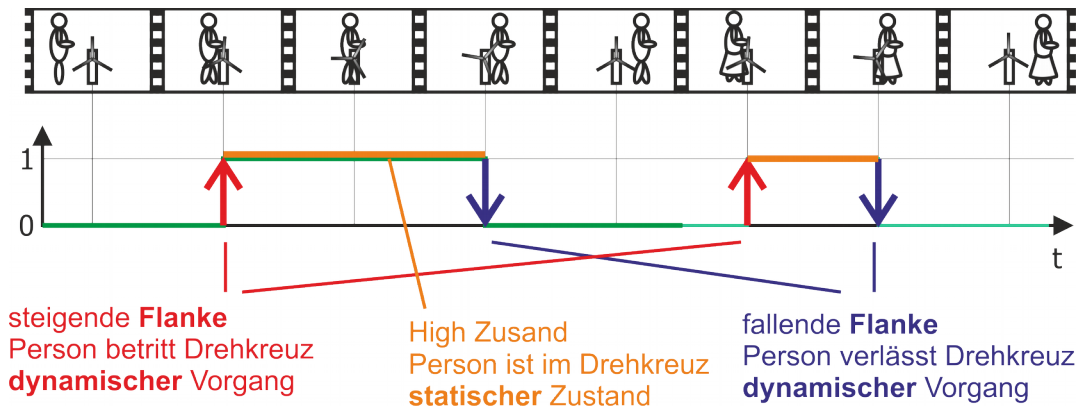
Einschätzungsfragen:

| In der Klassenarbeit gefragt: Ich kann ... | 😊😊 | 😊 | 😐 | 😞 |
|--|----|---|---|---|
| vorwärts zählen konfigurieren. | | | | |
| rückwärts zählen konfigurieren. | | | | |
| unterscheiden wann „laden“ und wann „reset“ notwendig ist. | | | | |
| eine Zahl am Ladeeingang festlegen. | | | | |
| entscheiden wann vorwärts oder rückwärts gezählt wird. | | | | |
| den Zählerstand als Dezimalzahl angeben. | | | | |
| den Zähler auf einen maximalen Zählwert begrenzen. | | | | |
| entscheiden wie viele Bit der Zählerbaustein benötigt. | | | | |
| | | | | |
| In der Klassenarbeit nicht gefragt. | | | | |
| Ich kann vorwärts zählen und rückwärts zählen mischen. | | | | |

7 Lösungen Einführung Zählerbausteine

7.1 Funktion des Drehkreuzes

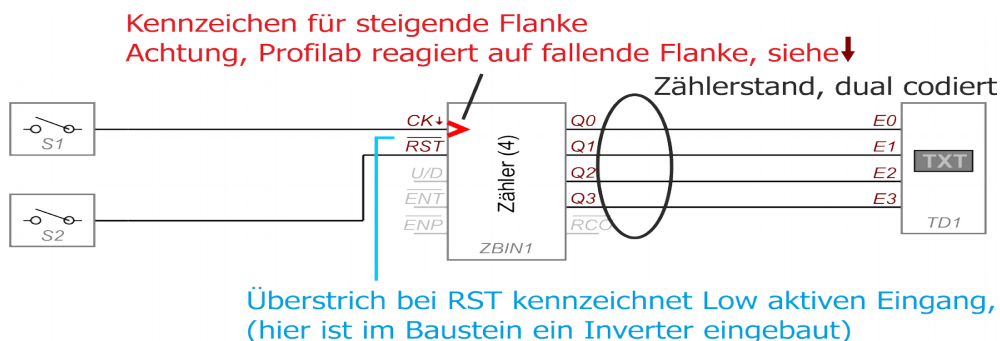
Die nebenstehende Darstellung zeigt das Verhalten des Drehkreuzes. Vervollständigen Sie das Impulsdiagramm.



7.2 Analyse der Schaltung

Kopieren Sie die Schaltung Seilbahn.prj und die Datei Seilbahn.bmp vom Tauschverzeichnis auf den Desktop. Schalten Sie während der Simulation die Schaltfläche und ein.

Untersuchen Sie die Schaltung und beantworten Sie die folgenden Fragen und vervollständigen Sie das Impulsdiagramm. Beschriften Sie die Schalter in Digilab mit ihrer Funktion.



a) Welche Funktion hat der Eingang CK?

Bei einer fallenden Flanke wird der Zählerstand um eins erhöht. Darstellung der fallenden Flanke am Symbol durch Pfeil nach unten.

b) Bei welcher Flanke erhöht der Zähler den Zählerstand?

Der Zählerstand wird bei einer steigenden Flanke erhöht.

c) Welche Funktion hat der Eingang $\overline{\text{RST}}$?

Rücksetzen. Wenn eine 0 an RST anliegt, wird der Zähler zurückgesetzt.

d) Wie wird der Zählerstand dauerhaft auf '0000' gehalten?

Reset muss dauerhaft auf 0 gehalten werden.

8 Kooperative Modellierung und Programmierung von Text-Adventure-Spielen

8.1 Einführung

Typischerweise beginnt der Unterricht in der Eingangsklasse im Fach Informationstechnik (Software) mit Konzepten der strukturierten Programmierung. Die Schülerinnen und Schüler lernen Zuweisungen, Sequenzen von Anweisungen, die ein- und zweiseitige Auswahl, Wiederholungen (Iterationen) und schließlich komplexe Datentypen wie Felder kennen. Vertieft wird dieses Lernen der theoretischen Konzepte im Fach Angewandte Informationstechnik beim Implementieren von Programmen in einer meist objektorientierten Programmiersprache wie z. B. Java. Um das objektorientierte Denken zu fördern, sollte spätestens im zweiten Halbjahr der Eingangsklasse mit der Einführung von Klassen und Objekten begonnen werden. Um die objektorientierte Modellierung und Implementierung in einen für die Schülerinnen und Schüler bekannten und motivierenden Kontext in den Unterricht einzubetten, wird im Folgenden beschrieben, wie dies beim Programmieren eines Text-Adventure-Spiels möglich ist.

Die Unterrichtseinheit „Kooperative Modellierung und Programmierung von Text-Adventure-Spielen“ gliedert sich grob in die abwechselnden Phasen der Programmierung eines Text-Adventure-Spiels in Partnerarbeit oder im Team und dem Spielen der Spiele. Im ersten Teil wird ein Text-Adventure-Spiel nach einer vorgegebenen Anleitung programmiert. Dadurch wird eine mögliche objektorientierte Software-Architektur eines Text-Adventure-Spiels vorgegeben, die später im zweiten Teil bei der Entwicklung eines eigenständigen Text-Adventure-Spiels im Team als Grundlage verwendet werden kann. Dazwischen wird die objektorientierte Software-Architektur des ersten Text-Adventure-Spiels in Form eines UML-Klassendiagramms veranschaulicht und Beurteilungskriterien für ein selbst programmiertes Text-Adventure-Spiel entwickelt.

Die folgenden beiden Tabellen geben einen Überblick über die verschiedenen Aktivitäten während der Unterrichtseinheit. Die Zeitangaben sind nicht verbindlich und können sich je nach Vertiefung einzelner Konzepte verändern.

8.2 Teil 1 (Programmierung eines Text-Adventure-Spiels nach Anleitung)

| | |
|--|--------|
| Partnerwahl Zielvereinbarung Text-Adventure-Spiel (Datei <i>Zielvereinbarung.pdf</i>) | 30 min |
| Programmierung eines Text-Adventure-Spiels nach Anleitung Anleitung – Teil 1 (Datei <i>TextAdventureSpiel_Anleitung1.pdf</i>) Lernkarten 01 – 04 (Verzeichnis <i>Lernkarten</i>) | 90 min |
| Programmierung eines Text-Adventure-Spiels nach Anleitung Anleitung – Teil 2 (Datei <i>TextAdventureSpiel_Anleitung2.pdf</i>) Lernkarten 05 - 06 (Verzeichnis <i>Lernkarten</i>) | 90 min |
| Programmierung eines Text-Adventure-Spiels nach Anleitung Anleitung – Teil 3 (Datei <i>TextAdventureSpiel_Anleitung3.pdf</i>) | 90 min |
| Reflexion über die Zielvereinbarung (Datei <i>Zielvereinbarung.pdf</i>) | 20 min |

| | |
|---|--------|
| Spiele der verschiedenen Text-Adventure-Spiele | 45 min |
| Entwicklung des UML-Klassendiagramms für das Text-Adventure-Spiel | 90 min |

8.3 Teil 2 (Entwicklung und Programmierung eines eigenen Text-Adventure-Spiels)

| | |
|--|-------------|
| Erarbeitung von Beurteilungskriterien zur Bewertung eines Text-Adventure-Spiels (z. B. Datei <i>Beurteilungsbogen.pdf</i> als ein mögliches Ergebnis dieses Prozesses) | 45 min |
| Gruppenbildung | 10 min |
| Entwicklung und Programmierung eines eigenen Text-Adventure-Spiels | ca. 500 min |
| Selbsteinschätzung (Datei <i>Selbsteinschätzung.pdf</i>) | 30 min |
| Spiele und Bewerten der Text-Adventure-Spiele anderer Gruppen (eigene Datei wie z.B. <i>Beurteilungsbogen.pdf</i>) | 90 min |
| Bewertung durch die Lehrkraft | |

9 Programmierung eines Text-Adventure-Spiels nach Anleitung

9.1 Beschreibung

Im ersten Teil wird nach einer vorgegebenen Anleitung ein Text-Adventure-Spiel programmiert. Die Geschichte spielt auf einem englischen Anwesen in einem Haus mit einem Keller, einem Erdgeschoss und einem Dachgeschoss. Die Etagen des Hauses enthalten Rätsel, die mit Hilfe von Gegenständen, die sich ebenfalls in den Etagen befinden, gelöst werden müssen. Durch die Anleitung wird die Klassenstruktur und dadurch ganz allgemein die Architektur eines Text-Adventure-Spiels vorgegeben. Dabei werden die bis dahin gelernten objektorientierten Konzepte wiederholt und vertieft. Kreativ werden die Schülerinnen und Schüler in der Arbeit, die Rätsel und dazugehörigen Hilfestellungen zu erfinden.

Programmiert wird in Partnerarbeit, wobei die Paare durch die Lehrperson vorgegeben werden. Es wird empfohlen, Paare aus jeweils einem programmiertechnisch stärkeren und schwächeren Schülerinnen und Schüler zu bilden.

Da bis zur Unterrichtseinheit nicht alle Konzepte zur Umsetzung eines Text-Adventure-Spiels im Unterricht behandelt werden können, werden die noch fehlenden Informationen wie z. B. das Einlesen von Zahlen über die Konsole auf sogenannte Lernkarten beschrieben. Die Lernkarten liegen im Unterrichtsraum aus und können von den Schülerinnen und Schülern bei Bedarf geholt werden. Außerdem gibt es auch Lernkarten, die bereits eingeführte Konzepte wiederholen.

Bei der Partnerarbeit beim Programmieren achtet die Lehrkraft nach einer Eingewöhnungsphase darauf, dass beide Partner Programmierarbeit übernehmen, d. h. an der Tastatur und Maus sitzen und den Programmcode im Editor eingeben. Somit wird sichergestellt, dass sich beide Schülerinnen und Schüler aktiv an der Partnerarbeit beteiligen.

Zusätzlich wird vor der Programmierung von jeder Schülerin und jedem Schüler eine Zielvereinbarung erarbeitet, in der der Einsatz für das Text-Adventure-Spiel und der erhoffte fachliche Fort-

schritt im Bereich der Informationstechnik durch die Arbeit schriftlich fixiert wird. Nach der Programmierung wird von den Schülerinnen und Schülern ein Zwischenbericht erstellt und möglicherweise um eine Lernberatung gebeten.

Sind Paare schneller mit ihrem Text-Adventure-Spiel fertig, so können sie selbstständig kleine Erweiterungen an ihrem Spiel vornehmen.

Da die Schülerinnen und Schüler selbstständig mit der Anleitung für das Text-Adventure-Spiel und den Lernkarten arbeiten, hat die Lehrkraft Zeit, einzelne Schülerinnen und Schüler zu fördern und ihnen zu helfen, ihre individuellen Lücken im Bereich Informationstechnik zu schließen.

9.2 Material

9.2.1 Anleitung des Text-Adventure-Spiels

Für die Programmierung des Text-Adventure-Spiels nach Anleitung sind drei bis vier Doppelstunden vorgesehen. Aus diesem Grund ist die Anleitung in folgende drei Teile aufgeteilt:

- TextAdventureSpiel_Anleitung1.pdf
- TextAdventureSpiel_Anleitung2.pdf
- TextAdventureSpiel_Anleitung3.pdf

Die Anleitung bezieht sich auf die Programmiersprache *Java* und die Entwicklungsumgebung *Eclipse*. Sollten andere Programmiersprachen und/oder Entwicklungsumgebungen eingesetzt werden, so müssen die Dokumente entsprechend angepasst werden.

9.2.2 Lernkarten

Auf den Lernkarten werden komprimiert Informationen zusammengefasst, die die Schülerinnen und Schüler zur Programmierung des Text-Adventure-Spiels brauchen, die jedoch noch nicht im Unterricht eingeführt wurden oder wiederholt werden sollen.

Die Lernkarten stellen beispielhaft dar, wie Schülerinnen und Schüler beim selbstständigen Erarbeiten und Wiederholen von Lerninhalten unterstützt werden können. Auch hier ist eine Anpassung und Neuentwicklung von Lernkarten in Abhängigkeit des Lernstandes der Klasse notwendig.

Unterrichtseinheit „Kooperative Modellierung und Programmierung von Text-Adventure-Spielen“
Teil 1: Programmierung eines Text-Adventure-Spiels nach Anleitung

Lernkarte 01 – Konstanten in Java

Häufig braucht man in der Programmierung an vielen Stellen eines Programms denselben konstanten Wert. Dieser Wert hat eine spezielle Bedeutung.

Beispiel:

- 19 Mehrwertsteuer
- 1 / 2 / 3 Mann / Frau / Kind
- 100 Maximale Temperatur einer Heizung

Anstatt nun immer wieder diesen konstanten Wert zu verwenden, ist es besser, sich einmal für eine Bezeichnung dieses Wertes zu entscheiden und dann immer nur den Bezeichner des Wertes und nicht den Wert selbst im Programmcode zu verwenden. Im Unterschied zu einer Variablen kann der Wert einer Konstanten nur einmal gesetzt und dann nicht mehr verändert werden.

Java:

Konstanten werden wie Attribute angelegt. Der einzige Unterschied besteht darin, dass mit dem Schlüsselwort `final` festgelegt wird, dass die Werte der Konstanten nicht verändert werden können, und dass ihnen direkt ein Startwert zugewiesen werden muss. Wie man im Beispiel unten erkennen kann, werden Konstanten in Java immer mit Großbuchstaben bezeichnet.

```
public class Person{
    // Konstanten
    private final int STATUS_LEDIG = 1;
    private final int STATUS_VERHEIRATET = 2;
    private final int STATUS_GESCHIEDEN = 3;
    private final int STATUS_VERWITWET = 4;

    // Attribute
    private String name;
    private int status;
    ...

    public void ausgeben(){
        System.out.print(name + " ist ");
        switch (status){
            case STATUS_LEDIG:
                System.out.println("ledig.");
                break;
            case STATUS_VERHEIRATET:
                System.out.println("verheiratet.");
                break;
            case STATUS_GESCHIEDEN:
                System.out.println("geschieden.");
                break;
            case STATUS_VERWITWET:
                System.out.println("verwitwet.");
                break;
            case default:
                System.out.println("in einem unbekannten Status.");
                break;
        }
    }
}
```

Abbildung 9.1: Lernkarte 01 - Konstanten in Java

Unterrichtseinheit „Kooperative Modellierung und Programmierung von Text-Adventure-Spielen“
Teil 1: Programmierung eines Text-Adventure-Spiels nach Anleitung

Lernkarte 02 – Attribute im Konstruktor initialisieren

Konstruktoren einer Klasse werden zum Erzeugen von Objekten verwendet. Dabei sollen die erzeugten Objekte in einen definierten Anfangszustand gebracht werden. Dies erreicht man dadurch, dass den Attributen der Objekte bei ihrer Erzeugung Startwerte zugewiesen werden.

Beispiel:

```
public class Kaempfer{
    // Attribute
    private String name;
    private int staerke;
    private Waffe waffe;

    // Konstruktoren
    public Kaempfer(String pName){
        name = pName;
        staerke = 20;
        waffe = new Waffe("Schwert",3);
    }

    public Kaempfer(String pName, int pStaerke, Waffe pWaffe){
        name = pName;
        staerke = pStaerke;
        waffe = pWaffe;
    }
    ...
}
```

Die Zuweisung der Startwerte zu den Attributen kann in drei Varianten erfolgen. Dies zeigt das Beispiel oben:

1. Der Startwert eines Attributs kann im Parameter übergeben werden.

Diese Variante wurde im ersten Konstruktor für das Attribut `name` und im zweiten Konstruktor für alle Attribute angewandt.

```
name = pName;
staerke = pStaerke;
waffe = pWaffe;
```

2. Der Startwert eines Attributs mit einem primitiven Datentyp (byte, short, int, long, float, double, boolean, char) kann im Konstruktor gesetzt werden.

Diese Variante wurde im ersten Konstruktor für das Attribut `staerke` angewandt, da ein Standardkämpfer immer eine Stärke von 20 hat.

```
staerke = 20;
```

3. Der Startwert eines Attributs mit dem Datentyp einer Klasse kann im Konstruktor auf ein neu erzeugtes Objekt gesetzt werden.

Diese Variante wurde im ersten Konstruktor für das Attribut `waffe` angewandt, da ein Standardkämpfer als Waffe ein Schwert mit einem Zerstörungsgrad von 3 besitzt. Das `Waffe`-Objekt muss allerdings erst mit Hilfe von `new` und dem Aufruf des entsprechenden Konstruktors der Klasse `Waffe` erzeugt werden.

```
waffe = new Waffe("Schwert",3);
```

Abbildung 9.2: Lernkarte 02 - Attribute im Konstruktor initialisieren

Unterrichtseinheit „Kooperative Modellierung und Programmierung von Text-Adventure-Spielen“
Teil 1: Programmierung eines Text-Adventure-Spiels nach Anleitung

Lernkarte 03 – Private Operationen in Klassen

Private Operationen haben die Einschränkung, dass sie nur von anderen Operationen derselben Klasse aufgerufen werden können.

Sie bieten jedoch folgende Vorteile:

- **wiederverwendbarer Programmcode**, da sie eine kleine abgeschlossene Teilaufgabe erledigen und somit an verschiedenen Stellen eingesetzt werden können.
- Reduktion der Komplexität von Operationen, indem private Operationen jeweils eine Teilaufgaben des Gesamtproblems lösen (**Modularisierung**); die Lösungen der Teilaufgaben in den privaten Operationen können dann in der komplexen Operation verwendet werden. Der Programmcode der komplexen Operation bleibt also kurz und übersichtlich. Eine Voraussetzung für das Gelingen der Modularisierung ist die sinnvolle Benennung der privaten Operationen. Der Name muss möglichst eindeutig die Funktion der Operation beschreiben.

```
public class Statistik{
    private int messdaten[];
    public Statistik(){
        messdaten = new int[365];
    }

    private int berechneMin(){...}
    private int berechneMax(){...}

    private int getIndex(int pZahl){
        int index;
        int i = 0;
        while ( i<messdaten.length ){
            if ( messdaten[i]==pZahl ){
                index = i;
                i++;
            }
        }
        return index;
    }
    public int spannweite(){
        int min = berechneMin();
        int max = berechneMax();
        return (max-min);
    }
    public int getIndexMax(){
        int max = berechneMax();
        return getIndex(max);
    }
}
```

Der Programmcode der privaten Operation `berechneMax` wird sowohl in der Operation `spannweite` als auch in der Operation `getIndexMax` wieder verwendet.

Das komplexe Problem, den Array-Index des größten Elementes der Messdaten (`messdaten`) zu finden, wurde hier in die beiden Module „Finden des Maximums“ und „Finden des Index eines Wertes“ zerlegt. Die Lösung in der Operation `getIndexMax` ist hier sehr gut nachvollziehbar und einfach zu verstehen.

Abbildung 9.3: Lernkarte 03 – Private Operationen in Klassen

Unterrichtseinheit „Kooperative Modellierung und Programmierung von Text-Adventure-Spielen“
Teil 1: Programmierung eines Text-Adventure-Spiels nach Anleitung

Lernkarte 04 – Menüauswahl in der Konsole (Tastatureingabe von Zahlen)

In einem Textmenü wählt der Benutzer am besten über die Eingabe von Zahlen seine nächste Aktion in der Konsole aus.

Mit Hilfe eines Objekts der Klasse `Scanner` (aus dem Paket `java.util`) kann man einfach die Eingaben, die von der Tastatur (`System.in`) kommen, scannen.

```
Scanner input = new Scanner(System.in);
```

Um nun die vom Benutzer zur Menüauswahl eingegebene Zahl zu erhalten, kann mit der Operation `public int nextInt()` der Klasse `Scanner` die Zahl gelesen werden.

```
int choice = input.nextInt();
```

Beispiel:

Im Spiel *Tic Tac Toe* gibt es die Möglichkeit, dass Spieler 1 oder Spieler 2 das Spiel beginnt oder das Spiel abgebrochen wird. Entsprechend der Tastatureingabe des Spielers, die vom `Scanner`-Objekt `input` gelesen wird, wird dann das Spiel gestartet oder abgebrochen.

```
import java.util.Scanner;

public class TicTacToe{
    ...

    public void showMenu(){
        System.out.println("Men");
        System.out.println("(0) Spieler 1 beginnt");
        System.out.println("(1) Spieler 2 beginnt");
        System.out.println("(2) Abbrechen");

        Scanner input = new Scanner(System.in);
        int choice = input.nextInt();

        switch (choice){
            case 0:
                start(1);
                break;
            case 1:
                start(2);
                break;
            case 2:
                System.exit(0);
                break;
            default:
                System.out.println("Falsche Eingabe!!!");
                break;
        }
    }

    public void start(int pStartSpielerNr){
        ...
    }
}
```

Abbildung 9.4: Lernkarte 04 – Menüauswahl in der Konsole (Tastatureingabe von Zahlen)

Unterrichtseinheit „Kooperative Modellierung und Programmierung von Text-Adventure-Spielen“
Teil 1: Programmierung eines Text-Adventure-Spiels nach Anleitung

Lernkarte 05 – null als Standardwert für Attribute mit komplexem Datentyp

Auf der **Lernkarte 02 - Attribute im Konstruktor initialisieren** hast du schon gelernt, dass Attribute einen **primitiven Datentyp** (byte, short, int, long, float, double, boolean, char) oder den Datentyp einer Klasse (**komplexer Datentyp, Referenzdatentyp**) haben können. Gleiches gilt natürlich auch für Variablen und Parameter.

Das Attribut eines primitiven Datentyps hat einen Standardwert, der vom jeweiligen Datentyp abhängig ist (z.B. 0 oder 0.0). Ein guter Programmierstil ist jedoch daran zu erkennen, dass auch Attributen mit primitiven Datentypen im Konstruktor explizit ein Wert zugewiesen wird.

```
public class Auto{
    private int geschwindigkeit;
    public Auto(){
        geschwindigkeit = 0;
    }
}
```

Bei Attributen mit komplexen Datentypen (Referenzattribute) ist dies anders. Die Attribute verweisen auf ein Objekt. Im Attribut wird nicht das Objekt, sondern eine Referenz, d.h. ein Verweis auf das Objekt gespeichert. Existiert das Objekt noch nicht (es wurde noch nicht mit `new` und dem Aufruf des Konstruktors angelegt), so sollte in dem Attribut das so genannte `null`-Literal als Standardwert gespeichert werden.

Der Wert `null` bedeutet nichts Anderes, als dass das Attribut mit dem komplexen Datentyp noch auf **nichts**, also kein Objekt, verweist.

```
public class Kaempfer{
    private int staerke;
    private Waffe waffe;
    private Ruestung ruestung;
    public Kaempfer(int pStaerke){
        staerke = pStaerke;
        waffe = new Waffe("Schwert");
        ruestung = null;
    }

    public void addRuestung(Ruestung pRuestung){
        ruestung = pRuestung;
    }

    public void setWaffe(Waffe pWaffe){
        waffe = pWaffe;
    }
}
```

Da die Rüstung des Kämpfers nicht bei seiner Erzeugung feststeht, sondern von ihm erst zu einem späteren Zeitpunkt gekauft und mit der Operation `addRuestung` hinzugefügt wird, wird das Attribut `ruestung` mit dem `null`-Literal im Konstruktor initialisiert. Anders verhält es sich mit der Waffe des Kämpfers. Standardmäßig erhält jeder Kämpfer ein Schwert als Waffe. Diese Waffe kann jedoch zu einem späteren Zeitpunkt mit Hilfe der Operation `setWaffe` verändert werden.

Abbildung 9.5: Lernkarte 05 – null als Standardwert für Attribute mit komplexem Datentyp

Unterrichtseinheit „Kooperative Modellierung und Programmierung von Text-Adventure-Spielen“
Teil 1: Programmierung eines Text-Adventure-Spiels nach Anleitung

Lernkarte 06 – Delegation von Aufgaben an Objekte anderer Klassen

Ein Objekt einer Klasse benötigt manchmal zur Erfüllung seiner Aufgaben die Hilfe anderer Objekte. Es ruft dann die Operationen anderer Objekte auf, erhält von diesen möglicherweise Ergebnisse zurück und verarbeitet die Ergebnisse in irgendeiner Form, um seine Aufgabe zu erfüllen.

In der Fachsprache der Informationstechnik bezeichnet man die Aufrufe der Operationen anderer Objekte als Delegation von Aufgaben, da die anderen Objekte Arbeit leisten, die dem aufrufenden Objekt beim Lösen seiner Aufgabe behilflich ist.

Beispiel:

```
public class PlanungFarbeinkauf{
    private Kreis kreis;
    private Rechteck r;
    public PlanungFarbeinkauf(double pRadius,double pB,double pH){
        kreis = new Kreis(pRadius);
        r = new Rechteck(pB, pH);
    }
    public double getUeberdeckteFlaeche(){
        double flaeche;
        flaeche = kreis.getFlaeche();
        flaeche = flaeche + r.getFlaeche();
        return flaeche;
    }
}

public class Kreis{
    private double radius;
    public Kreis(double pRadius){
        radius = pRadius;
    }
    public double getFlaeche(){
        return Math.PI*pRadius*pRadius;
    }
}

public class Rechteck{
    private double breite;
    private double hoehe;
    public Rechteck(double pBreite, double pHoehe){
        breite = pBreite;
        hoehe = pHoehe;
    }
    public double getFlaeche(){
        return (breite*hoehe);
    }
}
```

Im Beispiel kann man sich vorstellen, dass mehrere Flächen (hier ein Kreis und ein Rechteck) mit einer Farbe überstrichen werden müssen. Damit genügend Farbe eingekauft wird, muss zuerst bestimmt werden, wie groß die bedeckte Fläche ist. Dies erfolgt in der Operation `getUeberdeckteFlaeche` der Klasse `PlanungFarbeinkauf`. Diese delegiert jedoch die Arbeit an das `Kreis`-Objekt `kreis` und an das `Rechteck`-Objekt `r`, die beide die Operation `getFlaeche` besitzen und mit dieser ihre Fläche berechnen und zurückgeben. Somit hat die Operation `getUeberdeckteFlaeche` nur noch die Aufgabe, die Werte der beiden Flächen zu addieren, um sie dann ebenfalls zurück zu geben.

Abbildung 9.6: Lernkarte 06 - Delegation von Aufgaben an Objekte anderer Klassen

9.2.3 Zielvereinbarung für die Partnerarbeit

Mit der Zielvereinbarung für die Partnerarbeit werden die Schülerinnen und Schüler unterstützt, sich besser selbst einzuschätzen, ihre Stärken und Schwächen zu identifizieren und sich realistische Ziele zu setzen.

Unterrichtseinheit „Kooperative Modellierung und Programmierung von Text-Adventure-Spielen“
Teil 1: Programmierung eines Text-Adventure-Spiels nach Anleitung

Zielvereinbarung für die Partnerarbeit

Name: _____ Datum: _____

Mein Ziel ist es, mich bei der Programmierung des Text-Adventure-Spiels nach Anleitung

| sehr | | | | | | überhaupt nicht | |
|------|---|---|---|---|---|-----------------|---|
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | | | | | | | |

in die Partnerarbeit einzubringen.

Dafür werde ich folgenden Einsatz bringen (z.B. programmieren (aktiv/passiv), entwickeln von Rätseln, testen des Programms, einarbeiten in neue Themengebiete, zeitlicher Einsatz außerhalb des Unterrichts etc.):

Unterschriften

Schülerin/Schüler: _____

Partnerin/Partner: _____

Überprüfung der Ziele

☐ Mein Ziel und mein Einsatz stimmen nach wie vor überein.

☐ Mein Ziel und mein Einsatz scheinen auseinander zu laufen. Daraus haben sich folgende Konsequenzen für das Text-Adventure-Game ergeben:

Abbildung 1: Zielvereinbarung für die Partnerarbeit

9.2.4 Musterlösung des Text-Adventure-Spiels

Die Anleitung zur Erstellung des Text-Adventure-Spiels liegt als Musterlösung in Java mit Javadoc-Kommentaren vor: TextAdventure_MusterloesungJava.zip und TextAdventure_MusterloesungJava-Doc.zip

10 Spielen der Text-Adventure-Spiele der anderen Paare

Um die Programmierarbeit der Schülerinnen und Schüler wertzuschätzen, werden im Unterricht 45 Minuten dem Spielen der Text-Adventure-Spiele eingeräumt. Die Schülerinnen und Schüler versuchen dabei, die Rätsel der Spiele der anderen Paare zu lösen.

In Abhängigkeit des Klassenklimas kann aus dem Spielen ein Wettbewerb veranstaltet werden, in dem das Paar am Ende gewinnt, das die meisten Rätsel anderer Spiele gelöst hat. Die Statistik des Wettbewerbs kann an der Tafel festgehalten und das Sieger-Paar nach 45 Minuten gekürt werden.

11 Ableitung des UML-Klassendiagramms aus dem programmierten Text-Adventure-Spiel

Im Fach Informationstechnik (Software) wird aus dem programmierten Text-Adventure-Spiel das UML-Klassendiagramm abgeleitet. Da alle Paare ihr Programm nach der Anleitung entwickelt haben, entsteht hier ein für alle Schülerinnen und Schüler einheitliches UML-Klassendiagramm. Dabei wird das Konzept der Assoziation zwischen Klassen eingeführt. Das Text-Adventure-Spiel eignet sich hierfür, da es komplex genug ist, um den Unterschied zwischen primitiven und komplexen Datentypen von Attributen im UML-Klassendiagramm zu veranschaulichen.

12 Entwicklung eines Beurteilungsbogens für ein eigenes Text-Adventure-Spiel

12.1 Beschreibung

Im zweiten Teil der Unterrichtseinheit wird ein eigenes Text-Adventure-Spiel im Team entwickelt. Für dieses wird zunächst im Klassenverband ein Katalog von Beurteilungskriterien erarbeitet, nach dem die Spiele später bewertet werden. Im Katalog könnten z. B. Kriterien stehen zu

- Programmiertechniken,
- sauberem Programmcode, Einhaltung von Programmierrichtlinien,
- Spannung/Kreativität des Text-Adventure-Spiels.

Die in der Klasse entwickelten Beurteilungskriterien werden von der Lehrperson gesammelt, sortiert und in einem Beurteilungsbogen zusammengefasst. Dieser Beurteilungsbogen muss am Ende des Projekts von den Schülerinnen und Schülern zur Bewertung des Spiels anderer Teams eingesetzt werden.

Zusätzlich kann der Beurteilungsbogen den einzelnen Teams in der Projektarbeit helfen, sich auf die wesentlichen Aspekte der Entwicklung ihres Text-Adventure-Spiels zu fokussieren und sich

nicht in Randgebiete zu vertiefen. Er sollte also auch bei der Planung des Projekts berücksichtigt werden.

12.2 Beispiel eines von Schülerinnen und Schülern entwickelten Beurteilungsbogens

Unterrichtseinheit „Kooperative Modellierung und Programmierung von Text-Adventure-Spielen“
Teil 2: Entwicklung und Programmierung eines eigenen Text-Adventure-Spiels

Beurteilungsbogen (Bewertung in Schulnoten)

Name des zu bewertenden Text-Adventure-Spiels: _____

Bewertung durch (Name der Schülerin, des Schülers): _____

Programmierung

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Layout des Programmcodes | | | | | | |
| Übersicht im Programmcode | | | | | | |
| Sauberkeit, Aufgeräumtheit des Programmcodes | | | | | | |
| Verständlichkeit des Programmcodes | | | | | | |
| Guter Programmcode | | | | | | |
| Korrektheit | | | | | | |
| Stabilität des Programms (z.B. keine Abstürze) | | | | | | |
| Funktioniert alles, was dem Spieler angeboten wird? | | | | | | |
| Macht das Programm das, was man von ihm erwartet? | | | | | | |
| Komplexität | | | | | | |
| Funktionalität | | | | | | |
| Schwierigkeit des Programmcodes | | | | | | |
| Komplexe Spielmechanik | | | | | | |
| Benutzeroberfläche | | | | | | |
| Verständlichkeit | | | | | | |
| Kommentare | | | | | | |
| Ausreichende Kommentierung des Programmcodes | | | | | | |

Story

| | 1 | 2 | 3 | 4 | 5 | 6 |
|--|---|---|---|---|---|---|
| Interessante, kreative und sinnvolle Story | | | | | | |
| Spannung | | | | | | |
| Besondere Elemente, die man nicht erwartet | | | | | | |
| Verständnis der Story | | | | | | |
| Schwierigkeit der Aufgaben im Spiel | | | | | | |

Text-Adventure-Spiel

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Spielspaß | | | | | | |
| Intuitives Spiel, das Spielprinzip ist schnell zu verstehen | | | | | | |
| Logischer Aufbau des Text-Adventure-Spiels | | | | | | |
| Interessanter Spielverlauf | | | | | | |
| Angemessene Schwierigkeit | | | | | | |
| Lösbarkeit des Text-Adventure-Spiels | | | | | | |

Abbildung 12.1: Beurteilungsbogen eines Text-Adventure-Spiels

13 Entwicklung und Programmierung eines eigenen Text-Adventure-Spiels

13.1 Beschreibung

Im zweiten Teil der Unterrichtseinheit wird nun in ca. fünf Doppelstunden ein Text-Adventure-Spiel mit einer eigenen Story und einer eigenen Software-Architektur entwickelt. Dazu wird eine neue Lerngemeinschaft aus drei Schülerinnen und Schülern gebildet. Die Teamzusammensetzung kann in Abhängigkeit der Klassenstruktur per Zufall, von der Lehrkraft oder von den Schülerinnen und Schülern selbst bestimmt werden.

Während der Programmierarbeit können optional Techniken wie die paarweise Programmierung, Code Reviews oder Software-Tests zur Sicherung der Qualität der Programmcodes eingeführt und ausprobiert werden.

13.2 Selbsteinschätzung

Mit Hilfe des Selbsteinschätzungsbogens können die Schülerinnen und Schüler über ihre Rolle im Team reflektieren. Durch eine Aufteilung in Rollen, die man während der Entwicklung des Spiels eingenommen hat, und Rollen, in denen man sich verbessert hat, können alle Schülerinnen und Schüler eine Entwicklung feststellen. Außerdem werden Ziele herausgearbeitet, indem Beispiele genannt werden müssen, in denen man sich verbessern möchte. Zuletzt wird noch analysiert, in welchen Rollen man in Zukunft mehr arbeiten möchte und wie man das erreichen kann.

Kann eine Schülerin oder ein Schüler den Selbsteinschätzungsbogen nicht ausfüllen, so ist ein Coaching-Gespräch mit der Lehrkraft notwendig, um die Selbstwahrnehmung der Schülerin oder des Schülers zu verbessern. Das Gespräch sollte stärkenorientiert erfolgen.

Unterrichtseinheit „Kooperative Modellierung und Programmierung von Text-Adventure-Spielen“
Teil 2: Entwicklung und Programmierung eines eigenen Text-Adventure-Spiels

Selbsteinschätzungsbogen

Name: _____ Datum: _____

Ich habe mich in meinem Team bei der Entwicklung und Programmierung eines eigenen Text-Adventure-Spiels folgendermaßen eingebracht:

| Rolle | sehr | | | | | überhaupt nicht | | |
|--------------------|------|---|---|---|---|-----------------|---|---|
| | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| Software-Architekt | | | | | | | | |
| Programmierer | | | | | | | | |
| Game Designer | | | | | | | | |
| Test-Entwickler | | | | | | | | |
| Tester | | | | | | | | |
| Koordinator | | | | | | | | |
| Konfliktmanager | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

In der angegebenen Rolle habe ich mich folgendermaßen verbessert:

| Rolle | sehr | teilweise | kaum | überhaupt nicht |
|--------------------|------|-----------|------|-----------------|
| Software-Architekt | | | | |
| Programmierer | | | | |
| Game Designer | | | | |
| Test-Entwickler | | | | |
| Tester | | | | |
| Koordinator | | | | |
| Konfliktmanager | | | | |
| | | | | |
| | | | | |

Erkläre an zwei Beispielen, wie du die Verbesserung erreicht hast:

In welcher dieser Rollen möchtest du am liebsten in nächster Zeit weiterarbeiten? Welche Tätigkeit bietet dir die Möglichkeit, dies zu tun?

Bei der Entwicklung und Programmierung des eigenen Text-Adventure-Spiels hat mir Folgendes sehr viel Spaß gemacht:

Abbildung 13.1: Selbsteinschätzungsbogen

14 Beurteilung der Text-Adventure-Spiele durch die Schülerinnen und Schüler

In einer Doppelstunde spielt jede Schülerin und jeder Schüler mindestens zwei Spiele anderer Teams und bewertet diese mit Hilfe des gemeinsam erarbeiteten Beurteilungsbogens (vgl. Abschnitt 5.2).

Enthält der Beurteilungsbogen auch Kriterien zum Programmcode, werden die Schülerinnen und Schüler gezwungen, sich zumindest oberflächlich mit fremden Programmcodes auseinanderzusetzen. Dies ermöglicht ihnen, alternative Vorgehensweisen in der Programmierung kennenzulernen, ihre eigene Arbeit im Vergleich zu den anderen einzuschätzen und ihren aktuellen Leistungsstand zu reflektieren.

15 Bewertung des Text-Adventure-Spiels als Klassenarbeit

Die Bewertung des Text-Adventure-Spiels als Klassenarbeit kann z. B. folgendermaßen aufgeteilt werden:

- Titel des Text-Adventure-Spiels mit einer kurzen Beschreibung des Spielinhalts (Abgabe in der 2. Doppelstunde, Notenanteil: 5 %)
- Text-Adventure-Spiel (Notenanteil: 70 %)
- Spielanleitung für das Text-Adventure-Spiel (Notenanteil: 5 %)
- UML-Klassendiagramm des Text-Adventure-Spiels (Notenanteil: 10 %)
- Bewertung von zwei Text-Adventure-Spielen anderer Teams mit Hilfe des Beurteilungsbogens (Notenanteil: 10 %)

Die Bewertung der Spiele durch die Schülerinnen und Schüler kann in der Note für das Text-Adventure-Spiel berücksichtigt werden.

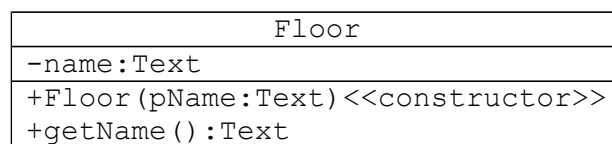
16 Anhang Text-Adventure-Spiel-Anleitung 1

In dieser Aufgabe wirst du in Partnerarbeit dein erstes *Text-Adventure-Spiel* implementieren. Bitte befolge die Vorgaben für das Spiel genau. Zu einem späteren Zeitpunkt, wenn du noch mehr über Java gelernt hast, wirst du ein ganz individuelles *Text-Adventure-Spiel* nach deinen Vorstellungen implementieren. Es ist also noch Geduld notwendig.

Dein erstes *Text-Adventure-Spiel* spielt auf einem Anwesen (engl. *property*). Dort gibt es ein Haus mit den Stockwerken (engl. *floor*) Keller (engl. *basement*), Erdgeschoß (engl. *first floor*) und Dachgeschoß (engl. *top floor*). Natürlich ist das Dachgeschoß und der Keller nur über das Erdgeschoß des Hauses zu erreichen. Die Geschichte um das Anwesen ist von euch selbst frei zu erfinden.

Aufgabe 1 (Vorbereitung von Eclipse): Lege in *Eclipse* ein Projekt *TextAdventureGame* mit einer Klasse *AdventureGame* an. Achte in der Klasse *AdventureGame* darauf, dass du das Häkchen für die Generierung der `main`-Operation setzt.

Aufgabe 2 (Programmierung der Klasse `Floor`): Programmiere die Klasse `Floor` für das *Text-Adventure-Spiel* nach dem unten angegebenen UML-Klassendiagramm. Später werden der Keller, das Erdgeschoss und das Dachgeschoss Objekte der Klasse `Floor` sein. Im Attribut `name` wird der Name des Stockwerks gespeichert.



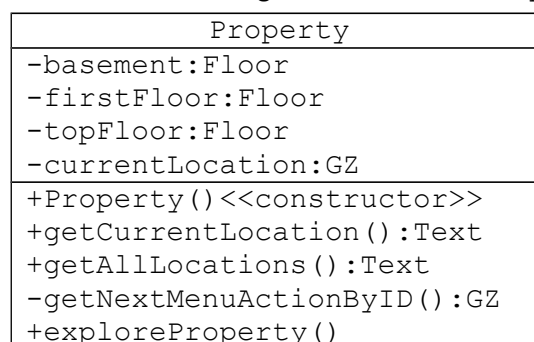
Aufgabe 3 (Programmierung der Klasse `Property`):

(a) Erstelle im Projekt *TextAdventureGame* eine Klasse *Property*.

(b) Lege die Konstanten `LOCATION_BASEMENT`, `LOCATION_FIRSTFLOOR` und `LOCATION_TOPFLOOR` als ganzzahlige Werte 1, 2 und 3 für die verschiedenen Orte des *Text-Adventure-Spiels* in der Klasse *Property* ganz oben an.

Lernzeit: Lese auf der **Lernkarte 01 - Konstanten in Java** nach, wie Konstanten in einer JAVA-Klasse definiert und verwendet werden.

(c) Programmiere das folgende UML-Klassendiagramm der Klasse *Property* in Java.



Der Programmcode innerhalb des Konstruktors und der Operationen wird noch leer gelassen. Damit keine Fehler entstehen, müssen in den Operationen `getCurrentLocation` und `getAllLocations` jeweils ein Text zurückgegeben werden. Dies erreicht man mit der Programmzeile `return ""`;, denn dadurch wird ein leerer Text zurückgegeben. Gleiches gilt für die Operation `getNextMenuActionById`. Hier muss mit `return` eine Zahl zurückgegeben werden.

(d) Programmiere den Konstruktor `Property() <<constructor>>`. Hier werden den Attributen

der Klasse `Floor` die neu erstellten Objekte zugewiesen.

Lernzeit: Lese auf der **Lernkarte 02 – Attribute im Konstruktor initialisieren** nach, wie Attribute mit einem primitiven Datentyp und Attribute, deren Datentyp eine Klasse ist, im Konstruktor initialisiert werden können.

Außerdem muss der aktuelle Aufenthaltsort im Attribut `currentLocation` gespeichert werden. Das *Text-Adventure-Spiel* beginnt im Erdgeschoß. Der Wert wird mit Hilfe der in (b) erstellten Konstanten gesetzt.

- (e) Die Operation `getCurrentLocation` gibt den Namen des Orts als Text zurück, an dem sich der Spieler gerade auf dem Anwesen befindet. Um den Namen herauszufinden, muss man das entsprechende `Floor`-Objekt nach seinem Namen fragen und diesen dann zurückgeben.
- (f) Die Operation `getAllLocations` gibt alle Stockwerke des Hauses in einem Text zurück. Der Text soll folgenden beispielhaften Aufbau haben:

Im Haus gibt es folgende Stockwerke:

- Keller
- Erdgeschoss
- Dachgeschoss

Dabei sind *Keller*, *Erdgeschoss* und *Dachgeschoss* die Namen der `Floor`-Objekte, die bei diesen selbst mit der Operation `getName` nachgefragt werden können.

- (g) Lege in der Klasse `Property` die Konstanten `MENU_SHOW_CURRENTLOCATION`, `MENU_SHOW_ALLLOCATIONS` und `MENU_EXIT` mit den Werten 1, 2 und 0 an, um verschiedene Spielooptionen in einem textuellen Auswahlmenü anbieten zu können.
- (h) Die Operation `getNextMenuActionByID` bietet dem Spieler des *Text-Adventure-Spiels* das textuelle Hauptmenü der Spielooptionen an. Sie gibt später die vom Spieler ausgewählte Option in Form der entsprechenden Zahl (ID) zurück. Im Moment wird noch eine beliebige Zahl zurückgegeben. Implementiere den Text für folgendes Menü mit `System.out.println` in der Operation `getNextMenuActionByID`. Verwende zur Ausgabe der Zahlen die in (g) angelegten Konstanten.

```
=====
=                MENUE                =
=====
Was wollen sie tun?
(1) Aktuellen Ort anzeigen
(2) Alle möglichen Orte anzeigen
(0) Programm beenden
```

Der Spieler kann die gewünschte Aktion durch Eingabe der entsprechenden Zahl auswählen.

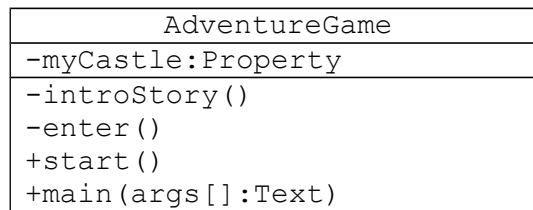
- (i) In der Operation `exploreProperty` soll das Hauptmenü mit den Spielooptionen dargestellt werden. Lege in der Operation `exploreProperty` zunächst eine `int`-Variable `menuActionID` an. Rufe dann die Operation `getNextMenuActionByID` auf und speichere die zurückgegebene ID der Spielaktion in der Variablen `menuActionID` ab.

Die Operation `getNextMenuActionByID` hat die Sichtbarkeit `private`, da sie nur innerhalb einer anderen Operation (hier `exploreProperty`) derselben Klasse verwendet wird. Informiere dich auf der nächsten Lernkarte über `private` Operationen.

Lernzeit: Lese auf der **Lernkarte 03 – Private Operationen in Klassen** nach, warum sowohl öffentliche als auch `private` Operationen in der objektorientierten Programmierung sinnvoll sind.

Aufgabe 4 (Programmierung der Klasse `AdventureGame`): Jetzt wird es Zeit, dass das Programm getestet werden kann. Dazu fehlt nur noch die Implementierung der Klasse `AdventureGame`.

(a) Erweitere die Klasse `AdventureGame` entsprechend dem folgenden UML-Klassendiagramm:



Die Operation `main` gibt es bereits. Die anderen Operationen und der Konstruktor werden erst einmal ohne Programmcode in die Klasse eingefügt.

(b) In der privaten Operation `introStory` sollen eine Begrüßung des Spielers des *Text-Adventure-Spiels* und eine kurze Einführung in das Spiel implementiert werden. Gebe hier mit `System.out.println` einen entsprechenden Text in der Konsole aus, der den Spieler ausreichend informiert.

(c) Programmiere die andere private Operation `enter`. In ihr wird das Objekt `myCastle` erzeugt. Falls du nicht mehr weißt, wie es geht, kannst du dir

- den Programmcode im Konstruktor der Klasse `Property` oder
- die **Lernkarte 02 – Attribute im Konstruktor initialisieren** (vgl. Aufgabe 3 (d))

noch einmal anschauen.

Anschließend wird die Erkundung des Anwesens `myCastle` gestartet. Schicke dazu die Botschaft `exploreProperty` (Operation der Klasse `Property`) an das Objekt `myCastle`. Der Aufruf erfolgt nach dem bekannten Muster:

```
objektname.operationsname();
```

(d) Der Start des *Text-Adventure-Spiels* erfolgt in der Operation `start`. Hier müssen nur die beiden privaten Operationen `introStory` und `enter` nacheinander aufgerufen werden.

Um nun endlich etwas zu sehen, muss ein `AdventureGame`-Objekt in der `main`-Operation mit

```
AdventureGame game = new AdventureGame();
```

erzeugt werden. Rufe anschließend die Operation `start` für dein `game`-Objekt auf.

Teste nun dein *Text-Adventure-Spiel*.

Aufgabe 5 (Erweiterung der Klasse `Property`): Ein Menü hat keinen Nutzen, so lange man keine Aktion auswählen kann. Dies soll sich nun ändern.

(a) Lerne zunächst einmal, wie man Zahlen, die der Benutzer über die Tastatur eingibt, in einer Variablen speichern kann.

Lernzeit: Lese auf der **Lernkarte 04 – Menüauswahl in der Konsole (Tastatureingabe von Zahlen)** nach, wie du herausbekommen kannst, welche Eingaben (Zahl, Text) ein Benutzer über die Tastatur vorgenommen hat.

Erweitere die Operation `getNextMenuActionByID`, so dass nach der Textausgabe des Menüs in einer ganzzahligen Variablen `menuActionID` der Wert der Aktion gespeichert wird, die der Spieler gerade ausgewählt hat. Lass die Operation jetzt den richtigen Wert, die `menuActionID` der Aktion, zurückgeben.

(b) Lege in der Klasse `Property` eine private Operation `doMenuAction(pActionID:GZ)` an. Als

Parameter wird ihr die vom Spieler ausgewählte Aktion in Form einer ID übergeben.

Implementiere in der Operation `doMenuAction` in einer Mehrfachauswahl (`switch-case`) über den Parameter `pActionID` für die drei Menüpunkte (vgl. Aufgabe 3 (h)) die entsprechende Funktionalität. Wird das Programm beendet, so wird ein Text zur Verabschiedung ausgegeben und anschließend das Java-Programm mit dem Befehl `System.exit(0)` gestoppt.

Für die anderen beiden Menüpunkte wurden (vgl. Aufgabe 3 (e) und (f)) bereits entsprechende Operationen in der Klasse `Property` implementiert, die hier nur aufgerufen werden müssen.

- (c) Füge den Aufruf der neuen privaten Operation `doMenuAction` in `exploreProperty` ein. Er steht natürlich hinter `getNextMenuActionByID`, da mit der Operation zunächst die Auswahl der Aktion durch den Spieler in die Variable `menuActionID` gelesen wird und der Wert nun an den Parameter von `doMenuAction` übergeben werden muss.

Jetzt kann das *Text-Adventure-Spiel* endlich getestet werden.

- (d) Du hast leicht feststellen können, dass das Anwesen nur sehr schlecht zu erkunden ist, wenn nach jeder Aktion des Spielers das *Text-Adventure-Spiel* wieder neu gestartet werden muss. Sinnvoller wäre es, die vom Spieler ausgewählte Aktion auszuführen und dann erneut das Hauptmenü anzuzeigen.

Dies wird dadurch erreicht, dass die Aktionsauswahl durch den Spieler und die Ausführung der Aktion in einer Endlosschleife steht. Eine Endlosschleife hat folgenden Aufbau:

```
while (true){
    ...
}
```

Da die Bedingung in der Schleife immer wahr ist, wird die Schleife nie abgebrochen. Passe die Operation `exploreProperty` entsprechend an.

```
=====
=                MENUE                =
=====
Was wollen sie tun?
(1) Aktuellen Ort anzeigen
(2) Alle möglichen Orte anzeigen
(3) In den Keller
(4) In das Erdgeschoss
(5) In das Dachgeschoss
(0) Programm beenden
```

Lege zunächst die Konstanten `MENU_GOTO_BASEMENT`, `MENU_GOTO_FIRSTFLOOR` und `MENU_GOTO_TOPFLOOR` für die neuen Aktionen an.

- (e) Erweitere in der Operation `doMenuAction` die `switch-case`-Anweisung für die neuen Menüoptionen. Beachte, dass man z. B. nicht vom Dachgeschoss direkt in den Keller gehen kann. Es gibt weitere Einschränkungen, die ebenfalls im Programmcode auszuschließen sind. Informiere den Spieler durch entsprechende Hinweise in der Konsole.

Aufgabe 6 (Erweiterung der Klasse `AdventureGame`):

Nach der Einführung in das *Text-Adventure-Spiel* sollte der Spieler die Möglichkeit erhalten, das Haus im Erdgeschoss zu betreten oder das Spiel zu beenden.

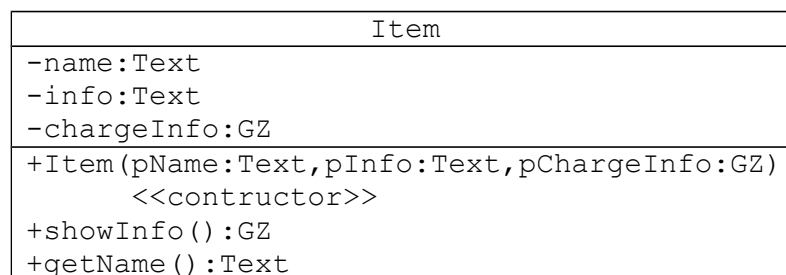
Gebe in der Operation `enter` ein entsprechendes Menü am Bildschirm aus und lese die vom Spieler ausgewählte Aktion von der Konsole in eine Variable `menuActionID` ein. Betrete dann entsprechend der Aktion das Haus (bisheriger Programmcode) oder verabschiede den Spieler.

17 Anhang Text-Adventure-Spiel-Anleitung 2

Das *Text-Adventure-Spiel* ist bisher recht langweilig, da man nur zwischen den verschiedenen Stockwerken hin und her gehen kann. Es wird nun um Rätsel (engl. *mystery*) auf den Stockwerken erweitert. Werden die Rätsel gelöst, erhält der Spieler einen Bonus, ansonsten wird eine Gebühr erhoben. Außerdem wird es bis zu drei verschiedene Gegenstände (engl. *item*) auf einem Stockwerk geben, die einen Hinweis zur Lösung des Rätsels geben. Auch diese Hinweise werden eine Gebühr kosten.

Um die Programmierung möglichst einfach zu gestalten, werden die Rätsel in Form von Multiple-Choice-Fragen mit drei alternativen Antworten gestellt.

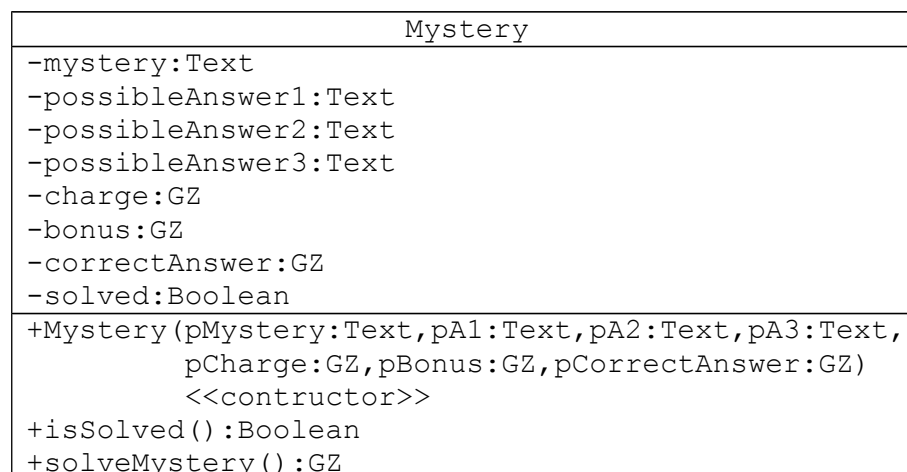
Aufgabe 1 (Programmierung der Klasse *Item*): Die Gegenstände mit den zusätzlichen Informationen (Attribut *info*) zur Lösung der Rätsel des *Text-Adventure-Spiels* haben einen Namen wie z. B. *Kerzenständer* oder *Blumenvase*. Wie hoch die Gebühr für die Information ist, wird im Attribut *chargeInfo* gespeichert. Daraus ergibt sich das sehr einfach zu implementierende UML-Klassendiagramm der Klasse *Item*:



Mit Hilfe der Operation *showInfo* wird der Hinweis im Attribut *info* mit *println* am Bildschirm ausgegeben. Zusätzlich wird die Gebühr für die Information von der Operation zurückgegeben.

Lege die Klasse *Item* in deinem Projekt an und programmiere sie wie oben beschrieben.

Aufgabe 2 (Programmierung der Klasse *Mystery*): Das folgende Klassendiagramm beschreibt die Klasse *Mystery*, in der die Rätsel in Form von Multiple-Choice-Fragen mit drei Antwortalternativen realisiert werden.



- Lege eine Klasse *Mystery* in deinem Projekt an. Programmiere die angegebenen Attribute. Es gibt das Rätsel (engl. *mystery*), die drei Antwortmöglichkeiten (engl. *possible answer*), die Gebühr (engl. *charge*), falls das Rätsel nicht korrekt gelöst wurde, den Bonus (engl. *bonus*), den der Spieler erhält, wenn er die richtige Antwort (engl. *correct answer*) wählt und die Information, ob das Rätsel bereits gelöst (engl. *solved*) wurde.
- Der Konstruktor der Klasse *Mystery* dient dazu, den Attributen der Klasse Startwerte zuzuweisen. Bis auf das letzte Attribut erhalten alle anderen Attribute ihren Startwert über die entsprechenden Paramete-

ter. Programmiere den Konstruktor der Klasse und vergesse nicht, auch das Attribut `solved` korrekt zu initialisieren.

(c) Programmiere die Operation `solveMystery` nach folgenden Angaben:

Um das Rätsel zu lösen, wird der Spieler zunächst in der Konsole darüber informiert, wie hoch der Bonus für eine korrekte Lösung ist und welche Gebühr für eine falsche Antwort anfällt.

Jetzt kannst du das Rätsel lösen.

- Bonus für eine korrekte Antwort: 8 Punkte

- Gebühr bei einer falschen Antwort: 5 Punkte

Anschließend werden das Rätsel und die drei Antwortalternativen angezeigt.

Rätsel:

Im Keller liegt die Zahl 401. Ist dies eine Primzahl? Systematisch teilt man die 401 zunächst durch 2. Bleibt bei der Division ein Rest übrig, so muss man mit der Division von 3 weitermachen. Bis zu welcher Zahl muss man die Division und Überprüfung des Rests fortsetzen, um sicher zu sein, dass es sich bei 401 um eine Primzahl handelt?

(1) bis 401

(2) bis 20

(3) bis 21

Die Antwort 1, 2 oder 3 des Spielers wird mit Hilfe des `Scanner`-Objekts über die Tastatur in eine Variable gelesen. Falls du nicht mehr weißt, wie man Zahlen, die der Benutzer über die Tastatur eingibt, in einer Variablen speichern kann, so

- lese entweder auf der **Lernkarte 04 – Menüauswahl in der Konsole (Tastatureingabe von Zahlen)** nach oder
- schaue dir noch einmal den Programmcode in der Operation `getNextMenuActionByID` der Klasse `Property` an.

Zuletzt muss noch die Antwort des Spielers ausgewertet werden. Die Nummer der richtigen Antwort steht im Attribut `correctAnswer`. Sowohl bei einer richtigen als auch bei einer falschen Antwort wird dem Spieler eine Nachricht am Bildschirm angezeigt und die Gebühr (eine positive ganze Zahl) für die falsche Antwort oder der Bonus für die richtige Antwort in Form einer negativen Zahl (es ist eine negative Gebühr, also etwas Positives) von der Operation `solveMystery` zurückgegeben. Außerdem muss bei einer richtigen Antwort der Wert des Attributs `solved` angepasst werden, da das Rätsel gelöst ist.

(d) Mit Hilfe der Operation `isSolved() : Boolean` wird Auskunft darüber gegeben, ob der Spieler das Rätsel bereits gelöst hat. Diese Information ist im Attribut `solved` zu finden und muss von der Operation zurückgegeben werden.

Aufgabe 3: Da du nun die programmiertechnischen Voraussetzungen für Rätsel, Hilfestellungen zur Lösung und Bewertung in den Klassen `Item` und `Mystery` geschaffen hast, musst du jetzt ein neues, interessantes Rätsel mit spannenden Antwortalternativen erfinden.

Löse die folgenden Aufgaben schriftlich in einem separaten Textdokument.

- Überlege dir eine interessante Fragestellung für ein Rätsel und notiere dir die Frage.
- Erfinde drei Antworten zum Rätsel, so dass nicht auf den ersten Blick klar wird, welches die korrekte Lösung des Rätsels ist.
- Welche Gegenstände könnten Hinweise zur Lösung des Rätsels geben? Welche Informationen können an den Gegenständen hinterlegt werden? Entwickle zu deinem Rätsel bis zu drei Gegenstände mit zusätzlichen Informationen.

Aufgabe 4 (Erweiterung der Floor):

- (a) Jedem Stockwerk im Haus kann nun ein Rätsel und bis zu drei Gegenstände mit Hinweisen zur Lösung des Rätsels hinzugefügt werden. Erweitere deswegen die Klasse `Floor` um die Attribute `mystery`, `item1`, `item2` und `item3`.
- (b) Um beim Erzeugen eines Stockwerks gleich ein Rätsel hinterlegen zu können, wird ein zweiter, neuer Konstruktor in der Klasse `Floor` definiert, der sowohl den Namen des Stockwerks als auch das Rätsel in zwei Parametern übergeben bekommt.

```
Floor(pName:Text, pMystery:Mystery)<<constructor>>
```

Die Gegenstände werden mit Hilfe einer eigenen Operation `addItem` hinzugefügt. Aus diesem Grund gibt es bei der Erzeugung des Stockwerks noch keine Gegenstände. In den Gegenständen (Attribute `item1`, `item2` und `item3`) ist also nichts gespeichert.

Lernzeit: Lese auf der **Lernkarte 05 – null als Standardwert für Attribute mit komplexem Datentyp** nach, wie Attribute mit dem Datentyp einer Klasse (Referenzattribute, Attribute mit komplexem Datentyp) initialisiert werden.

Ergänze nun wie beschrieben den neuen Konstruktor in der Klasse `Floor` und initialisiere die Attribute für die Gegenstände mit dem Standardwert `null`.

Verändere auch den bereits existierenden Konstruktor `Floor(pName:Text)`. Initialisiere hier sowohl das Attribut `mystery` als auch die Attribute `item1`, `item2` und `item3` mit dem Standardwert `null`.

- (c) Mit der Operation `addItem(pName:Text, pInfo:Text, pChargeInfo:GZ)` kann ein Gegenstand dem Stockwerk hinzugefügt werden. Handelt es sich um den ersten Gegenstand, so soll er im Attribut `item1` gespeichert werden, ist es der zweite, so wird er in `item2` geschrieben. Der dritte Gegenstand kommt dann in `item3`. Versucht man weitere Gegenstände hinzuzufügen, so wird eine Fehlermeldung mit `println` am Bildschirm ausgegeben.

Programmiere die Operation `addItem` der Klasse `Floor`. Ob z. B. in `item1` bereits ein Gegenstand gespeichert wurde, kann durch einen Vergleich mit dem Wert `null` ermittelt werden (vgl. **Lernkarte 05 – null als Standardwert für Attribute mit komplexem Datentyp**).

- (d) Befindet sich der Spieler in einem Stockwerk, so möchte er wissen, welche Gegenstände sich dort befinden und wie er einen Gegenstand mit dem entsprechenden Hinweis zur Lösung des Rätsels auswählen kann. Hierbei sind folgende Schritte notwendig:

- Zählen, wie viele Gegenstände es auf dem Stockwerk gibt,
- Ausgabe des nachfolgenden Textes, falls man sich gerade im Erdgeschoss (im Attribut `name` steht der Text *Erdgeschoss*) befindet und es zwei Gegenstände im Erdgeschoss gibt (in den `item1` und `item2` ist kein `null`-Literal gespeichert).

Erdgeschoss: Es gibt 2 Gegenstände

Wähle einen der folgenden Gegenstände als Hilfe aus:

- Ausgabe der Gegenstände in folgender Form (es wird angenommen, es gibt eine *Lampe* und ein *Gemälde* im Erdgeschoss; diese stehen in den Attributen `item1` und `item2`):
 - (1) *Lampe*
 - (2) *Gemälde*
 - (3) *Nichts tun*

Sind überhaupt keine Gegenstände vorhanden, so soll dies ebenfalls durch eine Bildschirmausgabe dem Spieler mitgeteilt werden.

Implementiere in der neuen privaten Operation `-showPossibleItems():GZ` der Klasse `Floor`

die beschriebene Funktionalität. Gebe die entsprechenden Texte am Bildschirm aus. Am Ende wird von der Operation die Anzahl der Gegenstände zurückgegeben.

- (e) Als nächstes wird die Operation `+exploreFloor() : GZ` implementiert. In ihr werden mit der privaten Operation `showPossibleItems` die Gegenstände des Stockwerks angezeigt. In Abhängigkeit davon, ob es überhaupt einen Gegenstand gibt, muss anschließend die Auswahl des Spielers mit einem `Scanner`-Objekt gelesen und in einer Variablen gespeichert werden (vgl. Aufgabe 2 (c)). Anschließend wird der Hinweis des Gegenstands mit der Operation `showInfo` der Klasse `Item` angezeigt. Die Operation `showInfo` gibt die Gebühr zurück, so dass die Operation `exploreFloor` diese entgegennehmen und ebenfalls mit `return` zurückgeben kann.
- (f) In der Operation `+solveMystery() : GZ` der Klasse `Floor` ist nicht viel zu tun, da man die Operation `solveMystery` der Klasse `Mystery` verwenden kann. In dieser wurde bereits in Aufgabe 2c) die Anzeige des Rätsels, das Einlesen der Antwort durch die Spielereingabe einer Zahl über die Tastatur und die Auswertung der Antwort realisiert. Programmiere also die Operation `+solveMystery() : GZ` in der Klasse `Floor`, indem du dem `mystery`-Objekt die Arbeit mit der Operation `solveMystery` der Klasse `Mystery` übergibst. Diese liefert die Gebühr (falsche Antwort; positive Zahl) des Lösungsversuchs des Rätsels oder den Bonus (richtige Antwort; negative Zahl) zurück. Dieser Wert muss dann von der Operation `solveMystery` der Klasse `Floor` zurückgegeben werden. Bevor du die Arbeit beginnst, solltest du natürlich überprüfen, ob es auf dem Stockwerk überhaupt ein Rätsel gibt.

Lernzeit: Falls es dir sonderbar vorkommt, dass in einer Operation eine Operation mit demselben Namen einer anderen Klasse aufgerufen wird, so informiere dich auf der **Lernkarte 06 – Delegation von Aufgaben an Objekte anderer Klassen**.

- (g) Die Operation `+hasMystery() : Boolean` der Klasse `Floor` gibt zurück, ob das Stockwerk überhaupt ein Rätsel besitzt oder nicht. Dies hängt vom Zustand des Attributs `mystery` in der Klasse `Floor` ab. Falls du nicht mehr weißt, wie man herausfinden kann, ob ein Stockwerk ein Rätsel hat, lese die **Lernkarte 05 – null als Standardwert für Attribute mit komplexem Datentyp**.
- (h) Die Operation `+isSolved() : Boolean` der Klasse `Floor` gibt `true` zurück, falls das Stockwerk ein Rätsel besitzt und der Spieler dieses Rätsel während des Spiels bereits gelöst hat oder überhaupt kein Rätsel auf dem Stockwerk existiert. Ist das Rätsel noch nicht gelöst, wird `false` zurückgegeben. Die Aufgabe, herauszubekommen, ob ein vorhandenes Rätsel gelöst ist, wird auch hier an ein anderes Objekt, das `mystery`-Objekt, delegiert (vgl. Aufgabe 4 (f)). Verwende hierfür die öffentliche Operation `isSolved` der Klasse `Mystery`.
- (i) Zusammenfassend sieht das UML-Klassendiagramm der Klasse `Floor` nun folgendermaßen aus:

| Floor |
|--|
| -name:Text -mystery:Mystery -item1:Item -item2:Item -item3:Item |
| +Floor(pName:Text) <<constructor>> +Floor(pName:Text, pMystery:Mystery) <<constructor>> +getName():Text +addItem(pName:Text, pInfo:Text, pChargeInfo:GZ) -showPossibleItems():GZ +exploreFloor():GZ +solveMystery():GZ +hasMystery():Boolean +isSolved():Boolean |

18 Anhang Text-Adventure-Spiel-Anleitung 3

Aufgabe 1 (Erweiterung der Klasse `Property`): Das Rätsel, das in Aufgabe 3 der Anleitung – Teil 2 entwickelt wurde, wird nun im Keller platziert. Da der Keller im Konstruktor der Klasse `Property` erzeugt wird, muss hier weiterprogrammiert werden.

- (j) Lege im Konstruktor der Klasse `Property` eine Variable `mysteryBasement` vom Datentyp `Mystery` an. Erzeuge mit Hilfe deiner Überlegungen aus Aufgabe 3 (Anleitung – Teil 2) ein `Mystery`-Objekt und ordne dieses der Variablen `mysteryBasement` zu.
- (k) Übergebe dieses `Mystery`-Objekt jetzt dem `Floor`-Konstruktor des Attributs `basement` im zweiten Parameter.
- (l) Stelle noch die in Aufgabe 3 (Anleitung – Teil 2) entwickelten Gegenstände in den Keller. Diese können mit Hilfe der Operation `addItem` der Klasse `Floor` dem `basement`-Objekt hinzugefügt werden.

Aufgabe 2 (Erweiterung der Klasse `Property` um die Operation `doActionOnFloor`): Bisher kann der Spieler sich nur zwischen den verschiedenen Stockwerken des Hauses hin und her bewegen. Jetzt wird die Funktionalität erweitert, so dass er die Stockwerke erkunden und die gestellten Rätsel lösen kann. Das Erkunden der Stockwerke bedeutet, dass der Spieler sich die Informationen der Gegenstände anzeigen lässt und somit Tipps zu den Rätseln erhält.

- (a) Füge der Klasse `Property` die private Operation `doActionOnFloor(pFloor:Floor)` hinzu, da es sich sowohl beim Erkunden eines Stockwerks als auch beim Lösen des Rätsels um eine Aktion auf dem Stockwerk handelt.
- (b) Lege in der Klasse `Property` die Konstanten `ACTION_EXPLORE`, `ACTION_SOLVEPROBLEM`, `ACTION_MAINMENU` als ganzzahlige Werte 1, 2 und 3 für die verschiedenen Interaktionsmöglichkeiten auf einem Stockwerk an. Falls du nicht mehr weißt, wie Konstanten angelegt werden, lese noch einmal die **Lernkarte 01 – Konstanten in Java** durch.
- (c) Befindet sich der Spieler gerade im Keller, so wird in der Operation `doActionOnFloor` folgender Text ausgegeben, um den Spieler zu informieren, was er gerade tun kann:

*Sie befinden sich am folgenden Ort: Keller
Was wollen sie tun?
(1) Keller erkunden
(2) Rätsel lösen
(3) Hauptmenü*

Programmiere in der Operation `doActionOnFloor` die Textausgabe. Natürlich soll der Name des aktuellen Stockwerks (im Beispiel Keller) angegeben werden. Verwende zur Ausgabe der Zahlen 1, 2 und 3 die in (b) angelegten Konstanten.

- (d) Speichere die Aktionsauswahl des Spielers über die Tastatur in einer Variablen `actionID`. Falls du nicht mehr weißt, wie man Zahlen von der Tastatur einlesen kann, lese noch einmal die **Lernkarte 03 – Menüauswahl in der Konsole (Tastatureingaben von Zahlen)** durch.
- (e) Zuletzt muss in Abhängigkeit der Aktionsauswahl entweder das Stockwerk erkundet, das Rätsel gelöst oder zum Hauptmenü zurückgekehrt werden.

Für diese Aufgaben hast du bereits Vorarbeit geleistet, da die Funktionalität in der Klasse `Floor` in den Operationen `exploreFloor` (Erkundung des Stockwerks) und `solveMystery` (Lösen des Rätsels) programmiert wurden. Wie du weißt, bieten die Operationen dem Spieler die Möglichkeit, das Stockwerk zu erkunden oder das Rätsel zu lösen und berechnen dafür die Gebühr oder den Bonus für die Hilfe oder die Lösung des Rätsels.

Da drei verschiedene Aktionen möglich sind, ist es sinnvoll, eine Mehrfachauswahl über die Variable `actionID` mit Hilfe der in (b) erstellten Konstanten zu programmieren. In den einzelnen Fällen können

dann für das Stockwerk `pFloor` (es wurde in der Operation übergeben) entweder die Operation `exploreFloor` oder `solveMystery` der Klasse `Floor` aufgerufen werden. Für die Auswahl des Hauptmenüs kann im Moment noch kein Programmcode eingefügt werden.

Lege in der Operation `doActionOnFloor` noch eine neue Variable `charge` an, in der die Gebühr für die Erkundung der Gegenstände oder das fehlerhafte Lösen des Rätsels gespeichert wird. Verwalte in `charge` die Rückgabewerte der Operationen `exploreFloor` und `solveMystery`, da diese zu einem späteren Zeitpunkt auf dem Konto des Spielers gutgeschrieben werden müssen.

Aufgabe 3 (Erweiterung der Klasse `Property`): Sowohl nach einem Stockwerkswechsel als auch im Hauptmenü erhält der Spieler die Möglichkeit, das aktuelle Stockwerk zu erkunden und das Rätsel des Stockwerks zu lösen. Dazu sind die folgenden Erweiterungen notwendig:

- (a) Lege für das Hauptmenü eine weitere Konstante `MENU_EXPLORE_FLOOR` mit dem Wert 6 an.
- (b) Erweitere die Operation `getNextMenuActionByID` um die Aktionsauswahl *Aktuelles Stockwerk erkunden*. Verwende zur Anzeige der Menüoption die Konstante `MENU_EXPLORE_FLOOR`, so dass die Erkundung des aktuellen Stockwerks durch die 6 ausgewählt werden kann.
- (c) Erweitere die Mehrfachauswahl in der Operation `doMenuAction` um die Erkundung des aktuellen Stockwerks. Verwende auch hier die Konstante `MENU_EXPLORE_FLOOR`.
- (d) Suche nun in der Operation `doMenuAction` alle Stellen, an denen entweder ein Stockwerkswechsel oder die Erkundung des aktuellen Stockwerks stattfindet und rufe dort die in Aufgabe 2 erstellte private Operation `doActionOnFloor` mit dem entsprechenden Stockwerk auf.
- (e) Teste dein *Text-Adventure-Spiel*. Wie du siehst, läuft es noch nicht optimal.

Aufgabe 4 (Erweiterung der Klasse `Property`): Im *Text-Adventure-Spiel* wäre es schön, wenn man mehrere Gegenstände nacheinander anschauen könnte, um schneller an die verschiedenen Hilfestellungen zu gelangen. Bis im Menü

```
Sie befinden sich am folgenden Ort: Keller
Was willst Du tun?
(1) Keller erkunden
(2) Rätsel lösen
(3) Hauptmenü
```

die Option *Hauptmenü* ausgewählt wurde, soll nun nach jeder Aktion dem Spieler immer wieder das Menü angeboten werden. Dazu muss die Operation `doActionOnFloor` um folgende Schleife erweitert werden (Programmcode in Ausschnitten):

```
boolean action = true;
while ( action==true ){
    switch (actionID){
        case ACTION_EXPLORE:
            ...
            break;
        case ACTION_MAINMENU:
            action=false;
            break;
    }
}
```

Nur im Fall der Auswahl der Option *Hauptmenü* wird die Variable `action` auf `false` gesetzt und somit die Schleife beendet.

Aufgabe 5 (Programmierung der Klasse `Player`): Das *Text-Adventure-Spiel* ist bereits spielbar, aber der Spieler kann noch nicht gewinnen oder verlieren. Er läuft auf dem Anwesen herum, schaut sich die Hilfestellungen der Gegenstände an und löst die Rätsel. Was das Spielziel ist, bleibt jedoch offen. Dies wird sich nun ändern, denn das *Text-Adventure-Spiel* wird gewonnen, falls der Spieler alle Rätsel des Anwesens gelöst hat, oder verloren, falls der Spieler keine Punkte mehr auf seinem Konto hat.

In der Anleitung – Teil 2 wurde bereits vorbereitet, dass die Hilfestellungen und falschen Antworten beim Lösen des Rätsels Gebühren verursachen und die Lösung eines Rätsels einen Bonus einbringt. Um dies Guthaben auf einem Konto (engl. *bank balance*) zu verwalten, wird nun die letzte Klasse des Projekts, die Klasse `Player` mit den Attributen `name` und `bankBalance` angelegt. Beachte zur Lösung der Aufgabe das UML-Klassendiagramm der Klasse `Player` unten.

- Lege die Klasse `Player` mit den Attributen `name` und `bankBalance` an.
- Erstelle den Konstruktor der Klasse, mit dem ein `Player`-Objekt erzeugt wird. Der Name des Spielers und der Kontostand werden dabei in den Parametern übergeben.
- Implementiere die beiden `get`-Operationen der Klasse.
- Mit Hilfe der Operation `getInfo` werden die Informationen über den Spieler in Textform zurückgeben. Es könnte z. B. der Text „*Meier hat noch folgendes Guthaben: 34*“ zurückgegeben werden, falls der Spieler *Meier* heißt und noch *34 Punkte* auf seinem Konto hat.
- Implementiere die Operation `changeBalance (pAmount:GZ)`, die den Kontostand des Spielers um `pAmount` verändert.

| Player |
|---|
| -name:Text -bankBalance:GZ |
| +Player(pName:Text,pBalance:GZ) <<constructor>> +getName():Text +getBalance():GZ +getInfo():Text +changeBalance(pAmount:GZ) |

Aufgabe 6 (Erweiterung der Klasse `Property`):

- Lege ein Attribut `player` der Klasse `Property` in der Klasse `Property` an, damit der Spieler in das *Text-Adventure-Spiel* integriert wird.
- Im Konstruktor der Klasse `Property` muss das `Player`-Objekt erzeugt werden. Dazu soll der Name des Spielers von der Konsole eingelesen werden. Dies funktioniert fast wie bei Zahlen:

```
Scanner input = new Scanner(System.in);
String name = input.next();
```

Statt mit der Operation `nextInt()` wie bei Zahlen, liest man den Text mit der Operation `next()` vom `Scanner` ein. In `name` steht nun der Name des Spielers. Jetzt kann der Konstruktor für das Attribut `player` aufgerufen werden.

Erzeuge das `Player`-Objekt im Attribut `player` der Klasse `Property`. Falls du nicht mehr weißt, wie es geht, lese es auf der **Lernkarte 02 – Attribute im Konstruktor initialisieren** nach. Wie du jetzt langsam erkennen solltest, wiederholt sich jede Programmier Technik immer wieder.

Aufgabe 7 (Erweiterung der Klasse `Property`): Nun ist alles für das Programmierfinale vorbereitet. Es kann beginnen.

- programmiere in der Klasse `Property` die private Operation `-checkGameOver()`. Das *Text-Adventure-Spiel* ist zu Ende, falls das Konto des Spielers leer ist oder der Spieler alle Rätsel gelöst hat.

Um festzustellen, ob das Konto des Spielers leer ist, kann das Attribut `player` nach dem Kontostand gefragt werden. Ist dieser negativ, so ist das Spiel zu Ende. Gebe in diesem Fall einen entsprechenden Text in der Konsole aus und beende das *Text-Adventure-Spiel* (vgl. Anleitung – Teil 1, Aufgabe 5 (b)).

Die Information, ob alle Rätsel gelöst wurden, muss man sich etwas zusammensuchen. Um herauszufinden, wie viele Rätsel es überhaupt gibt, muss jedes einzelne Stockwerk mit der Operation `hasMystery` der Klasse `Floor` befragt werden, ob sie ein Rätsel hat. Außerdem kann jedes Stockwerk befragt werden, ob ihr Rätsel bereits gelöst ist (Operation `isSolved` der Klasse `Floor`). Aus diesen Informationen lässt sich dann sehr leicht feststellen, ob alle Rätsel gelöst sind. Ist dies der Fall, gebe ebenfalls einen entsprechenden Text in der Konsole aus und beende das *Text-Adventure-Spiel*.

- Der Kontostand des Spielers kann sich nur ändern, falls er das Stockwerk erkundet (Operation `exploreFloor` der Klasse `Floor`) oder ein Rätsel löst (Operation `solveMystery` der Klasse `Floor`). Das Erkunden des Stockwerks oder Lösen des Rätsels erfolgt in der Operation `doActionOnFloor` der Klasse `Property`. Suche die entsprechenden Stellen im Programmcode. In der Variablen `charge` sollte bereits die Gebühr oder der Bonus der Aktion gespeichert sein (vgl. Aufgabe 2 (e)).

Verwende nun die Operation `changeBalance` aus der Klasse `Player`, um den Kontostand des Attributs `player` mit der gerade berechneten Gebühr oder dem Bonus in der Variablen `charge` zu aktualisieren.

Damit du später auch sehen kannst, wie sich der Kontostand ändert, gebe mit Hilfe der Operation `getInfo` der Klasse `Player` an einer geeigneten Stelle die Informationen über den Spieler in der Konsole aus. Diese beinhalten auch den Kontostand.

- Teste am Ende der Endlos-Schleife in der Operation `doActionOnFloor`, ob das Spiel bereits zu Ende ist. Verwende dazu die in (a) programmierte private Operation `checkGameOver`. Jetzt hast du alles geschafft und kannst dein Programm testen.

Aufgabe 8 (Mögliche Erweiterungen):

- Kommentare im Programmcode
- Javadoc-Kommentare
- Redesign, um die Gegenstände (Objekte der Klasse `Item`) in einem Array zu speichern
- Ausgabe des Text-Adventure-Spiel in einem separaten Textfenster
- Fehlerhafte Benutzereingaben abfangen
- ...