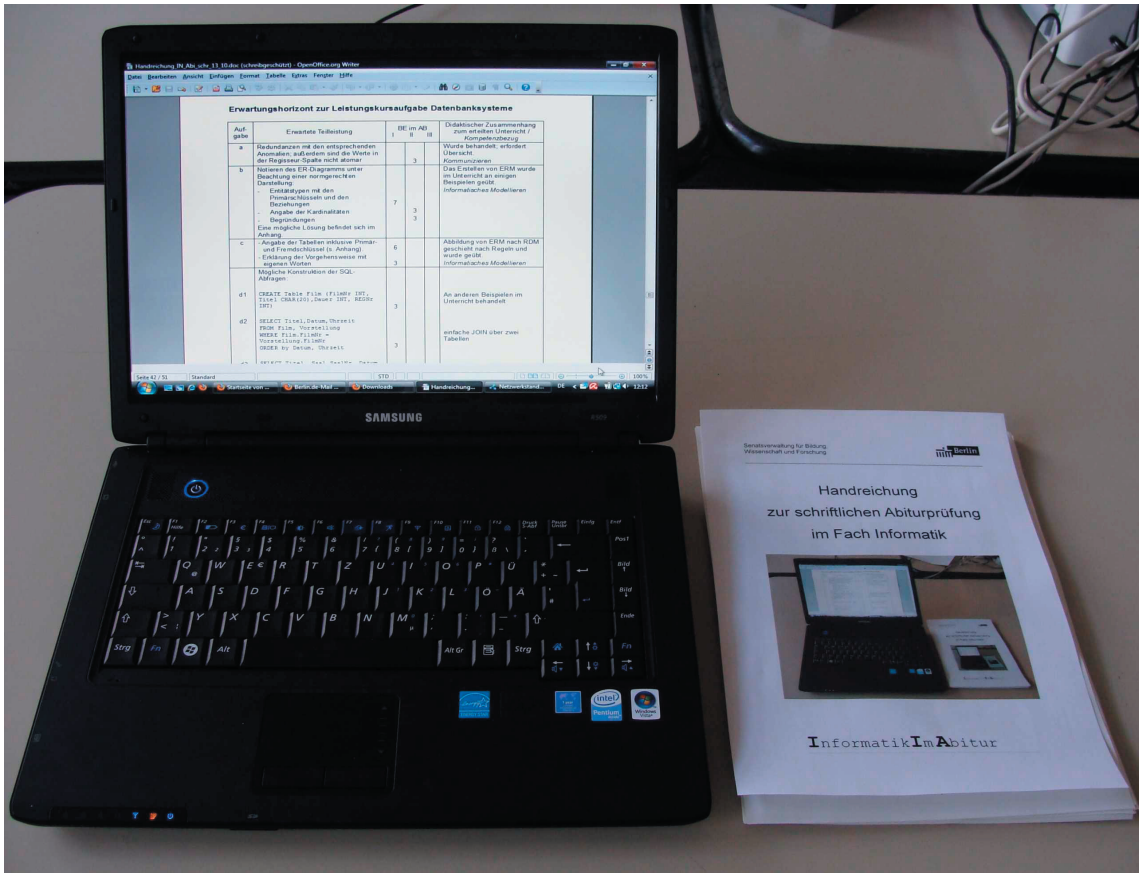
Berlin

Senatsverwaltung für Bildung,
Wissenschaft und Forschung

Bildung für Berlin



Handreichung
zur schriftlichen Abiturprüfung
im Fach Informatik

Impressum

Herausgeber

Senatsverwaltung für Bildung,
Wissenschaft und Forschung,
Beuthstraße 6 – 8
10117 Berlin

www.berlin.de/sen/bwf

Verantwortlich
Christian Bänsch
Referat VI A: Allgemein bildende Unterrichtsfächer

Redaktion
Christian Bänsch

Autoren
Christian Steinbrucker, Alexander Dietz, Annemarie Lotze

Druck
Oktoberdruck

Auflage
1.000
November 2009

Inhaltsverzeichnis

1. Allgemeine Hinweise

Informatik ist in Berlin dezentrales Abiturfach. Die hier vorliegende Handreichung soll den Lehrerinnen und Lehrern bei der Erstellung der Unterlagen für die schriftliche Informatik-Prüfung helfen. Die Empfehlungen und Hinweise auf LK- und GK-Niveau sollen auch dazu dienen, dass die Unterschiede der Aufgabenvorschläge hinsichtlich Anspruchsniveau, Umfang und Qualität zwischen den prüfenden Schulen nicht unnötig groß werden.

Die Handreichung ersetzt nicht die Lektüre der Abiturrichtlinien (EPA, AV Prüfungen). Der Umfang ist durch die enthaltenen Aufgabenbeispiele recht groß geworden. Auch für die neueren Themenfelder „Informatik und Gesellschaft“ und „Sprachen und Automaten“ sind Beispielaufgaben enthalten. Alle Beispielaufgaben sollen Anregungen für eigene Aufgabenstellungen liefern; sie können natürlich nicht in dieser Form als Abituraufgabe eingereicht werden.

1.1 Die wichtigsten formalen Vorgaben

Neben der ständigen fachlichen sowie methodisch-didaktischen Weiterentwicklung des Schulfaches Informatik gab es in den vergangenen Jahren auch mehrere neue und veränderte formale Vorgaben. Beginnend mit dem „neuen“ Rahmenlehrplan im Schuljahr 2006/07 liegen bisher vier Fachbriefe vor, und in der Abiturprüfung 2010 gelten wiederum veränderte Ausführungsvorschriften über schulische Prüfungen (AV Prüfungen nebst Fachanlage 3b).

Wir verwenden folgende Begriffskonventionen: Es sind zum Abitur zwei **Vorschläge** einzureichen, von denen jeder aus zwei oder drei **Aufgaben** besteht. Jede Aufgabe gliedert sich in mehrere **Teilaufgaben**. Die Lösungen der Teilaufgaben setzen sich ggf. aus mehreren **Teilleistungen** zusammen, denen im Erwartungshorizont **Bewertungseinheiten** zugewiesen werden.

Die Gesamtzahl der Bewertungseinheiten eines Vorschlags soll die Feinheit der Prozentskala ausnutzen und dem gemäß ca. 90 bis 120 betragen. Weitere Unterteilungen (halbe BE) sind nicht vorgesehen, auch nicht im Gutachten der schriftlichen Arbeit. Im Erwartungshorizont ist insbesondere in den Anforderungsbereichen II und III der Bezug zum erteilten Unterricht auszuweisen.

Für jede Aufgabe muss deren Gewichtung den Prüflingen bekannt gemacht werden und auf dem Text des Vorschlages, den die Prüflinge erhalten, vermerkt sein. Bei umfangreichen Aufgaben empfiehlt es sich, auch die Gewichtung der Teilaufgaben zu vermerken.

In Berlin gibt es für das Fach Informatik keine Operatorenlisten. Auch die EPA Informatik nennt keine Operatoren, wohl aber die EPA Berufliche Informatik, an der man sich ggf. orientieren kann.

Eine wichtige Vorgabe für die Abiturvorschläge besteht darin, dass beide Vorschläge sich auf unterschiedliche Schwerpunkte des Schwerpunktkurshalbjahres beziehen sollen[1]. Das zweite Jahr der Qualifikationsstufe bietet nun eine deutlich höhere Anzahl von möglichen Schwerpunktthemen als das erste, in dem es lediglich die Bereiche „Datenbanken und Softwareentwicklung I“ und „Datenbanken und Softwareentwicklung II“ gibt. Wenn das 1. oder das 2. Semester Schwerpunktkurshalbjahr werden soll, dann ist frühzeitig bei der Planung des Unterrichts darauf zu achten, dass hinreichend verschiedene Schwerpunkte im Unterricht gesetzt werden.[2]

[1] (vgl. AV Prüfungen, Anlage 3b, Nr. 2.1 Absatz 2)
[2] Siehe Hinweise und Beispiele in Kapitel 4

1.2 Zur Unterscheidung von Grund- und Leistungskurs

Die allgemeinen Merkmale der Unterscheidung zwischen Grundkurs- und Leistungskursaufgaben werden in der EPA Informatik dargestellt[3] und hier im Abschnitt 2 an einem ausformulierten Aufgabenbeispiel verdeutlicht. Das unterschiedliche Anspruchsniveau der beiden Kursarten muss sowohl im Umfang als auch im Schwierigkeitsgrad der Aufgaben beachtet werden. Der höhere Umfang im Leistungskurs wird durch die längere Bearbeitungszeit erforderlich: Grundkurs: 180 Minuten, Leistungskurs: 240 Minuten.

Nicht alle Aufgabenteile des Leistungskurses sollen ein höheres Niveau aufweisen als im Grundkurs. Vielmehr muss es auch im Leistungskurs möglich sein, einfache Anforderungen im AB I zu erreichen. Diese unterscheiden sich wenig von den Anforderungen im Grundkurs. Werden Grund- und Leistungskurse zeitgleich an einer Schule im Abitur geprüft, kann ein identischer Aufgabenkern bestehen.

1.3 Was macht einen guten Vorschlag aus?

Eine Prüfungsaufgabe in der schriftlichen Abiturprüfung soll fachliche und methodische Kompetenzen in möglichst großer Breite an geeigneten fachlichen Inhalten überprüfen (vgl. EPA Informatik, Kapitel 1, S. 4-6).

Ein der Aufgabe zugrunde liegendes Themenfeld wird zunächst charakterisiert durch **wesentliche** und **ergänzende Inhalte**. Darüber hinaus ergeben sich unterschiedliche mögliche **Kontexte** und Möglichkeiten zum **Kompetenzerwerb.**

Zur Bearbeitung der Aufgabe ist eine entsprechende Vielfalt von Methoden notwendig[4].

In einer schriftlichen Prüfungsaufgabe ist der Nachweis möglichst vieler Kompetenzen gewünscht, z. B. sowohl Entwurfsarbeit durchzuführen als auch vorhandenes Entwurfsmaterial zu analysieren. Ebenso soll eine Programmierung (auf Codeebene!) erstellt

werden, wie es auch zusätzlich sinnvoll ist, gegebenen Code analysieren zu lassen.

---

[3] Aus den EPA Informatik, Kapitel 1.3, S. 6:

In den Abituraufgaben unterscheiden sich Grundkurs- und Leistungskursfach insbesondere durch
- den Grad der Vorstrukturierung bei der Problembearbeitung,
- die Offenheit der Aufgabenstellung,
- die Anforderungen an Selbstständigkeit bei der Bearbeitung der Aufgaben,
- den Umfang und die Art der bereitgestellten Hilfsmitteln und Informationen,
- den Grad der Abstraktion der zu behandelnden Inhalte und Begriffe,
- den Grad der Formalisierung von Sachverhalten und Darstellungen,
- den Grad der Komplexität der Problemstellungen,
- die Vielfältigkeit der verwendeten Methoden und die Vielfalt an Untersuchungs- und Lösungsstrategien

[4] Aus den EPA Informatik, Kapitel 3.2, S. 12:

Folgende Arten von Aufgaben oder Teilaufgaben können u. a. vorkommen, wobei teilweise Überschneidungen möglich sind:
- Modellierung einer konkreten Problemstellung
- Implementierung einer konkreten bereits modellierten Problemstellung
- Darstellung, Erläuterung und sachgerechte Anwendung von informatischen Begriffen und Verfahren
- Untersuchung und Beschreibung vorgegebener informatischer Konstrukte
- Visualisierung von Sachverhalten und informatischen Zusammenhängen
- Interpretation, Vergleich und Bewertung von Daten, Ergebnissen, Lösungswegen oder Verfahren
- Übertragung von Ergebnissen auf einen anderen Sachverhalt

	Entwurf / Modellierung	Programmierung / Implementierung
Analyse	z. B. gegebene • Klassendiagramme, • ERM-Diagramme, • Struktogramme, • Zustandsdiagramme, • freier Text	z. B. • gegebener Quelltext, • Schreibtischtest
Konstruktion	• Erzeugung, • Änderung oder • Ergänzung von Diagrammen (s. o.)	z. B. • Quelltextbearbeitung, • Quelltextergänzung, • Quelltextschaffung

Die bereits in 1.1 genannte Vorgabe der Zuordnung eines Themas zum Schwerpunktkurshalbjahr sei hier an einem Beispiel erläutert. Angenommen das Schwerpunktsemester sei das zweite Kurshalbjahr. Die Schüler erwerben Kompetenzen zur Analyse und auch zur Modellierung umfangreicherer Datenbanken und sie stellen die Ergebnisse in ER-Modellen dar bzw. interpretieren vorgegebene ER-Modelle. Im Vorschlag A sei eine Datenbank zu analysieren, deren Struktur nur aus 3 Entitäten und einem trinären Relationship besteht. In dem Vorschlag B sei ein Szenario gegeben, dessen Modellierung zu einem Datenbankentwurf führt, der ebenfalls aus 3 Entitäten und einem trinären Relationship besteht.

Nun werden durch die beiden Vorschläge zwar unterschiedliche Kompetenzbereiche angesprochen, was die Vorgehensweise der Analyse und des Entwurfes angeht. In inhaltlicher Dimension verkürzt sich der Unterschied aber erheblich. Daraus folgt, dass es zur Unterscheidung der Schwerpunkte entscheidend darauf ankommen wird, auch die weiteren inhaltlichen Unterschiede dieser Aufgaben klar und umfangreich zu machen.

1.4 Überblick zu den Beispielaufgaben

Die oben genannten allgemeinen Hinweise werden im Folgenden an drei Beispielaufgaben erläutert:

Das erste Beispiel demonstriert die Beachtung der formalen Vorgaben bei der Erstellung eines **Gesamtvorschlags für einen Grundkurs**, z. B. konkrete Erläuterungen zu den Bewertungseinheiten, die Semesterzuordnungen und die Gewichtung aller Teilaufgaben untereinander. Die Aufgaben stammen aus den Themenfeldern „Objektorientierte Softwareentwicklung“ und „Sprachen und Automaten“.

Im zweiten Beispiel werden die Merkmale der **Unterscheidung zwischen Grundkurs- und Leistungskursaufgaben** an einer ausformulierten Aufgabe aus dem Themenfeld „Datenbanken“ verdeutlicht.

Merkmale eines guten Vorschlags werden im dritten Beispiel an Aufgaben aus den Themenfeldern „Informatik, Mensch und Gesellschaft“ (im GK) und „Applikative Programmierung“ (im LK) illustriert.

2. Ein kompletter Grundkursvorschlag

2.1 Vorbemerkungen

Der vollständige Vorschlag für einen Grundkurs Informatik besteht aus zwei Aufgaben. Den Schwerpunkt bildet das Lerngebiet der „Softwareentwicklung“ aus dem 2. Informatikjahr. Die zweite Aufgabe gehört zum Lerngebiet „Automaten und Sprachen“ aus dem 3. Informatikjahr. Im Anschluss an die Aufgabenstellung zeigt der Erwartungshorizont die Zuordnungen der Bewertungseinheiten. Dabei wird die Gewichtung zwischen den Teilaufgaben und den beiden Aufgaben des Vorschlags erklärt. Der didaktische Kommentar dokumentiert exemplarisch Bezüge zum notwendigen Unterricht. Die Implementierungsanteile der Teilaufgaben 1.2 bzw. 1.4 könnten auch am Rechner bearbeitet werden. Die Teilaufgabe 1.4 ist auch dann vollständig lösbar, wenn die Implementierung der Teilaufgabe 1.2 fehlt.

Ausgehend von typischen Merkmalen der Objektorientierung bieten Aufgaben des Themengebietes „Softwareentwicklung“ vielfältige Möglichkeiten. Allerdings ist darauf zu achten, dass fachinhaltlich nicht Probleme ausgewählt werden, deren tatsächliche softwaretechnische Realisation besser mit Datenbanken gelöst wird. Hier sind es häufig Listen und Listenoperationen, die zur Verarbeitung größerer Datenmengen dienen sollen. Ohne Zweifel ist das Verständnis des Listenbegriffs ein zutreffendes Thema dieses Kurses. Daher werden Listen häufig im Unterricht softwaretechnisch realisiert. Die Abgrenzung zwischen allgemeinem OOA-OOD-OOP-Zyklus und Listenbasierten Programmen und Datenbanken ist sicher zu stellen. Daher wird im vorliegenden Fall eine Aufgabe gezeigt, die einem anderen geeigneten und weit verbreiteten softwaretechnischen Thema entstammt: ein Spiel.

Der Rahmenplan Informatik, Kapitel 3.2 *Abschlussorientierte Standards, Objektorientierte Modellierung und Programmierung*, wird in dieser Aufgabe interpretiert als die Kompetenz zur Anwendung der Klassenbeziehungen Assoziation, Aggregation und Vererbung. Die Anwendung schließt dabei nicht nur den Entwurf, sondern auch die Analyse und die Konstruktion ein.

Die Ausprägung in einem Glücksspiel eröffnet die Möglichkeit der Vernetzung mit dem Datenschutz und rechtlichen Fragen. Zunächst spielt hier genau ein Spieler gegen einen Anbieter mit der Glücksspiel-typischen Regel: der Anbieter gewinnt langfristig. Der Schwierigkeitsgrad der Teilaufgaben nimmt in der Folge zunächst zu, um nach den programmierrelevanten Teilen zu vernetzten Frageteilen überzugehen. Um den Einstieg in die Aufgabe sicher zu stellen, soll zunächst der gegebene Code analysiert werden und Ergebnisse werden in einem vorbereiteten Klassendiagramm vervollständigt.

Es schließt sich die Fragestellung nach einer (abstrakten) Klasse *Person* als Vorfahr der beiden gegebenen Fachklassen *Anbieter* und *Spieler* an. Diese Teilaufgabe erfordert im Entwurf klares Verständnis des Vererbungsprinzips. In der gefragten anschließenden Implementierung dieser neuen abstrakten Klasse werden bewusst keine algorithmischen Probleme gelöst. Das Verlagern gegebenen Codes an die richtige Stelle, das Entwerfen des neuen Konstruktors sowie die Verlagerung der Assoziation zum Vorfahren bilden den Kern dieser Teilaufgabe.

Bevor der Klassenentwurf durch eine ausbauende Teilaufgabe untersucht wird, stellt ein Schreibtischtest weiteres Verständnis der Thematik sicher. Dieser für die weitere Bearbeitung kaum relevante Zwischenschritt ist eine häufig im Unterricht anzutreffende Analysetechnik. Das vorbereitete Protokollblatt erleichtert den Zugang.

Im Weiteren wird die Spielregel ergänzt, indem der Anbieter nun mit drei Würfeln würfelt. Dies führt im Entwurf zu der mehrfachen Instanzierung der Klasse *Wuerfel*. Außerdem muss die neue Spielregel in den Methoden der Klassen *Anbieter*, *Spieler* und *Spielbrett* umgesetzt werden. Dies bedeutet Eingriffe in die Algorithmik. Die Klassen *Wuerfel* und *Person* bleiben unverändert. An dieser Stelle enden die engen Fragen zu Analyse und Konstruktion. Mit der

anschießenden, sehr offenen Frage, wie sich der Programmentwurf verändert, wenn „Chuck A Luck“ mit zwei Spielern gespielt wird, gibt der Prüfling zu erkennen, inwieweit er eigene Transferleistungen entwickeln kann. Zunächst wird die zweite Instanz der Klasse *Spieler* zu erwarten sein. Damit verbunden ist die Eigenschaft eines eigenen Kontos jedes Spielers. Die Spielreihenfolge und das „Sprengen“ des Anbieterkontos markieren die zu erwartenden Antworten. Diese Teilaufgabe ist, wie auch die letzte Teilaufgabe danach, ausschließlich in eigenen Worten zu lösen, was auch methodisch einen Unterschied zu den vorangegangenen Teilaufgaben zeigt.

Die schon benannte Vernetzung mit dem Lernbereich „Informatik und Gesellschaft“ beendet die erste Aufgabe dieses Vorschlags, indem nach der Zulässigkeit gefragt wird, das Spiel auf der Homepage der Schule in einer Online-Version mit realen Geldeinsätzen über Kreditkarten anzubieten. Der Gewinnabfluss an den Förderverein der Schule berührt den Lebensbereich der Schüler noch hinreichend, Onlinespiele jedenfalls gehören dazu. Es sind zwei Antwortbereiche zu erwarten, einerseits die Überlegung zum Glücksspiel, das mit der Gewinnerwartung erkennbar ausgewiesen wird und andererseits die Frage der Datenhaltung von realen Geldeinsätzen mit Hilfe einer Software, die von Schülern erzeugt wird.

Die Modellierung der Software zum Spiel „Chuck A Luck“ kann ganz unterschiedlich erfolgen: in 25 Kursen wären vielleicht 25 verschiedene Entwürfe anzutreffen. Für den dargestellten Entwurf und die anschließende Programmierung waren folgende Gründe entscheidend:

Die verschiedenen Arten der Klassenbeziehungen sollen verwendet werden (Aggregation, Assoziation, Vererbung). Die Anzahl der Objektinstanzen ist nicht nur gleich 1, sondern einmal auch gleich 3 und es wird auch eine abstrakte Klasse verwendet. Über alles das hinaus soll die Umsetzung für die in Berlin verbreiteten objektorientierten Programmiersprachen möglichst gleich sein (DELPHI, JAVA, PYTHON). Diese Aufgabe wird allein durch Bearbeitung der Fachklassen (MVC) gelöst. Bewusst wurde auf jegliche Benutzerschnittstelle verzichtet.

Die zweite Aufgabe dieses Vorschlages stammt aus dem Themenfeld „Sprachen und Automaten“. Die Problemstellung ist im Kern nicht neu[5]. Zunächst soll in der Teilaufgabe 2.1 der Aufbau und die Arbeitsweise eines endlichen Automaten verbal beschrieben werden. Das geschieht unabhängig von der im Folgenden eröffneten Problemstellung, die sich durch die weitere Aufgabenstellung zieht. Eine übliche Darstellungsform eines endlichen Automaten ist das Zustandsdiagramm. Es enthält die Zustände und die möglichen Übergänge in Abhängigkeit von der Eingabe. Ein solches Diagramm soll in der Aufgabe 2.2 ausgehend vom SOS-Morse-Code entwickelt werden. Die folgenden Testdaten schärfen in der Teilaufgabe 2.3 die Analyse des gegebenen Automaten. Die beiden abschließenden Teilaufgaben untersuchen den Morse-Code anhand konkreten Materials im Kontext „Codierung“. Zusammenfassend ist zu bemerken, dass auch diese Aufgabe, mit Ausnahme des Zustandsdiagramms, weitgehend verbal gelöst wird, was den Fachintentionen gut entspricht (siehe Kap. 4).

Der Erwartungshorizont dieses Vorschlags ist hinsichtlich der verschiedenen Kriterien einzuhalten: Die Summen der prozentualen Anteile der Aufgaben und der Anforderungsbereiche. Die Übersichtstabelle am Schluss ist obligatorisch. Darüber hinaus müssen weitere formale Bedingungen eingehalten werden, die in dieser Handreichung beispielhaft untersucht werden, aber nicht im Erwartungshorizont dokumentiert werden müssen:

	AV Prüfungen	Beispiel in der Handreichung
Mindestumfang jeder Aufgabe	20 %	38 %
Schwerpunktkurshalbjahr	51 % … 75 %	62 %
Summe der Bewertungseinheiten	90 … 125	100
maximale Bündelung	10 %	10 %

[5] Eckart Modrow, Automaten Schaltwerke Sprachen, Dümmler; 1988

2.2 Aufgabenvorschlag Grundkurs

Aufgabe 1: Softwareentwicklung (62 %)

„Chuck A Luck“ ist ein verbreitetes Glücksspiel[6], das verschiedene Versionen kennt: bspw. variiert die Anzahl der Würfel und die Anzahl der Spieler.

1.1 Es gilt folgende einfache Regel: Der Spieler setzt 1 € auf eine Augenzahl, dann wirft der Anbieter (der Bankhalter) einen Würfel. Zeigt der Würfel die gesetzte Augenzahl, erhält der Spieler seinen Einsatz und 3 € Gewinn vom Anbieter des Spiels. Der dokumentierte „Chuck A Luck“-Prototyp kennt genau einen Würfel.

In der Anlage A ist ein unvollständiges Klassendiagramm, in der Anlage B ist der zugehörige Programmcode gegeben. Vervollständigen Sie das Klassendiagramm mit den Klassen `Spieler`, `Anbieter`, `Wuerfel` und `Spielbrett` in Anlage A hinsichtlich der fehlenden Beziehungen. Dokumentieren Sie Ihren Lösungsweg durch Angabe und Kommentierung der relevanten Klassen-Codezeilen.

1.2 Fassen Sie die Gemeinsamkeiten zwischen Anbieter und Spieler in einer neuen Klasse zusammen. Zeichnen Sie das neue Klassendiagramm mit den veränderten Beziehungen.

Geben Sie Attribute und Methoden nur für diese drei Klassen an.
Implementieren Sie diese drei Klassen.

1.3 Führen Sie einen Schreibtischtest mit textlicher Angabe des Laufzeitverhaltens gemäß folgender Vorgabe durch. Die vom Spieler gesetzte Augenzahl ist 6 und

Fall a) Es wird eine 5 gewürfelt.

Fall b) Es wird eine 6 gewürfelt.

Die Initialisierung des Spiels erfolgt in der Klasse Steuerung (siehe Anlage B).

Verwenden Sie hierzu das Blatt in der Anlage C.

Beziehen Sie sich bei der Bearbeitung auf Ihre Lösung aus der Teilaufgabe 1.2. Falls Sie diese Aufgabe nicht gelöst haben, können Sie sich auf den Quellcode aus Teilaufgabe 1.1 beziehen.

[6] Nach einhelliger Auffassung liegt ein Glücksspiel vor, wenn die Entscheidung über Gewinn oder Verlust des Spiels nach den Spielbedingungen nicht wesentlich von den Fähigkeiten und den Kenntnissen des Spielers abhängt, sondern allein oder hauptsächlich vom Zufall.

1.4 Es gelte nun die bekannte „Chuck A Luck“-Regel:

„Die Spieler tätigen ihre Einsätze auf einem Tableau[7] mit den Zahlen Eins bis Sechs, dann wirft der Bankhalter drei Würfel. Zeigt ein Würfel die gesetzte Augenzahl, gewinnt ein Spieler einfach; zeigen zwei Würfel die gesetzte Augenzahl, gewinnt der Spieler den doppelten Einsatz; zeigen alle drei Würfel die gesetzte Augenzahl, so gewinnt der Spieler den dreifachen Einsatz; zeigt kein Würfel die gesetzte Augenzahl, so ist der Einsatz verloren. Der Bankvorteil beträgt bei diesem Spiel 7,9 %.“

Quelle: http://de.wikipedia.org/wiki/Chuck_a_Luck

Gehen Sie bitte bei folgender Spielsituation von dieser Regel aus. Es spielt ein einzelner Spieler mit drei Würfeln, beim Kontostand des Spielers oder des Anbieters unter Null ist das Spiel beendet:

Gesetzte Augenzahl	Einsatz	Würfel 1	Würfel 2	Würfel 3	Gewinn/Verlust
2	1	3	5	6	-1
2	1	2	5	6	+1
2	1	5	2	2	2
2	1	2	2	2	3
6	3	6	5	1	3
6	3	6	6	5	6
6	3	5	2	2	-3

Passen Sie Ihre Implementierung an den Stellen an, wo Eigenschaften oder Methoden verändert werden müssen (drei Würfel, variabler Einsatz).

1.5 Beschreiben Sie mit eigenen Worten, wie sich der Programmentwurf verändert, wenn „Chuck A Luck“ mit 2 Spielern gespielt wird!

1.6 Ein Informatikkurs hat das Spiel implementiert und möchte auf der Homepage der Schule eine Online-Version mit realen Geldeinsätzen über Kreditkarten anbieten. Der Anbieter gewinnt im Verlauf einer längeren Spielphase in der verwendeten Version 7,9 % des Einsatzes. Dieser Gewinn fließt an den Förderverein der Schule.
Erläutern Sie die Zulässigkeit dieses Vorhabens.

Teilaufgabe	1.1	1.2	1.3	1.4	1.5	1.6
Anteil an der Gesamtleistung	10 %	16 %	12 %	10 %	6 %	8 %

[7] hier: Spielbrett

Aufgabe 2: Sprachen und Automaten (38 %)

Bis zur Einführung eines internationalen Satellitengestützten Seenot-Funksystems im Jahr 1999 war fast 100 Jahre lang als internationales Notrufzeichen der SOS-Morse-Code (• • • – – – • • •) festgelegt. Die entsprechenden Notruffrequenzen mussten regelmäßig abgehört und auf das Auftreten eines Notrufes überprüft werden. Nun soll ein Automat diese Aufgabe übernehmen. Dazu werden folgende Festlegungen getroffen:

- ein Punkt symbolisiert ein kurzes Morsesignal
- ein Strich symbolisiert ein langes Morsesignal
- ein Buchstabe wird durch eine festgelegte Kombination von Punkten und Strichen symbolisiert
- die Codes aufeinander folgender Buchstaben werden durch ein Trennsymbol "W" getrennt
- der Notrufcode • • • – – – • • • enthält kein Trennsymbol, er besteht also aus 9 aufeinander folgenden kurzen bzw. langen Morsesignalen. Damit ein Schiffskoch, der bei seiner Reederei Schokoladen**sos**se bestellt, nicht versehentlich Alarm auslöst, wird vor und nach dem Notrufcode ein Trennsymbol W gesendet.

Ein gültiger Notruf, der beim Automaten einen Alarmzustand auslösen muss, lässt sich demnach in der Form W • • • – – – • • • W darstellen.

Ein solches Muster in der Folge der empfangenen Morsezeichen kann ein Akzeptor (als spezielle Form eines endlichen Automaten) erkennen, als Notrufsignal akzeptieren und als Alarmzustand anzeigen.

2.1 Beschreiben Sie den Aufbau und die Arbeitsweise eines endlichen Automaten.

2.2 Geben Sie das Zustandsdiagramm eines Akzeptors an, der das beschriebene Notrufsignal W • • • – – – • • • W akzeptiert.

2.3 Beschreiben Sie die Arbeitsweise des Akzeptors für die folgenden Eingaben:

Folge a) W • • • – • W

Folge b) W • • • – – – • • • W.

2.4 Gegeben ist die folgende Tabelle des Morse-Codes der Buchstaben von A bis Z. Warum können beim Morse-Code trotz der Codelänge von 4 Bits alle Buchstaben (A bis Z) codiert werden?

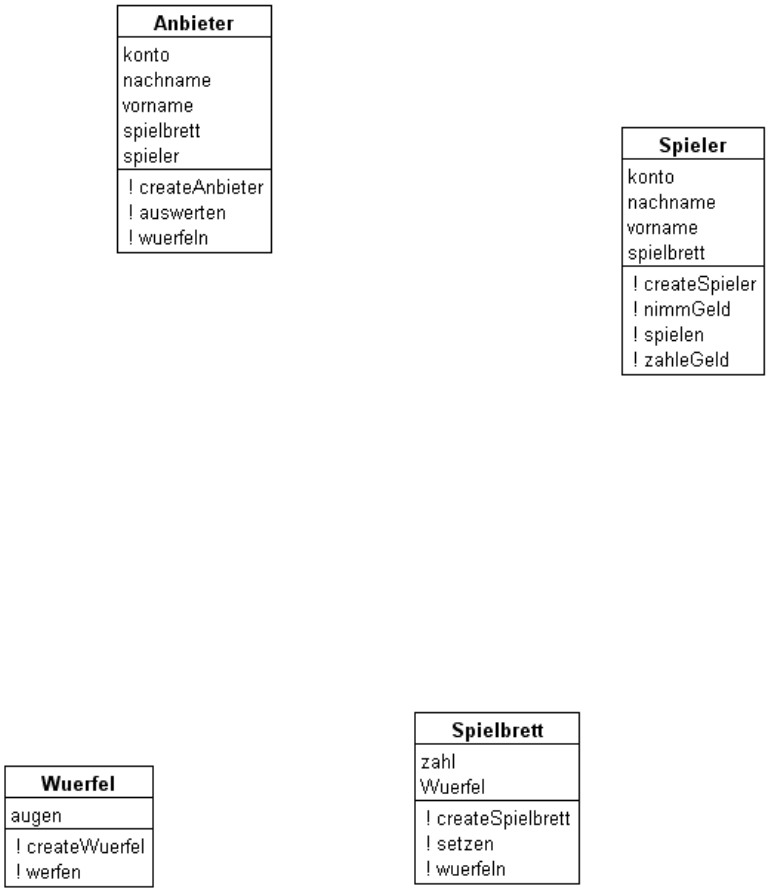
A	·–	B	–···	C	–·–·	D	–··	E	·	F	··–·	G	––·
H	····	I	··	J	·–––	K	–·–	L	·–··	M	––	N	–·
O	–––	P	·––·	Q	––·–	R	·–·	S	···	T	–	U	··–
V	···–	W	·––	X	–··–	Y	–·––	Z	––··				

2.5 Erläutern Sie am Beispiel des Codes • • – • • • die Schwierigkeiten, die beim Decodieren des Morse-Codes auftreten können. Welche Konsequenzen entstehen daraus für die binäre Codierbarkeit des Morse-Codes?

Teilaufgabe	2.1	2.2	2.3	2.4	2.5
Anteil an der Gesamtleistung	9 %	15 %	8 %	3 %	3 %

Anlage A zu Aufgabe 1.1: Klassendiagramm

Name: ….....



Anlage B zur Aufgabe 1.1: Quellcode

Code in DELPHI:

```
1     type
2       TAnbieter = class
3         private
4           konto: Integer;
5           nachname: String;
6           vorname: String;
7           spielbrett: TSpielbrett;
8           spieler: TSpieler;
9         public
10          constructor createAnbieter
11                      (v: String; n: String; k: Integer; sb: TSpielbrett;
12                       Sp: TSpieler);
13          procedure auswerten;
14          procedure wuerfeln;
15          function gibKontostand : Integer;
16          function gibNachname : String;
17          function gibVorname : String;
18      end;
19
20    constructor TAnbieter.createAnbieter
21              (v: String; n: String; k: Integer; sb: TSpielbrett; sp: TSpieler);
22    begin
23      inherited;
24      vorname  := v;
25      nachname := n;
26      konto    := k;
27      spielbrett := sb;
28      spieler := sp;
29    end;
30
31    function TAnbieter.gibVorname : String;
32    begin    Result := vorname;  end;
33
34    function TAnbieter.gibNachname : String;
35    begin    Result := nachname;  end;
36
37    function TAnbieter.gibKontostand : Integer;
38    begin    Result := konto;  end;
39
40    procedure TAnbieter.auswerten;
41    begin
42      if (Spielbrett.gibTipp = Spielbrett.gibWuerfel)
43        Then Begin
44               spieler.nimmGeld (3);
45               konto := konto - 3;
46             End
47        Else Begin
48               spieler.zahleGeld(1);
49               konto := konto + 1;
50             End;
51    end;
52
53    procedure TAnbieter.wuerfeln;
54    begin    spielbrett.wuerfeln;  end;
55
```

```
56    type
57      TSpieler = class
58        private
59          konto    : Integer;
60          nachname : String;
61          vorname  : String;
62          spielbrett : TSpielbrett;
63        public
```

```
64          constructor createSpieler (v: String; n: String; k: Integer; s: TSpielbrett);
65          procedure zahleGeld (betrag: Integer); virtual;
66          procedure nimmGeld (betrag: Integer); virtual;
67          procedure spielen (tipp: Integer); virtual;
68          function gibKontostand : Integer; virtual;
69          function gibNachname : String; virtual;
70          function gibVorname : String; virtual;
71        end;
72
73    constructor TSpieler.createSpieler (v: String; n: String; k: Integer; s: TSpielbrett);
74    begin
75      inherited;
76      konto      := k;
77      nachname   := n;
78      vorname    := v;
79      spielbrett := s;
80    end;
81
82    function TSpieler.gibVorname : String;
83    begin   Result := vorname;  end;
84
85    function TSpieler.gibNachname : String;
86    begin   Result := nachname;  end;
87
88    function TSpieler.gibKontostand : Integer;
89    begin   Result := konto;  end;
90
91    procedure TSpieler.nimmGeld (betrag: Integer);
92    begin   konto := konto + betrag;  end;
93
94    procedure TSpieler.zahleGeld (betrag: Integer);
95    begin   konto := konto - betrag;  end;
96
97    procedure TSpieler.spielen (tipp: Integer);
98    begin   spielbrett.setzen(tipp);  end;
99
```

```
100   type
101     TWuerfel = class
102       private
103         augen : Integer;
104       public
105         constructor createWuerfel; virtual;
106         procedure werfen; virtual;
107         function gibAugen : Integer; virtual;
108     end;
109
110   implementation
111
112   constructor TWuerfel.createWuerfel;
113   begin  inherited;  randomize;  end;
114
115   procedure TWuerfel.werfen;
116   begin  augen := Random (6) + 1;  end;
117
118   function TWuerfel.gibAugen: Integer;
119   begin  Result := augen;  end;
120
121   type
122     TSpielbrett = class
123       private
124         zahl : Integer;
125         Wuerfel: TWuerfel;
126       public
127         constructor createSpielbrett;
128         procedure setzen (tipp: Integer);
129         procedure wuerfeln;
130         function gibTipp : Integer;
131         function gibWuerfel : Integer;
132       end;
```

```
133
134   constructor TSpielbrett.createSpielbrett;
135   begin
136     inherited;
137     Wuerfel := TWuerfel.Create;
138   end;
139
140   procedure TSpielbrett.setzen (tipp: Integer);
141   begin  zahl := tipp;  end;
142
143   function TSpielbrett.gibTipp : Integer;
144   begin  Result := zahl;  end;
145
146   function TSpielbrett.gibWuerfel: Integer;
147   begin  Result := Wuerfel.gibAugen;  end;
148
149   procedure TSpielbrett.wuerfeln;
150   begin  wuerfel.werfen;  end;
151
152   var Form1:      TForm1;
153       Spieler:    TSpieler;
154       Anbieter:   TAnbieter;
155       Spielbrett: TSpielbrett;
156
157   procedure TForm1.FormCreate(Sender: TObject);
158   begin
159     Spielbrett := TSpielbrett.createSpielbrett;
160     Spieler    := TSpieler.createSpieler
161                   ('Moritz', 'Zuse', 100, spielbrett);
162     Anbieter   := TAnbieter.CreateAnbieter
163                   ('Max', 'Turing', 100, spielbrett, spieler);
164
165     EdKontoAnbieter.Text := IntToStr(spieler.gibKontostand);
166     EdKontoSpieler.Text  := IntToStr(anbieter.gibKontostand);
167     LbKontoAnbieter.Caption := 'Konto des Anbieters' + anbieter.gibVorname;
168     LbKontoSpieler.Caption  := 'Konto des Spielers' + spieler.gibVorname;
169     LbWuerfel.Caption := 'Würfel A:';
170     LbTipp.Caption := 'Tipp';
171     BtnSpielen.Caption := 'spielen';
172     BtnBeenden.Caption := 'Spielende';
173     EdWuerfel.Text := '';
174     ComboBox1.Items.Add('1');
175     ComboBox1.Items.Add('2');
176     ComboBox1.Items.Add('3');
177     ComboBox1.Items.Add('4');
178     ComboBox1.Items.Add('5');
179     ComboBox1.Items.Add('6');
180     ComboBox1.ItemIndex := 0;
181   end;
182
183   procedure TForm1.BtnSpielenClick(Sender: TObject);
184   Var tipp: Integer;
185   begin
186     tipp := ComboBox1.ItemIndex + 1;
187     spieler.spielen(tipp);
188     anbieter.wuerfeln;
189     anbieter.auswerten;
190     EdWuerfel.Text := IntToStr (Spielbrett.gibWuerfel);
191     EdKontoAnbieter.Text := IntToStr(spieler.gibKontostand);
192     EdKontoSpieler.Text := IntToStr(anbieter.gibKontostand);
193   end;
```

Alternative: Code in JAVA

```
1     public class Anbieter
2     { private String vorname, nachname;
3       private int konto;
4       private Spielbrett spielbrett;
5       private Spieler spieler;
6
7       public Anbieter(String v, String n, int k, Spielbrett sb, Spieler sp)
8       { vorname = v;
9         nachname = n;
10        konto = k;
11        spielbrett = sb;
12        spieler = sp; }
13
14      public String gibVorname()
15      { return vorname;}
16
17      public String gibNachname()
18      { return nachname; }
19
20       public int gibKontostand()
21      { return konto; }
22
23      public void wuerfeln()
24      { spielbrett.wuerfeln();  }
25
26      public void auswerten()
27      { if (spielbrett.gibTipp() == spielbrett.gibWuerfel())
28          // Der Spieler gewinnt den Einsatz von 1 EUR plus 3 EUR
29          { spieler.nimmGeld(3);
30            konto = konto - 3;}
31        else   // Der Spieler verliert seinen Einsatz von 1 EUR
32        { spieler.zahleGeld(1);
33          konto = konto + 1; }
34      }
35    }
```

```
36    public class Spieler
37    { private String vorname, nachname;
38      private int konto;
39      private Spielbrett spielbrett;
40
41      public Spieler(String v, String n, int k, Spielbrett s)
42      { vorname = v;
43        nachname = n;
44        konto = k;
45        spielbrett = s;  }
46
47      public String gibVorname()
48      { return vorname;  }
49      public String gibNachname()
50      { return nachname; }
51      public int gibKontostand()
52      { return konto;    }
53
54      public void nimmGeld(int betrag)      // Geld gewonnen
55      { konto = konto + betrag; }
56
57      public void zahleGeld(int betrag)     // Geld verloren
58      { konto = konto - betrag; }
59
60      public void spielen(int tipp)         // Der Einsatz betraegt immer 1 EUR pro Spiel
61      { spielbrett.setzen(tipp);  }         // tipp steht für den Wuerfeltipp
62    }
```

```
63
64    import java.util.Random;
65    public class Wuerfel
66    { private int augen;
```

```
67      private Random r;
68
69      public Wuerfel()
70      {  r = new Random(); }
71
72      public void werfen()
73      {  augen = r.nextInt(6)+1;         // natuerliche Zufallszahl zwischen 1 und 6  }
74
75      public int gibAugen()
76      {  return augen;  }
77    }
```

```
78    public class Spielbrett
79    { private int zahl;
80      private Wuerfel wuerfel;
81
82      public Spielbrett()
83      {  wuerfel = new Wuerfel(); }
84
85      // Setzen auf eine Zahl
86      public void setzen(int tipp)
87      {  zahl = tipp; }
88
89      public void wuerfeln()
90      {  wuerfel.werfen();  }
91
92      public int gibTipp()
93      {  return zahl; }
94
95      public int gibWuerfel()
96      {  return wuerfel.gibAugen(); }
```

```
97    public class Steuerung
98    { private Spieler spieler;
99      private Spielbrett spielbrett;
100     private Anbieter anbieter;
101
102     public Steuerung() {
103                                  // Klassenbeziehungen
104      spielbrett = new Spielbrett();
105      spieler = new Spieler("Moritz","Zuse",100,spielbrett);
106      anbieter = new Anbieter("Max","Turing",100,spielbrett, spieler);
107
108   // Konsolen-Spielsimulation mit vom Spieler gesetzter Augenzahl 6 und geworfener 5
109        spieler.spielen(6);
110        anbieter.wuerfeln();
111        anbieter.auswerten();
112
113        System.out.println(spieler.gibKontostand());
114        System.out.println(anbieter.gibKontostand());  }
115
116     public static void main(String [] args)
117     {  new Steuerung();  }
118   }
```

Alternative: Code in PYTHON

```
1     class Anbieter:
2
3       def __init__(self, vorname, nachname, konto, spielbrett, spieler):
4               self.__vorname = vorname
5               self.__nachname = nachname
6               self.__konto = konto
7               self.__spielbrett = spielbrett
8               self.__spieler = spieler
9       def gib_vorname(self): return self.__vorname
10      def gib_nachname(self): return self.__nachname
11      def gib_kontostand(self): return self.__konto
12      def wuerfeln(self): self.__spielbrett.wuerfeln()
13      def auswerten(self):
```

```
14            if self.__spielbrett.gib_tipp() == \
15               self.__spielbrett.gib_wuerfel():
16                    # Der Spieler gewinnt den Einsatz von 1 EUR plus 3 Eur
17                    self.__konto -= 3
18                    self.__spieler.nimm_geld(3)
19            else:
20                    # Der Spieler verliert seinen Einsatz von 1 EUR
21                    self.__spieler.zahle_geld(1)
22                    self.__konto += 1
23
```

```
24    class Spieler:
25     def __init__(self, vorname, nachname, konto, spielbrett):
26            self.__vorname = vorname
27            self.__nachname = nachname
28            self.__konto = konto
29            self.__spielbrett = spielbrett
30     def gib_vorname(self):  return self.__vorname
31     def gib_nachname(self): return self.__nachname
32     def gib_kontostand(self): return self.__konto
33     def nimm_geld(self, betrag): self.__konto += betrag        # Geld gewonnen
34     def zahle_geld(self, betrag): self.__konto -= betrag       # Geld verloren
35     def spielen(self, tipp): self.__spielbrett.setzen(tipp)    # tipp steht für Wuerfeltipp
36
```

```
37
38    from random import Random
39    class Wuerfel:
40     def __init__(self):
41            self.__augen = None
42            self.__r = Random()
43     def werfen(self): self.__augen = self.__r.randrange(1, 7)
44     def gib_augen(self): return self.__augen
45
```

```
46    from wuerfel import Wuerfel
47    class Spielbrett:
48     def __init__(self):
49            self.__zahl = None
50            self.__wuerfel = Wuerfel()
51     def setzen(self, tipp):
52            # Setzen auf eine Zahl
53            self.__zahl = tipp
54     def wuerfeln(self): self.__wuerfel.werfen()
55     def gib_tipp(self): return self.__zahl
56     def gib_wuerfel(self): return self.__wuerfel.gib_augen()
57
```

```
58    from spielbrett import Spielbrett
59    from spieler import Spieler
60    from anbieter import Anbieter
61
62    class Steuerung:
63     def __init__(self):
64            self.__spielbrett = Spielbrett()
65            self.__spieler = Spieler('Moritz', 'Zuse', 100, self.__spielbrett)
66            self.__anbieter = Anbieter('Max', 'Turing', 100, self.__spielbrett, self.__spieler)
67            self.__spieler.spielen(6)
68            self.__anbieter.wuerfeln()
69            self.__anbieter.auswerten()
70
71            print 'Kontostand Spieler:',
72
73            print self.__spieler.gib_kontostand()
74            print 'Kontostand Anbieter:',
75            print self.__anbieter.gib_kontostand()
76
77    if __name__ == '__main__': steuerung = Steuerung()
```

Anlage C: Arbeitsblatt zu Aufgabe 1.3: Schreibtischtest (JAVA-Version)

Name: ….....

aufrufendes Objekt	Methodenaufruf	Ergebnis / Erläuterung
`steuerung`	`spielbrett = new Spielbrett();`	
`spielbrett`	`wuerfel = new Wuerfel();`	

`steuerung`	`spieler = new Spieler(..);`
	`anbieter = new Anbieter(..)`
	`spieler.spielen(6);`
		Der Spieler setzt auf „6“.

2.3 Erwartungshorizont

Aufgabe	Erwartete Teilleistung	BE im AB I	II	III	Didaktischer Zusammenhang zum erteilten Unterricht / *Kompetenzbezug*
1.1	Nach einer umfangreichen Analyse des unbekannten Quellcodes werden die Beziehungen im Klassendiagramm eingetragen (siehe Musterlösung). Dokumentation durch Angabe und Kommentierung der relevanten Klassen-Codezeilen	4 6			UML-konforme Klassendiagramme sind ständige Übungen bei Analyse und Modellierung. „Chuck A Luck“ ist aus dem Unterricht nicht bekannt. Die Methoden enthalten keine komplexe Algorithmik. *Informatisches Modellieren*
1.2	Neues Klassendiagramm entwickeln und zeichnen (Klasse Person, veränderte Klassen Anbieter und Spieler), siehe Musterlösung. Implementierung der Klassen Person, Anbieter und Spieler, siehe Musterlösung. - semantische Richtigkeit des Diagramms - semantische Richtigkeit des Codes - syntaktische Richtigkeit des Diagramms und des Codes	6	5 5		Die semantisch richtige Lösung einer unbekannten Aufgabe erfordert sicheres Verständnis der Vererbung. Die Syntax von Diagrammen und Code zu beachten ist eine ständige Anwendung im Unterricht. *Informatiksysteme verstehen*
1.3	Durchführung und Protokollierung des Schreibtischtests (siehe Musterlösung). Die Fälle a) und b) werden nacheinander untersucht. - 1. untersuchter Fall - 2. untersuchter Fall (dies schließt die Erkenntnis ein, dass große Teile der Lösung sich wiederholen)		8 4		Schreibtischtests gehören zur ständigen Übung, es liegt wegen der vier beteiligten Klassen ein komplexes Problem vor. *Informatiksysteme verstehen*
1.4	Neuprogrammierung an zahlreichen Stellen (siehe Musterlösung)			10	Die Lösung erfordert sicheres Verständnis des objektorientierten Programmierens in einer unbekannten und offenen Problemstellung. *Informatiksysteme verstehen*
1.5	Verbale Beschreibung der Wartungseingriffe: Zwei Objekte vom Typ Spieler; jeder Spieler hat sein Konto. Algorithmische Anpassungen bei der Klasse Anbieter in der Methode auswerten und in der Klasse Steuerung beim Spielablauf.		6		In einer offenen Aufgabenstellung wird die geübte Beachtung mehrfacher Exemplare von Klassen angewandt. *Informatisches Modellieren*

1.6	Verbale Beschreibung und Bewertung, insbesondere des Datenschutzproblems und der Beteiligten beim Glücksspiel. Das Verfahren ist nicht zulässig, Gründe z. B. - (Staatliche) Schulen dürfen keine kommerziellen Ein- und Ausgaben verbuchen. - Für die Durchführung eines Online-Spiels auf der Homepage müssten personenbezogene Daten der Spieler erhoben und gespeichert werden (z. B. Name, Kreditkartennummer), deren Verfügbarkeit in der Verantwortung der Schüler liegt. - Öffentliches Glücksspiel um Geld widerspricht dem Bildungs- und Erziehungsauftrag. - Das Spiel ist durch den ausgewiesenen Bankvorteil von 7,9 % unfair. - Selbst auf der Homepage des Fördervereins oder einer Privatperson sind solche Glücksspiele verboten.		8		Rechtliche Betrachtungen, insbesondere des Datenschutzes, werden im Unterricht durchgeführt. *Kommunizieren*
	Summe Aufgabe 1: 62 BE	16	36	10	
2.1	Ein endlicher Automat wird beschrieben durch: - die endliche, nichtleere Menge der Eingabezeichen (Eingabealphabet) - die endliche, nichtleere Menge der Ausgabezeichen (Ausgabealphabet) - die endliche, nichtleere Menge der möglichen Zustände (Zustandsmenge) - die Überführungstabelle, die die Änderung der Zustände in Abhängigkeit vom aktuellen Zustand und dem eingegebenen Zeichen beschreibt - einen Anfangs- und endlich viele Endzustände. Arbeitsweise: Der Automat ist im Anfangszustand; nach dem Lesen eines Zeichens vom Eingabeband ändert sich sein Zustand gemäß der Überführungstabelle und auf das Ausgabeband wird das entsprechende Zeichen geschrieben. Der Automat stoppt, wenn die Eingabe vollständig gelesen wurde.	5 4			Endliche Automaten sind verpflichtender Bestandteil des Unterrichts in in-3. *Informatisches Modellieren*
2.2	Selbständige Entwicklung des Zustandsdiagramms (siehe Musterlösung und Modrow S. 116[8]).				Anwendung bekannter Modellierung auf eine neues Problem

[8] Eckart Modrow, Automaten Schaltwerke Sprachen, Dümmler; 1988, ISBN 3-427-42912-1

	- Allgemeine Zustände, Start- und Endzustand - Zustandsübergänge mit Legende		6 9		*Informatiksysteme verstehen*
2.3	Textliche Beschreibung des Verhaltens eines erkennenden Automaten (gemäß 2.3) a) Ausgehend vom Startzustand S0 wird nach dem Lesen des Trennzeichens W der Zustand SW erreicht; danach werden nacheinander die Zustände S1, S2, S3 und S4 durchlaufen; das nachfolgende Eingabezeichen (Punkt) führt zurück zum Startzustand S0, nach dem Einlesen des Trennzeichens W ist der Automat wieder im Zustand SW . b) Ausgehend vom Startzustand S0 wird nach dem Lesen des Trennzeichens W der Zustand SW erreicht; danach werden nacheinander die Zustände S1 bis S9 durchlaufen; das nachfolgende Eingabezeichen w führt zum Endzustand SE, d.h. der Automat hat das Eingabewort akzeptiert.	4 4			Die Analyse eines Zustandsdiagramms sowie einer textlichen Automatenbeschreibung sind wiederholt durchgeführt worden. *Informatiksysteme verstehen*
2.4	Textliche Beschreibung der Besonderheiten des Morse-Codes als ein Code mit veränderlicher Codelänge. Mit 4 Bit lassen sich 2^4 = 16 verschiedene Buchstaben codieren. Da aber beim Morse-Code die meisten Buchstaben nur einen Code der Länge 1, 2 oder 3 haben, genügen 4 Bit. Dabei haben häufiger vorkommende Buchstaben einen möglichst kurzen Code.		3		Die Analyse von Codes wurde z. B. im Rahmen des Vertiefungsgebietes „Kryptologie und Datensicherheit“ in in-3 durchgeführt. Kryptologie kann auch ein möglicher Kontext zu Sprachen und Automaten sein. *Mit Informationen umgehen*
2.5	Erläuterung der fehlenden Präfix-Eigenschaft beim Morsecode am vorliegenden Beispiel. Erst durch das Einfügen von Pausen bzw. eines Trennsymbols zwischen Buchstaben wird die Nachricht eindeutig decodierbar. Deshalb ist eine binäre Codierbarkeit nicht mehr möglich.		3		Die Lösung erfordert die Anwendung von fundamentalen Regeln zur eindeutigen Decodierbarkeit von Codes. *Mit Informationen umgehen*
	Summe Aufgabe 2: 38 BE	17	21	0	

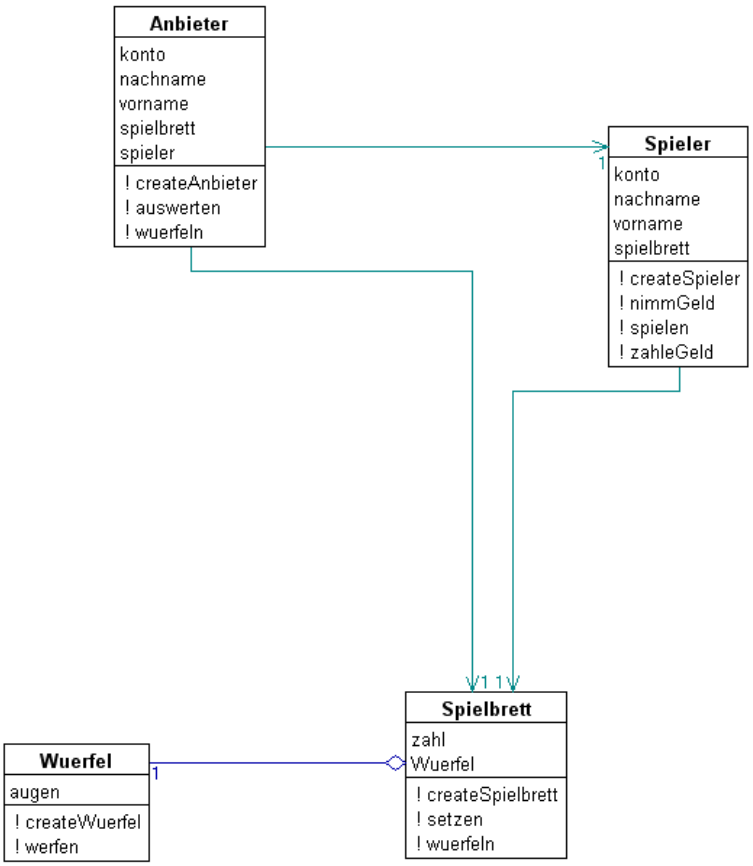
Übersichtstabelle:

	AB 1	AB 2	AB3	BE gesamt	Anteil in %
Aufgabe 1	16	36	10	62	62
Aufgabe 2	17	21	0	38	38
Anforderungsprofil in BE	33	57	10	100	
Anforderungsprofil in %	33	57	10	100	

Die Aufgabe 1 bezieht sich allein auf das **erste Jahr** der Qualifikationsphase und dort allein auf den Lernbereich **Softwaretechnik**. Diese Aufgabe bildet den prüfungsdidaktischen Schwerpunkt der Arbeit (62 %). Die Aufgabe 2 bezieht sich allein auf das **zweite Jahr** der Qualifikationsphase und dort auf das Pflichtgebiet **Sprachen und Automaten,** u. U. mit einem Bezug zum Vertiefungsgebiet **„**Kryptologie und Datensicherheit“.

2.4 Lösungsbeispiele und Diagramme

Musterlösung zu Aufgabe 1.1, Klassendiagramm und Beziehungen



Möglicher Lösungsweg in DELPHI:

`Wuerfel` ist Teil von `Spielbrett` (Aggregation)

```
125 Wuerfel: Twuerfel
137 Wuerfel := Twuerfel.Create;
```

`Anbieter` kennt genau einen `Spieler` (Assoziation)

```
  8 spieler: TSpieler;
 28 spieler := sp;
```

`Anbieter` kennt genau ein `Spielbrett` (Assoziation)

```
  7 spielbrett: TSpielbrett;
 27 spielbrett := sb;
```

`Spieler` kennt genau ein `Spielbrett` (Assoziation)

```
 62 spielbrett : TSpielbrett;
 79 spielbrett := s;
```

Alternative in JAVA

`Wuerfel` ist Teil von `Spielbrett` (Aggregation)

```
80 private Wuerfel wuerfel;
82 public Spielbrett()
83{  wuerfel = new Wuerfel(); }
```

`Anbieter` kennt genau einen `Spieler` (Assoziation)

```
 5 private Spieler spieler;
12 spieler = sp;
```

`Anbieter` kennt genau ein `Spielbrett` (Assoziation)

```
 4 private Spielbrett spielbrett;
11 spielbrett = sb;
```

`Spieler` kennt genau ein `Spielbrett` (Assoziation)

```
39 private Spielbrett spielbrett;
45 spielbrett = s;
```

Alternative in PYTHON:

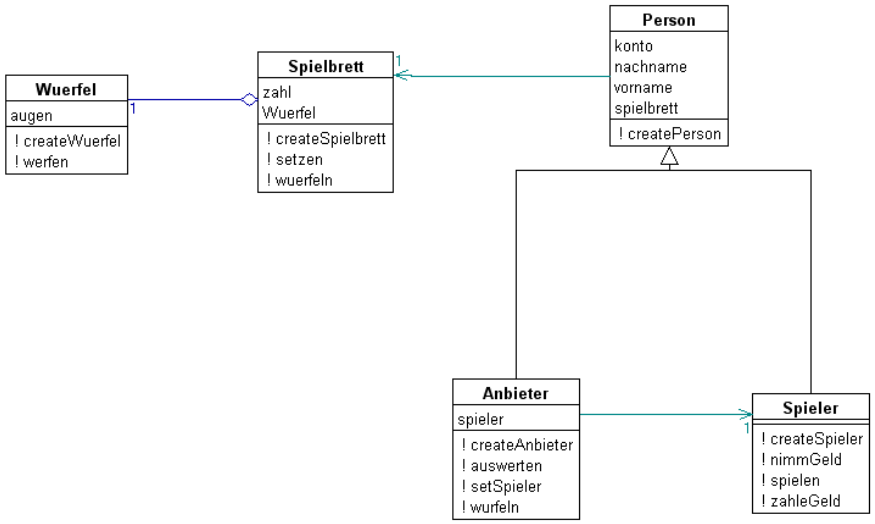
`Wuerfel` ist Teil von `Spielbrett` (Aggregation) `50 self.__wuerfel = Wuerfel()`

`Anbieter` kennt genau einen `Spieler` (Assoziation) `8 self.__spieler = spieler`

`Anbieter` kennt genau ein `Spielbrett` (Assoziation) `7 self.__spielbrett = spielbrett`

`Spieler` kennt genau ein `Spielbrett` (Assoziation) `29 self.__spielbrett = spielbrett`

Musterlösung zu Aufgabe 1.2, erweitertes Klassendiagramm:



Musterlösung zu Aufgabe 1.2 (Code in DELPHI)

```
type
  TPerson = class
    protected
      vorname : String;
      nachname : String;
      konto : Integer;
      spielbrett : TSpielbrett;
    public
      constructor createPerson
      (v: String; n: String; k: Integer; sb: TSpielbrett);
      function gibKontostand : Integer;
      function gibNachname : String;
      function gibVorname : String;
  end;

constructor TPerson.createPerson
            (v: String; n: String; k: Integer; sb: TSpielbrett);
begin
  vorname  := v;
  nachname := n;
  konto    := k;
  spielbrett := sb;
end;

function TPerson.gibNachname : String;
begin  Result := nachname;  end;
function TPerson.gibVorname : String;
begin  Result := vorname;  end;
function TPerson.gibKontostand : Integer;
```

```
begin  Result := konto;  end;
```

```
type
  TAnbieter = class (TPerson)
    private
      spieler: TSpieler;
    public
      constructor createAnbieter
      (v: String; n: String; k: Integer; sb: TSpielbrett; sp: TSpieler);
      procedure wurfeln;
      procedure auswerten;
      procedure setSpieler (s: TSpieler);
  end;

constructor TAnbieter.createAnbieter
            (v: String; n: String; k: Integer; sb: TSpielbrett; sp: TSpieler);
begin
  inherited CreatePerson (v, n, k, sb);
  spieler := sp;
end;

procedure TAnbieter.auswerten;
begin
  if (Spielbrett.gibTipp = Spielbrett.gibWuerfel)
    Then Begin
           spieler.nimmGeld (3);
           konto := konto - 3;
         End
    Else Begin
           spieler.zahleGeld(1);
           konto := konto + 1;
         End;
end;

procedure TAnbieter.setSpieler (s: TSpieler);
begin  spieler := s;  end;

procedure TAnbieter.wurfeln;
begin  spielbrett.wuerfeln;  end;
```

```
type
  TSpieler = class (TPerson)
    private
    public
      constructor createSpieler
       (v: String; n: String; k: Integer; s: TSpielbrett);
      procedure zahleGeld (betrag: Integer);
      procedure nimmGeld (betrag: Integer);
      procedure spielen (tipp: Integer);
  end;

constructor TSpieler.createSpieler
            (v: String; n: String; k: Integer; s: TSpielbrett);
begin
  inherited create;
  konto := k;
  nachname := n;
  vorname := v;
  spielbrett := s;
end;

procedure TSpieler.zahleGeld (betrag: Integer);
begin  konto := konto - betrag;  end;

procedure TSpieler.nimmGeld (betrag: Integer);
begin  konto := konto + betrag;  end;

procedure TSpieler.spielen (tipp: Integer);
begin  spielbrett.setzen(tipp);  end;
```

Musterlösung zu Aufgabe 1.2 (Code in JAVA)

```
public abstract class Person
{ protected  String vorname, nachname;
  protected int konto;
  protected  Spielbrett spielbrett;


  public Person(String v, String n, int k, Spielbrett sb)
  { vorname = v;
    nachname = n;
    konto = k;
    spielbrett = sb;  }

  public String gibVorname()
  { return vorname; }
  public String gibNachname()
  { return nachname; }
  public int gibKontostand()
  { return konto; }
}
```

```
public class Anbieter extends Person
{  private Spieler spieler;

  public Anbieter(String v, String n, int k, Spielbrett sb, Spieler sp)
  { super(v,n,k,sb);
    spieler = sp;    }

  public void wuerfeln()
  { spielbrett.wuerfeln();  }

  public void auswerten()
  { if (spielbrett.gibTipp() == spielbrett.gibWuerfel())
      // Der Spieler gewinnt den Einsatz von 1 EUR plus 3 EUR
      { spieler.nimmGeld(3);
        konto = konto - 3;  }
    else
    { // Der Spieler verliert seinen Einsatz von 1 EUR
      spieler.zahleGeld(1);
      konto = konto + 1;  }
  }
}
```

```
public class Spieler extends Person
{  public Spieler(String v, String n, int k, Spielbrett sb)
  { super(v,n,k,sb); }
  // Geld gewonnen
  public void nimmGeld(int betrag)
  {  konto = konto + betrag;  }
  // Geld verloren
  public void zahleGeld(int betrag)
  {  konto = konto - betrag; }
  // Der Einsatz betraegt immer 1 EUR pro Spiel; tipp steht für den Wuerfeltipp
  public void spielen(int tipp)
  {  spielbrett.setzen(tipp); }
}
```

Musterlösung zu Aufgabe 1.2 (Code in PYTHON)

```
class Person:
	def __init__(self, vorname, nachname, konto, spielbrett):
		self.__vorname = vorname
		self.__nachname = nachname
		self.__konto = konto
		self.__spielbrett = spielbrett
	def spielbrett(self):
		return self.__spielbrett
	def gib_vorname(self):
		return self.__vorname
	def gib_nachname(self):
		return self.__nachname
	def gib_kontostand(self):
		return self.__konto
	def nimm_geld(self, betrag):
		# Geld gewonnen
		self.__konto += betrag
	def zahle_geld(self, betrag):
		# Geld verloren

	self.__konto -= betrag
```

```
from person import Person

class Anbieter(Person):
	def __init__(self, vorname, nachname, konto, spielbrett, spieler):
		Person.__init__(self, vorname, nachname, konto, spielbrett)
		self.__spieler = spieler
	def wuerfeln(self):
		self.spielbrett().wuerfeln()
	def auswerten(self):
		if self.spielbrett().gib_tipp() == \
		   self.spielbrett().gib_wuerfel():
			# Der Spieler gewinnt den Einsatz von 1 EUR plus 3 Eur
			self.gib_geld(3)
			self.__spieler.nimm_geld(3)
		else:
			# Der Spieler verliert seinen Einsatz von 1 EUR
			self.__spieler.zahle_geld(1)
			self.nimm_geld(1)
```

```
from person import Person

class Spieler(Person):
	def __init__(self, vorname, nachname, konto, spielbrett):
		Person.__init__(self, vorname, nachname, konto, spielbrett)

	def spielen(self, tipp):
		# Der Einsatz betraegt immer 1 EUR pro Spiel; tipp steht
		# fuer den Wuerfeltipp
		self.spielbrett().setzen(tipp)
```

Musterlösung zu Aufgabe 1.3, Schreibtischtest (JAVA-Version)

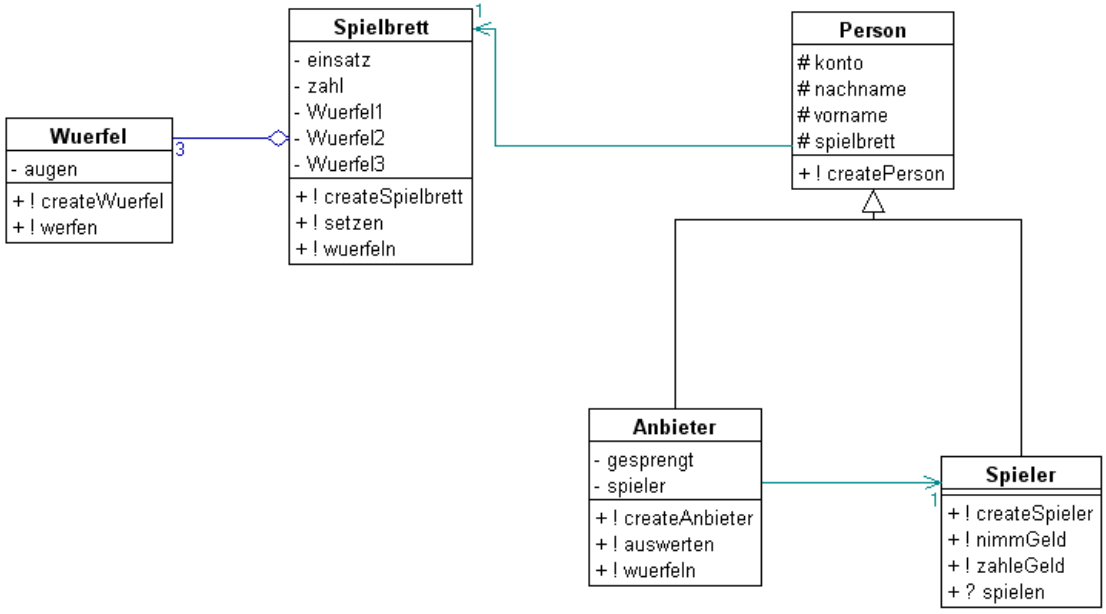
aufrufendes Objekt	Methodenaufruf	Ergebnis / Erläuterung
steuerung	`spielbrett = new Spielbrett();`	
spielbrett	`wuerfel = new Wuerfel();`	

steuerung	`spieler = new Spieler(..);`
	`anbieter = new Anbieter(..)`
	`spieler.spielen(6);`
		Der Spieler setzt auf „6“.

spieler	`spielbrett.setzen(6);`	
steuerung	`anbieter.wuerfeln()`	Der Anbieter würfelt.
anbieter	`spielbrett.wuerfeln();`	
spielbrett	`wuerfel.werfen()`	
Fall a)		
wuerfel	`augen = r.nextInt(6)+1;`	Der Würfel zeigt die Augenzahl „5“.
steuerung	`anbieter.auswerten();`	
anbieter	`if (spielbrett.gibTipp() ==` `spielbrett.gibWuerfel())`	Vergleich des Tipps „6“ mit der Augenzahl „5“.
	`spieler.gibGeld(1);`	Der Spieler hat 1 EUR verloren.
spieler	`konto = konto -1 ;`	
anbieter	`konto = konto + 1;`	Der Anbieter hat 1 EUR gewonnen.
Fall b)		
wuerfel	`augen = r.nextInt(6)+1;`	Der Würfelt zeigt die Augenzahl „6“.
steuerung	`anbieter.auswerten();`	
anbieter	`if (spielbrett.gibTipp() ==` `spielbrett.gibWuerfel())`	Vergleich des Tipps „6“ mit der Augenzahl „6“.
	`spieler.nimmGeld(3);`	Der Spieler hat 3 EUR gewonnen.
spieler	`konto = konto + 3`	
anbieter	`konto = konto - 3;`	Der Anbieter hat 3 EUR verloren.

Musterlösung zu Aufgabe 1.4

(Dieses Klassendiagramm gehört nicht zur erwarteten Schülerleistung)



Musterlösung zu Aufgabe 1.4, (Code in Delphi)

Die erwarteten Codefragmente sind in dieser Beispiellösung zur Übersicht in den vollständigen Programmcode eingebettet.

```
type
  TAnbieter = class (TPerson)
    private
      spieler: TSpieler;
      gesprengt : Boolean;
    public
      constructor createAnbieter
      (v: String; n: String; k: Integer; sb: TSpielbrett; sp: TSpieler);
      procedure wuerfeln;
      procedure auswerten;
      function istGesprengt: Boolean;
  end;

constructor TAnbieter.createAnbieter
         (v: String; n: String; k: Integer; sb: TSpielbrett; sp: TSpieler);
begin
  inherited CreatePerson (v, n, k, sb);
  spieler := sp;
  gesprengt := false;
end;

procedure TAnbieter.auswerten;              // Zeigt ein Würfel die gesetzte Augenzahl, gewinnt
Var spielergewinn: Integer;                 // der Spieler einfach; zeigen zwei Würfel die
    einsatz: Integer;                       // gesetzte Augenzahl, gewinnt der Spieler den
begin                                       // doppelten Einsatz; zeigen alle drei Würfel die
  if (not gesprengt) then                   // gesetzte Augenzahl, so gewinnt der Spieler den
  Begin                                     // dreifachen Einsatz; zeigt kein Würfel die
    spielergewinn := 0;                     // gesetzte Augenzahl, so ist der Einsatz verloren.
    einsatz := spielbrett.gibEinsatz;
    if (Spielbrett.gibTipp = Spielbrett.gibWuerfel1) Then spielergewinn := spielergewinn + einsatz;
    if (Spielbrett.gibTipp = Spielbrett.gibWuerfel2) Then spielergewinn := spielergewinn + einsatz;
    if (Spielbrett.gibTipp = Spielbrett.gibWuerfel3) Then spielergewinn := spielergewinn + einsatz;
    if (spielergewinn > 0) then
     Begin
       konto := konto - (spielergewinn + einsatz);
```

```
      spieler.nimmGeld (spielergewinn + einsatz);
      if (konto <= 0) then gesprengt := true;
    End
    else Begin
           spieler.gibgeld (einsatz);
           konto := konto + einsatz;
         End;
  End;
end;

function TAnbieter.istGesprengt: Boolean;
begin  Result := gesprengt;  end;

procedure TAnbieter.wuerfeln;
begin
  spielbrett.wuerfeln;
end;
```

```
type
  TSpielbrett = class
    private
      zahl : Integer;
      einsatz: Integer;
      Wuerfel1: TWuerfel;
      Wuerfel2: TWuerfel;
      Wuerfel3: TWuerfel;
    public
      constructor createSpielbrett;
      procedure setzen (n: Integer; tipp: Integer);
      procedure wuerfeln;
      function gibTipp : Integer;
      function gibEinsatz : Integer;
      function gibWuerfel1 : Integer;
      function gibWuerfel2 : Integer;
      function gibWuerfel3 : Integer;
   end;

constructor TSpielbrett.createSpielbrett;
begin
  inherited;
  Wuerfel1 := TWuerfel.Create;
  Wuerfel2 := TWuerfel.Create;
  Wuerfel3 := TWuerfel.Create;
end;

procedure TSpielbrett.setzen (n: Integer; tipp: Integer);
begin
  zahl := tipp;
  einsatz := n;
end;

function TSpielbrett.gibTipp : Integer;
begin  Result := zahl;  end;

procedure TSpielbrett.wuerfeln;
begin
  wuerfel1.werfen;
  wuerfel2.werfen;
  wuerfel3.werfen;
end;

function TSpielbrett.gibWuerfel1: Integer;
begin  Result := Wuerfel1.gibAugen;  end;

function TSpielbrett.gibWuerfel2: Integer;
begin  Result := Wuerfel2.gibAugen;  end;

function TSpielbrett.gibWuerfel3: Integer;
begin  Result := Wuerfel3.gibAugen;  end;

function TSpielbrett.gibEinsatz: Integer;
begin  Result := Einsatz;  end;
type
  TSpieler = class (TPerson)
    private
```

```
    public
      constructor createSpieler
       (v: String; n: String; k: Integer; sb: TSpielbrett);
      procedure gibGeld (betrag: Integer);
      procedure nimmGeld (betrag: Integer);
      function spielen (einsatz: Integer; tipp: Integer): Boolean;
  end;

constructor TSpieler.createSpieler
            (v: String; n: String; k: Integer; sb: TSpielbrett);
begin  inherited createPerson(v,n,k,sb);  end;

procedure TSpieler.gibGeld (betrag: Integer);
begin  konto := konto - betrag;  end;

procedure TSpieler.nimmGeld (betrag: Integer);
begin  konto := konto + betrag;  end;

function TSpieler.spielen (einsatz: Integer; tipp: Integer): Boolean;
begin
  if ((konto - einsatz) >= 0)
    Then
      Begin
        Result := true;
        spielbrett.setzen(einsatz, tipp);
      End
    Else Result := false;
end;
```

Musterlösung zu Aufgabe 1.4 (Code in JAVA)

```
public class Anbieter extends Person
{ private Spieler spieler;
  private boolean gesprengt;

  public Anbieter(String v, String n, int k, Spielbrett sb, Spieler sp)
  { super(v,n,k,sb);
    spieler = sp;
    gesprengt = false;  }

   public boolean istGesprengt(){
    return gesprengt;
  }

  /* Zeigt ein Würfel die gesetzte Augenzahl, gewinnt der Spieler einfach;
  zeigen zwei Würfel die gesetzte Augenzahl, gewinnt der Spieler den doppelten Einsatz;
  zeigen alle drei Würfel die gesetzte Augenzahl, so gewinnt der Spieler den dreifachen
  Einsatz; zeigt kein Würfel die gesetzte Augenzahl, so ist der Einsatz verloren.
  */

   public void wuerfeln()
  { spielbrett.wuerfeln();  }

  public void auswerten()
  { if (!gesprengt)
    {  int spielergewinn = 0;
       int einsatz = spielbrett.gibEinsatz();
       if (spielbrett.gibTipp() == spielbrett.gibWuerfel1())
       {  spielergewinn = spielergewinn + einsatz;     }
       if (spielbrett.gibTipp() == spielbrett.gibWuerfel2())
       {  spielergewinn = spielergewinn + einsatz;   }
       if (spielbrett.gibTipp() == spielbrett.gibWuerfel3())
       {  spielergewinn = spielergewinn + einsatz;   }

       if (spielergewinn > 0)
       { konto = konto -(spielergewinn+einsatz);
         spieler.nimmGeld(spielergewinn + einsatz);
         if (konto <= 0)
         { gesprengt = true;  }
       }
       else
       { // Der Spieler verliert seinen Einsatz
         spieler.gibGeld(einsatz);
         konto = konto + einsatz;    }
    }
  }
```

```
}
```

```
public class Spieler extends Person
{ public Spieler(String v, String n, int k, Spielbrett sb)
  { super(v,n,k,sb); }

 // Geld gewonnen
  public void nimmGeld(int betrag)
  { konto = konto + betrag;  }
  // Geld verloren
  public void gibGeld(int betrag)
  { konto = konto - betrag;  }
  // Der Einsatz betraegt "einsatz" EUR pro Spiel; "tipp" steht für den Wuerfeltipp
  // Falls das Limit mit dem Einsatz erschöpft wird, kann nicht weiter gespielt werden.
  public boolean spielen(int einsatz, int tipp)
  {  if ((konto - einsatz) >= 0)
    {  spielbrett.setzen(einsatz,tipp);
       return true;   }
    else
      return false;
  }
}
```

```
public class Spielbrett
{ private int zahl;
  private int einsatz;
  private Wuerfel wuerfel1, wuerfel2, wuerfel3;

  public Spielbrett()
  { wuerfel1 = new Wuerfel();
    wuerfel2 = new Wuerfel();
    wuerfel3 = new Wuerfel();  }

  // Setze n EUR auf eine Zahl tipp
  public void setzen(int n, int tipp)
  { zahl = tipp;
    einsatz = n;  }

  public void wuerfeln()
  { wuerfel1.werfen();
    wuerfel2.werfen();
    wuerfel3.werfen();  }
  public int gibTipp()
  { return zahl; }
   public int gibEinsatz()
  { return einsatz;  }
  public int gibWuerfel1()
  { return wuerfel1.gibAugen();  }
  public int gibWuerfel2()
  { return wuerfel2.gibAugen();  }
  public int gibWuerfel3()
  { return wuerfel3.gibAugen();  }
}
```

Musterlösung zu Aufgabe 1.4 (Code in PYTHON)

```
from person import Person

class Anbieter(Person):
	def __init__(self, vorname, nachname, startkapital, spielbrett, spieler):
		Person.__init__(self, vorname, nachname, startkapital, spielbrett)
		self.__spieler = spieler
#		self.__gesprengt = startkapital <= 0

	def ist_gesprengt(self):
#		return self.__gesprengt
		return self.gib_kontostand() <= 0
	def wuerfeln(self):
		self.spielbrett().wuerfeln()
	def auswerten(self):
		if self.ist_gesprengt():
			return
		spielergewinn = 0
		einsatz = self.spielbrett().gib_einsatz()
		if self.spielbrett().gib_tipp() == \ self.spielbrett().gib_wuerfel1():
		  spielergewinn += einsatz
		if self.spielbrett().gib_tipp() == \ self.spielbrett().gib_wuerfel2():
		  spielergewinn += einsatz
		if self.spielbrett().gib_tipp() == \ self.spielbrett().gib_wuerfel3():
		  spielergewinn += einsatz
# Alternativ Spielbrett Variante b:
#		tipp = self.spielbrett().gib_tipp()
#		for i in range(self.spielbrett().wuerfelanzahl()):
#			if tipp == self.spielbrett().gib_wuerfel(i): spielergewinn += einsatz
		if spielergewinn > 0:
			self.gib_geld(spielergewinn)
			self.__spieler.nimm_geld(spielergewinn)
		else:
			# Der Spieler verliert seinen Einsatz von 1 EUR
			self.__spieler.gib_geld(einsatz)
			self.nimm_geld(einsatz)
```

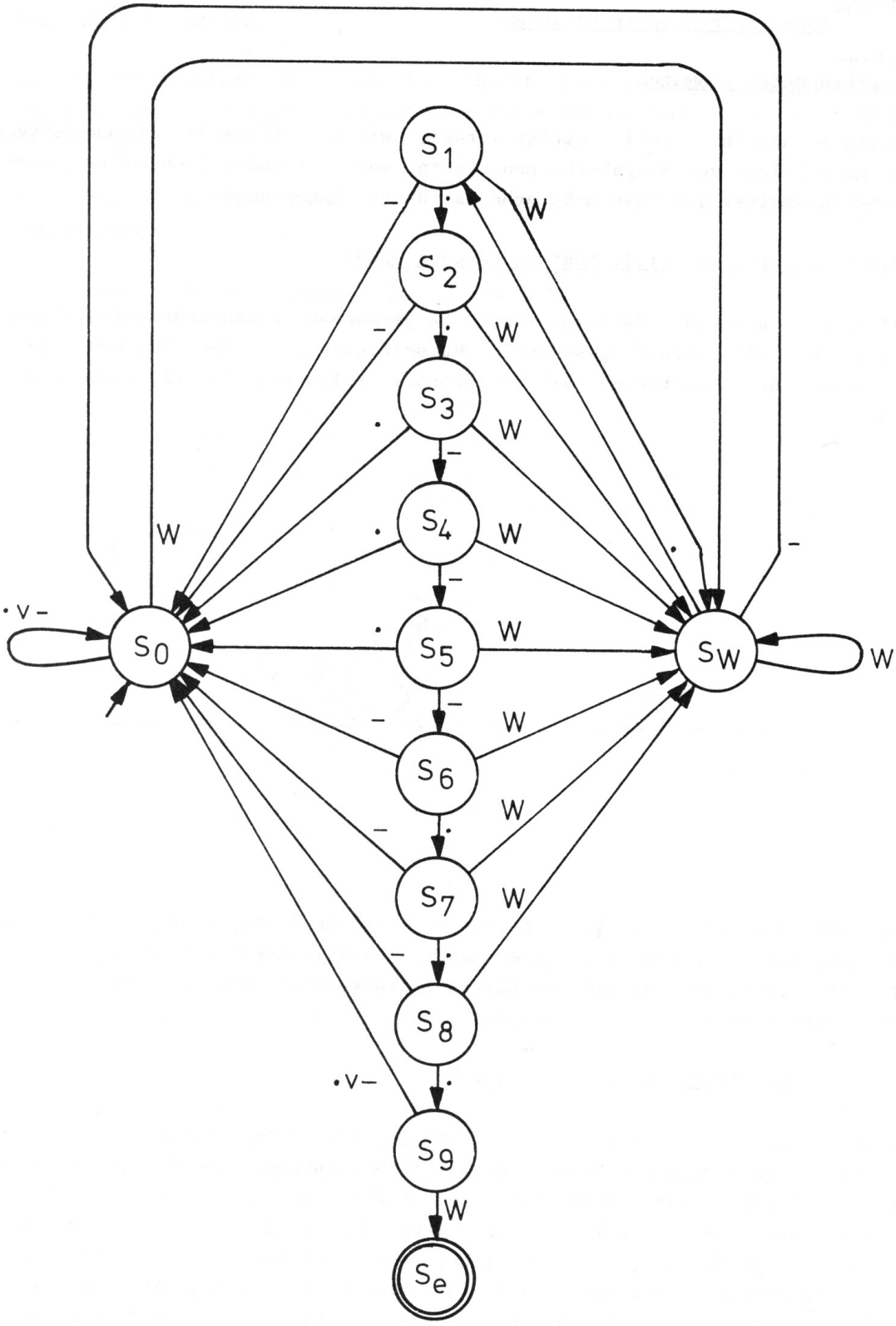
```
from person import Person

class Spieler(Person):
	def __init__(self, vorname, nachname, startkapital, spielbrett):
		Person.__init__(self, vorname, nachname, startkapital, spielbrett)
	def spielen(self, einsatz, tipp):
		# Der Einsatz pro Spiel ist beliebig, aber positiv; tipp steht
		# für den Wuerfeltipp
		if self.gib_kontostand() - einsatz >= 0:
			self.spielbrett().setzen(einsatz, tipp)
			return True
		else:
			return False
```

```
from wuerfel import Wuerfel

class Spielbrett:
	def __init__(self):
		self.__zahl = None
		self.__einsatz = None
		self.__wuerfel1 = Wuerfel()
		self.__wuerfel2 = Wuerfel()
		self.__wuerfel3 = Wuerfel()
	def setzen(self, einsatz, tipp):
		# Einen Einsatz auf eine Zahl setzen
		self.__zahl = tipp
		self.__einsatz = einsatz
	def wuerfeln(self):
		self.__wuerfel1.werfen()
		self.__wuerfel2.werfen()
		self.__wuerfel3.werfen()
	def gib_tipp(self): return self.__zahl
	def gib_einsatz(self):  return self.__einsatz
	def gib_wuerfel1(self): return self.__wuerfel1.gib_augen()
	def gib_wuerfel2(self): return self.__wuerfel2.gib_augen()
	def gib_wuerfel3(self): return self.__wuerfel3.gib_augen()
```

Musterlösung zu Aufgabe 2.3, Zustandsdiagramm



3. Unterscheidung Grundkurs / Leistungskurs

In diesem Beispiel werden die Merkmale der Unterscheidung zwischen Grundkurs- und Leistungskursaufgaben (vgl. Abschnitt 1.2) an einem ausformulierten Aufgabenbeispiel verdeutlicht. Die Bewertungseinheiten dienen hier allein der Gewichtung der Teilaufgaben untereinander.

Der Bezug zum Gesamtvorschlag bleibt wegen der fehlenden zweiten oder dritten Aufgabe fiktiv. Darüber hinaus ergibt sich – wie im dokumentierten Beispiel – die Möglichkeit, die Bewertungseinheiten für die Aufgabe unterschiedlich hoch anzusetzen (hier 31 BE im GK bzw. 39 BE im LK). Umfang und Schwierigkeitsmaß dieser Aufgaben unterscheiden sich mehr als es das Verhältnis der Bearbeitungszeiten (180 min bzw. 240 min) unter Berücksichtigung des jeweils unterschiedlich durchgeführten Unterrichts ausweist. Zur besseren Vergleichbarkeit wurden gleiche Aufgabenteile im Grund- und Leistungskurs mit gleicher Anzahl von BE versehen. Die Gesamtsumme der BE eines Vorschlages liegt zwischen 90 und 125 (vgl. AV Prüfungen, Fachanlage 3b). Das im Abschnitt 1.3 empfohlene Prinzip, verschiedene Handlungsdimensionen des Schülers zu integrieren, wurde bereits innerhalb dieser **einen** Aufgabe beachtet: z. B. Analyse, Konstruktion, textliche Beschreibung und Beurteilung, grafische Darstellung und Implementierung.

Die vorliegenden Aufgaben beziehen sich beide auf die Verwaltung eines Wochenprogramms in einem Kino. Der gemeinsame Kern ist in den vier identischen Entitäten Film, Regisseur, Vorstellung und Saal gegeben. Zwar sind die Aufgabenteile a) bis d1) zunächst fast identisch, die zugrunde gelegte Problemstellung ist im Leistungskurs aber komplexer, denn das Problem der sitzplatzgenauen Kartenbuchung gibt es in der Grundkursaufgabe nicht.

Hieraus entsteht für den Leistungskurs ein umfangreicherer Datenbankentwurf und komplexere Anfragen an die Datenbank sind möglich (siehe d1 bis d4). Beispielsweise wird in d4) durch einen anspruchsvollen Join mit geschachtelter Unterabfrage der Anforderungsbereich III erreicht. Weiterhin entfällt im Grundkurs im Aufgabenteil b) die im Leistungskurs ausdrücklich geforderte Begründung.

Inhaltliche Unterscheidungen, wie sie der Rahmenlehrplan, Kapitel 4.1 für den Leistungskurs vorsieht, finden sich in der Teilaufgabe c). Hier wird vorausgesetzt, dass die Schüler im vorangegangenen Unterricht das Konzept der Normalisierung erarbeitet und Normalisierungen vorgenommen haben.

Wenn Teilaufgaben Implementierungen erwarten, ist es hilfreich, die syntaktische und semantische Richtigkeit zu unterscheiden (siehe Kapitel 2, Aufgabe 1.2). In der Aufgabe d) des Grundkurses wird deshalb eine Variante (siehe vollständiger Erwartungshorizont im Anschluss an die Aufgabenstellung hinten) vorgestellt, die bei gleicher Anzahl von Bewertungseinheiten eine Unterscheidung nach syntaktischer und semantischer Richtigkeit vornimmt.

3.1 Grundkursaufgabe Datenbanksysteme (31 %)

Ein Berliner Kino erfasst das Wochenprogramm bisher in Form von dieser Tabelle:

Filmtitel	Dauer in min	Regisseur (Name, Land, Geburtsjahr)	Datum	Uhrzeit	Saal-Nr.	Anzahl Sitzplätze
Ein Quantum Trost	106	Marc Forster,GER,1969	04.12.	20:00	1	475
Ein Quantum Trost	106	Marc Forster,GER,1969	04.12.	20:00	2	412
Wall-E	110	Andrew Stanton, USA ,1965	04.12.	19:30	6	278
Wall-E	110	Andrew Stanton, USA ,1965	04.12.	22:00	3	121
Casablanca	104	Michael Curtiz, USA, 1988	04.12	21:00	5	80
Drachenläufer	127	Marc Forster,GER,1969	04.12.	19:00	4	174
…	…	…	…	…	…	…

Ein Informatikkurs wird beauftragt, eine relationale Datenbank zur Verwaltung des Kinoprogramms zu erstellen. Folgende Vorgabe ist dabei einzuhalten: Zu jeder Vorstellung eines Films soll die Gesamtzahl der bisher verkauften Kinokarten gespeichert werden.

a) Erläutern Sie die Nachteile, die sich ergeben, wenn die Daten in der oben angegebenen Form gespeichert sind.

b) Gliedern Sie die Tabellendaten in Entitäten und geben Sie die Beziehungen zwischen ihnen im Entity-Relationship-Modell (ERM) an. Markieren Sie die jeweiligen Schlüssel. Bestimmen Sie die Kardinalitäten der modellierten Beziehungen.

c) Erstellen Sie an Hand des ERM das relationale Datenbankmodell für die Verwaltung des Kinoprogramms. Erläutern Sie Ihre Vorgehensweise.

d) Geben Sie jeweils eine geeignete SQL-Anweisung an, die folgende Abfragen erfüllt:

   d1) Erstellen Sie die Tabelle „Film“.

   d2) Fügen Sie einen neuen Eintrag in die Tabelle „Film“ ein.

   d3) Wann läuft der Film „Ein Quantum Trost“? Angezeigt werden soll der Filmtitel, Datum und Uhrzeit. Wählen Sie eine geeignete Sortierung.

Teilaufgabe	a	b	c	d
Anteil an der Gesamtleistung ca.	3 %	10 %	9 %	9 %

Erwartungshorizont zur Grundkursaufgabe Datenbanksysteme

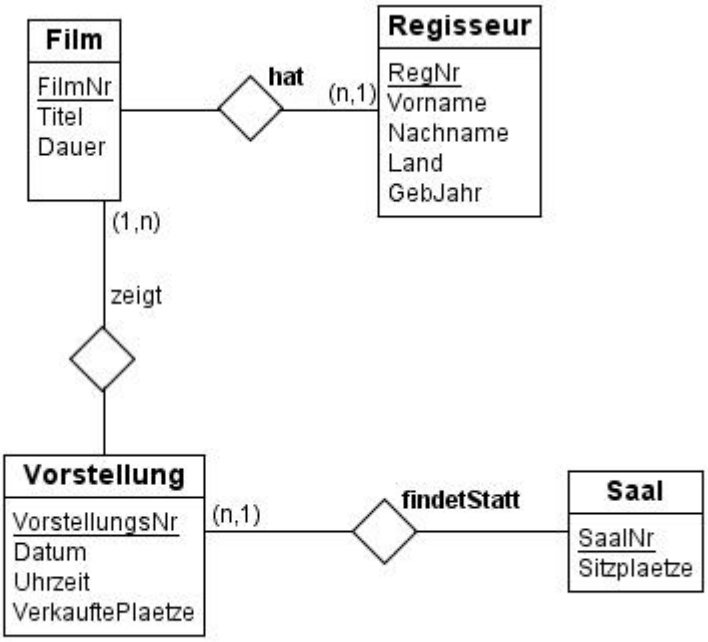
Aufgabe	Erwartete Teilleistung	BE im AB I	II	III	Didaktischer Zusammenhang zum erteilten Unterricht / *Kompetenzbezug*
a	Redundanzen mit den entsprechenden Anomalien; außerdem sind die Werte in der Regisseur-Spalte nicht atomar		3		Wurde behandelt; erfordert aber Übersicht. *Kommunizieren*
b	Notieren des ER-Diagramms unter Beachtung einer normgerechten Darstellung: - Entitätstypen mit den Primärschlüsseln und den Beziehungen - Angabe der Kardinalitäten (Eine mögliche Lösung im Anhang.)	7	3		Das Erstellen von ERM wurde im Unterricht an einigen Beispielen geübt. *Informatisches Modellieren*
c	- Angabe der Tabellen inklusive Primär- und Fremdschlüssel (s. Anhang). - Erklärung der Vorgehensweise mit eigenen Worten	6 3			Abbildung von ERM nach RDM geschieht nach Regeln und wurde geübt. *Informatisches Modellieren* *Kommunizieren*
d1	Mögliche Konstruktion der SQL-Abfragen: `CREATE Table Film (FilmNr INT, Titel CHAR(20),Dauer INT, RegNr INT)`	3			An anderen Beispielen im Unterricht behandelt
d2	`INSERT INTO Film (FilmNr, Titel, Dauer) VALUES (110,"High Noon", 100)`	3			An anderen Beispielen im Unterricht behandelt
d3	`SELECT Titel,Datum,Uhrzeit FROM Film, Vorstellung WHERE Film.FilmNr = Vorstellung.FilmNr ORDER by Datum, Uhrzeit`		3		JOIN über zwei Tabellen *Informatiksysteme verstehen*
	Summe: 31 BE	22	9	0	

Variante des Erwartungshorizonts (siehe Vorbemerkung) bei gleicher Anzahl von Bewertungseinheiten zur Unterscheidung nach syntaktischer und semantischer Richtigkeit:

d1	Mögliche Konstruktion der SQL-Abfragen: `CREATE Table Film (FilmNr INT, Titel CHAR(20),Dauer INT, RegNr INT)`				An anderen Beispielen im Unterricht behandelt.
d2	`INSERT INTO Film (FilmNr, Titel, Dauer) VALUES (110,"High Noon", 100)`				An anderen Beispielen im Unterricht behandelt.
d3	`SELECT Titel,Datum,Uhrzeit FROM Film, Vorstellung WHERE Film.FilmNr = Vorstellung.FilmNr ORDER by Datum, Uhrzeit` syntaktisch richtig semantisch richtig	3	6		Im Unterricht wurden entsprechende Implementierungsübungen durchgeführt. *Informatiksysteme verstehen*

Beispiellösungen zur Grundkursaufgabe Datenbanksysteme

Teilaufgabe b)



Teilaufgabe c)

```
Film (FilmNr, Titel, Dauer, RegNr)
Regisseur (RegNr, Vorname, Nachname, GebJahr, Land)
Vorstellung (VorstellungsNr, Datum, Uhrzeit, VerkauftePlaetze, FilmNr,
SaalNr)
Saal (SaalNr, Sitzplaetze)
```

Primärschlüssel sind unterstrichen, Fremdschlüssel sind kursiv.

Die drei Relationen sind 1:n-Beziehungen zwischen den Entitätstypen E_1 und E_2, wobei E_2 jeweils obligatorisch ist. Deshalb können sie im relationalen Datenbankmodell durch einen Fremdschlüsselverweis in der E_2 zugeordneten Tabelle dargestellt werden.

3.2 Leistungskursaufgabe: Datenbanksysteme (39 %)

Ein Berliner Kino erfasst das Wochenprogramm bisher in Form von zwei Tabellen:

Tabelle 1:

Filmtitel	Dauer in min	Regisseur (Name, Land, Geburtsjahr)	Datum	Uhrzeit	Saal-Nr.	Anzahl Sitzplätze
Ein Quantum Trost	106	Marc Forster,GER,1969	04.12.	20:00	1	475
Ein Quantum Trost	106	Marc Forster,GER,1969	04.12.	20:00	2	412
Wall-E	110	Andrew Stanton, USA ,1965	04.12.	19:30	6	278
Wall-E	110	Andrew Stanton, USA ,1965	04.12.	22:00	3	121
Casablanca	104	Michael Curtiz, USA, 1988	04.12.	21:00	5	80
Drachenläufer	127	Marc Forster,GER,1969	04.12.	19:00	4	174
…	…	…	…	…	…	…

Tabelle 2:

Saalnummer	Datum	Uhrzeit	Reihe	Platznummer	Kategorie	Gebucht
1	04.10.	20:00	1	1	Loge	ja
1	04.10.	20:00	1	2	Loge	nein
1	04.10	20:00	1	3	Loge	ja
…	…	…	…	…	…	…
8	04.10.	22:00	12	16	Parkett	nein

Ein Informatikkurs wird beauftragt, eine relationale Datenbank zur Verwaltung des Kinoprogramms zu erstellen. Folgende Vorgaben sind dabei einzuhalten: Für jeden Sitzplatz sind die Kategorie (Loge bzw. Parkett), die Reihe und seine spezielle Platznummer in der Reihe anzugeben. Zu jeder Vorstellung eines Films soll die Gesamtzahl der bisher verkauften Kinokarten gespeichert werden.

a) Erläutern Sie die Nachteile, die sich ergeben, wenn die Daten in der oben angegebenen Form gespeichert sind.

b) Gliedern Sie die Tabellendaten in Entitäten und geben Sie die Beziehungen zwischen ihnen im Entity-Relationship-Modell (ERM) an. Markieren Sie die jeweiligen Schlüssel. Bestimmen Sie die Kardinalitäten der modellierten Beziehungen. Begründen Sie Ihre Entscheidungen.

c) Erstellen Sie an Hand des ERM das relationale Datenbankmodell für die Verwaltung des Kinoprogramms. Erläutern Sie Ihre Vorgehensweise.

d) Geben Sie jeweils eine geeignete SQL-Anweisung an, die folgende Abfragen erfüllt:
   d1) Erstellen Sie die Tabelle „Film“.
   d2) Wann läuft der Film „Ein Quantum Trost“? Angezeigt werden soll der Filmtitel, Datum und Uhrzeit. Wählen Sie eine geeignete Sortierung.
   d3) Gesucht ist eine Liste der nicht ausverkauften Vorstellungen des Films mit dem Titel „Ein Quantum Trost“. Angezeigt werden sollen der Filmtitel, die Saalnummer sowie Uhrzeit und Datum der Vorstellung.
   d4) Bestimmen Sie alle Filmvorstellungen, die mindestens 50 Zuschauer mehr hatten als die durchschnittliche Zuschauerzahl aller Vorstellungen. Gesucht sind Titel des Films, Anzahl der verkauften Plätze, Saalnummer sowie Datum und Uhrzeit der Vorstellung.

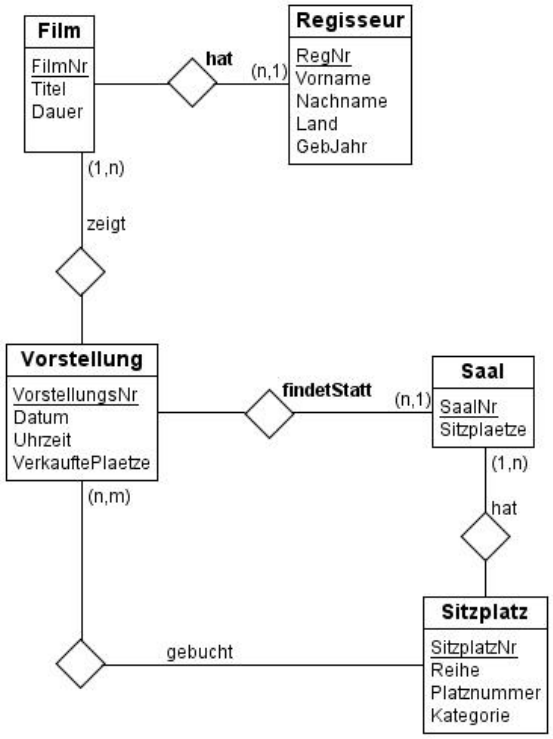
Teilaufgabe	a	b	c	d
Anteil an der Gesamtleistung ca.	3 %	13 %	9 %	14 %

Erwartungshorizont zur Leistungskursaufgabe Datenbanksysteme

Aufgabe	Erwartete Teilleistung	BE im AB I	II	III	Didaktischer Zusammenhang zum erteilten Unterricht / *Kompetenzbezug*
a	Redundanzen mit den entsprechenden Anomalien; außerdem sind die Werte in der Regisseur-Spalte nicht atomar		3		Wurde behandelt; erfordert Übersicht. *Kommunizieren*
b	Notieren des ER-Diagramms unter Beachtung einer normgerechten Darstellung: - Entitätstypen mit den Primärschlüsseln und den Beziehungen - Angabe der Kardinalitäten - Begründungen Eine mögliche Lösung befindet sich im Anhang.	7	3 3		Das Erstellen von ERM wurde im Unterricht an einigen Beispielen geübt. *Informatisches Modellieren*
c	- Angabe der Tabellen inklusive Primär- und Fremdschlüssel (s. Anhang). - Erklärung der Vorgehensweise mit eigenen Worten	6 3			Abbildung von ERM nach RDM geschieht nach Regeln und wurde geübt. *Informatisches Modellieren*
d1	Mögliche Konstruktion der SQL-Abfragen: `CREATE Table Film (FilmNr INT, Titel CHAR(20),Dauer INT, REGNr INT)`	3			An anderen Beispielen im Unterricht behandelt
d2	`SELECT Titel,Datum,Uhrzeit FROM Film, Vorstellung WHERE Film.FilmNr = Vorstellung.FilmNr ORDER by Datum, Uhrzeit`	3			einfache JOIN über zwei Tabellen
d3	`SELECT Titel, Saal.SaalNr, Datum, Uhrzeit FROM Vorstellung, Saal, Film WHERE Saal.SaalNr = Vorstellung.SaalNr AND Vorstellung.FilmNr = Film.FilmNr AND Titel = 'Ein Quantum Trost' AND VerkauftePlaetze < Sitzplaetze`		4		anspruchsvoller JOIN
d4	`SELECT Titel, VerkauftePlaetze, Saal.SaalNr, Datum,Uhrzeit FROM Vorstellung, Saal, Film WHERE Saal.SaalNr = Vorstellung.SaalNr AND Vorstellung.FilmNr = Film.FilmNr AND VerkauftePlaetze >= (SELECT AVG(VerkauftePlaetze)- 50 FROM Vorstellung)`			4	anspruchsvoller Join mit INNER-SELECT *Informatiksysteme verstehen*
	Summe 39 BE	22	13	4	

Beispiellösungen zur Leistungskursaufgabe Datenbanksysteme

Teilaufgabe b)



Teilaufgabe c)

Film (FilmNr, Titel, Dauer, *RegNr*)

Regisseur (RegNr, Vorname, Nachname, GebJahr, Land)

Vorstellung (VorstellungsNr, Datum, Uhrzeit, VerkauftePlaetze,*FilmNr*, *SaalNr*)

Saal (SaalNr, Sitzplaetze)

Sitzplatz (SitzplatzNr, Reihe, Platznummer, Kategorie, *SaalNr*)

Buchung(*VorstellungsNr, SitzplatzNr*)

Primärschlüssel sind unterstrichen, Fremdschlüssel sind kursiv.

Die Beziehung „gebucht“ ist eine n:m-Beziehung und wird in eine eigenständige Tabelle übersetzt. Der zusammengesetzte Primärschlüssel besteht aus den beiden Fremdschlüsseln *VorstellungsNr* und *SitzplatzNr.*

Die anderen vier Beziehungen sind 1:n-Beziehungen zwischen den Entitätstypen E_1 und E_2, wobei E_2 jeweils obligatorisch ist. Deshalb können sie im relationalen Datenbankmodell durch einen Fremdschlüsselverweis in der E_2 zugeordneten Tabelle dargestellt werden.

4. Was macht einen guten Vorschlag aus?

In diesem Abschnitt werden Merkmale eines guten Vorschlags (vgl. Abschnitt 1.3) an Aufgaben aus den Themenfeldern „Informatik, Mensch und Gesellschaft“ (im GK) und „Applikative Programmierung“ (im LK) illustriert.

Bewertungseinheiten zu den Aufgaben dieses Kapitels sind bewusst nicht angegeben, weil dazu der Bezug zum Gesamtvorschlag erforderlich ist. Diese Thematik ist Gegenstand des Kapitels 2.

4.1 Grundkursaufgabe zum Thema „Informatik und Gesellschaft“

Die Aufgabe bezieht sich inhaltlich auf das verbindlich integrativ oder als Vertiefungsgebiet zu unterrichtende Themenfeld Informatik, Mensch und Gesellschaft. Sie zeigt die Möglichkeit, frei von jeglicher Programmierung ein informatisch wichtiges Themengebiet aus der Lebenswelt der Schülerinnen und Schüler zu berücksichtigen.

Hierbei bewerten die Schüler Risiken und Chancen von Informatiksystemen. Sie reflektieren ihr Recht auf informationelle Selbstbestimmung und untersuchen in einem konkreten Fall Fragen des Datenschutzes. Die Verwendung von kommerziellen Logos, Texten und Materialien aus Wirtschaft und Industrie stellt für den Aufgabenstellenden selbst die Frage der berechtigten Verwendung. Eine Berechtigung ist ggf. vom Rechteinhaber einzuholen[9]. Durch die Authentizität des verwendeten Materials wird die Nähe zur Lebenssituation der Schüler besonders deutlich. Dies vermeidet sonst notwendige Abstraktionen durch selbst konstruierte Fälle.

Die Aufgabenstellung lässt es noch offen, ob neben dem verwendeten Flyer weiteres Material, z. B. Gesetzestexte oder wirtschaftskundliche Veröffentlichungen bereitgestellt werden. Diese Aufgabe hat den für einen Grundkurs begrenzten Umfang und Schwierigkeitsgrad.

Grundkursaufgabe: Auswirkungen von Datensammlungen[10]

Studieren Sie das in Anlage A dargestellte Material zum Bonusprogramm *HappyDigits*.

a) Erläutern Sie Vorteile, die die an dem Programm beteiligten Firmen erzielen.

b) Setzen Sie die angebotenen Prämien in Beziehung zu den im Gegenzug zu tätigenden Einkäufen und kommentieren Sie das Kosten-Nutzen-Verhältnis auf Seiten der teilnehmenden Firmen sowie der teilnehmenden Kunden.

c) Erläutern Sie, welche Voraussetzung gegeben sein muss, damit das Verhalten der beteiligten Firmen im Einklang mit dem Bundesdatenschutzgesetz steht.

[9] Das hier dargestellte Material darf für diese Handreichung verwendet werden.

[10] Hier folgt im realen Aufgabenvorschlag die Angabe des Aufgabengewichts, die von der sonstigen Aufgabestellung abhängig ist.

Ihr Online-Shopping-Führer: So kommen Sie zu vielen ExtraDigits!

Partner	Sammel-Chancen[1]
Beautynet	4 Digits pro 1 €
Condor	500 ExtraDigits pro Buchung
Neu: DB Autozug	900 ExtraDigits pro Buchung
dress-for-less.de	5 Digits pro 1 €
Neu: EPSON Store	4 Digits pro 1 €
Fleurop	5 Digits pro 1 €
getgo.de	1 Digit pro 1 €
Hagebau.de	3 Digits pro 1 €
Hamburger Akademie für Fernstudien	5.000 ExtraDigits pro Abschluss
HappySize	4 Digits pro 1 €
ImmobilienScout24	bis zu 7.000 ExtraDigits
Lenscare	11 Digits pro 1 €
LOVEFILM	1.000 ExtraDigits
Neu: myby	3 Digit pro 1 €
mydays	8 Digits pro 1 €
Notenbuch.de	5 Digits pro 1 €
onlineKiosk	16 Digits pro 1 €
posterXXL	7 Digits pro 1 €
Premiere	bis zu 3.400 ExtraDigits
Neu: ReifenDirekt.de	1 Digit pro 1 €
Rossmann Versand	3 Digits pro 1 €
SCHLECKER.com	3 Digits pro 1 €
Spreadshirt Designer	12 Digits pro 1 €
Swarovski	6 Digits pro 1 €
Thalia.de	bis zu 2 Digits pro 1 €
Tom Tailor	5 Digits pro 1 €
Neu: T-Online Shop	450 ExtraDigits pro Bestellung
Neu: TravelScout24	bis zu 3.500 ExtraDigits
Valentins	10 Digits pro 1 €
Vistaprint.de	250 ExtraDigits
zooplus	2 Digits pro 1 €

Jetzt sammeln: happydigits.de/shop[2]

@ Gleich reinschauen auf www.happydigits.de/shop[2]

[1]Digits werden auf den Nettoumsatz ohne MwSt., Versandkosten und optionale Servicegebühren vergeben. Änderungen sind vorbehalten. Ihre Digits werden Ihnen 8 Wochen nach Kauf gutgeschrieben. | [2]Bitte beachten Sie, dass Sie nur über das Aufrufen dieses HappyDigits Links Digits bei den Online-Partnern sammeln können.

Das Bonusprogramm
HappyDigits

Hier blühen Ihnen noch mehr Vorteile!

Digits sammeln

ExtraDigits auf Empfehlung! HappyDigits sucht RegioPartner.

Gibt es in Ihrer Nachbarschaft Geschäfte, in denen Sie besonders gerne einkaufen? Dann empfehlen Sie uns diese doch gleich weiter, damit Sie auch dort viele Digits sammeln können! Wird Ihr empfohlenes Wunsch-Geschäft tatsächlich zum HappyDigits RegioPartner, belohnen wir Sie mit **1.000 ExtraDigits!**

@ **Jetzt Wunsch-Geschäft empfehlen: www.happydigits.de/regio**

Bei allen HappyDigits Partnern sammeln:

T · KARSTADT · KAISER'S · KaDeWe · QUELLE. · RUNNERS POINT · SIXT rent a car · Hertie · SAT.1 · Gesund ist bunt APOTHEKEN · neckermann.de · Yello Strom · TENGELMANN

Prämien auswählen

Frohe Ostern mit den HappyDigits Frühjahrs-Prämien!

1.690 Digits

Clatronic Eierkocher

In diesem Edelstahl-Eierkocher verstecken sich Ostereier am liebsten. Ausgestattet mit Kontrollleuchte und Summer, inkl. Ei-Halter mit Eipicker und Messbecher. Für bis zu 7 Eier. 350 Watt.

Best.-Nr. 1-1117883401 1.690 Digits

je 820 Digits

Plüschhase

Hopp, hopp ins Körbchen. Diese süßen Plüschhasen sind aus kuschlig-weicher Microfaser, inkl. Ohren 14 cm hoch und bevorzugen liebevolle Handwäsche. Wahlweise in Braun oder Beige.

Plüschhase braun
Best.-Nr. 1-1117888341 820 Digits
Plüschhase beige
Best.-Nr. 1-1117880941 820 Digits

1.590 Digits

Solarradio

Frischer Frühlings-Hit: Dank Solarzellen spielt dieses UKW-Radio schon heute Ihre Lieblingslieder umweltfreundlich ohne Batterie. Ausstattung inklusive Teleskopantenne, herausklappbarem Aufsteller und Ohrhörer. Batteriebetrieb möglich, Maße: ca. 12 x 7 x 2,5 cm.

Best.-Nr. 1-1117873471 1.590 Digits

2.290 Digits

Gartentasche

Rein ins Grüne: Die waschbare Gartentasche ist das perfekte Ordnungssystem für Garten, Balkon und Terrasse. Denn sie ist ideal als Arbeitsfläche und Unterlage für Garten- und Erdarbeiten, beim Laubrechen, Holztragen oder als Stauraum für Ihre Gartenartikel. Aus Polyester. Ø: ca. 1,5 m.

Best.-Nr. 1-1117885391 2.290 Digits

Gleich bestellen unter: 01805-88 00 30* oder auf www.happydigits.de/praemien

Tipp: Im Internet finden Sie über 300 weitere Prämien und praktische Zuzahlungs-Möglichkeiten.

Mögliche Antworten:

a) Durch das Prämienprogramm wird eine höhere Kundenbindung erreicht: Die Kunden kaufen ein Produkt eher bei einem teilnehmenden Anbieter, da er so auch seinen HappyDigit-Stand erhöhen kann. Gleichzeitig werden die Firmen über das Programm beworben, so dass die Teilnahme am Programm auch neue Kunden anlockt.
Darüber hinaus lassen sich aus der dauerhaften Zuordnung von Einkäufen zu den Kunden Kundenprofile erstellen. Dies lässt eine personalisierte und damit deutlich effektivere Form der Werbung aber auch insgesamt umfangreiche Marktanalysen zur Optimierung von Marketing und Produktpalette zu.

b) Im Schnitt werden etwa 3 bis 6 Digits pro Euro Netto-Einkaufswert gutgeschrieben. Für einen Plüschhasen muss also ein Einkauf im Wert von ca. 135 bis 275 € netto, d. h. ca. 160 bis 325 € inkl. MwSt. getätigt werden. Für die Gartentasche sind es entsprechend Einkäufe im Wert von ca. 450 bis 900 € inkl. MwSt. Geht man nun davon aus, dass die Prämien eher im Niedrigpreis-Sektor eingekauft werden und von einfacher Qualität sind, so scheinen sie im Verhältnis zu den zu investierenden Beträgen wenig attraktiv für den Kunden.

c) Die Verwendung der personenbezogenen Daten muss auf die Verwaltung der Digits und den Versand der Prämien beschränkt sein. Eine weitergehende Nutzung für Werbezwecke oder eine Weitergabe an Dritte bedarf der ausdrücklichen Genehmigung der Kunden. Um dies zu gewähren, muss der Zugang zu den Daten durch entsprechende Organisation auf Mitarbeiter beschränkt sein, die mit der Verwaltung der Daten beauftragt sind (z.B. Authentifizierung mit Login und Passwort an allen Rechnern, von denen aus auf die Daten zugegriffen werden kann, Verschließen von Räumen bei Abwesenheit, Vergabe von benutzergebundenen Zugriffsrechten für im Netzwerk freigegebene Ressourcen, …).

4.2 Leistungskursaufgabe zum Paradigma der applikativen / prädikativen Programmierung

Die vorliegende Aufgabe bezieht sich inhaltlich auf den im Leistungskursfach verbindlichen Schwerpunkt der deklarativen Programmierung (funktional oder logisch) aus dem Themenfeld Softwareentwicklung. Auf der Grundlage des tatsächlichen, aber hier fiktiven vorangegangenen Unterrichts werden die abschlussorientierten Standards eines Leistungskurses erfasst mit der Thematik des nichtimperativen Paradigmas. Wie schon im Abschnitt 1.3 dargestellt, wird eine Aufgabe durch wesentliche und ergänzende Inhalte und Kompetenzen charakterisiert. Die Bewertung durch den Autor der Beispielaufgabe sieht folgende wesentliche Bestandteile vor:

- eine wichtige rekursive Algorithmik,
- die dynamische Datenstruktur Liste,
- die Anwendung eines nichtimperativen Sprachparadigmas,
- eine Programmierung im Kleinen.

Den ergänzenden Rahmen bilden:

- ein verbaler Paradigmenvergleich,
- eine selbständige Komplexitätsabschätzung,
- die Verwendung von Hilfsfunktionen,
- das Aufstellen von Testfällen.

In der Aufgabe wird die nicht imperative Programmierung am Beispiel des Sortierverfahrens Mergesort mit dem Inhalt Algorithmen und Datenstrukturen vernetzt. Dabei werden rekursive Verfahren angewandt, die nur für den Leistungskurs verbindlich sind (vgl. Rahmenlehrplan 3.2, abschlussorientierte Standards).

In Teilaufgabe a) analysiert der Prüfling einen vorgegebenen und aus dem Unterricht bekannten Algorithmus. Bei der Beschreibung der einzelnen Funktionen in der konkreten Implementierung des Algorithmus in Teilaufgabe b) zeigt er, dass er den vorgegebenen Sachverhalt und die informatischen Zusammenhänge beschreiben kann. Dabei wendet er unter Umständen im Unterricht erlernte Methoden der Visualisierung von Algorithmen in Tabellen, Diagrammen o.ä. an. In der in Teilaufgabe c) geforderten Auswahl von Testläufen muss der Prüfling alle wesentlichen Sonderfälle beachten und angemessen darstellen.

Im Leistungskursfach befassen sich die Schülerinnen und Schüler systematischer und vertiefter als im Grundkursfach mit komplexen Sachverhalten und theoretischen Fragestellungen. Diese Aspekte werden in den weiteren Teilaufgaben berücksichtigt.

Teilaufgabe d) handelt von der Zuordnung von Algorithmen zu Komplexitätsklassen. Der Prüfling interpretiert die gegebene Klasse O(log n * n) bezüglich des Mergesort. Auch hier ist eine anschauliche Darstellung gefordert, z. B. anhand eines konkreten Beispiels in Baumstruktur. Zusätzlich vergleicht und bewertet er O(log n * n) gegebenenfalls mit Komplexitätsklassen anderer Verfahren. Die Offenheit der Aufgabenstellung ermöglicht den Vergleich mit Quicksort oder einem einfachen Sortierverfahren der Komplexitätsklasse O(n*n). Die formale Genauigkeit der erwarteten Lösung hängt vom vorangegangenen Unterricht ab.

In Teilaufgabe e) wendet der Prüfling ein deklaratives Sprachparadigma zur Modellierung und Implementierung an. Zur Lösung der Aufgabe sind die Ergebnisse der vorherigen Teilaufgaben (rekursive Implementierung des Mergesort) und Erfahrungen aus dem vorangegangenen Unterricht (z. B. Implementierung der binären Suche in einer imperativen Sprache) auf einen anderen Sachverhalt zu übertragen. Dazu werden vorgegebene Schnittstellen (Hilfsfunktionen) der verwendeten Programmiersprache angewandt. Dies setzt entsprechende Übung und damit Kenntnis davon voraus. Im Leistungskurs erscheint dies angemessen.

In Teilaufgabe f) steht die Unterscheidung von Vor- und Nachteilen funktionaler bzw. regelbasierter Modellierung im Vordergrund. Ausgehend vom Mergesort-Algorithmus liegt eine Aufgabe mit großer Offenheit vor. Nur der Leistungskurs bietet ausreichende Erfahrungen mit verschiedenen Paradigmen **und** Algorithmen. Bei der Lösung zeigt der Schüler, dass er den Zusammenhang in angemessener Fachsprache und in eigenen Worten darstellen kann.

Leistungskursaufgabe Mergesort [11]

Gegeben ist eine Implementierung des Sortierverfahrens Mergesort (siehe Anlage):

a) Erläutern Sie den Algorithmus dieses Sortierverfahrens!

b) Beschreiben Sie die Arbeitsweise der einzelnen Funktionen (bzw. Prädikate) `merge`, `split` und `mergesort`.

c) Um die Wirksamkeit der Implementierung zu untersuchen, sind Testläufe erforderlich. Stellen Sie neben der gegebenen Liste L = [16, 37, 31, 10, 27] weitere für die Listenverarbeitung typische Fälle zusammen und begründen Sie Ihre Auswahl.

d) Hinsichtlich der erforderlichen Zeit wird die Komplexitätsklasse dieses Algorithmus mit O(n * log n) angegeben. Begründen Sie diese Angabe.

e) Um ein Element in einer **sortierten** Liste zu suchen, bietet sich das Verfahren der binären Suche an:

Man vergleicht jeweils das mittlere Element der Liste mit dem zu suchenden Element. Ist das mittlere Element das gesuchte, so ist man fertig, ansonsten wendet man den Algorithmus rekursiv auf die Teilliste an, in der das Element auf Grund des Vergleiches liegen muss (je nachdem, ob das Vergleichselement kleiner oder größer als das gesuchte Element ist).

Implementieren Sie für dieses Verfahren eine rekursive Funktion `binSuch`, die den Wert `True` liefert, falls sich das gesuchte Element in der sortierten Liste befindet. Sie dürfen die Hilfsfunktionen aus der Anlage verwenden.

f) Beschreiben Sie ausgehend vom Beispiel der oben angegebenen Implementierung von Mergesort wesentliche Merkmale der applikativen / prädikativen Programmierung gegenüber der imperativen Programmierung.

[11] Auch hier folgt im realen Aufgabenvorschlag die Angabe des Aufgabengewichts, die aber von der sonstigen Aufgabenstellung abhängig ist.

Anlage zur Leistungskursaufgabe Mergesort

Implementierung in einer funktionalen Programmiersprache, Code in Haskell

```
merge :: Ord a  => [a] -> [a] -> [a]
merge [] ys  = ys
merge xs []  = xs
merge (x:xs) (y:ys)
  | x<=y = x:merge xs (y:ys)
  | x>y  = y:merge (x:xs) ys

split :: Ord a => [a] -> ([a],[a])
split [] = ([],[])
split [x] = ([x],[])
split (x:y:zs) = (x:xs,y:ys)
  where (xs,ys) = split zs

mergesort:: Ord a => [a] -> [a]
mergesort [] = []
mergesort [x] = [x]
mergesort xs= merge (mergesort links) (mergesort rechts)
     where (links,rechts) = split xs
```

Anlage zur Teilaufgabe e (Hilfsfunktionen, Code in Haskell)

```
binSuch :: Ord a => a -> [a] -> Bool
take::Int -> [a]->[a]     -- liefert die ersten n Elemente einer Liste
drop::Int -> [a]->[a]     -- liefert die Liste ohne die ersten n Elemente
```

alternative Implementierung in einer deklarativ/logischen Programmiersprache, Code in PROLOG

```
merge_sort([], []).
merge_sort([Element], [Element]).
merge_sort(Liste, SortierteListe):-
split(Liste, Liste1, Liste2),
merge_sort(Liste1, SortierteListe1),
merge_sort(Liste2, SortierteListe2),
merge(SortierteListe1,SortierteListe2,SortierteListe).

merge([Kopf1|Liste1], [Kopf2|Liste2], [Kopf1|Liste3]):-
Kopf1 =< Kopf2, !,
merge(Liste1, [Kopf2|Liste2], Liste3).
merge(Liste1, [Kopf2|Liste2], [Kopf2|Liste3]):-
merge(Liste1, Liste2, Liste3).
merge([], Liste, Liste).
merge(Liste, [], Liste).

split([], [], []).
split([Element], [Element], []).
split([Element, Element2|Rest1], [Element|Rest2], [Element2|Rest3]):-
split(Rest1, Rest2, Rest3).
```

Alternative: Funktionale Implementierung, Code in Python

```
def merge(L1,L2):
       if   L1 == []: return L2
       elif L2 == []: return L1
       else:
              if L1[0] < L2[0]:
                     return [L1[0]] + merge(L1[1:],L2)
              else  return [L2[0]] + merge(L1,L2[1:])

def split(L):
       if L == []: return ([],[])
       elif L[1:] == []: return ([L[0]], [])
       else:
              (M, N) = split(L[2:])
              return ([L[0]] + M, [L[1]] + N)

def merge_sort(L):
       if L == [] : return []
       elif L[1:] == [] : return [L[0]]
       else:
              (M,N) = split(L)
              Return merge(merge_sort(M), merge_sort(N))
```

Musterlösung zur Teilaufgabe e, Code in Haskell

```
binSuch :: Ord a => a -> [a] -> Bool
binSuch v [] = False
binSuch v [x]
   | v==x = True
   | otherwise = False
binSuch v xs
  | v==mitte = True
  | v<mitte = binSuch v (take half xs)
  | otherwise = binSuch v (drop half xs)
  where
      half = div (length xs) 2
      mitte = head (drop half xs)
```

5 Literatur

5.1 Vorgaben etc., Stand: 24.06.2008

EPA Informatik
http://www.kmk.org/fileadmin/veroeffentlichungen_beschluesse/1989/1989_12_01-EPA-Informatik.pdf

EPA Berufliche Informatik
http://www.kmk.org/fileadmin/veroeffentlichungen_beschluesse/1979/1979_06_01-EPA-berufliche-Informatik.pdf

Vereinbarung über die Abiturprüfung der gymnasialen Oberstufe in der Sekundarstufe II
http://www.kmk.org/doc/beschl/abi-03.pdf

Senatsverwaltung für Bildung, Wissenschaft und Forschung, LISUM Berlin-Brandenburg:

Rahmenlehrplan Sek II Berlin 2006
http://www.berlin.de/imperia/md/content/sen-bildung/schulorganisation/lehrplaene/sek2_informatik.pdf

Fachbriefe Informatik
http://www.bjsinfo.verwalt-berlin.de/index.aspx?id=135

Anregungen zur Rahmenlehrplanumsetzung Sek II
http://bildungsserver.berlin-brandenburg.de/rahmenplaene_inf.html

Ausführungsvorschrift Prüfungen
http://www.berlin.de/imperia/md/content/senbildung/rechtsvorschriften/av_pruefungen_ab2010.pdf

5.2 Aufgabenbeispiele, Stand: 24.06.2008

Nordrhein-Westfalen
http://www.standardsicherung.schulministerium.nrw.de/abitur-gost/fach.php?fach=15

Mecklenburg-Vorpommern
http://www.bildung-mv.de/export/sites/lisa/de/abitur2008/Beispielaufgaben_2008_pdf-Format/Informatik.pdf

Baden-Württemberg
http://www.schule-bw.de/schularten/berufliche_schulen/vollzeitschulen/berufliche_gymnasien/musteraufg_abi/mu_aufg_infomanag.htm

Sachsen
DUDEN PAETEC Schulbuchverlag, zentrale Aufgaben