Aufgabe 1

1. Beschreiben … werden.

*Kunde hat die Attribute name, adresse und telefon (jeweils String), zudem wird von der Kontenliste das erste und das aktuelle Konto gespeichert. Ein Kunde wird mit dem Namen erzeugt, dieser darf nicht mehr geändert werden (nur gelesen). Alle anderen Attribute können gelesen und geschrieben werden. Alle Atrribute sind private, alle Klassen public.*

*Konto ist eine abstrakte Klasse mit den Attributen kontoNr und kontoStand, sowie einem Zeiger auf das nächste Konto der Liste. Ein Konto wird mit seiner Kontonummer erzeugt, diese kann nicht mehr geändert werden. Kontostand kann gelesen und über einzahlen / auszahlen geändert werden. Der Zeiger auf das nächste Konto kann gelesen und geschrieben werden. Attribute sind protected bzw. private, die Methoden sind public.*

*Es besteht eine rekursive Assoziation von Konto zu sich selbst, damit sind die Konten in einer Liste gespeichert. Diese kann mit den Methoden firstKonto und nextKonto durchlaufen werden.*

*Zudem besteht eine Assoziation von Kunde zu Konto. Diese wird durch die Methoden appendKonto, deleteKonto und getKontoCurrent verwaltet.*

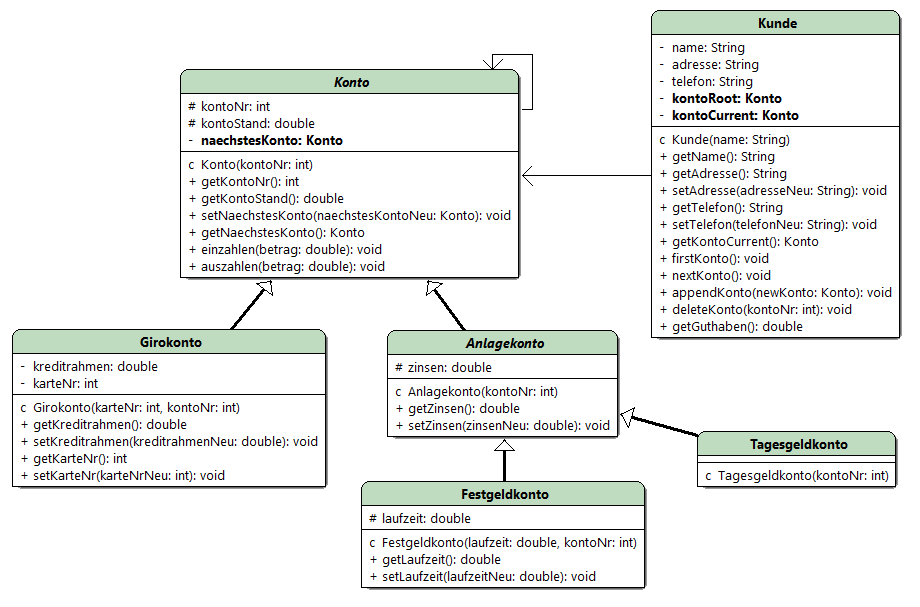
1. Implementieren … werden.)

|  |
| --- |
| **public** **abstract** **class** Konto **{**    **protected** **int** kontoNr**;**  **protected** **double** kontoStand**;**  **private** Konto naechstesKonto**;**    **public** Konto**(int** kontoNr**)** **{**  **this.**kontoNr **=** kontoNr**;**  **this.**kontoStand **=** 0**;**  **this.**naechstesKonto **=** **null;**  **}**  ...  **public** **void** einzahlen**(double** betrag**)** **{**  kontoStand **=** kontoStand **+** betrag**;**  **};**    **public** **void** auszahlen**(double** betrag**)** **{**  **if** **(**kontoStand **>=** betrag**)** **{**  kontoStand **=** kontoStand **-** betrag**;**  **}**  **};**  **}** |

1. Implementieren … zurückgibt.

|  |
| --- |
| **public** **double** getGuthaben**()** **{**  firstKonto**();**  **double** g **=** kontoCurrent**.**getKontoStand**();**    **while** **(k**ontoCurrent**.**getNaechstesKonto**()** **!=** **null)** **{**  kontoCurrent = kontoCurrent**.**getNaechstesKonto**());**  g **=** g **+** kontoCurrent**.**getKontoStand**();**  **}**    **return** g**;**  **}** |

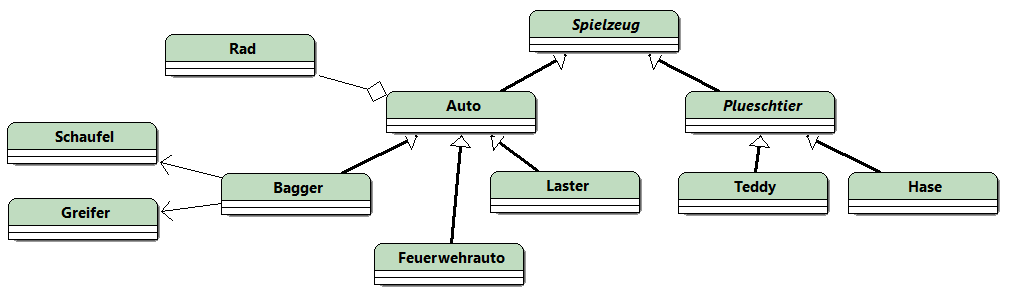
1. Ergänzen … Vererbung.



|  |
| --- |
| **public** **class** Festgeldkonto **extends** Anlagekonto **{**    **protected** **double** laufzeit**;**    **public** Festgeldkonto**(double** laufzeit**,** **int** kontoNr**)** **{**  **super(**kontoNr**);**  **this.**laufzeit **=** laufzeit**;**  **}**  **public** **double** getLaufzeit**()** **{**  **return** laufzeit**;**  **}**  **public** **void** setLaufzeit**(double** laufzeitNeu**)** **{**  laufzeit **=** laufzeitNeu**;**  **}**  **}** |

Aufgabe 2

1. Entwerfen … Methoden).



1. Erläutern … Oberklasse.

*Vererbung: Weitergabe von Strukturen, neue Klassen entstehen durch Abwandlung vorhandener Klassen, Weitergabe bzw. Überschreiben von Attributen und Methoden*

*Generalisierung: Aufwärtsbeziehung zwischen Objekten (Feld -> Fläche)*

*Spezialisierung: Abwärtsbeziehung zwischen Objekten (Fläche -> Feld)*

*abstrakte Klasse: Klasse, von der keine Objekte / Instanzen angelegt werden (Gebäude)*

*Aufruf von Methoden der Oberklasse: super(), Funktionalität der Oberklasse wird genutzt*

1. Erläutern … Beziehung.

*Komposition würde existentielle Abhängigkeit bedeuten, d. h. wird der Bagger gelöscht, ist auch die Schaufel mit weg – dies ist nach Aufgabenstellung nicht gewollt.*

*Aggregation (oder Assoziation), da Bagger die Schaufel hat (kennt)*

Aufgabe 3

1. Abstrakte … Eigenschaft.

*Der Anwender kennt nur den Wertebreich und die darauf definierten Operationen / Methoden eines ADT. Wie die einzelnen Methoden arbeiten ist außerhalb des ADT nicht bekannt; Zugriff auf Attribute geschieht ausschließlich über Methoden.*

1. Im … Methode.

*einfach verkettet Liste, insertBefore + Begründung*

1. Erläutern … ein.
2. *previous-Zeiger des nächsten Elements auf vorheriges Element*
3. *next-Zeiger des vorherigen Elements auf nächstes Element*
4. *current auf erstes Element*
5. *(Sonderfall erstes und letztes)*

Inhalt

Inhalt

Inhalt

**root**

**current**

Inhalt

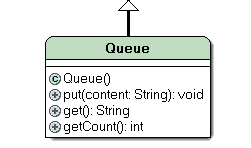
1. Implementieren … Liste.

|  |
| --- |
| **public** **void** delete**()** **{**  **if** **(!**isEmpty**())** **{**  **if** **(**current **==** root**)** **{**  root **=** root**.**getNext**();**  **if** **(**root **!=** null**)**  root**.**setPrevious**(null);**  **}** **else** **{**  **if** **(!**isLast**())**  current**.**getNext**().**setPrevious**(**current**.**getPrevious**());**  current**.**getPrevious**().**setNext**(**current**.**getNext**());**  **}**  **}**  current **=** root**;**  count**--;**  **}** |

1. Erläutern … erfolgen: Geben … an.

*Lösung1.doc – Lösung2.xls – Gertrud.jpg // Beschreibung der Vorgehensweise*

1. Ähnlich … macht.

*Abstrakte Klassen, z. B. AbstractListe, sind Klassen, von der keine Objekte erzeugt werden (können). Sie bilden die Basis für andere Klassen, in unserem Fall Liste, Stack und nun auch Queue. Abstrakte Klassen besitzen nur eine sehr allgemeine Funktionalität, die in der Regel nicht sinnvoll eingesetzt werden kann. Queue dagegen nutzt AbstractListee um eine spezielle Funktionalität bereitzustellen.*

1. Implementieren Sie die Methoden put() und get() mit Hilfe der Klasse AbstractList.

Mögliche Lösung:

|  |
| --- |
| **public** **void** put**(**String content**)** **{**  **super.**append**(**content**);**  **}**    **public** String get**()** **{**  **super.**first**();**    String content **=** **super.**getContent**();**  **super.**delete**();**  **super.**last**();**    **return** content**;**  **}** |

1. Stellen … erfolgen:

*Birgit.doc / 2 – Blatt.xls / 2 – Rechnung.xls / 3 – Lösung.doc / 4*