Karlsruher Institut für Technologie

Programmieren – Wintersemester 2017/18

Architecture-driven Requirements Engineering (ARE)
`https://sdqweb.ipd.kit.edu/wiki/Programmieren`
Jun.-Prof. Dr.-Ing. Anne Koziolek

Übungsblatt 3

Ausgabe: 06.12.2017 – 13:00
Abgabe: 21.12.2017 – 06:00

- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich. Die Kommentare sollen einheitlich in entweder englischer oder deutscher Sprache verfasst werden.
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klasse, Methoden und Attribute.
- Verwenden Sie keine Klassen der Java-Bibliotheken ausgenommen Klassen der Pakete `java.lang.*` und `java.io.*`, es sei denn die Aufgabenstellung erlaubt ausdrücklich weitere Pakete.
- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie die Regeln zu Variablen-, Methoden- und Paketbenennung ein und wählen Sie aussagekräftige Namen.
- Allgemeiner Hinweis: Bei Regelverletzung wird der Praktomat Ihre Abgabe zurückweisen.

Abgabemodalitäten

Die Praktomat-Abgabe wird am **Mittwoch, den 13. Dezember 2017, um 13:00 Uhr**, freigeschaltet.

- Geben Sie Ihre Antworten zu Aufgabe A als `*.java`-Dateien ab.
- Geben Sie Ihre Antworten zu Aufgabe B als `*.java`-Dateien ab.
- Geben Sie Ihre Antworten zu Aufgabe C als `*.java`-Dateien ab.

Achten Sie unbedingt darauf, diese Dateien im Praktomat bei der richtigen Aufgabe hochzuladen. Falsch hochgeladene Abgaben werden nicht bewertet. Denken Sie daran, den Checkstyle-Regelsatz zu beziehen. Bitte beachten Sie, dass das erfolgreiche Bestehen der öffentlichen Tests für eine erfolgreiche Abgabe nötig ist. Planen Sie für Ihren ersten Abgabeversuch entsprechend Zeit ein. Bitte beachten Sie: Sollten Sie Probleme bei der Abgabe bzw. beim Hochladen Ihrer Dateien haben, verwenden Sie einen anderen Browser. Der Internet Explorer 11 kann Probleme verursachen.

Objektorientierte Modellierung
Achten Sie darauf, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung, als auch Funktionalität bewertet werden.

A Objektorientierte Modellierung: Bankinstitut (8 Punkte)

In dieser Aufgabe soll eine modifizierte Variante der Aufgabe D des letzten Aufgabenblattes modelliert werden. Beim letzten Übungsblatt wurden die Klassen `Main`, `Account` und `AccountHolder` mit ihren Membern spezifiziert. Für diese Aufgabenstellung werden die Klassen `Account` und `Bank` definiert. Versehen Sie im Folgenden Attribute und/oder Methoden mit passenden Sichtbarkeitsmodifizierern. Es soll nicht das `default`-Package genutzt werden. Legen Sie ein eigenes Package mit der Bezeichnung `edu.kit.informatik` an. Überlegen Sie sich bei dieser Aufgabe, ob es sich bei den jeweiligen Attributen um Klassen- oder Objektvariablen handelt. Erweitern Sie alle Klassen um geeignete `getter`- und/oder `setter`-Methoden. Sind `getter`- und/oder `setter`-Methoden für ein Attribut mit dem Namen `Attributname` sinnvoll vorhanden, nennen Sie die entsprechenden Methodennamen `getAttributname` und/oder `setAttributname`. Zum Beispiel lauten die `getter`- und `setter`-Methoden für ein Attribut `String name` `getName` und `setName`. Verfassen Sie jeweils ein Kommentar, der kurz darlegt, warum Sie sich für oder gegen die `getter`-/`setter`-Methoden entschieden haben. Eine Spezifikation der `main`-Methode ist nicht erforderlich.
Denken Sie daran, **keine Klassen** aus `java.util.*` zu verwenden.

A.1 Klasse `Account`

Die Klasse `Account` stellt ein Konto dar und besitzt folgende Attribute: `int accountNumber` für die Darstellung der Kontonummer, `int bankCode` für die Darstellung der Bankleitzahl und `int balance` für die Darstellung des Kontostandes. Wie in Übungsblatt 2 können Sie annehmen, dass sowohl Kontonummer, Bankleitzahl und Kontostand positive ganze Zahlen sind, als auch die Kontonummern stets eindeutig sind.
Der Konstruktor der Klasse `Account` nimmt Bankleitzahl und Kontonummer genau in dieser Reihenfolge entgegen und initialisiert diese.

Zudem besitzt die Klasse zwei Methoden, die im Folgenden näher spezifiziert werden:

- `boolean withdraw(int amount)`: Diese Methode repräsentiert das Auszahlen. Dabei wird der Kontostand um den übergebenen Betrag (`amount`) verringert. Wird der Kontostand durch die Auszahlung negativ, findet die Auszahlung nicht statt, d.h. der Kontostand wird nicht geändert. In diesem Fall wird `false` zurückgegeben. Andernfalls wird `true` zurückgegeben.
- `void deposit(int amount)`: Diese Methode repräsentiert das Einzahlen. Dabei wird der Kontostand um den übergebenen Betrag (`amount`) erhöht.

A.2 Klasse `Bank`

Implementieren Sie eine Klasse `Bank` zur Darstellung der Banken. Die Klasse `Bank` hat eine Bankleitzahl (`bankCode`) vom Typ `int` und ein Array von Bankkonten (`Account`) mit einer initialen Länge von 8.
Implementieren Sie einen geeigneten Konstruktor, der die Bankleitzahl entgegennimmt und diese initialisiert.

Implementieren Sie zudem folgende Methoden für die Klasse `Bank`:

- `int createAccount(int accountNumber)`: Diese Methode repräsentiert das Anlegen eines Kontos mit der Kontonummer `accountNumber` innerhalb der Bank. Jedes Konto, das innerhalb dieser Bank erstellt wird, bekommt die Bankleitzahl der Bank. Das neue Konto wird im ersten freien Feld des Arrays der Konten innerhalb der Bank eingefügt. Die Methode gibt schließlich zurück, zu welchem Array-Index das Konto hinzugefügt wurde. Werden mehr Konten erzeugt als die Größe des Arrays, wird intern die Länge des Arrays um seine bisherige Länge verdoppelt. Das neue Konto wird, wie oben beschrieben, dem Array hinzugefügt.
- `boolean removeAccount(int accountNumber)`: Diese Methode repräsentiert das Löschen eines Kontos innerhalb dieser Bank. Existiert die zu löschende Kontonummer, wird `true` zurückgegeben. Zudem wird das zugehörige Konto vom Array von Konten entfernt. Sind nach dem Entfernen weniger als 1/8 des Arrays besetzt, wird die Länge des Arrays halbiert. Dabei wird die Länge des Arrays nie kleiner als die initiale Länge des Arrays, d.h. 8. Beim Löschen eines Array-Elementes darf keine Lücke entstehen. Dies bedeutet,

dass alle Elemente, die nach dem zu löschenden Account kommen, nach „vorne“ aufrücken. Existiert kein Konto mit der vorgegebenen Kontonummer, wird `false` zurückgegeben. Sie können davon ausgehen, dass keine neuen Konten mit einer bereits gelöschten Kontonummer erzeugt werden.

- `boolean containsAccount(int accountNumber)`: Diese Methode gibt an, ob es ein Konto mit der Kontonummer `accountNumber` im Array existiert. Existiert ein solches Konto wird `true` zurückgegeben, andernfalls `false`.
- `boolean internalBankTransfer(int fromAccountNumber, int toAccountNumber, int amount)`: Diese Methode repräsentiert das Überweisen. Dabei entspricht `fromAccountNumber` der Kontonummer des Quellkontos und `toAccountNumber` der Kontonummer des Zielkontos und `amount` dem Betrag, der überwiesen werden soll. Der Kontostand des Quellkontos, von dem der Betrag überwiesen wird, wird um den übergebenen Betrag verringert und der Kontostand des Zielkontos, worauf der Betrag überwiesen wird, wird um diesen Betrag erhöht. Weiter können Sie davon ausgehen, dass nur Überweisungen innerhalb der Konten einer Bank stattfinden. Existieren die Kontonummer des Quellkontos und/oder Zielkontos nicht oder wird der Kontostand des Quellkontos durch diese Überweisung negativ, findet keine Überweisung statt. In diesem Fall wird `false` zurückgegeben. Weitere Randfälle brauchen Sie für diese Methode nicht behandeln. Findet die Überweisung statt, wird `true` zurückgegeben.
- `int length()`: Diese Methode gibt die aktuelle Länge des Arrays (und nicht die Anzahl der besetzten Array-Felder) zurück.
- `int size()`: Diese Methode gibt die Anzahl der besetzten Arrayfelder zurück.
- `Account getAccount(int index)`: Diese Methode gibt das gespeicherte Element mit dem Index `index` im Array zurück. In allen anderen Fällen (z.B. wenn das entsprechende Array-Feld leer ist oder wenn der Index im Array gar nicht vorkommt) wird `null` zurückgegeben.

B Tic-Tac-Toe (4 Punkte)

Tic-Tac-Toe ist ein einfaches Spiel, in dem 2 Spieler ihre Zeichen auf die freien Zellen eines quadratischen Spielfeldes der Größe 3x3 Zellen abwechselnd platzieren. In diesem Spiel hat ein Spieler das Zeichen „X“ und der andere Spieler das Zeichen „O“. Der Spieler, der als erster 3 Zeichen in eine Zeile, Spalte oder Diagonale platzieren kann, gewinnt.

In dieser Aufgabe sind alle Zellen des Spielfeldes eindeutig durchnummeriert, so wie in Abbildung 1 dargestellt ist.

0	1	2
3	4	5
6	7	8

Abbildung 1: Eindeutige Durchnummerierung der Zellen des Spielfelds

Im Gegensatz zur klassischen Variante, in dem das Spiel endet, sobald ein Spieler gewinnt, betrachten wir in dieser Aufgabe eine modifizierte Variante, in der das Spiel fortgesetzt wird, bis alle Felder besetzt sind. Sie können aber annehmen, dass es trotz einer Spielfortsetzung nie zu einer Spielsituation kommt, in der 2 Spieler gewinnen. Abbildung 2 illustriert 3 Beispiele, bei denen Spieler 1 mit dem Zeichen „X“ gewinnt.

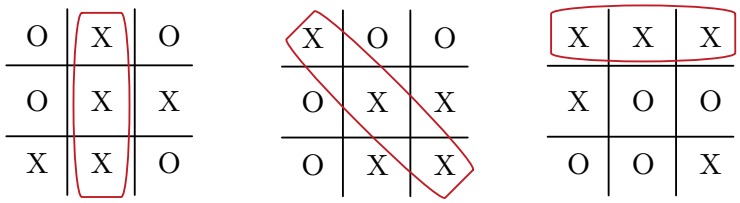

Abbildung 2: 3 Beispiele für Gewinnsituationen

B.1 Aufgabenstellung

Ihr Programm nimmt genau 9 `Integer`-Zahlen als Kommandozeilenargumente entgegen. Jede Zahl gibt eine eindeutige Zelle auf dem Spielfeld an, siehe hierzu Abbildung 1. Der erste Spieler beginnt immer das Spiel. Die Spieler platzieren Ihre Zeichen abwechselnd auf einer freien Zelle des Spielfelds. Das erste Kommandozeilenargument beschreibt die Zellennummer, auf die der Spieler 1 sein Zeichen setzt. Das zweite Kommandozeilenargument beschreibt die Zellennummer, auf die der Spieler 2 sein Zeichen platziert, usw.

Implementieren Sie ein Programm, welches überprüft, welcher Spieler das Spiel gewinnt.

Denken Sie daran, keine Klassen aus `java.util.*` zu verwenden. Nutzen Sie **nicht** das `default`-Package.

B.2 Ausgabe

Es wird genau eine Zeile auf das Terminal ausgegeben. In dieser Zeile wird ausgegeben, welcher Spieler in welchem Zug das Spiel gewonnen hat. Die Züge sind beginnend mit 1 lückenlos und fortlaufend durchnummeriert. Gewinnt der Spieler 1 im Zug `Zugnummer`, wird `P1 wins Zugnummer` ausgegeben. Falls Spieler 2 im Zug `Zugnummer` gewinnt, wird `P2 wins Zugnummer` ausgegeben und falls das Spiel unentschieden ausgeht, wird `draw` ausgegeben.

B.3 Beispielablauf

Der Aufruf `java programmname 0 4 2 3 1 7 5 8 6` führt zum in der Abbildung 3 dargestellten Spielablauf, wobei Spieler 1 das Zeichen „X“ und Spieler 2 das Zeichen „O“ hat. Wie Sie sehen, wird das Spiel fortgesetzt, obwohl der erste Spieler bereits im fünften Zug das Spiel gewonnen hat. Für dieses Beispiel gibt Ihr Programm `P1 wins 5` aus.

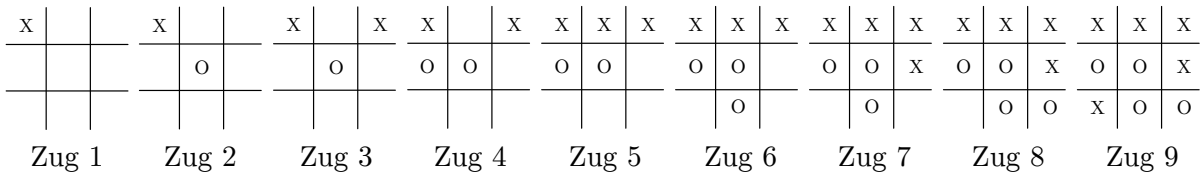

Abbildung 3: Beispiel eines Spielverlaufs

C Kryptosystem (8 Punkte)

In dieser Aufgabe soll in den ersten Ansätzen ein asymmetrisches kryptographisches Verfahren (auch Public-Key Verfahren (PKV)) umgesetzt werden.
Das besagte Verfahren (RSA-Kryptosystem[1]) stammt aus den 1970er Jahren und hat seinen Namen den drei Erfindern Ronald **R**ivest, Adi **S**hamir und Leonard **A**dleman zu verdanken.
Allgemein lässt sich ein Kryptosystem formal mithilfe eines Quintupels (P, C, K, E, D) beschreiben:

- P ist eine endliche Menge von Klartexten (engl. plaintext), die den Klartextraum darstellt.
- C ist die endliche Menge von Kryptotexten (Geheimtexte, Chiffrate), die den Kryptotextraum darstellt.
- K ist eine endliche Menge von Schlüsseln, die den Schlüsselraum darstellt.
- $E = \{E_k : k \in K\}$ ist eine Menge von Funktionen, deren Elemente Verschlüsselungsfunktionen mit $E_k : P \to C$ darstellen.
- $D = \{D_k : k \in K\}$ ist eine Menge von Funktionen, deren Elemente Entschlüsselungsfunktionen mit $D_k : C \to P$ darstellen.
- Für jedes $e \in K$ existiert ein $d \in K$, so dass für alle $p \in P$ die Gleichung $D_d(E_e(p)) = p$ gilt.

Möchte man mit Hilfe von RSA eine Nachricht m chiffrieren, so benötigt man zunächst ein Schlüsselpaar, bestehend aus einem öffentlichen Schlüssel (e, N) und einem privaten Schlüssel (d, N). Die Verschlüsselung selbst geschieht mit dem öffentlichen, die Entschlüsselung mit dem privaten Schlüssel. Die Kenntnis des privaten Schlüssels ermöglicht die Entschlüsselung des Chiffrates. Die Schwierigkeit besteht nun darin, aus dem öffentlichen Schlüssel den privaten Schlüssel zu generieren.

Das RSA-Modul N wird aus dem Produkt der Faktoren p und q (d.h. $N = p \cdot q$) gebildet, während beide Faktoren Primzahlen darstellen, für die $p \neq q$ gilt. Der Verschlüsselungsexponent e des öffentlichen Schlüssels (e, N) ist so zu wählen, dass e teilerfremd zum Wert der eulerschen Funktion von N sei: $\varphi(N) = (p-1) \cdot (q-1)$ mit $1 < e < \varphi(N)$. Der Entschlüsselungsexponent d des privaten Schlüssels (d, N) ergibt sich aus dem multiplikativen Inversen von e bezüglich des Moduls $\varphi(N), also : e \cdot d \equiv 1 \pmod{\varphi(N)}$.

Kryptotext c aus Klartext m lässt sich anschließend wie folgt herstellen, wobei gilt $1 < m < N$:
$c \equiv m^e \pmod{(p \cdot q)}$, d.h. $c \equiv m^e \pmod{(N)}$. (Beispiel: $7^{23} \pmod{143} \equiv 2 =$ c)
Klartext m aus Kryptotext c wird im Folgenden berechnet mit:
$m \equiv c^d \pmod{(p \cdot q)}$, d.h. $m \equiv c^d \pmod{(N)}$. (Beispiel: $2^{47} \pmod{143} \equiv 7 =$ m)

[1] Rivest, Ronald L., Adi Shamir, and Leonard Adleman. “A method for obtaining digital signatures and public-key cryptosystems” Communications of the ACM 21.2 (1978): 120-126.

Nachdem das asymmetrische kryptographische Verfahren kurz vorgestellt wurde, sollen zunächst nur die folgenden zwei Punkte umgesetzt werden:

- Implementieren Sie den erweiterten Euklidischen Algorithmus (EGCD) zur Bestimmung des multiplikativen Inversen einer Zahl bezüglich eines Moduls: $a \cdot b \equiv 1 \pmod{c}$. Beispiel: $11 \cdot 2 = 22 \equiv 1 \pmod{7}$.
  Erläuterung: Der erweiterte euklidische Algorithmus berechnet für zwei Zahlen $a, b \in \mathbb{N}$ den größten gemeinsamen Teiler ggt(a, b) (es gilt: $ggt(a, b) = ggt(b, a \ mod \ b)$) und die Darstellung des ggt mit ggt(a, b) als ganzzahlige Linearkombination von a und b; also $ggt(a, b) = u \cdot a + v \cdot b$ mit $u, v \in \mathbb{Z}$ und $a, b \in \mathbb{N}$.
- Implementieren Sie das Sieb des Eratosthenes (siehe Vorlesung), um die Primfaktorzerlegung einer Zahl zu ermöglichen.

Hinweis: `Long` reicht als Datentyp hier häufig nicht aus. Verwenden Sie daher intern den Datentyp `BigInteger` von `java.math.*`. Verwenden Sie nicht das `default`-Package.

C.1 Beispielablauf

Für den Aufruf `java programmname sieve 20`, gibt Ihr Programm alle Primzahlen bis 20 aus.
Also: `2 3 5 7 11 13 17`
Für den Aufruf `java programmname egcd 84 20`, gibt Ihr Programm ggt u v aus.
Also: `4 1 -4`
mit der Rechnung ($1 \cdot 84$ - $4 \cdot 20 = 4$).