Klausur B-AI2 PM 2 SS 2019, Prof. Dr. Bernd Kahlbrandt

Matrikelnummer	Vorname	Name

Aufgabe	1.1	1.2	1.3	2	3.1	3.2	4	$\sum$	Note
Punkte (ca.)	3	7	6	20	6	6	12	60	
Erreicht									

Datum: **03.07.2019** Uhrzeit: **9:00**
Raum: **BT 7, 11.Stock** Dauer (Minuten): **120**

Hinweise

1. Verfügbare Hilfsmittel: Java API Doc, Java Source Code, Laufwerk M u. a. mit Java Sprachspezifikation (jls11.pdf, jls12.pdf), Spezifikation der JVM (jvm11.pdf, jvm12.pdf), Skript, Folien (pdf),
2. Erlaubte „Mitbringsel“: Ein DIN A4 Blatt mit Notizen, beidseitig handbeschrieben, Wörterbuch Englisch oder Muttersprache, elektronisches Wörterbuch ohne Internetzugang („Flugmodus“), Papier, Stifte.
3. Notwendige Bestandteile:
   3.1. In **jeder** .java-Datei steht unmittelbar vor dem „Top-Level Element“ (Klasse, Annotation etc.) ein javadoc-Kommentar mit Ihrem Namen nach einem *@author* tag.
   3.2. Ihre Entwurfsentscheidungen sind in javadoc-Kommentaren oder //-Kommentaren knapp und nachvollziehbar begründet.
   Meine Vorstellung dazu: Ein javadoc-Kommentar (s. o. unter 3.1) mit den Entwurfsentscheidungen für die Klasse, pro Attribut bzw. Methode entsprechend.
   3.3. Die Verwendung der Annotation *@SupressWarnings* ist nur zulässig, wenn Sie bei Verwendung dieser Annotation in einem javadoc-Kommentar überzeugend darlegen, warum die jeweilige Warnung gefahrlos unterdrückt werden kann!
   3.4. Die üblichen Java-Codekonventionen (Einrückung, Namensgebung etc.) müssen eingehalten werden!
4. Achten Sie bitte auf den genauen Text der Aufgaben und etwaige Hinweise!
5. Die Testfälle für die Aufgaben befinden sich im Paket *de.hawh.kahlbrandt.ss2019bai2-pm2exam.test*.

Den Code schreiben Sie bitte im Wurzelpaket des Projekts: *de.hawh.kahlbrandt.ss2019bai2pm2-exam*.

Viel Erfolg!

1 Lambdas, Streams, Typen

1. Schreiben Sie bitte in einer Klasse *A01* eine Klassenmethode *Comparator<Integer> naturalOrder()*, die einen Comparator zurück liefert, der die natürliche Ordnung ganzer Zahlen implementiert! Schreiben Sie bitte den Code für den Comparator selbst!

   Bemerkung: Das ist logisch natürlich völlig unnötig, liefert aber nützliche Einsichten in die Funktionsweise von Java oder erinnert zumindest an diese.

2. Die Folge
$$a_n = \begin{cases} 3, & n = 0, \\ 2 \cdot a_{n-1} + 1, & n > 0, n \text{ ungerade} \\ (a_{n-1}^2 - 3)/2, & n > 0, n \text{ gerade} \end{cases}$$
   wächst schnell. Schreiben Sie bitte in der Klasse *A01* eine Klassenmethode *public static long [] generateCoprimes(int number)*, die die ersten *number* Elemente dieser Folge sicher berechnet und eine *ArithmeticException* wirft, wenn der Bereich von *long* verlassen wird! Geben Sie bitte in der Nachricht (message) mit, für die wievielte Zahl sie geworfen wurde!

3. Schreiben Sie bitte in der Klasse *A01* eine Klassenmethode *Map<String, Long> wordFrequency(String filename)*, die die Datei mit dem übergebenen Dateinamen einliest, die Häufigkeit der Worte **ohne** Berücksichtigung von Groß- und Kleinschreibung in der Datei mittels map-reduce ermittelt und die Worte mit der jeweiligen Häufigkeit in einer *Map* zurückgibt!

   Sie können annehmen, dass die Datei Zeichen in UTF-8 enthält.

2 Generische Klasse schreiben

Schreiben Sie bitte eine generische Klasse *Queue<E>*! Das heißt im Einzelnen:

1. Die Klasse implementiert das generische Interface *IQueue* aus dem Paket *interfaces*.

   1.1. `public IQueue<E> enqueue(E element)`: Fügt ein Element in die Queue ein und liefert eine Referenz auf die IQueue zurück, so dass Aufrufe verkettet werden können.

   1.2. `public IQueue<E> dequeue() throws QueueEmptyException`: Entfernt das zuerst eingefügte Element aus der Queue und liefert eine Referenz auf die IQueue zurück. Ist die Queue leer, so wird eine *RuntimeException* namens *QueueEmptyException* geworfen.

   1.3. `peek() throws QueueEmptyException`: Liefert das zuerst eingefügte Element. Ist die Queue leer, so wird eine *RuntimeException* namens *QueueEmptyException* geworfen.

   1.4. `public boolean isEmpty()`: Liefert *true*, wenn die Queue leer ist, andernfalls *false*.

   1.5. `public int size()`: Liefert die Größe der Queue.

2. Für die Speicherung der Elemente verwenden Sie bitte eine einfach vorwärts verkettete Liste mit einem Start-Knoten und einem End-Knoten.

   - „vorwärts verkettet“ heißt dabei: Ein Knoten hat eine Referenz auf den nächsten Knoten.
   - Der letzte Knoten (der End-Knoten) hat also eine (noch) ungenutzte Referenz. Verwenden Sie diese bitte dazu, den vorletzten Knoten zu referenzieren. Das Element im End-Knoten kann leer sein. Er muss nicht zur Speicherung von Elementen verwendet werden.

3. Selbstverständlich hat die Klasse die üblicherweise in einer Java-Klasse überschriebenen Methoden! Zwei Queues gelten als gleich, wenn Sie die gleichen Elemente in der gleichen Reihenfolge enthalten.

4. Die Testfälle sollen Sie dabei unterstützen, Fehler in Ihrer Implementierung zu entdecken.

3 Datum, Uhrzeit, Zeiträume

Schreiben Sie bitte folgende Methoden in einer Utility-Klasse *DateAndTimeUtils*!

1. *getAge*: Diese bekommt ein Geburtsdatum als Parameter übergeben und liefert die zum Zeitpunkt des Aufrufs der Methode aktuelle detaillierte Lebensdauer (Jahre, Monate, Tage) in einer Map mit jeweiliger Einheit und Anzahl zurück.

2. *getBinaryPalindromYears* Diese bekommt zwei *LocalDate*s (Anfang und Ende) und liefert einen *Stream<String>* der für die Jahre zwischen Anfang und Ende einschließlich die Jahre als Binärstring liefert, in denen der 25.07. ein Sonntag ist und das Jahr in Binärdarstellung ein Palindromjahr ist! Verwenden Sie dazu bitte einen unbegrenzten Stream und sorgen Sie auf geeignete Weise für sein Begrenzung!

   Zur Erläuterung was ein binäres Palindromjahr ist: Das letzte Palindromjahr in Binärdarstellung war $(2015)_{10} = (11111011111)_2$

   Den Rückgabetyp habe ich so gewählt, damit Sie nicht noch eine Collection aus einem Stream erzeugen müssen.

4 Enums

Ein Pferd beherrscht vier Gangarten:

Stand Das Pferd steht.

Schritt Das Pferd geht im Schritt(gang).

Trab Das Pferd trabt.

Galopp Das Pferd galoppiert.

Ein Pferd reagiert unverzüglich auf die Kommandos *Hü* und *Brr*. *Hü* schaltet einen Gang höher von Stand bis Galopp, *Brr* schaltet von Schritt und Trab in Stand, und von Galopp in Trab. Alle anderen Befehle veränden den Zustand des Pferdes nicht.

Schreiben Sie bitte eine Klasse Pferd! Die Objekte der Klasse sollen eine der vier Gangarten haben und auf die beiden Kommandos durch Wechsel der Gangart reagieren können. Die Gangarten und die Befehle implementieren Sie bitte als *enums*!

Schreiben Sie bitte Code, z. B. in einer *main*-Methode, der demonstriert, dass Ihr „Pferd“ sich spezifikationsgemäß verhält!