Hochschule für angewandte Wissenschaften
Würzburg-Schweinfurt
Programmieren 2
Sommersemester 2017
10.07.2017
10:00-11:30

Matrikelnummer: ____________________

(optional) Name, Vorname: ____________________

Diese Prüfung besteht aus 8 Seiten (inkl. diesem Deckblatt) und 4 Fragen. Sie können maximal 90 Punkte erreichen. Zum Bestehen sind 45 Punkte erforderlich.

Zugelassene Hilfsmittel: Wörterbücher (sofern Deutsch nicht Ihre Muttersprache ist; ohne eigene Anmerkungen).

Generelle Erleichterungen:

- import-Anweisungen müssen nicht geschrieben werden
- Sie dürfen „mit Verstand" abkürzen mit "… " arbeiten, z. B.
  `int[] int_feld = {1, 2, ..., 24};`
- Auf der letzen Seite der Klausur finden Sie einen Auszug aus der Java-Doku und Informationen über weitere relevante Klassen und Interfaces.

Wichtig: Tragen Sie die Lösungen zu der Aufgabe 4 in dieser Aufgabenstellung ein! Geben Sie diese Aufgabenstellung unbedingt mit ab. Lösen Sie NICHT die Klammern, welche diese Aufgabenstellung zusammen hält.

Punktetabelle (nicht von Ihnen auszufüllen!)

Frage	1	2	3	4	Gesamt
Punkte	35	24	15	16	90
Erreicht					

1. Die IBAN (Internationale Bankkontonummer) ist eine standardisierte Notation für Bankkontonummern.

[6 Punkte] (a) Schreiben Sie einen selbstdefinierten checked Ausnahmetyp namens `FalscheIBANException`. Dieser besteht aus zwei öffentlichen Konstruktoren (Konstruktor ohne Parameter und Konstruktor mit einem String-Parameter). Der Konstruktor ohne Parameter speichert die Ausnahme-Beschreibung „FEHLER: Datei enthält ungültige IBAN“. Bei dem zweiten Konstruktor ist die Ausnahme-Beschreibung gleich dem übergebenen (String-) Parameter.

[7 Punkte] (b) Schreiben Sie eine Methode `ibanCheck` welche eine übergebene IBAN auf Korrektheit prüft.

Eine gültige deutsche IBAN setzt sich folgendermaßen zusammen:

- 2-stelliger Ländercode in Großbuchstaben: DE für Deutschland
- 20-stellige Kontoidentifikation (beliebige Zeichen[1])

Beispiel für eine gültige deutsche IBAN:
DE2130120400000BYI15228 Handelt es sich um eine gültige IBAN, dann liefert die Methode den booleschen Wert `true` zurück. Im Fehlerfall wirft die Methode eine Exception vom Typ `FalscheIBANException` mit **aussagekräftigem Text, unter Angabe der falschen IBAN** und propagiert diese Ausnahme.

[13 Punkte] (c) Programmieren Sie die Methode `ibanAusDateiLesen`, welche eine Datei daraufhin überprüft ob sie gültige IBANs enthält. Die Datei besteht aus einer beliebigen Anzahl von IBANs die **zeilenweise als Zeichenkette** gespeichert sind. Der Dateiname wird in Form eines Strings an die Methode übergeben. Diese soll alle in der Datei enthaltenen IBANs lesen und mit Hilfe der Methode `ibanCheck` aus Teilaufgabe b) auf Gültigkeit prüfen. Sind alle IBANs in der Datei gültig gibt die Methode eine Erfolgsmeldung auf der Standardausgabe aus. Wirft die Methode `ibanCheck` eine Exception, dann propagiert die Methode `ibanAusDateiLesen` diese Exception an die aufrufende Methode. Die Methode `ibanAusDateiLesen` soll weiterhin Ausnahmen vom Typ `FileNotFoundException`, `IOException` und `NullPointerException` fangen und mit einem entsprechenden Fehlertext auf der Standardausgabe entschärfen. Beachten Sie bitte, dass auch im Fehlerfall die Ströme zu schließen sind.

---

[1] Um die Aufgabenstellung zu vereinfachen, nehmen wir an, dass eine IBAN nach dem Ländercode aus beliebigen Zeichen bestehen kann.

In Abhängigkeit von der geworfenen Exception soll der Text folgendermaßen lauten:

- FileNotFoundException: „Datei <dateiname> nicht gefunden“
- IOException: „Fehler beim Lesevorgang von Datei: <dateiname>“
- NullPointerException : „Datei <dateiname> existiert nicht“

[5 Punkte] (d) Schreiben Sie eine Methode `dateienTest`, welche beliebig viele Dateien mit Hilfe der Methode `ibanAusDateiLesen` aus c) daraufhin untersucht ob sie gültige IBANs enthalten. Die Dateinamen werden der Methode in Form eines String-Feldes übergeben. Ausnahmen welche von der Methode `ibanAusDateiLesen` propagiert werden, werden entschärft indem der Name der fehlerhaften Datei auf der Standardausgabe angezeigt wird.

[4 Punkte] (e) Schreiben Sie einen JUnit-Test für die Methode `ibanCheck`. Berücksichtigen Sie dabei auch, dass die Methode im Fehlerfall eine checked Exception propagiert. Beispiele für mögliche Methoden aus der Assert-Klasse:

```
static void assertTrue(String message, boolean condition);
static void assertEquals(Object expected, Object actual);
static void fail(String message);
```

2. Um natürlich-sprachliche Texte zu analysieren, ist es wichtig zu wissen wie oft ein Wort darin vorkommt. Glücklicherweise müssen Sie den Text nicht selbst analysieren, sondern nur die Ergebnisse in einer geeigneten Datenstruktur ablegen. Schreiben Sie eine Klasse `WortVorkommen`, welche zu jedem Wort alle zugehörigen Positionen abspeichert. Zwischen einem Wort und all seinen Positionen besteht eine 1:n Beziehung, da ein Wort mehrmals vorkommen kann.

   Ihrer Aufgabe besteht darin diese Instanzen von Wörtern und deren Positionen in einer passenden Datenstruktur für eine bessere Analyse abzulegen.

   Die Klassen `Wort` und `Position` sind folgendermaßen definiert:

```
public final class Wort implements Comparable<Wort> {
   String wort, wortTyp, sprache;
   int laenge;

   // Methode compareTo aus Comparable
}

public final class Position implements Comparable<Position> {
   int zeilennummer, spaltennummer;

   // Methode compareTo aus Comparable
}
```

   Diese Datenstruktur soll folgendermaßen genutzt werden:

```
1 Wort und = new Wort("und");
2 WortVorkommen woerter = new WortVorkommen();
3
4 woerter.einfuegen(und, new Position(1, 3));
5 woerter.einfuegen(und, new Position(3, 7));
6
7 Position letztePosition = woerter.holeLetztePosition(und);
8 Collection<Position> allePositionen = woerter.holeAlle(und);
9 int anzahlVorkommen = woerter.anzahlVorkommen(und);
```

[7 Punkte] (a) Nutzen Sie eine passende Datenstruktur und implementieren Sie die Klasse `WortVorkommen` mit gegebenenfalls notwendigen Attributen. Diese Klasse darf von anderen Klassen erben und bekannte Interfaces implementieren. Als Hilfestellung können Sie den Anhang nutzen.

[13 Punkte] (b) Implementieren Sie die Methoden `einfuegen`, `holeAlle` und `holeLetzte`.

[2 Punkte] (c) Implementieren Sie die Methode `compareTo`, um zwei Positionen vergleichen zu können. Die Vergleichskriterien können Sie frei wählen.

[2 Punkte] (d) Implementieren Sie die Methode `woerter.anzahlVorkommen` aus Code-Zeile 9. Sie können die Implementierungen aus den vorherigen Aufgaben nutzen.

3. Sie wurden beauftragt eine neue Schnittstelle zu erstellen. Diese soll von anderen Entwicklern möglichst einfach und direkt benutzt werden können, sodass auch neue Mitarbeiter sich schnell zurecht finden.

   Wichtig: Realisieren Sie dies mit Hilfe des **Builder-Patterns** und nutzen Sie Ihr Wissen über **Method-Chaining** und **Fluent Interfaces**.

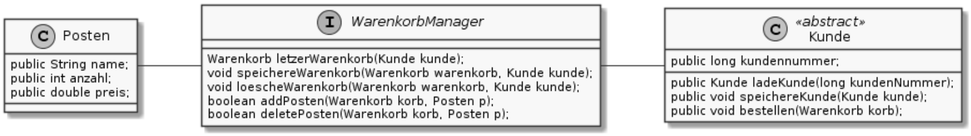
[6 Punkte] (a) Zeigen Sie mit einem Code-Beispiel wie das Erstellen eines Warenkorbes und einer Bestellung mit einem Builder-Pattern geschehen würde. Sie müssen den Builder an sich **nicht** implementieren. Die Nutzung des Builders soll so gestaltet werden, dass ein Warenkorb nur in Verbindung mit einem Kunden existieren kann. Außerdem soll ein Posten nicht ohne einen Warenkorb existieren können. Die Bestellung kann ausgelöst werden, wenn ein valider Warenkorb erzeugt wurde. Das Auslösen der Bestellung darf nicht Teil des Builder-Patterns sein, da der Bestellprozess zum späteren Zeitpunkt abgebrochen werden kann.

Beispiel eines Builders aus der Vorlesung:

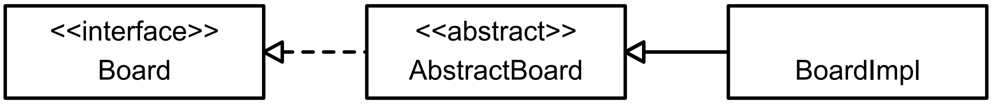
```
Person p = new Person.PersonBuilder("John", "Doe", "01.01.1900")
  .withKoerpergroesse(188).withWohnort("Berlin").build();
```

[9 Punkte] (b) Erstellen Sie ein Diagramm, welches die Klassen und Beziehungen der neuen Schnittstelle darstellt. Zudem sollten Sie auch alle für die Klasse relevanten Informationen einbeziehen. Dazu zählen Konstruktoren, Attribute und Methoden inklusive deren Sichtbarkeit sowie Daten- bzw. Rückgabetyp. Sofern Sie verschachtelte Klassen verwenden, markieren Sie diese deutlich im Diagramm.

Sie dürfen gerne ein Klassendiagramm erstellen und können folgendes Beispiel als Hilfestellung verwenden.



4. Nachfolgendes Klassendiagramm zeigt einen typischen Ausschnitt aus einem Java-Programm. Eine abstrakte Klasse implementiert ein Interface und eine konkrete Klasse erbt von der abstrakten Klasse.

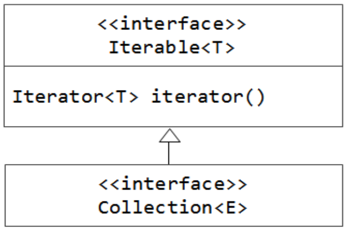


[4 Punkte] (a) Beschreiben Sie kurz abstrakte Klassen und Interfaces in Java anhand der Kriterien **Instanzen**, **Methoden**, **Attribute** und **Vererbung**.

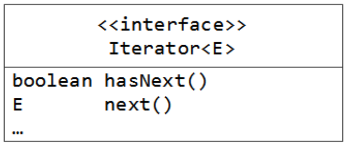
	Interfaces	Abstrakte Klassen
Instanzen		
Methoden		
Attribute		
Vererbung		

[12 Punkte] (b) Betrachten Sie nun das gesamte obige Klassendiagramm. Führen Sie eine Argumentation an, ob eine solche Struktur sinnvoll ist. Versetzen Sie sich dabei gedanklich in ein sehr großes Projekt (z. B. mehr als 50 Entwickler), welches von einem internationalen, verteilten Team umgesetzt wird und bereits unter starkem Zeitdruck steht.

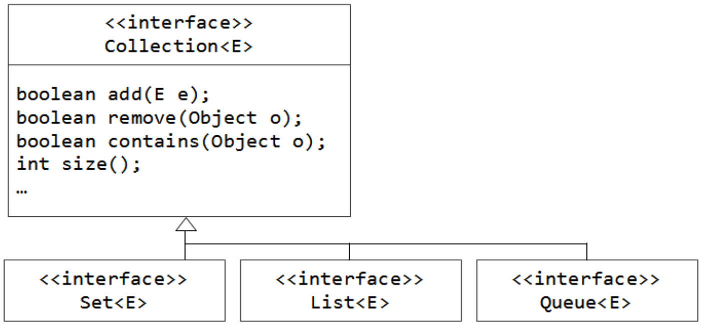
Führen Sie **zwei** Argumente an. Beachten Sie bitte, dass ein vollständiges Argument aus einer **Behauptung**, einer **Begründung** und einem **Beleg** bzw. **Beispiel** besteht.



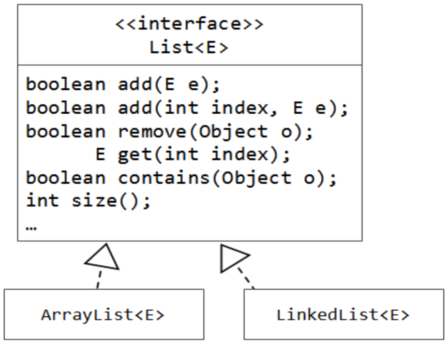
```
Iterator<T> iterator()
```



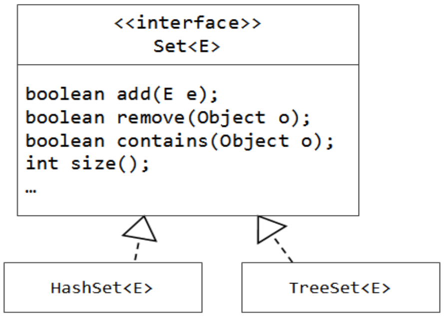
```
boolean hasNext()
E       next()
…
```



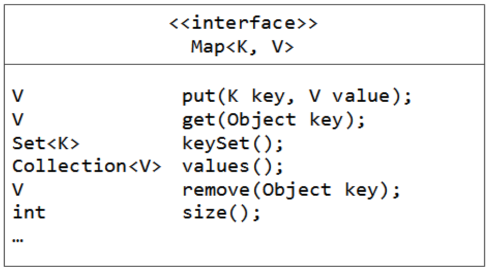
```
boolean add(E e);
boolean remove(Object o);
boolean contains(Object o);
int size();
…
```



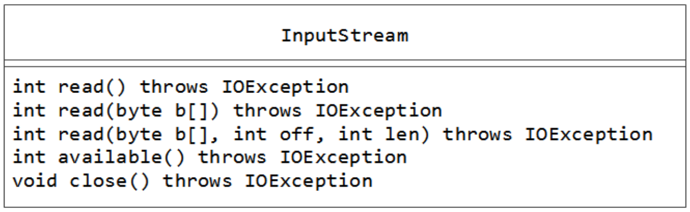
```
boolean add(E e);
boolean add(int index, E e);
boolean remove(Object o);
      E get(int index);
boolean contains(Object o);
int size();
…
```



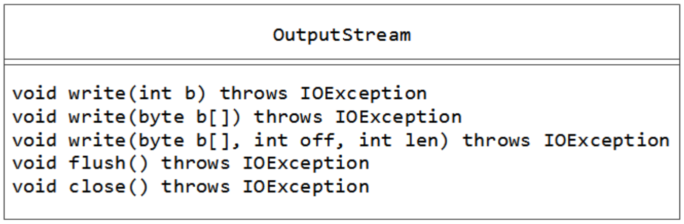
```
boolean add(E e);
boolean remove(Object o);
boolean contains(Object o);
int size();
…
```



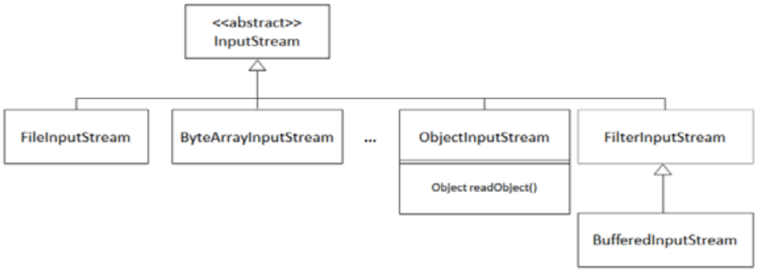
```
V               put(K key, V value);
V               get(Object key);
Set<K>          keySet();
Collection<V>   values();
V               remove(Object key);
int             size();
…
```

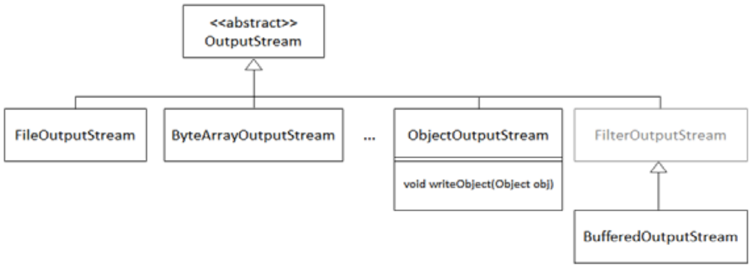


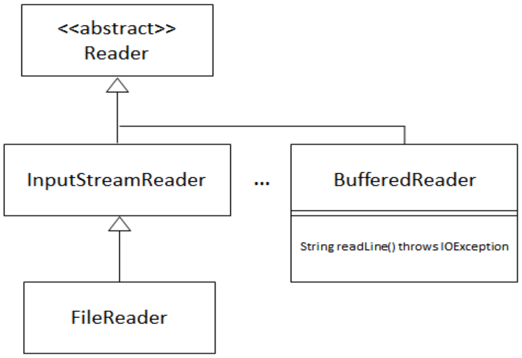
```
int read() throws IOException
int read(byte b[]) throws IOException
int read(byte b[], int off, int len) throws IOException
int available() throws IOException
void close() throws IOException
```

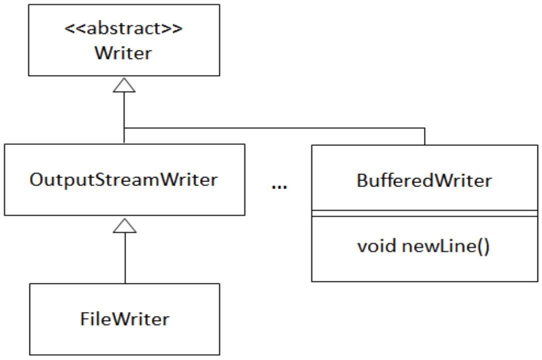


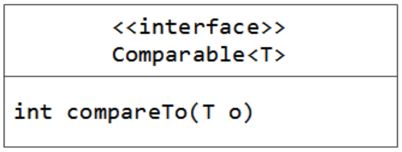
```
void write(int b) throws IOException
void write(byte b[]) throws IOException
void write(byte b[], int off, int len) throws IOException
void flush() throws IOException
void close() throws IOException
```



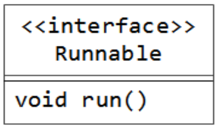




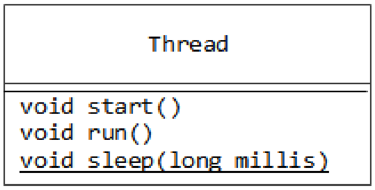




```
int compareTo(T o)
```



```
void run()
```



```
void start()
void run()
void sleep(long millis)
```