Prüfung
Data Processing (DP)

Prof. Dr.-Ing. Christian Heller <christian.heller@ba-leipzig.de>

Student	
Vor- und Nachname	Ihre Daten werden von der Klausuraufsichtsperson schriftlich auf der Anwesenheitsliste festgehalten.
Matrikelnummer	
Studienrichtung und Jahr	CS 2018-2
Anmeldename	Das Login "klaus..." muss unbedingt schriftlich auf der Anwesenheitsliste festgehalten werden, da sonst keine Zuordnung des Logins zu Ihrem Namen und damit keine Korrektur der Klausur möglich ist!

Prüfung	
Datum	September 2019
Dauer [min]	120
Hilfsmittel	Dokumentation im lokalen Netzwerk (Intranet) sowie Recherche im Internet. NICHT gestattet: * Kommunikation in jeglicher Form * Anmeldung via SSH auf dem Rechner "fileserv" * Anmeldung mit Klausur-Login nach Ende der Prüfung Dies kann leicht geprüft werden (last cs16*, Server-Log-Dateien). Bitte unterlassen Sie also Täuschungsversuche in Ihrem eigenen Interesse.
Bemerkungen	Hinterlegen Sie alle Programme und Antworten in elektronischer Form! Es wird kein Papier angenommen. Möchten Sie Lösungen erläutern, so nutzen Sie Quelltext-Kommentare oder legen eine Text-Datei an. Speichern Sie sämtliche Daten im HOME-Verzeichnis des Nutzers, d. h. unter Windows auf Laufwerk H:\ (NICHT auf C:\ oder "Eigene Dateien")! Idealerweise legen Sie dort ein Unterverzeichnis namens "klausur" an. Lesen Sie die Aufgaben komplett durch, bevor Sie sie lösen! Die Reihenfolge der Lösung ist Ihnen überlassen. Probieren Sie immer, eine Aufgabe zu lösen, da auch auf richtige Teile nicht vollständiger Lösungen Punkte vergeben werden! Falls vom Prinzip her richtig, so werden auch alternative Lösungen akzeptiert. Sie dürfen beliebig viele Bildschirmausgaben von Werten in den Quelltext einbauen, um ein Programm besser nachvollziehen zu können. Bitte duplizieren Sie Ihre Quelltextdateien (workspace) NICHT, da beim Korrigieren dann beide durchsucht werden müssen, was sinnlosen Aufwand verursacht. Diese Aufgabenstellung in Papierform können Sie nach dem Ende der Klausur behalten.

Bewertung						
Aufgabe	1	2	3	4	5	Summe
Punkte	20	20	20	20	20	100

Im Rahmen der Klausur sind fünf verschiedene Aufgaben zu lösen.

Aufgabe 1: File and Date / Time [20]

Zweck: Verwenden von Java-Klassen zur Ausgabe eines Dateidatums auf der Konsole.

a) Nehmen Sie einen Dateipfad wie „/etc/hosts“ als Kommandozeilenargument entgegen! [2]

b) Erzeugen Sie damit als Parameter ein Dateireferenz-Objekt vom Typ „File“! [2]

c) Speichern Sie das Datum der letzten Änderung der Datei in einer lokalen Variablen! [2]

d) Erzeugen Sie unter Verwendung dieses Wertes ein „Date“-Objekt! [2]

e) Erzeugen Sie ein für unsere Region geeignetes Kalenderobjekt! [2]

f) Setzen Sie seine Zeit unter Verwendung des obigen „Date“-Objektes! [2]

g) Geben Sie die Zeichenkette „Letzte Änderung: “ ohne Zeilenumbruch aus! [2]

h) Geben Sie anschließend das durch das Kalenderobjekt repräsentierte Datum im Format TT.MM.JJJJ mit dem Punkt als Trenner aus! [2]

i) Verwenden Sie Konstanten der „Calendar“-Klasse, um die einzelnen Elemente des Kalenderobjektes zu bestimmen! [2]

j) Geben Sie die Uhrzeit im Format HH:MM:SS mit dem Doppelpunkt als Trenner aus! [2]

Ergebnis: Das ausgegebene Datum hat die Form „Letzte Änderung: 24.10.2018 20:38:42“.

Aufgabe 2: Container [20]

Zweck: Verwendung einer generischen Schablonenklasse.

a) Erstellen Sie zwei Klassen namens „Socke“ und „Ohrring“, deren „toString“-Methode zur Identifizierung eine eindeutige Zeichenkette zurückgibt! [2]

b) Erstellen Sie eine Schablonenklasse (template class) namens „GenerischesPaar“, deren parametrisierbarer Typ „T“ heißt! [2]

c) Geben Sie ihr zwei private Attribute „l“ und „r“ vom Typ T! [2]

d) Schreiben Sie einen initialisierenden Konstruktor, der zwei Parameter entgegennimmt! [2]

e) Implementieren Sie die Zugriffsmethoden zum Lesen der Attributwerte! [2]

f) Implementieren Sie die „toString“-Methode in der Weise, dass die String-Darstellung auch der beiden Attributwerte (implizit oder explizit) ausgegeben wird! [2]

g) Erzeugen Sie in der „main“-Methode der „Starter“-Klasse ein „Socke“-Paar und geben Sie es auf der Konsole aus! [2]

h) Erzeugen Sie desweiteren ein „Ohrring“-Paar und geben Sie es auf der Konsole aus! [2]

i) Bestimmen Sie eine der Socken des „Socke“-Paares! Geben Sie sie auf der Konsole aus! [2]

j) Versuchen Sie, ein gemischtes Paar aus einer Socke und einem Ohrring zu erzeugen! Schreiben Sie die Fehlermeldung als Kommentar in den Quelltext! [2]

Ergebnis: Die Schablonenklasse wurde mit Objekten verschiedener Typen verwendet.

Aufgabe 3: Algorithm [20]

Zweck: Anwendung des „Quicksort“-Algorithmus’.

Kurzbeschreibung (Wikipedia): Quicksort ist ein schneller, rekursiver, nicht-stabiler Sortieralgorithmus, der nach dem Prinzip Teile und herrsche arbeitet. Pseudocode:

```
funktion quicksort(links, rechts)
    falls links < rechts dann
        teiler:= teile(links, rechts)
        quicksort(links, teiler-1)
        quicksort(teiler+1, rechts)
    ende
ende
```

a) Definieren Sie in der „main“-Methode ein Array mit den folgenden Werten:

int[] a = { 34, 65, 43, -23, 8, 454, 34, 2, -9, 7, 6, 4, 12, 234, 54, 23, 76, 8, 98, 32 };

Geben Sie seine Werte auf der Konsole aus! [2]

b) Rufen Sie noch VOR der Ausgabe die (noch zu erstellende) statische Methode „sort_quick“ auf! Übergeben Sie ihr drei Argumente [2]:

- das Array selbst: a
- den Index des ersten Elements (left index): l
- den Index des letzten Elements (right index): r

c) Erstellen Sie die Methode „sort_quick“ mit passender Signatur! Sichern Sie per Bedingung ab, dass der linke Index kleiner als der rechte ist! [2]

d) Initialisieren Sie drei Variablen, wie folgt:

```
int i = l - 1; int j = r; int tmp = 0;
```

Schreiben Sie eine „while“-Endlosschleife! Fügen Sie in den Schleifenkörper als Abbruchbedingung (I >= j) ein! [2]

e) Fügen Sie vor der Abbruchbedingung eine „do-while“-Schleife mit der Laufbedingung (a[i] < a[r]) ein! [2]

f) Fügen Sie danach, aber ebenfalls vor der Abbruchbedingung eine zweite „do-while“-Schleife mit der Laufbedingung ((a[j] > a[r]) & (j > i)) ein! [2]

g) Vertauschen Sie unterhalb der Abbruchbedingung die Elemente mit Index i und j! [2]

h) Vertauschen Sie unterhalb der „while“-Endlosschleife die Elemente mit Index i und r! [2]

i) Rufen Sie die „sort_quick“-Methode auf! Übergeben Sie ihr das Array selbst als erstes Argument; als zweites den „left index“ und als drittes die Schleifenlaufvariable (i - 1)! [2]

j) Rufen Sie die „sort_quick“-Methode abschließend ein zweites Mal auf! Übergeben Sie ihr erneut das Array selbst als erstes Argument; als zweites die Schleifenlaufvariable (i + 1) und als drittes den „right index“! [2]

Ergebnis: Das gegebene Ganzzahlen-Feld (Array) wurde korrekt sortiert.

Aufgabe 4: Stream [20]

Zweck: Schreiben eines einfachen Komprimierers.

Gegebene Eingabedatei „in.txt“:

```
+-------+-------+-------+
|       |       |       |
|   1   |   2   |   3   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   4   |   5   |   6   |
|       |       |       |
+-------+-------+-------+
```

Kodierte Ausgabedatei „out.txt“:

```
+@-+@-+@-+
|@ |@ |@ |
|   1   |   2   |   3   |
|@ |@ |@ |
+@-+@-+@-+
|@ |@ |@ |
|   4   |   5   |   6   |
|@ |@ |@ |
+@-+@-+@-+
```

a) Erstellen Sie eine Startklasse mit „main“-Methode, in der zwei Kommandozeilenargumente in Empfang genommen werden! Dabei repräsentiert das erste den Namen der Eingabe-, das zweite jenen der Ausgabedatei. [2]

b) Instanziieren Sie die ungepufferten Stream-Klassen „FileInputStream” und „FileOutputStream”! Verwenden Sie dabei die oben bestimmten Dateinamen! [2]

c) Kapseln Sie die beiden Objekte jeweils in den gepufferten Klassen „PushbackInputStream” beziehungsweise „BufferedOutputStream”! [2]

d) Lesen Sie via „PushbackInputStream“-Objekt ein Byte ein und speichern Sie es in einer lokalen Variablen namens b! Deklarieren Sie außerdem eine Schleifenlaufvariable namens c sowie eine Variable n für das jeweils nächste Byte! [2]

e) Verwenden Sie eine Schleife, die so lange läuft, wie das zuvor bestimmte Byte gültig ist! Bestimmen Sie darin zu Beginn den Wert des nächsten Bytes n und am Ende den Wert für das aktuelle Byte b neu! [2]

f) Verwenden Sie innerhalb der ersten eine zweite (geschachtelte) Zählschleife, um die Anzahl gleicher Bytes innerhalb der nachfolgenden Bytes herauszufinden (eine Art Vorschau also)! Erhöhen Sie dazu die Laufvariable c beginnend bei 1 in Einerschritten so lange, bis das nächste Byte n ungültig ist! Lesen Sie darin das nächste Byte ein und weisen es der Variablen n zu! Verlassen Sie die Schleife, sobald 255 Durchläufe stattgefunden haben ODER das nächste Byte n einen anderen Wert als das aktuelle Byte b hat! [2]

g) Bauen Sie im Anschluss an die innere Zählschleife eine Verzweigung ein, die prüft, ob in der Vorschau mindestens vier gleiche Bytes gefunden wurden! Falls ja, so kodieren Sie diese gemäß folgender Dreiteilung [2]:

1. `das Zeichen '@' zur Einleitung der Kodierungsfolge`
2. `das eigentliche Byte, welches im Text wiederholt auftrat`
3. `die Anzahl des Auftretens des sich wiederholenden Bytes`

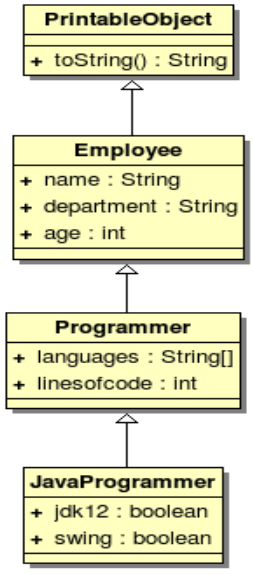
h) Falls nein, so schreiben Sie sich nicht wiederholende Bytes der Reihe nach in die Ausgabedatei! [2] Hinweis: Schreiben der einzelnen Bytes per Zählschleife bis zum Maximum c.

i) Stellen Sie das letzte Byte n mittels „unread“-Methode zurück, wenn es einen gültigen Wert hat! [2] Hinweis: Dies ist nötig, da die obige Einleseschleife abgebrochen wurde, weil c die Anzahl von 255 erreicht hatte oder n sich von b unterschied.

j) Denken Sie an das Schließen aller Dateiressourcen am Ende des Programms! [2]

Ergebnis: Die Textdatei wurde erfolgreich kodiert.

Aufgabe 5: Meta Programming [20]



Zweck: Ausgabe von Objekteigenschaften mittels Meta-Techniken.

a) Erstellen Sie die im nebenstehenden UML-Klassendiagramm gezeigten Klassen! Berücksichtigen Sie eventuelle Vererbungsbeziehungen! [2]

b) Implementieren Sie sämtliche Attribute samt Sichtbarkeiten! [2]

c) Erstellen Sie Startklasse und „main”-Methode, in der ein Objekt des Types „JavaProgrammer” erzeugt und auf Konsole ausgegeben wird! [2]

d) Weisen Sie ihm folgende Werte zu [2]:

```
name: "Max Mustermann"
department: "Entwicklung"
age: 32
linesofcode: 55000
jdk12: true
swing: false
```

e) Weisen Sie der Eigenschaft „languages“ ein Array mit folgenden Werten als Literal zu: {"C", "Pascal", "Perl", "Java"}! [2]

f) Überschreiben Sie die geerbte „toString“-Methode in der Klasse „PrintableObject“ (und NUR dort)! Machen Sie dies durch eine passende Annotation kenntlich! [2]

g) Bestimmen Sie darin mittels geeigneter Meta-Methode die Klasse des aktuellen Objektes! [2]

h) Iterieren Sie per „for-each“-Schleife durch sämtliche Eigenschaften (NICHT Methoden) der Klasse! Geben Sie jeweils Name UND Wert als sinnvoll formatierte Zeichenkette zurück! [2]

i) Geben Sie dabei auch die in „Array”-Attributen enthaltenen Werte aus! [2] Hinweis: Methoden „isArray“ und „Arrays.toString“

j) Umhüllen Sie die „for-each“- durch eine „while“-Schleife, die so lange läuft, wie die bestimmte Klasse nicht „null“ ist! Bestimmen Sie in jedem Zyklus die nächste Superklasse! [2]

Ergebnis: Durch Aufruf der „toString“-Methode wurden alle Objekteigenschaften ausgegeben.

Viel Erfolg!