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 2018-1
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	Juni 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	10	30	30	10	20	100

Die Nutzung der Java-Klassenbibliothek wird innerhalb fünfer Aufgaben geprüft. Dabei liegt der Fokus auf der Verarbeitung von Daten verschiedenster Formate.

Aufgabe 1: Container [10]

In dieser Aufgabe soll ein Behälter (Container) verwendet werden, dessen Inhalt auf zwei Weisen auszugeben ist, wobei beide Varianten den gleichen Inhalt liefern sollten.

a) Erstellen Sie ein kleines Anwendungsprogramm mit einem typsicheren Container vom Typ „HashMap“, um vier Personen folgende Gehälter zuzuordnen [2]:

```
Meier, August: 5000.00
Schmitz, Anton: 4500.00
Balder, Hugo: 4700.00
Schulze, Wolfgang: 4500.00
```

b) Bestimmen Sie die Menge der Schlüssel und speichern Sie sie in einer lokalen Variablen! [2]

c) Geben Sie die Gehälter unter Verwendung der Schlüssel per „for-each”-Schleife auf der Konsole aus! [2]

d) Bestimmen Sie das „EntrySet“ und speichern Sie es in einer lokalen Variablen! [2]

e) Geben Sie die Gehälter per „for-each”-Schleife via „entrySet” auf der Konsole aus! [2]

Ergebnis: Beide Schleifen liefern die gleiche Ausgabe auf der Konsole.

Aufgabe 2: Stream [30]

Zweck: Implementieren zweier Verschlüsselungsalgorithmen.

a) Erstellen Sie eine abstrakte Klasse namens „Encoder“ mit den Methoden „encode“ und „decode“, die jeweils ein „String“-Argument entgegennehmen und auch zurückliefern! [2]

b) Erstellen Sie eine Klasse „XorEncoder“, die von „Encoder“ erbt! Geben Sie ihr eine Ganzzahl namens „key“ als Attribut, welches über einen Konstruktor per Parameter initialisiert wird! [2]

c) Implementieren Sie die „encode“-Methode so, dass die übergebene Zeichenkette zunächst als String Array lokal gespeichert wird! [2]

d) Iterieren Sie über das Array und wenden Sie auf die einzelnen Zeichen die XOR-Verschlüsselung an! [2]

Hinweis: XOR-Verknüpfung (Kontravalenz) des aktuellen Array-Elementes mit dem Attribut „key“ mittels Operator „^“.

e) Kapseln Sie den Array-Inhalt in einer neuen Zeichenkette und verwenden Sie sie als Rückgabewert der Methode! [2]

Hinweis: Die Klasse „String“ bietet einen entsprechenden Konstruktor an.

f) Delegieren Sie den Aufruf der „decode“-Methode weiter an die „encode“-Methode! [2]

Hinweis: Dies ist möglich, da bei der einfachen XOR-Verschlüsselung Ver- und Entschlüsselung identisch sind.

g) Implementieren Sie nun die Klasse „SwapEncoder“, deren "encode"-Methode paarweise jeweils zwei Zeichen vertauschen und zwischen diese ein weiteres zufälliges Zeichen einfügen soll! [2]

h) Iterieren Sie wiederum durch die als lokales Array gespeicherte Zeichenkette, wobei die Zählschleife in Zweierschritten springen und von 0 bis (c.length – 1) laufen soll! [2]

i) Setzen Sie die neu gebildete Zeichenkette zusammen aus: (1) ursprünglicher Zeichenkette, (2) zweitem Zeichen, (3) zufälligem Zeichen, (4) erstem Zeichen! [2]

j) Prüfen Sie, ob die Länge der Zeichenkette ungerade war und fügen Sie in diesem Fall das letzte Zeichen manuell der Ergebniszeichenkette hinzu! [2]

Hinweis: Prüfen auf ungerade Zeichenkette per: ((c.length % 2) != 0)

k) Implementieren Sie die „decode“-Methode auf ähnliche Weise, wobei die Zählschleife in Dreierschritten von 0 bis (c.length – 2) laufen soll! Fügen Sie auch hier ein bei ungerader Zeichenkettenlänge eventuell vergessenes letztes Zeichen hinzu! [2] Hinweis:

```
// Zuerst das Zeichen c1 von Stelle 3
r = r + c[i + 2];
// Dann das Zeichen c2
r = r + c[i];
// Das Zeichen c3 entfaellt
```

l) Fordern Sie den Anwender in der „main“-Methode der Startklasse dazu auf, einen der beiden Algorithmen zu wählen! Nehmen Sie die ganzzahlige Auswahl per „java.util.Scanner“-Objekt entgegen und unterscheiden Sie per „switch-case“-Konstrukt! [2]

m) Erzeugen Sie, je nach Auswahl, das passende Encoder-Objekt! Nehmen Sie im Falle des „XorEncoder“ außerdem noch per „Scanner“ eine Ganzzahl als Schlüssel in Empfang! [2]

n) Rufen Sie in beiden Fällen eine neu zu implementierende, statische „demo“-Methode der Startklasse auf und übergeben Sie ihr das jeweilige „Encoder“-Objekt als Argument! [2]

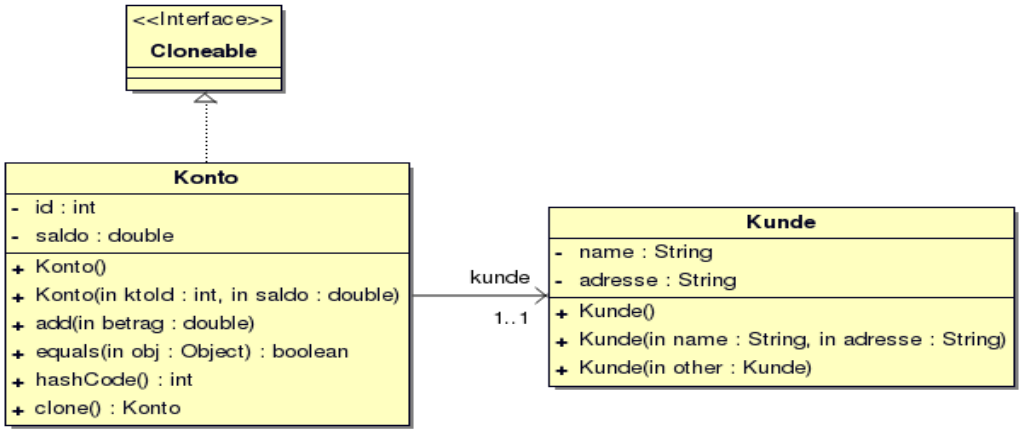
o) Nehmen Sie dort wiederum per „Scanner“ eine Textzeile entgegen (Hinweis: „nextLine()“)! Ver- und Entschlüsseln Sie sie mittels „Encoder“-Objekt! Testen Sie Original- und dekodierte Zeichenkette auf Gleichheit! [2]

Hinweis: Schließen aller „Scanner“-Objekte nicht vergessen.

Ergebnis: Die Ver- und Entschlüsselung von Texten funktioniert einwandfrei.

Aufgabe 3: Cloning [30]

Ziel dieser Aufgabe ist es, ein Konto mitsamt seinem zugeordneten Kunden zu duplizieren.



a) Bauen Sie das im UML-Klassendiagramm vorgegebene Klassengerüst nach! [2]

b) Fügen Sie die Attribute ein! [2]

c) Berücksichtigen Sie dabei auch die eingezeichnete Assoziation! [2]

d) Beachten Sie die Sichtbarkeiten! [2]

e) Fügen Sie öffentliche Zugriffsmethoden zur Umgehung der Kapselung für alle Attribute hinzu! [2]

f) Geben Sie beiden Klassen je einen Standardkonstruktor! [2]

g) Implementieren Sie die weiteren angegebenen Konstruktoren! Nutzen Sie deren übergebene Parameterwerte zur Initialisierung der Attribute! [2]

h) Lassen Sie die „add”-Methode den Saldo des Kontos um den übergebenen Betrag erhöhen! [2]

i) Prüfen Sie in der „equals”-Methode mittels „instanceof”-Operator auf Gleichheit des Types! Überprüfen Sie desweiteren, ob die „id”-Eigenschaft des aktuellen und des übergebenen Objektes den gleichen Wert haben! [2]

j) Bilden Sie den Hash-Code in gleichnamiger Methode wie folgt [2]: this.id % 100

k) Lassen Sie die Klasse „Konto“ von der Schnittstelle „Cloneable“ erben! Implementieren Sie die „clone”-Methode so, dass eine tiefe Kopie erstellt wird! [2]

l) Erstellen Sie eine „Launcher”-Klasse mit „main”-Methode und erzeugen Sie darin je ein Objekt vom Typ [2]:

- „Kunde” mit den Parametern: "Hugo Meier", "Hauptstr. 12, 40880 Ratingen
- „Konto” mit den Parametern: 4711, 10000.0

m) Weisen Sie das Kundenobjekt dem Kontoobjekt zu! [2]

n) Erzeugen Sie ein zweites Kontoobjekt durch Klonen des bereits vorhandenen! [2]

o) Ändern Sie das vom ersten Kontoobjekt referenzierte Kunde-Objekt, beispielsweise durch Ändern der Hausnummer auf den Wert „42”, ab! Geben Sie, zur Kontrolle, die Attributwerte aller Objekte auf der Konsole aus! [2]

Ergebnis: Die auf Konsole ausgegebenen Objekte unterscheiden sich nur durch die Änderung.

Aufgabe 4: Date and Time [10]

Zweck: Schreiben einer simplen, Konsole-basierten Stoppuhr.

Stoppuhr

a) Warten Sie per „read“-Methode des Standard-Input-Streams auf eine Nutzereingabe! [2]

b) Erzeugen Sie ein „Date“-Objekt und geben Sie es auf der Konsole aus! [2]

c) Warten Sie per „read“-Methode erneut auf eine Nutzereingabe! Erzeugen Sie ein zweites „Date“-Objekt und geben Sie es auf der Konsole aus! [2]

d) Bilden Sie die Differenz aus Start- und Stoppzeit und geben Sie sie auf der Konsole aus! [2]

e) Fangen Sie geworfene Ausnahmen ab! [2]

Ergebnis: Per Eingabetaste wird die Stoppuhr zur Messung der Zeit gestartet | gestoppt.

Aufgabe 5: Multi-Threading [20]

Zweck: Drei Objekte sollen parallel Ausgaben auf der Konsole produzieren.

a) Erstellen Sie eine Klasse namens „Film“, welche die Schnittstelle „Runnable“ erbt! [2]

b) Geben Sie ihr ein Attribut vom Typ „Thread“, welches im Konstruktor erzeugt wird! [2]

c) Verwenden Sie dazu den „Thread“-Konstruktoraufruf mit „target“ und „name“! [2]

d) Nehmen Sie den Wert von „name“ als Parameter entgegen! [2]

Hinweis: Der Wert von „target“ ist die aktuelle Objektreferenz selbst.

e) Starten Sie den als Attribut referenzierten Thread in der „start“-Methode! [2]

f) Verwenden Sie in der zu implementierenden „run“-Methode eine Zählschleife mit FÜNF Durchläufen! [2]

g) Geben Sie bei jedem Durchlauf den Namen des aktuellen Threads und den Wert der Laufvariablen auf der Konsole aus! Geben Sie NACH der Schleife die Nachricht „Fertig“ aus! [2]

Hinweis: Thread.currentThread()

h) Lassen Sie den Thread eine zufällige Zeit zwischen 0 sund 1 s lang schlafen! [2]

i) Fangen Sie geworfene Ausnahmen ab! [2]

j) Erzeugen und starten Sie in einer Startklasse mit „main“-Methode drei Objekte des Typs „Film“ mit beliebigen Namen! [2]

Ergebnis: Die Ausgaben der Ablaufstränge erfolgen abwechselnd schnell.

Viel Erfolg!