Aufgabenblatt 1

- Abgabetermin: **Montag, 02.05.2016 12:00 Uhr (mittags)**
- Zur Prüfungszulassung müssen in einem Aufgabenblatt mind. 25% der Punkte erreicht werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte bestanden werden.
- Die Aufgaben müssen in *Zweiergruppen* bearbeitet werden.
- Jede Hausaufgabe enthält eine optionale Zusatzaufgabe, die bei erfolgreicher Bearbeitungen 1 Klausurpunkt zählt.
- Abgabe:
  - Die Abgabe erfolgt über CodeOcean[1]. CodeOcean ermöglicht die Bearbeitung und Ausführung der Quelldateien im Browser. Bei Bedarf können die Sources (Quelltexte) und Unit Tests von der Übungs-Webseite[2] heruntergeladen werden, um lokal zu entwickeln.
  - CodeOcean benutzt Java Version 8.
  - Alle Aufgaben sind über das CodeOcean einzureichen. CodeOcean ermöglicht das Anfügen von Kommentaren bei der Abgabe, darüber hinaus können auch Kommentare in den Source-Code geschrieben werden.
  - Geben Sie bei **jeder** Aufgabe beide Namen Ihrer Gruppe an.
  - Achten Sie darauf, dass Ihr Quelltext hinreichend kommentiert ist.

Aufgabe 1: Vorbereitung

2 P

Melden Sie sich mit Ihrer HPI-Emailadresse beim Abgabesystem an und schreiben Sie sich in den Kurs ein (dies ist für jedes Teammitglied nötig). Sie können die Aufgabe im Browser bearbeiten oder das Archiv assignment1_sources.zip[3] herunterladen um es lokal zu bearbeiten, und die fertigen Dateien im Abgabesystem einfügen. Das Archiv enthält das Gerüst (Klassen- sowie Methodendeklarationen und Unit Tests), auf dem die folgenden Aufgaben aufbauen.

Aufgabe 2: Dreieckszahlen

4 P

Gegeben sei die folgende Definition der Dreieckszahlen: $\Delta(n) = \sum_{i=1}^{n} i, n \in \mathbb{N}^+$.

- Implementieren Sie die Funktion `String hex(int n)` der Klasse `assignment1.TriangularNumber` (im Ordner `src`) zur iterativen Berechnung der Dreieckszahlen ($\Delta(n)$).
- Stellen Sie sicher, dass ungültige Werte für n ($n \in \mathbb{Z}_{\leq 0}$) mit dem Werfen einer `IllegalArgumentException` behandelt wird.
- Achten Sie darauf, dass das erwartete Ergebnis der Funktion die *hexadezimale* String-Repräsentation der Dreieckszahl ist (`TriangularNumber.hex(7)` $\Rightarrow$ `1c`).
- Versichern Sie sich, dass alle Tests der Klasse `assignment1.TriangularNumberTest` (im `test`-Ordner) erfolgreich durchlaufen.

[1] https://open.hpi.de/courses/pt2-2016
[2] https://hpi.de/naumann/teaching/current-courses/ss-16/uebung-programmierechnik-ii.html
[3] https://hpi.de/fileadmin/user_upload/fachgebiete/naumann/lehre/SS2016/PTII_Uebung/assignment1_sources.zip

Aufgabe 3: File I/O in Java

4 P

- Implementieren Sie die Funktion `int sumFile(String fileName)` der Klasse `assignment1.Files`, die eine Datei einliest und sämtliche *positiven* Zahlen dieser Datei aufaddiert. Nehmen Sie an, dass in jeder Zeile genau eine ganze Zahl steht.
- Falls die angegebene Datei nicht gefunden werden kann soll die Funktion den Wert -1 zurück geben.
- Implementieren Sie ebenfalls die Funktion `int sumFile2(String fileName)`, die eine Datei einliest und sämtliche *positiven reellen* Zahlen aufaddiert. Gehen Sie davon aus, dass die angegebene Datei eine reelle Zahl pro Zeile beinhaltet. Runden Sie **jede** einzelne Zahl zu der nächstgelegenen ganzen Zahl auf bzw. ab, bevor sie zur Gesamtsumme addiert wird. *Hinweis: Beachten Sie die Bedeutungen von Komma und Punkt in Zahlenrepräsentation zwischen Deutschland und englischsprachigen Ländern.*
- Stellen sie sicher, dass alle Tests der Klasse `assignment1.Files` (siehe test-Ordner) erfolgreich abgearbeitet werden.

Aufgabe 4: Methoden-Overloading in Java

4 P

Implementieren Sie die in `assignment1.Operators` folgenden vorgegebenen Methodendefinitionen, die jeweils den +-Operator in Java verwenden, um übergebene Werte zu addieren oder konkatenieren. Nehmen Sie im Folgenden an, dass die übergebenen int-Werte positiv sind und sämtliche übergebene Strings Integer-Zahlen darstellen. Stellen sie sicher, dass alle Tests der Klasse `assignment1.Operators` (siehe `test`-Ordner) erfolgreich abgearbeitet werden.

- `int addValuesIntInt(int a, int b)`
- `int addValuesIntString(int a, String b)`
- `int addValuesStringStringtoInt(String a, String b)`
- `String addValuesStringStringtoString(String a, String b)`
- `String concatValuesStringInttoString(String a, int b)`
- `String concatValuesIntInttoString(int a, int b)`
- `int concatValuesIntInttoInt(int a, int b)`

Aufgabe 5: Fakultät

4 P

Die Fakultät einer Zahl n sei definiert durch: $n! = \prod_{i=1}^{n} i, n \in \mathbb{N}^+$.

Weiterhin sei festgelegt, dass $0! = 1$.

- Implementieren Sie die Funktion `fact(int n)` der Klasse `assignment1.Factorial` (src-Ordner) zur Berechnung der Fakultät ($n!$), die das Ergebnis als String zurückgibt.
- Für die Übung schränken wir den Definitionsbereich wie folgt ein: $0 \leq n \leq 100$. Alle ungültigen Eingaben für n sollen mit einer `IllegalArgumentException` behandelt werden.
- Stellen sie sicher, dass alle Tests der Klasse `assignment1.FactorialTest` (siehe test-Ordner) erfolgreich abgearbeitet werden.

Aufgabe 6: Rekursion in Java

5 P

Die Fibonacci-Zahlen sind definiert als:

$$fib(n) = \begin{cases} 1, & n <= 2 \\ fib(n-1) + fib(n-2), & n > 2 \end{cases}, n \in \mathbb{N}^+$$

- Implementieren Sie die Funktion `recursive(int n)` der Klasse `assignment1.Fibonacci` (siehe Ordner src) zur *rekursiven* Berechnung der Fibonacci Zahl ($fib(n)$).
- Implementieren Sie die zweite Funktion (`iterative(int n)`) der Klasse `assignment1.Fibonacci`, die die Fibonacci-Zahlen effizient *ohne Rekursion* berechnet.
- Für die Übung schränken wir den Definitionsbereich wie folgt ein: $0 < n \leq 50$. Alle ungültigen Eingaben für n sollen mit einer `IllegalArgumentException` behandelt werden.
- Stellen sie sicher, dass alle Tests der Klasse `assignment1.FibonacciTest` im test-Ordner erfolgreich abgearbeitet werden.
- Beobachten Sie die Laufzeit beider Funktionen (z.B. bei der Berechnung von $fib(47)$) und diskutieren Sie ihre Beobachtungen kurz in Form eines Quellcodekommentars.
- Betrachten Sie Ihre Funktion zur Berechnung der *Dreieckszahlen* erneut. Schreiben Sie eine Funktion `hexRecur(int n)`, die die Berechnung rekursiv statt iterativ ausführt. Bei der Ausführung von `hexRecur(100000)` tritt ein unerwartetes Verhalten auf. Erklären Sie Ihre Beobachtungen und Interpretation. Schreiben Sie diese als Kommentar in die Quelldatei.

Zusatzaufgabe: S-Bahn

Die S-Bahn Berlin hat eine neue S-Bahn-Linie eingerichtet, die direkt von einer Studenten-WG zum HPI fährt. Anstatt der gesamten Bahnlinie eine Nummer zu geben sind die Bahnen, die täglich fahren, chronologisch numeriert, d.h., die erste Bahn hat die Nummer 1, die nächste die Nummer 2, etc. Jedoch hat die neue S-Bahn-Linie der Pünktlichkeit der S-Bahn nicht geholfen. Durch wiederholte Beobachtungen haben die Studenten herausgefunden, dass Bahnen, deren Nummer nicht aus jeweils paarweise unterschiedlichen Ziffern bestehen, typischerweise ausfallen. Die Bahn mit der Nummer $n = 143$ wird beispielsweise püntklich fahren, die Bahn mit der Nummer $n = 144$ jedoch nicht. Die einzige Ausnahme sind Bahnen, deren Nummern ein Palindrom, also von vorne und hinten gelesen identisch sind. Dementsprechend wäre die Bahn 141 wieder pünktlich.

Die Studenten möchten nun wissen, wie viele Bahnen sie nehmen könnten, deren Nummer zwischen den Zahlen A und B liegt. A definiert die früheste S-Bahn, die die Studenten bedingt durch nächtliche Partys und ein Mindestbedürfnis an Schlaf nehmen können und B die späteste S-Bahn, die durch den Vorlesungsbegin definiert ist.

Vervollständigen Sie die Methode `int numVerbindungen(int A, int B)` der Klasse `assignment1.Sbahn`, um die Zahl der pünktlich fahrenden Sbahnen zu ermitteln, die die Studenten nehmen können.

Da nicht mehr als eine S-Bahn pro Minute fahren kann stellen Sie sicher, dass $1 \leq A \leq B \leq 86.400$, indem für fehlerhafte Eingaben eine `IllegalArgumentException` geworfen wird.