Aufgabenblatt 1

- Abgabetermin: **Donnerstag, 24.04.14 12:00 Uhr (mittags)**
- Zur Prüfungszulassung muss ein Aufgabenblatt mit mind. 25% der Punkte bewertet werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte.
- Die Aufgaben müssen in *Zweiergruppen* bearbeitet werden.
- Abgabe:
  - Geben Sie im Abgabesystem[1] ein ZIP-Datei ab, welches den Quelltext aller implementierten Klassen (Java 7), die Unit Tests (JUnit 4), sowie ein Ant-script (Ant 1.8) enthält.
  - Geben Sie bei der Einreichung Ihrer Lösung den Namen des anderen Gruppenmitglieds unter "Authors" an (pro Gruppe nur eine Abgabe).
  - Textuelle Lösungen reichen Sie bitte in einer entsprechend benannten Textdatei (aufgabe_x.txt) im ZIP-Archiv ein.
  - Achten Sie darauf, dass Ihr Quelltext gut kommentiert ist.

Aufgabe 1: Vorbereitung

3 P

Melden Sie sich beim Abgabesystem[1] für die PT2 Übungen an (alle Teammitglieder). Laden Sie sich das Archiv assignment1_sources.zip[2] herunter, es enthält das Gerüst (Klassen- sowie Methodendeklarationen und UnitTests), auf dem die folgenden Aufgaben aufbauen. Schreiben Sie ein Ant-script[3] (ant.xml), welches Ihre Klassen übersetzt und die Testsuite ausführt. Sie können bei dem Script davon ausgehen, dass junit.jar, hamcrest-core.jar und ant-junit.jar im Klassenpfad aufgeführt sind.

Aufgabe 2: Dreieckszahlen

4 P

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

a) Implementieren Sie die Funktion `hex(int n)` der Klasse `assignment1.TriangularNumber` (im Ordner src) zur Berechnung der Dreieckszahlen ($\Delta(n)$).

b) Stellen Sie sicher, dass ungültige Werte für n ($n \in \mathbb{Z}_{\leq 0}$) mit dem Werfen einer `IllegalArgumentException` behandelt wird.

c) Achten Sie darauf, dass das erwartete Ergebnis der Funktion die *hexadezimale* String-Repräsentation der Dreieckszahl ist (`TriangularNumber.hex(7)` $\Rightarrow$ `1c`).

d) Versichern Sie sich, dass alle Tests der Klasse `assignment1.TriangularNumberTest` (im test-Ordner) erfolgreich abgearbeitet werden.

[1] `https://www.dcl.hpi.uni-potsdam.de/submit/`
[2] `http://hpi-web.de/fileadmin/hpi/FG_Naumann/lehre/SS2014/PTII/assignment1_sources.zip`
[3] `http://ant.apache.org/manual/tutorial-HelloWorldWithAnt.html`

Aufgabe 3: Fakultät

4 P

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

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

a) Implementieren Sie die Funktion `hex(int n)` der Klasse `assignment1.Factorial` (src-Ordner) zur Berechnung der Fakultät ($n!$).

b) 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.

c) Achten sie darauf, dass das erwartete Ergebnis der Funktion die hexadezimale String-Repräsentation der Dreieckszahl ist (`assignment1.Factorial.hex(7)` $\Rightarrow$ `13b0`).

d) Stellen sie sicher, dass alle Tests der Klasse `assignment1.FactorialTest` (siehe test-Ordner) erfolgreich abgearbeitet werden.

Aufgabe 4: Fibonacci Zahlen

5 P

Die Fibonacci-Zahlen sind definiert als:

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

a) Implementieren Sie die Funktion `recursive(int n)` der Klasse `assignment1.Fibonacci` (siehe Ordner src) zur *rekursiven* Berechnung der Fibonacci Zahl ($fib(n)$).

b) Implementieren Sie die zweite Funktion (`iterative(int n)`) der Klasse `assignment1.Fibonacci`, die die Fibonacci-Zahlen effizient *ohne Rekursion* berechnet.

c) 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.

d) Stellen sie sicher, dass alle Tests der Klasse `assignment1.FibonacciTest` im test-Ordner erfolgreich abgearbeitet werden.

e) Beobachten Sie die Laufzeit beider Funktionen (z.B. bei der Berechnung von $fib(47)$) und diskutieren Sie ihre Beobachtungen kurz. Speichern Sie die Lösung als Textdatei (aufgabe_4e.txt) und fügen Sie die Datei dem ZIP-Archiv hinzu.

Aufgabe 5: Leiteraufstieg

4 P

Gegeben sei eine Leiter mit n Sprossen. Um eine Leiter hochzuklettern, kann man in einem Schritt eine bzw. zwei Sprossen überwinden. Dabei besitzt eine Leiter maximal 90 Sprossen.

a) Implementieren Sie die Funktion `combinationCount(int n)` der Klasse `assignment1.LadderSteps` (siehe Ordner src) zur Berechnung der Anzahl der unterschiedlichen Ein- und/oder Zwei-schritt Kombinationen, um die Leiter (der Größe n) zu beklimmen. Achtung, die Reihenfolge der Schritte ist *relevant*!

b) Stellen sie sicher, dass alle Tests der Klasse `assignment1.LadderStepsTest` im test-Ordner erfolgreich abgearbeitet werden.

Zusatzaufgabe: Halloween

Eine Gruppe von N Kindergartenkindern sammelt zu Halloween Süßigkeiten. Jedes Kind hat eine spezifische Zahl von gesammelten Süßigkeiten, die sich in ihren Beuteln befinden. Die Anzahl der Süßigkeiten entspricht $s[i]$ $(i = 0, 1, \ldots, N-1)$, also der Anzahl der Süßigkeiten des i-ten Kindes.

Um Streitigkeiten zwischen den Kindern vorzubeugen, entschließt sich der Kindergartenleiter die Zahl der Süßigkeiten pro Kind anzugleichen. Dafür kann er nur eine Operation ausführen: er gibt allen Kindern bis auf einem ein zusätzliches Bonbon. Anders gesagt, ein einziges Kind hat Pech gehabt und erhält keine zusätzliche Süßigkeit für seinen Beutel. Der Pechvogel kann natürlich von Operation zu Operation variieren. Der Kindergartenleiter könnte diese Operation nun beliebig oft anwenden um die Beutel anzugleichen, jedoch ist er ein beschäftigter Mann und möchte nur die minimale Anzahl an Operationen ausführen.

Ihre Aufgabe ist es nun die minimale Anzahl an Operationen in möglichst kurzer Zeit zu bestimmen. Implementieren Sie dazu die Methode `minOperationCount(int[] s)` der Klasse `assignment1.SweetsEqualizer` (im src-Ordner), die die minimale Anzahl von nötigen Operationen zurückgibt. Die Aufgabe gilt als gelöst, wenn die Funktion für jede mögliche Eingabe ($1 \leq N < 200$ und $\forall 0 \leq i \leq N : 0 \leq s[i] \leq 50000$) *i)* das richtige Ergebnis liefert und *ii)* in minimaler Zeit terminiert (also z.B. der durchschnittliche Desktoprechner des Kindergartenleiters die Berechnung in *deutlich* unter einer Sekunde abschließt). Zum Testen stehen Ihnen einige Unittests zur Verfügung (siehe Klasse `assignment1.SweetsEqualizerTest` im test-Ordner).