Aufgabe 1: Zahlenschloss

Ein Fahrrad-Zahlenschloss besteht aus Ringen mit den Ziffern 0 bis 9. Ringe können aufwärts und abwärts gedreht werden.

Beim Absperren soll das Schloss auf eine Kombination mit gleichen Ziffern gestellt werden. Hierzu soll ein Tool geschrieben werden, das zu einer beliebigen Startkombination die Einstellungen mit gleichen Ziffern findet, die mit der minimalen Anzahl an Drehungen am Schloss erreicht werden können. Es soll sowohl die Anzahl an Drehungen, als auch die Ziffern der Kombinationen bestimmt werden.

Beispiele

Startkombination	*Kombinationen*	*Anzahl an Drehungen*
1-1-1-1	1-1-1-1	mit 0 Drehungen erreicht
1-1-9-9	0-0-0-0 1-1-1-1 9-9-9-9	jeweils mit 4 Drehungen erreicht

Weitere Anforderungen

Die API des Tools kann frei gewählt werden, es muss aber möglich sein die Ziffern einer Startkombination zu übergeben (z.B. als String, int, Collection, Array…) und die minimale Anzahl an Drehungen sowie die Ziffern der Kombinationen programmatisch abzurufen. Diese API soll in Unit-Tests verwendet werden. Eine grafische Benutzeroberfläche ist nicht notwendig.

Aufgabe 2: Soziales Netzwerk

Teil A

Wir wollen ein neues soziales Netzwerk entwickeln. Unser Netzwerk kann beliebig viele Personen als Mitglied haben. Personen können beliebig viele Freunde haben. Freunde sind Personen aus dem Netzwerk.

Personen können Nachrichten versenden. Es gibt zwei Varianten an wen die Nachrichten versendet werden können:

- Variante 1: An Freunde und Freunde von Freunden.
- Variante 2: An alle, die man über Freundschaftsbeziehungen erreichen kann (Freunde von Freunden von Freunden und so weiter)

An der Person ist zu erfassen, wann und welche Nachrichten versendet wurden.

Teil B

Wenn eine Nachricht versendet wurde, ist zu ermitteln wieviel Personen insgesamt die Nachricht bekommen haben. Die ermittelte Anzahl soll an der Nachricht abgelegt werden.

Teil C : Simulation 1

Für unser soziales Netzwerk wollen wir auch eine Simulation implementieren.

Wir gehen davon aus, dass 10.000 Personen im sozialen Netzwerk sind.

Weiter nehmen wir an:

- 50 Personen haben 10 Freunde (Gruppe A)
- 30 Personen haben 20 Freunde (Gruppe B)
- 15 Personen haben 40 Freunde (Gruppe C)
- 5 Personen haben 80 Freunde (Gruppe D)

Die restlichen Personen können beliebig viele Freunde haben. Es ist auch möglich, dass eine Person keine Freunde hat.

Wir führen auch eine Zeitangabe für unsere Simulation ein. Der erste Tag hat den Wert 0, für jeden weiteren Tag zählen wir 1 dazu. Unsere Simulation startet am Tag 0 mit 10.000 Personen und der oben angegebenen Verteilung. Jeden Tag bekommen Personen neue Freunde und das Netzwerk wächst:

- Personen, die höchstens 25 Freunde haben, bekommen einen neuen Freund, der schon im unserem Netzwerk war und einen neuen Freund, der neu ins Netzwerk dazukommt
- Personen, die mehr als 25 Freunde haben, bekommen einen neuen Freund, der schon im Netzwerk war und drei, die neu ins Netzwerk dazukommen

Personen, die neu ins Netzwerk dazukommen, haben an ihrem ersten Tag im Netzwerk also einen Freund.

Jeden Tag versendet jede Person im Netzwerk zwei Nachrichten (eine Nachricht Variante 1 und eine Variante 2).

Für jeden Tag soll die Anzahl der Personen im Netzwerk und für eine beliebige Person die Anzahl der erreichten Personen in eine Datei (nur Text) ausgegeben werden.

Beispiel für die Ausgabe:

Tag 0 – Anzahl Personen 10.000
Nachricht Variante 1 hat 40 Personen erreicht und Nachricht Variante 2 hat 2000 Personen erreicht
Tag 1 – Anzahl Personen 11.000
Nachricht Variante 1 hat 60 Personen erreicht und Nachricht Variante 2 hat 3000 Personen erreicht

Die Simulation ist für eine variable Anzahl an Tagen durchzuführen.

Hinweise für die Entwicklung:

- Als Entwicklungsumgebung soll Eclipse verwendet werden
- Für die beiden Aufgaben (Zahlenschloss, soziales Netzwerk) sollen zwei Projekte angelegt werden
- Beide Aufgaben sollen in Java implementiert werden
- Unit Tests sollen mit JUnit 4 implementiert werden
- Für alle Methoden sollen JUnit Tests geschrieben werden.
- Auf JavaDoc kann verzichtet werden
- Der Fokus soll auf sauberem, testbarem und leserlichem Code liegen, der von anderen Entwicklern schnell verstanden werden kann. (Siehe auch das Buch Clean Code von Robert C. Martin)