Hausübung 1

Die Aufgabe soll allein bearbeitet werden, Sie dürfen sich aber mit ihren Kommiliton(inn)en austauschen. Plagiate werden nicht akzeptiert. Zum Abgabetermin halten Sie die Lösung vorführbereit an Ihrem Rechnerplatz bereit. Vorher fügen Sie die Lösung als Java-Archiv (jar-Format) der Moodle-Aktivität „Hausübung 1“ hinzu. Es wird empfohlen, schon jetzt ein git-Repository zu verwenden. In der nächsten HÜ wird dies verlangt.

Abgabetermin ist Dienstag, 22.11.2016.

Den Stand der Abgabe können Sie ebenfalls im Moodle einsehen. **Bitte beachten Sie folgende Abgabemodalitäten:**

1. UTF-8 vor Projektbeginn als Zeichenkodierung wählen! (Bei Windows-Rechnern.)
2. Der Dateiname ist ihre Matrikelnummer gefolgt von Ihrem Nachnamen, also z.B. 123456_Meier.jar.
3. Die Datei enthält den Quellcode (also alle .java Dateien), den Bytecode und die javadoc-Dokumentation.

Ansonsten kann Ihre Lösung nicht bearbeitet werden!

Die Hausübung ist erst anerkannt, wenn sie auch in einer Übungsgruppe von Ihnen erläutert wurde. Bitte bearbeiten Sie die Hausübung selbstständig, aber erst nachdem Sie die Aufgabenstellung und die Entwurfsvorgaben **vollständig gelesen** und **verstanden** haben!

Anleitung zur Abgabe der Hausübung:

1. Moodle-Kurs *Programmieren interaktiver Systeme* öffnen.
2. Aktivität *Hausübung 1* (auch unter Aktuelle Termine zu finden) öffnen.
3. Mit dem Button *Meine Lösung bearbeiten* kann die Lösung abgegeben werden. Die Lösung kann auch mehrmals abgegeben werden, die letzte Version wird überschrieben.
4. Unter Feedback können Sie den aktuellen Stand der Lösung einsehen. Ihre Lösung wurde akzeptiert, wenn der Wert von "Bewertung" auf YES steht. Unter *Abgabestatus → Abgabekommentar*" wird Ihre Lösung kommentiert. Sie sollten darauf reagieren!
5. Sollten Sie allgemeine Fragen zur Abgabe oder Implementierung haben, verwenden Sie bitte das soziale Forum von Moodle. Bei spezifischen Fragen oder Problemen kontaktieren Sie die Tutoren direkt per E-Mail (z.B. maximilian.zipp@mni.thm.de).

Abnahmekriterien

1. Alle Klassen gehören zum Paket `pis.hue1`**.**
2. Dateinamen enthalten **keine** Umlaute, also **nicht** `Würfel` als Klassenname!
3. Der Testfall ist bestanden.
4. Ihr Java-Archiv enthält alle **Quelltext-Dateien**, Bytecode-Dateien und javadoc-Dateien
5. Der Klartext soll für das Verschlüsseln **nicht** vorverarbeitet werden. Also bitte **nicht** Leerzeichen, Zeilentrenner o.ä. entfernen, ebensowenig alles in Groß- oder Kleinbuchstaben verändern: Entschlüsseln eines verschlüsselten Textes liefert *denselben* Ausgangstext.
6. Die **Klasseninvariante** der Wuerfel-Klasse steht bei der Definition der privaten Attribute dieser Klasse. **Sie muss angegegeben weden**. Ansonsten sind die Java-Kodierrichtlinien des FB MNI einzuhalten.

Aufgabenstellung

In [Leib] wird der *Doppelwürfel* als ein für Spione optimales Hand-Verfahren zur Verschlüsselung beschrieben:

"Geheimer Bestandteil des Verfahrens ist ein Losungswort (im Beispiel "Schwarzwald"). Man sortiert die Buchstaben des Losungswortes nach dem Alphabet, nummeriert sie durch und schreibt unter jeden Buchstaben des Losungswortes seine so ermittelte Nummer. Das ergibt die sogenannte Zahlenlosung; es handelt sich um eine Permutation der Zahlen von 1 bis *n*, wobei *n* die Länge des Losungswortes ist." Im Beispiel lautet die Zahlenlosung (8 3 5 9 1 7 11 10 2 6 4), in Ihrer Lösung nummerieren Sie die Spalten von 0 bis *n-1*, nehmen also die Permutation (7 2 4 8 0 6 10 9 1 5 3).

"Nach dieser Vorbereitung gewinnt man den Geheimtext, indem man unter die Losung zeilenweise den Klartext einträgt und spaltenweise in der Reihenfolge der Losungsnummern wieder ausliest. Der Empfänger des Geheimtextes schreibt zunächst die (Zahlen-)Losung nieder und zeichnet sich einen Würfelkasten, der so hoch ist, dass der Geheimtext gerade hineinpasst. In diesen Kasten trägt er spaltenweise in der Reihenfolge der Losungsnummern den Geheimtext ein und liest den Klartext zeilenweise ab.

Wird ein Würfel-Geheimtext erneut mit einem (anderen) Würfel verschlüsselt, hat man das Doppelwürfelverfahren."

Beispiel: Aus dem Klartext `eintreffendersendungverspaetetneuerterminfolgt` wird mit dem Losungswort *Schwarzwald* der Geheimtext `rneregnfirsrtdeulnsptnveoedtmeeregteaefntnfuei` ermittelt:

```
S C H W A R  Z  W A L D
8 3 5 9 1 7 11 10 2 6 4
-----------------------
e i n t r e  f  f e n d
e r s e n d  u  n g v e
r s p a e t  e  t n e u
e r t e r m  i  n f o l
g t
-----------------------
```

Verschlüsselt man den so erhaltenen Geheimtext wieder, diesmal mit dem Losungswort *Schwenningen*, so ergibt sich der zweite Geheimtext als `ndeeelmtsvtrngieedffprugnennsefiteereertoarutn`:

```
 S C H  W E N N I N G E  N
11 1 5 12 2 7 8 6 9 4 3 10
--------------------------
 r n e  r e g n f i r s  r
 t d e  u l n s p t n v  e
 o e d  t m e e r e g t  e
 a e f  n t n f u e i
--------------------------
```

Schreiben Sie ein Programm, das den Spion bei seiner Verschlüsselungsarbeit unterstützt!

Funktionalität der Benutzungsschnittstelle

Entwerfen Sie eine geeignete Benutzeroberfläche. Sie können wählen, ob Sie die neuere JavaFX-Klassenbibliothek verwenden wollen, oder die etwas angestaubte Swing-Bibliothek. Über die grafische Benutzeroberfläche (Swing oder JavaFX) kann der Benutzer:

1. Klartexte eingeben (z.B. in einem `JTextArea` Objekt bei Swing)
2. Geheimtexte eingeben (z.B. in einem anderen `JTextArea` Objekt),
3. die beiden Losungsworte eingeben und verändern (z.B. in je einem `JTextField` Objekt),
4. durch Drücken einer Schaltfläche (Objekt der Klasse `JButton`) eingebene Klar- bzw. Geheim-Texte nach obigem Verfahren verschlüsseln bzw. entschlüsseln und die Ergebnisse angezeigt bekommen,
5. das Programm beenden.

Entwurfsvorgaben und Hinweise:

1. **Lose Kopplung:** Trennen Sie scharf zwischen dem Codec, also der Komponente, die die Verschlüsselung vornimmt, und der GUI-Komponente, mit der unser Spion den Codec verwendet. Um diese Trennung zu erreichen, definieren Sie die Klasse `Wuerfel` als Implementierung einer (weiter unten definierten) Schnittstelle `Codec` mittels
   `class Wuerfel implements Codec {..}`

   In der GUI-Komponente (Namensvorschlag `CodecGUI`) soll dann auf die zwei (!)Würfel-Objekte (eines für jede Losung) *nur* über diese Schnittstelle zugegriffen werden. Dazu sollte die `CodecGUI` als Attribut zwei `Codec`-Objekte besitzen, die schon mit der Konstruktorinitialisierung des `CodecGUI`-Objektes festgelegt und danach nicht mehr geändert werden, d.h. eine Methode `setzeCodec(Codec)` ist *nicht* vorgesehen. Zur Laufzeit kann dann das Kodierverhalten der beiden `Codec`-Objektes mit der Methode `setzeLosung(String)` geändert werden. Mit der Einführung der Schnittstelle `Codec` wird erreicht, dass die GUI-Komponente für beliebige Implementierungen dieser Schnittstelle `Codec` verwendet werden kann. GUI-Komponente und `Codec`-Implementierungen sind nun voneinander entkoppelt und kooperieren nur indirekt über die `Codec`-Schnittstelle:

   ```
   interface Codec{
   public String kodiere(String klartext);
   public String dekodiere(String geheimtext);
   public String gibLosung();
   public void setzeLosung(String schluessel)throws
   IllegalArgumentException // bei ungeeignetem Schlüssel!
   }
   ```

   Ergänzen Sie diese Schnittstellendefinition um `javadoc`-Kommentare. Diese sollen geeignete Vor- und Nachbedingungen an die verlangten Schnittstellen-Methoden ausdrücken! Diese Bedingungen werden in der implementierenden Klasse `Wuerfel` natürlich *nicht* wiederholt. In der Klasse `Wuerfel` geben die Kommentare an, in welcher besonderen Art die Schnittstellenvorgabe erfüllt wird.
2. Bei der Berechnung der Zahlenlosung beachten Sie bitte, dass Groß- und Kleinschreibung innerhalb des Losungswortes irrelevant ist. *Schwarzwald* und *SCHWARZWALD* legen also dieselbe Zahlenlosung fest.
3. Das Losungswort (z.B. *Schwarzwald*) ist nur eine Merkhilfe für eine Permutation! (Falls Sie nicht mehr wissen, was das ist, schlagen Sie es nach! Sollte in diskreter Mathematik behandelt worden sein.) Diese Permutation legt fest, wie die Spalten vertauscht werden. Eine Permutation der Zahlen *0,..,n-1* lässt sich hervorragend als ein **`int`**-Array der Länge *n* beschreiben. Auch die Berechnung des Inversen einer Permutation (Vertauschung der Spalten rückgängig machen) ist in dieser Darstellung sehr einfach. So ist z.B. das Inverse von (7 2 4 8 0 6 10 9 1 5 3) die Permutation (4 8 1 10 2 9 5 0 3 7 6). Sind alle **`int`**-Arrays sinnvoll als Permutation interpretierbar? Formulieren Sie die Klasseninvariante! Wie

setzen Sie sie durch?

4. Es ist nicht nötig, `String`-Objekte in **`char`**-Arrays zu wandeln: Die Klasse `String` bietet schließlich eine Methode `charAt(int)`. Besser, Sie verwenden meist die Klasse `StringBuilder` ! Warum diese? Warum nicht `StringBuffer`?
5. Erzeugen und behandeln Sie eine Ausnahme `IllegalArgumentException`, wenn die Losungsworte ungeeignet sind. Was ungeeignet ist, müssen Sie natürlich selbst festlegen!
6. Dokumentieren Sie Ihre Lösung ausführlich mit `javadoc`-Kommentaren! Die daraus erzeugten HTML-Dateien geben Sie in der jar-Datei mit ab!
7. Zum Test Ihres Programmes entschlüsseln Sie bitte folgenden Text
   ```
   steuenelaoewecsvarhrtruidstidfereeedgetelzlieoaeazeseaiiesiejb
   bwcnhirziernzbveslrnoswsuseugtgeeeennemnfbhinahfennetbeutsocgv
   hueekjstgdahhneignruidtitlbstbaidnrensgupacehmlemiereeatcmnerl
   bninlln
   ```
   der mit den Losungsworten *Programmierung* und *Vergnuegen* verschlüsselt wurde.
8. Um zu prüfen, ob Ihnen die Entkopplung der Benutzeroberfläche (Klasse `CodecGUI`) von der Modell-Komponente `Wuerfel` gelungen ist, mischen Sie die Komponenten verschiedener Lösungen: Bauen Sie eine neue Lösung mit der Benutzeroberfläche der einen Lösung mit der `Wuerfel`-Klasse der anderen Lösung!

[Leib] Otto Leiberich, "Vom diplomatischen Code zur Falltürfunktion", Spektrum der Wissenschaft, Juni 6/1999, S. 26 - 34. (Der Kasten zu Losung 2 auf Seite 31 enthält einen Fehler: Dem Buchstaben 'w' aus "Schwenningen" wurde die Zahl 5 zugeordnet! Der Fehler ist in der obigen Tabelle nicht mehr enthalten.)
Letzte Änderung 24.10.2016