

Übungsserie Nr. 3

Ausgabe: 11. März 2015

Abgabe: 18. März 2015

Hinweise

Für diese Serie benötigen Sie das Archiv

<http://vs.inf.ethz.ch/edu/FS2015/I2/downloads/u3.zip>.

1. Aufgabe: (6 Punkte) String und StringBuffer

Strings gibt es in Java in zwei Varianten: den unveränderlichen (immutable) `String` und den veränderlichen (mutable) `StringBuffer`. Unveränderliche Objekte können nach ihrer Erzeugung nicht mehr modifiziert werden. Dem Compiler und der Laufzeitumgebung sind dann Optimierungen möglich, z.B. dass sich mehrere Objekte den selben Speicher teilen können. Falls jedoch eine Modifikation nötig ist, muss stattdessen eine Kopie angelegt werden, was Laufzeit und Speicher kostet. Es hängt also vom Anwendungsfall ab, welcher Typ effizienter ist.

In dieser Aufgabe implementieren Sie eine Variante der Caesar-Verschlüsselung¹. Hierbei handelt es sich um eine der einfachsten Verschlüsselungen: Beim Verschlüsseln eines Texts (*encryption*) wird jeder Buchstabe des Klartexts (*plaintext*) durch den Buchstaben ersetzt, der im Alphabet 3 Zeichen weiter hinten steht. Bei der Entschlüsselung (*decryption*) wird entsprechend im Schlüsseltext (*ciphertext*) jedes Zeichen um 3 reduziert.

Wir berufen uns hier auf die ASCII-Tabelle und ignorieren der Einfachheit halber Überläufe am Ende des Alphabets. Das Verschlüsseln von *Xaver5* führt somit zum Schlüsseltext *[dyhu8]*.

(1a) (3 Punkte) Implementieren Sie die Methode `decrypt`, welche einen gegebenen Schlüsseltext wieder in Klartext umwandelt. Verwenden Sie (ohne die Methodensignatur zu ändern) `StringBuffer` anstelle von `String`, um die Implementierung effizient zu gestalten. Testen Sie Ihre Implementierung mit Hilfe der Unittests.

Hinweis: Sie können sich an der Implementierung von `encrypt` orientieren, die allerdings `Strings` verwendet.

¹http://en.wikipedia.org/wiki/Caesar_cipher

(1b) (2 Punkte)

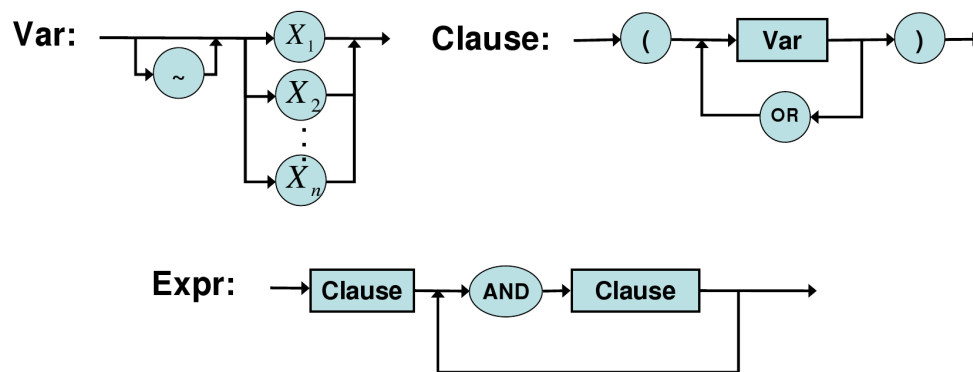
Das Framework enthält ausserdem eine Main-Methode in der Datei `Main.java`. Schauen Sie sich die Main-Methode genau an und beschreiben Sie ihr Verhalten. Führen Sie die Main-Methode aus – wie lautet die Ausgabe? Erklären Sie die unterschiedlichen Laufzeiten bei der Verschlüsselung und der Entschlüsselung.

(1c) (1 Punkt)

Warum werden Strings dann überhaupt verwendet? Beschreiben Sie ein Szenario, in dem die Nutzung von `String` effizienter ist als die Nutzung von `StringBuffer` und begründen Sie Ihre Antwort.

2. Aufgabe: (3 Punkte) Syntaxdiagramm

Betrachten Sie folgendes Syntaxdiagramm:



(2a) (2 Punkte) Geben Sie an, welche der folgenden Ausdrücke nach dem Diagramm *Clause* korrekt erzeugt werden können.

	erzeugbar	nicht erzeugbar		erzeugbar	nicht erzeugbar
X_2	<input type="radio"/>	<input type="radio"/>	$\sim (X_1 \text{ OR } \sim X_2)$	<input type="radio"/>	<input type="radio"/>
$(\sim X_1)$	<input type="radio"/>	<input type="radio"/>	$(X_2) \text{ OR } (\sim X_1 \text{ OR } X_2)$	<input type="radio"/>	<input type="radio"/>

(2b) (1 Punkt) Geben Sie an, welche der folgenden Ausdrücke nach dem Diagramm *Expr* korrekt erzeugt werden können.

	erzeugbar	nicht erzeugbar
$(X_1 \text{ OR } X_2) \text{ AND } (\sim X_1)$	<input type="radio"/>	<input type="radio"/>
$(X_1) \text{ AND } (\sim X_1 \text{ OR } \sim X_2) \text{ AND } (X_2)$	<input type="radio"/>	<input type="radio"/>

3. Aufgabe: (7 Punkte) Syntaxchecker

(3a) (1 Punkt) Ergänzen Sie das Syntaxdiagramm auf Folie 228 im Skript, sodass leere Bäume und leere Teilbäume generiert werden können. Ein leerer Baum bzw. Teilbaum soll dabei durch ein ‘-’ repräsentiert werden. Achten Sie darauf, dass durch Ihre Modifikation keine ungültigen Bäume generiert werden können.

(3b) (6 Punkte) Die Klasse *u3a3.KD* soll ein Syntaxchecker für Wurzelbäume in Klammerschreibweise werden. Vervollständigen Sie die Implementierung.

Hinweis: Schreiben Sie für *Baum*, *Nachfolger* und *Knoten* je eine eigene Funktion, die mit Hilfe der jeweils anderen Funktionen und analog zum jeweiligen Syntaxdiagramm den String sequenziell überprüft und die Anzahl der überprüften Zeichen zurück gibt.