Assignment 1

Abgabe bis zum 15. April 2025 um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 15. April 2025 um 23:59 Uhr auf folgender Webseite ab:
https://assignments.hci.uni-hannover.de
Ihre Abgabe muss aus einer einzelnen zip-Datei bestehen, die den Quellcode und ggf. ein PDF-Dokument bei Freitextaufgaben enthält. Beachten Sie, dass Dateien mit Umlauten im Dateinamen nicht hochgeladen werden können. Entfernen Sie die daher vor der Abgabe die Umlaute aus dem Dateinamen. Überprüfen Sie nach Ihrer Abgabe, ob der Upload erfolgreich war. Bei technischen Problemen, wenden Sie sich an Sebastian Preuss. Es wird eine Plagiatsüberprüfung durchgeführt. Gefundene Plagiate führen zum Ausschluss von der Veranstaltung. In dieser Veranstaltung wird ausschließlich die Java/JavaFX Version 17 verwendet. Code und Kompilate anderer Versionen sind nicht zulässig. Ihre Abgaben sollen die vorgegebenen Dateinamen verwenden. Wiederholter Verstoß gegen diese Regel kann zu Punktabzügen führen.

Aufgabe 1: Ver- und Entschlüsselung

Diese Aufgabe widmet sich der Entwicklung einer Ver- und Entschlüsselung von Charakter-Arrays. Des Weiteren soll eine Methode implementiert werden, die in der Lage ist Charakter-Arrays als einfache Strings auf der Kommandozeile auszugeben. Wie die Ver- und Entschlüsselung funktioniert, wird jeweils in den Teilaufgaben beschrieben.

a) Für die Funktionsweise wird ein sogenannter Pseudozufallszahlengenerator (hier: Linearer Kongruenzgenerator) benötigt. Dieser berechnet die nächste Zufallszahl x_i anhand folgender Formel

$$x_{i+1} = (a * x_i + b) \bmod m$$

Erstellen Sie zunächst die Klasse Encryption (Encryption.java) und implementieren Sie die Funktion `public static int nextInt(int a, int b, int m, int x)`, die den Zahlengenerator umsetzt. Nutzen Sie für die ganze Aufgabe die Initialwerte $x_0 = 3$, $a = 7$, $b = 28$, $m = 256$.

b) Implementieren Sie die Methode `public static void printCharArray(char[] letters)`. Diese Methode nimmt das Charakter-Array *letters* entgegen und gibt dieses mittels `System.out.print` auf der Kommandozeile aus. Dabei soll die Form {'H', 'A', 'L', 'L', 'O'} → "HALLO" eingehalten werden.

c) Realisieren Sie die Verschlüsselung innerhalb der Funktion `public static int[] encrypt(char[] letters, int[] keys)`. Das Array *letters* beinhaltet die zu verschlüsselnden Charaktere und *keys* Zahlen mit denen die Charaktere gleichen Indexes verschlüsselt werden sollen. Charaktere am Index i (c_i) werden mit dem korrespondieren Schlüssel k_i in den Charakter enc_i wie folgt verschlüsselt:

$$enc_i = \begin{cases} c_i \text{ XOR } k_i, & \text{wenn } i = 0 \\ c_i \text{ XOR } k_i \text{ XOR } enc_{i-1}, & \text{sonst} \end{cases}$$

XOR bezeichnet hierbei den Exklusiv-Oder-Gatter. Wird dieser auf zwei Binärzahlen angewandt, werden die Stellen gleichen Indexes gemäß der Wahrheitstabelle (XOR-Gatter) miteinander verrechnet. In Figure 1 ist eine

beispielhafte Rechnung zu finden. In Java entspricht XOR dem ^ Operator. Hierzu müssen Charaktere in Integer u.a durch `int char_as_int = ((int)'a')` umgewandelt werden. Ähnlich kann ein Integer in einen Charakter umgewandelt werden `char int_as_char = ((char)97)`. Diese Methodik nennt man Typecasting. Die verschlüsselten Charaktere sollen anschließend durch die Methode in einem Integer-Array gespeichert und zurückgegeben werden.

$$\begin{array}{r} 1100 \\ XOR\ 1001 \\ \hline 0101 \end{array}$$

Figure 1: Anwendung der XOR-Operation auf den Dezimalwert 12 und 9.

d) Implementieren Sie die Funktion `public static char[] decrypt(int[] letters, int[] keys)`, die die in *letters* gespeicherten verschlüsselten Charaktere wieder entschlüsselt. Überlegen Sie wieso die Entschlüsselung ähnlich wie die Verschlüsselung sein muss. Dafür können Sie sich das Ergebnis von `int result = 5 ^ 3 ^ 3` ausgeben lassen.

$$\begin{array}{r} \mathbf{1001} \\ XOR\ 0011 \\ \hline 1010 \\ XOR\ 0011 \\ \hline \mathbf{1001} \end{array}$$

Figure 2: Schrittweise Anwendung von 5 ^ 3 ^ 3. '^' stellt die Bitweise XOR-Operation dar.

Ergänzen Sie die Main-Funktion um ein beispielhafte Ver- und Entschlüsselung eines Charakter-Arrays Ihrer Wahl. Nutzen Sie den Zufallgenerator, um sich Zufallszahlen in der Form eines Integer-Arrays zur Ver- und Entschlüsselung zu generieren. Geben Sie Ihr Ergebnis der Entschlüsselung aus, um die Korrektheit Ihres Programms zu prüfen.

Aufgabe 2: Alternative Zahlendarstellung

Diese Aufgabe realisiert die Umwandlung einer alternativen Darstellung von Integer auf der Basis von Arrays und vice versa. Weiter wird die Addition mit 1 umgesetzt. Testen Sie die ersten 3 Teilaufgaben mit mind. 2 Beispielaufrufen innerhalb der Main-Methode.

a) Erstellen Sie innerhalb der Klasse IntList (IntList.java) die Methode `public static int countNumbers(int num)`, die die Länge einer Zahl zählt und zurück gibt. (Beispiel: countNumbers(123) → 3) Hierfür kann der Modulo-Operator (%) sinnvoll sein.

b) Nutzen Sie die Methode countNumbers, um die Methode `public static int[] intToList(int value)` zu implementieren. Die Methode soll ein Integer *value* in seine einzelnen Ziffern zerlegen, innerhalb eines Integer-Arrays speichern und dieses anschließend zurückgeben. (Beispiel: intToList(1239) → [1, 2, 3, 9])

c) Realisieren Sie die Methode `public static int listToInt(int[] numbers)`, welche das Ergebnis von intToList umkehren kann. Es fügt dementsprechend die einzelnen Ziffern wieder zu einer Zahl zusammen. (Beispiel: listToInt([1, 2, 3, 9]) → 1239)

d) Die Methode `public static int[] addOne(int[] numbers)` soll die Addition eines Integers *numbers*, welches in seine Bestandteile aufgeteilt wurde, und 1 realisieren. Das Ergebnis soll innerhalb eines Integer-Arrays gespeichert werden. Achten Sie darauf, dass dieses Array ggf. länger sein muss. (Beispiel: addOne([9, 9, 9]) → [1, 0, 0, 0], addOne([1, 2, 3]) → [1, 2, 4])

Fügen Sie außerdem ein Beispiel hinzu, welches die Zahl 9999 zunächst in die alternative Darstellung umwandelt, addOne nutzt, die Darstellung des Ergebnisses wieder umkehrt und anschließend auf der Kommandozeile ausgibt.

Aufgabe 3: Debugging

Jedes Assignment wird eine Debugging-Aufgabe enthalten. In diesen Aufgaben werden Sie einen fehlerhaften Java Codeabschnitt bekommen. Diese Fehler können **syntaktisch** oder **semantisch** sein. Jedoch beschreiben Kommentare immer das **korrekte** Verhalten des Programms. Sie dürfen den Code kompilieren und ausführen um Fehler zu finden. Arbeiten Sie in den Template-Dateien in der `Debug01.zip`.

a) Der Code enthält 7 Fehler. Kompilieren und führen Sie den Code aus. Suchen Sie anhand der Fehlermeldungen des Compilers oder der Laufzeitfehlermeldungen Fehler im Code und korrigieren Sie diese. Beschreiben Sie am Ende der jeweiligen Zeile, was Sie korrigiert haben.

Erstellen Sie während der Fehlerkorrektur einen Blockkommentar unter dem Code. Schreiben Sie die Zeile(n) des Fehlers auf und beschreiben Sie den Fehler. Kopieren Sie die Fehlermeldung, sofern es eine gab, anhand welcher Sie den Fehler erkannt haben. Die Dokumentation soll wie in folgendem Beispiel aussehen:

```
1  public class Debug { //K: class falsch geschrieben (ckass)
2
3  ...
4
5  /*
6  ...
7  Zeile 1: class Keyword falsch geschrieben (ckass):
8  Fehlermeldung:
9  **************
10 Debug.java:1: error: class, interface, or enum expected
11 public ckass Debug {
12        ^
13 **************
14 Der Compiler erwartet eines der drei oben angegebenen Keywords, hat aber nur das falsch
       ↪ geschriebene ckass bekommen.
15 ...
16 */
```