Programmierung 1
Wintersemester 2023/24
Prof. Dr.-Ing. Johannes Konert

Hausaufgabe 6

Vorgesehene Abgabe: 10.12.2023 (23:59 Uhr)

Abgabe:
Für dieses Hausaufgabenblatt ist **eine** *CeasarsCode.java* Datei abzugeben. Dafür gibt es bereits eine **Vorlagedatei in Moodle**, die Sie verwenden sollten, denn darin sind bereits in der main-Methode alle Testfälle reinprogrammiert. Kommentieren Sie ggf. einzelne Zeilen mit // aus, um Teile Ihres Programmcodes zu testen. Das Programm erwartet Aufrufparameter (siehe Erläuterungen direkt im Quellcode).
Die Bearbeitung und Lösung der Hausaufgabe ist insgesamt anspruchsvoll, da hier mehrere der Themen kombiniert werden, welche die letzten Wochen behandelt wurden (Schleifen, Methoden, Datentypen, Arrays, String-Operationen, Konsolen-Parameter). Nutzen Sie daher intensiv das Praktikum für Rückfragen, wenn es Schwierigkeiten gibt.
(Das Programm muss bei Abgabe für sich lauffähig sein. Nehmen Sie auch Quellcodebeispiele der SUs als Basis)

(einzige) Aufgabe 0: Caesars Verschlüsselung
Einleitung: In dieser Aufgabe implementieren Sie eine Text-Verschlüsselung, die auf Julius Ceasars Verschlüsselungsalgorithmus basiert. Dabei wird für jeden Großbuchstaben A bis Z jeweils ein Austauschzeichen definiert (Kleinbuchstaben und andere Zeichen bleiben in dieser Aufgabe einfach erhalten), z.B. könnte man folgende Ersetzungen vornehmen: A durch W, B durch X, C durch Z und so weiter bis Y durch B und Z durch Q. Dies kann man durch den Schlüssel „WXZJKACSRTPNOMLYVFHEGIUDBQ“ darstellen. An Position 0 des Schlüsseltextes steht das Austauschzeichen für A an Position 25 steht das Austauschzeichen für Z. Wendet man den Schlüssel auf den Text „HALLO ANNA“ an so ist der verschlüsselte Text entsprechen „SWNNL WMMW“ (Leerzeichen bleiben unverändert).
Um den Schlüssel in Java zu speichern wird in dieser Aufgabe ein char Array verwendet. Da Character (char) als Datentyp in Java ebenfalls Zahlen sind und lediglich als Zeichen (Buchstaben, usw.) ausgegeben werden, kann man über char auch iterieren (eine Schleife schreiben) und char als Index für Arrays verwenden (!). So können Sie das Austauschzeichen für A an *Position ‚A‘* speichern. Folgendes Codebeispiel in Abbildung 1 verdeutlicht dies.

```
// print chars A to Z
for (char c= 65; c <= 90; c++) {
  System.out.println(c);
}

// create a boring key char[] where each char is replaced by itself
char[] key = new char[256];
for (char c=0; c < 256; c++) {
  key[c] = c;
}
// print all 256 characters' replacements
for (int i=0; i < 256; i++) {
  System.out.println(key[i]);
}
```

Abbildung 1: Exemplarische Verwendung von Datentyp char als Index für Arrays: Dieser Code gibt zunächst alle Großbuchstaben von A bis Z auf der Konsole aus. Dann wird für jedes Zeichen das Zeichen selbst als Austauschzeichen in einem char[] key *definiert (wenig spannend zugegeben) und 256 Zeichen werden zeilenweise auf die Konsole geschrieben.*

(eigentliche Implementierungsaufgaben auf Seite 2)

Vorgeschlagenes Vorgehen zur Implementierung von Ceasars Code (erst komplett lesen):

1. Schreiben Sie eine Methode `public static char[] getKey(String s)`, die einen String aus 26 Zeichen als Argument nimmt (ansonsten liefert die Methode *null* als Rückgabe) und das Zeichen an Position 0 als Austauschzeichen für ‚A' nimmt, das Zeichen an Position 1 als Austauschzeichen für ‚B' usw. Das Resultat soll ein char[] mit 256 Elementen sein (da es 256 verschiedene char-Zeichen gibt). Ein A hat den Wert 65, Z ist 90, für alle anderen Zeichen neben A-Z soll das Zeichen selbst auch das Austauschzeichen sein. (siehe Codeausschnitt in Abbildung 1).
   Mit anderen Worten: Die Methode erstellt aus dem String das char[] mit 256 Einträgen, wobei die Austauschzeichen an Position 65-90 stehen und ergänzt es um alle anderen Zeichen 0-64 und 91-255.

   Skizze mit dem Beispiel WXZJKACSRTPNOMLYVFHEGIUDBQ (Auszug der Array-Indizes 63 bis 92):

?	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\
?	@	W	X	Z	J	K	A	C	S	R	T	P	N	O	M	L	Y	V	F	H	E	G	I	U	D	B	Q	[	\

2. Schreiben Sie eine Methode `public static String encrypt(String text, char[] key)`, die den String mit dem Schlüssel verschlüsselt. Mit anderen Worten: Ersetzen Sie jedes Zeichen in `text` mit seinem Ersetzungszeichen im `key`. *Tipp: In key können Sie das zu ersetzende Zeichen als Index verwenden. Der Wert an dieser Stelle im Array ist das zu verwendende Ersetzungszeichen.*

3. Schreiben Sie eine Methode `public static String decrypt(String cipher, char[] key)`, die den verschlüsselten Text mit dem Schlüssel wieder entschlüsselt. Mit anderen Worten: Ersetzen Sie alle Zeichen im `cipher` wieder durch das ursprüngliche Zeichen. *Tipp: Die Zeichen aus cipher sind im Array key irgendwo als Wert abgelegt (eindeutig) und der Index ist das ursprüngliche Zeichen.*

In der Vorgabedatei *CaesarsCode.java* finden Sie fertige Testfälle in der *main* Methode.
Mit einer funktionierenden Lösung liefern die Testfälle die folgende Ausgabe mit den angegebenen zwei Aufrufparametern (siehe Abbildung 2, zur Kontrolle):

```
$> java CaesarsCode WXZJKACSRTPNOMLYVFHEGIUDBQ "HELLO STUDENTS4711"
SKNNL HEGJKMEH4711
HELLO STUDENTS4711
```

Abbildung 2: Exemplarischer Aufruf des Programms CaesarsCode

Knobelaufgabe auf Seite 3

Knobelaufgabe (keine Punkte, aber lehrreich)

Die Lösung erhalten Sie, indem Sie die weißen Textzeilen unten (weiß auf weiß) herauskopieren und in einen Editor einfügen.

Das folgende Programm soll ein *Histogramm* aller Integer modulo 3 ausgeben, also die Verteilung aller Integer auf die Divisionsreste 0, 1 und 2. Dies sind nicht genau gleich viele, da 2^{32} nicht durch 3 teilbar ist.

Bevor Sie es abtippen und direkt laufen lassen, bitte erst durchsprechen und überlegen, was wo (anscheinend) passiert oder passieren soll. Erst dann starten und sich ggf. wundern. Das Programm enthält nämlich Fehler, die gerne übersehen werden bzw. „nicht gewusst" wurden. Welche sind das? Erklären Sie sich (gegenseitig) warum das zum Fehler führt! Beheben Sie die Fehler! Man lernt einiges über % und *Math.abs* dabei ☺.

```
public class ModuloHistogram {
  public static void main(String[] args) {
    int mod = 3;
    int[] histogram = new int[mod];
    int i = Integer.MIN_VALUE;
      do {
        histogram[Math.abs(i) % mod]++;
      } while (i++ != Integer.MAX_VALUE);
      for (int j= 0; j < mod; j++) {
        System.out.print(histogram[j] + " ");
      }
  }
}
```

LÖSUNG:

LÖSUNG ENDE