Übung 01

Abgabe (*tagging*) im GitLab spätestens Do., 04.05., 8 Uhr
Testat ab Do., 04.05., Details werden noch bekannt gegeben

Vorbereitung

GitLab/Git

Arbeiten Sie *Übung - Vorbereitung* (`vorbereitung.pdf`) durch. Hinterlegt im GWDG GitLab `https://gitlab.gwdg.de/app/2023ss/lecture` im Verzeichnis `uebung`.

Abgabe (*tagging*)

Sind Sie mit der Bearbeitung der Übung 01 in Ihrer lokalen Arbeitskopie fertig und haben alle **Änderungen ins GitLab übertragen**, kennzeichnen Sie das mit dem *annotated tag* `exercise01` wie folgt.

```
git tag -a exercise01 -m "Kommentar"
git push --tag
```

Das **`tagging` muss spätestens am Abgabetermin Do., 04.05., 8 Uhr** erfolgen.

Sie können danach in der Arbeitskopie weiterarbeiten, **der Tutor wird beim Testat seine lokale Kopie auf den Zeitpunkt des *tagging* zurücksetzen.**

Java

Informieren Sie sich in der *Java Platform, Standard Edition 11 API Specification* `https://docs.oracle.com/javase/11/docs/api/` über die Klasse `java.lang.Object`.

Aufgabe 1

Polynome über GF(2)

Implementieren Sie eine Klasse `PolynomialGF2` für die Verarbeitung von Polynomen über GF(2) dem endlichen Körper mit zwei Elementen, in unserem Fall den `boolean`-Werten `false` und `true`.

Für eine Polynom $a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_2x^2 + a_1x + a_0$ werden die Koeffizienten $a_{n-1}, \ldots a_0 \in$ GF(2) in einem `boolean`-Feld der Länge n gespeichert. Die Referenz auf das Feld ist `private`. Ein Polynom kann, nach dem es erzeugt worden ist, nicht mehr verändert werden, deshalb sind Methoden zum Setzen der Koeffizienten nicht nötig. Zum

Lesen der Koeffizienten wird eine `public` Methode `toArray` exportiert, die eine Kopie des Koeffizienten-Feldes zurückliefert.

Sehen Sie `final` Klassenvariablen für das Null-Polynom `ZERO` (Koeffizienten-Feld ist `null`) und das Eins-Polynom `ONE` ($n = 1$ und $a_0 = 1$) vor.

Implementieren Sie einen default–Konstruktor, der das Eins-Polynom `ONE` zurückliefert.

Implementieren Sie eine Konstruktor, der ein Koeffizienten-Feld übergeben bekommt und ein dazu passendes Polynom erzeugt. Es wird sichergestellt, dass das Feld nicht größer als nötig ist, d.h. es gilt a_{n-1} `== true`. Implementieren Sie dazu eine Hilfsmethode `trim`, als `private` Klassenmethode, die ein `boolean`-Feld übergeben bekommt und ein, eventuell kleineres, `boolean`-Feld zurückliefert.

Implementieren Sie die Methoden `isZero` und `isOne`, zum Testen ob das `PolynomialGF2`-Objekt das Null- bzw. Eins-Polynom ist.

Überschreiben Sie die von der Superklasse `java.lang.Object` geerbten Methoden `clone`, `equals` und `toString` sinnvoll. Stellen Sie die in der API für `equals` geforderten Eigenschaften (*reflexive, symmetric, transitive, consistent*) sicher.

Überschreiben Sie die von der Superklasse `java.lang.Object` geerbte Methode `hashCode`, sodass die Koeffizienten $(a_{n-1}, \ldots, a_0)$ als Dual-Codierung des Hashcodes (h) interpretiert werden, d.h. $h = \sum_{i=0}^{n-1} a_i 2^i$.

Fügen Sie (Objekt-) Methoden für die Operationen Addition (`plus`), Multiplikation (`times`) und den Rest der Divison (`mod`) von Polynomen über GF(2) hinzu. Diese Methoden rechnen jeweils mit dem aktuellen und einem übergebenen `PolynomialGF2`-Objekt, diese werden nicht verändert. Als Ergebnis wird ein neues `PolynomialGF2`-Objekt zurückgeliefert.

Für die Methode `mod` sind folgende Hilfsmethoden sinnvoll. Die Methode `degree` liefert den Grad des Polynoms zurück. Die Methode `shift`, mit einer ganzen Zahl k als Parameter, realisiert die Multiplikation des Polynom $a_{n-1}x^{n-1}+...+a_0$ mit x^k, durch eine Verschiebung der Koeffizienten. D.h. für nicht negatives k wird als Ergebnis ein neues `PolynomialGF2`-Objekt für das Polynom $a_{n-1}x^{n-1+k} + \ldots + a_0x^k$ zurückgeliefert, für negatives k ist das Verhalten unbestimmt.

Schreiben Sie sowenig Quelltext wie möglich doppelt.

Schreiben Sie eine Testklasse, die die unten unter *Test 1* und *Test 2* stehenden Ausgaben reproduziert, dabei muss die Formatierung nicht exakt übereinstimmen.

Kommentieren Sie die Klassen ausführlich.

Test 1

```
i | hash | x^i
----------------------
0 |    1 | 1
1 |    2 | x
2 |    4 | x^2
3 |    3 | x + 1
4 |    6 | x^2 + x
5 |    7 | x^2 + x + 1
6 |    5 | x^2 + 1
```

Auflistung der Element von $\mathrm{GF}(2^3)\backslash\{0\}$ mit Generator x (2) und irreduziblem Polynom x^3+x+1. In Zeile i steht das Polynom ($x^i \mod x^3+x+1$) und der zugehörige Hashcode.

Test 2

```
  |  0   1   2   3   4   5   6   7   8   9   a   b   c   d   e   f
-------------------------------------------------------------------
0 | 01  03  05  0f  11  33  55  ff  1a  2e  72  96  a1  f8  13  35
1 | 5f  e1  38  48  d8  73  95  a4  f7  02  06  0a  1e  22  66  aa
2 | e5  34  5c  e4  37  59  eb  26  6a  be  d9  70  90  ab  e6  31
3 | 53  f5  04  0c  14  3c  44  cc  4f  d1  68  b8  d3  6e  b2  cd
4 | 4c  d4  67  a9  e0  3b  4d  d7  62  a6  f1  08  18  28  78  88
5 | 83  9e  b9  d0  6b  bd  dc  7f  81  98  b3  ce  49  db  76  9a
6 | b5  c4  57  f9  10  30  50  f0  0b  1d  27  69  bb  d6  61  a3
7 | fe  19  2b  7d  87  92  ad  ec  2f  71  93  ae  e9  20  60  a0
8 | fb  16  3a  4e  d2  6d  b7  c2  5d  e7  32  56  fa  15  3f  41
9 | c3  5e  e2  3d  47  c9  40  c0  5b  ed  2c  74  9c  bf  da  75
a | 9f  ba  d5  64  ac  ef  2a  7e  82  9d  bc  df  7a  8e  89  80
b | 9b  b6  c1  58  e8  23  65  af  ea  25  6f  b1  c8  43  c5  54
c | fc  1f  21  63  a5  f4  07  09  1b  2d  77  99  b0  cb  46  ca
d | 45  cf  4a  de  79  8b  86  91  a8  e3  3e  42  c6  51  f3  0e
e | 12  36  5a  ee  29  7b  8d  8c  8f  8a  85  94  a7  f2  0d  17
f | 39  4b  dd  7c  84  97  a2  fd  1c  24  6c  b4  c7  52  f6  01
```

Auflistung der Element von $\mathrm{GF}(2^8)\backslash\{0\}$ als Exponentialtabelle für den Generator $x+1$ (03) mit irreduziblem Polynom $x^8+x^4+x^3+x+1$.

Die Zeilen-/Spaltenbeschriftungen (0-f) der Exponentialtabelle sind die beiden Hexadezimalziffern des Exponenten. Die Beschriftungen der Zeile ist die höherwertige Ziffer, die der Spalte die niederwertige Ziffer. Ein Tabelleneintrag ist der Hashcodes des potenzierten Generators in Hexadezimalschreibweise.

Beispiel

```
  | ...   1   2
---------------
0 | ...  03 ...
...
3 | ... ...  04
```

Zeile `0`, Spalte `1` ergibt Exponent `01` $= (01)_{16} = (1)_{10}$, erzeugt Tabelleneintrag $(x+1)^1 \mod (x^8 + x^4 + x^3 + x + 1) = x + 1 = (00000011)_2 = (3)_{16} =$ `03`.

Zeile `3`, Spalte `2` ergibt Exponent `32` $= (32)_{16} = (50)_{10}$, erzeugt Tabelleneintrag $(x+1)^{50} \mod (x^8 + x^4 + x^3 + x + 1) = x^2 = (00000100)_2 = (4)_{16} =$ `04`.

Hintergrund

Algorithmen die Bytes verarbeiten, z.B. das symmetrisches Verschlüsselungsverfahren *Advanced Encryption Standard* (AES, Rijndael-Algorithmus), arbeiten oft mit Arithmetik auf dem endlichen Körper GF(2^8) mit 256 Elementen. Wobei jedes Element eine der 256 möglichen Bitfolgen repräsentiert, die mit einem Byte (8 Bit) kodiert werden können.

Insbesondere um schnelle Multiplikation zu realisieren arbeiten diese Algorithmen mit im Voraus berechneten Exponentialtabellen (und Logarithmustabellen) aus denen das Ergebnis abgelesen werden kann. Siehe z.B. `https://de.wikipedia.org/wiki/Rijndael_MixColumns`.