3. Praktikum: Große Zahlen

$2^{33222} <=> 13^{8978}$

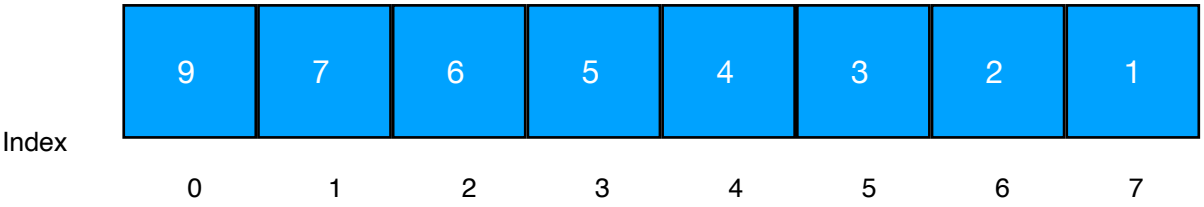
[$-8053873881207597^4 + 8043575814581751^5 + 12602123297335631^3$?]

Bearbeitungszeitraum:	28.11.22 - 12.12.22, 9.00 Uhr
Gesamtpunktzahl:	5 Abgabepunkte + 10 Leistungspunkte
Praktikumsziel:	Sie üben die Verwendung eindimensionaler Felder und von (Klassen-) Methoden.
Voraussetzungen:	Kenntnisse aus der Vorlesung Nutzung des Java-Compilers und der Java *virtual machine* Strukturierte Anweisungen, (statische) Methoden, Felder
Aufgabe:	Implementieren Sie ein Java-Programm, welches Operationen auf sehr großen natürlichen Zahlen bereitstellt und benutzt. Verwenden Sie zur Zahlenrepäsentation Felder von Ziffern (in Dezimal-schreibweise).
Abgabe:	Die vollständig implementierte Klasse **`Bigs.java`** senden Sie wie üblich bei **Moodle** als Lösung ein. *Bitte verwenden Sie genau diesen Dateinamen und achten Sie auf Groß- und Kleinschreibung.* ***ACHTUNG: Nur Gruppenabgaben möglich***

Implementieren Sie eine Klasse **`Bigs`**, die es ermöglicht, mit sehr großen natürlichen Zahlen zu rechnen. Dazu sollen die Zahlen in Feldern von Ziffern repräsentiert werden, wobei

1. an jeder Position nur Ziffern zwischen 0 und 9 erlaubt sind,
2. am Index 0 stets die Einerstelle der Zahl abzulegen ist (höherwertige Stellen folgen) und
3. stets nur so viele Feldpositionen wie nötig verwendet werden, so dass also nie führende Nullen (außer bei 0 selbst) erfasst werden.

Die Zahl 12345679 soll demzufolge die folgende Repräsentation haben:



Implementieren Sie dazu (mindestens) die im folgenden Programmfragment vorgegebenen Methoden:

```
class Bigs {
    // addiert die Ziffernfelder a und b
    public static int[ ] add (int[ ] a, int[ ] b)          { /* TODO */ }

    // gibt das Ziffernfeld n in lesbarer dezimaler Form aus
    // bei sehr langen Zahlen soll das Format verwendet werden, welches auch von
    // bc (s.u.) benutzt wird: Umbruch nach 68 Ziffern mit einem \ am Ende
    static void print (int[ ] n)                           { /* TODO */ }

    // konstruiert ein einstelliges Ziffernfeld aus der Ziffer d
    static int[ ] digit(int d)                             { /* TODO */ }

    // konstruiert das Ziffernfeld, welches den Wert Null repraesentiert
    static int[ ] Null()                                   { /* TODO */ }

    // konstruiert das Ziffernfeld, welches den Wert Eins repraesentiert
    static int[ ] One()                                    { /* TODO */ }
```

```
    // Rest des Ziffernfeldes n bei Division durch 10 (eine int-Zahl!)
    static int mod10(int[ ] n)                          { /* TODO */ }

    // ganzzahliger Quotient bei Division durch 10
    static int[ ] div10(int[ ] n)                       { /* TODO */ }

    // Umwandlung einer beliebigen int-Zahl in ein Ziffernfeld
    static int[ ] fromInt(int n)                        { /* TODO */ }

    // kopiert den Wert von n
    static int[ ] copy(int[ ] n)                        { /* TODO */ }

    // multipliziert das Ziffernfeld n mit einer (einstelligen!) int-Zahl
    static int[ ] times(int[ ] n, int d)                { /* TODO */ }

    // multipliziert das Ziffernfeld n mit 10
    static int[ ] times10(int[ ] n)                     { /* TODO */ }

    // multipliziert zwei Ziffernfeld
    static int[ ] times(int[ ] a, int[ ] b)             { /* TODO */ }

    // Quadratzahl eines Ziffernfeldes
    static int[ ] square(int[ ] a)                      { /* TODO */ }

    // Kubikzahl eines Ziffernfeldes
    static int[ ] cubic(int[ ] a)                       { /* TODO */ }

    // Test auf kleiner-Relation zweier Ziffernfelder: a < b ?
    static boolean less (int[ ] a, int[ ] b)            { /* TODO */ }

    // Test auf Gleichheit zweier Ziffernfelder
    static boolean equal (int[ ] a, int[ ] b)           { /* TODO */ }

    // Test auf Korrektheit eines Ziffernfeldes: Feld existiert und enthaelt
    // mindenstens eine Ziffer, alle Positionen liegen zwischen 0 und 9
    // keine fuehrenden Nullen (ausser bei Null selbst)
    static boolean ok (int[ ] n)                        { /* TODO */ }

    // gibt die (kleinste) Ziffer mit der groessten Haeufigkeit in n aus
    static void maxDigit(int[] n)                       { /* TODO */ }

    public static void main (String[] s) {
        int[ ] a = One();
        for (int i=0; i<33222; ++i) { a = times(a, 2); }
        System.out.println("2^33222 hat " + a.length + " Stellen");
        print(a);
        System.out.println();
        int[ ] b = fromInt(13);
        int[ ] c = copy(b);
        for (int i=1; i<8978; ++i) { c = times(c, b); }
        System.out.println("13^8978 hat " + c.length + " Stellen");
        print(c);
        System.out.println();
        System.out.println(less(a, c)); // beantwortet die Frage aus der
                                        // Aufgabenueberschrift
        maxDigit(a);
        maxDigit(c);
    }
}
```

Änderungen an den Signaturen der vorgegebenen Methoden sind **NICHT** erlaubt.

Die Verwendung von weiteren Hilfsklassen (z.B. **`java.math.BigInteger`**) aus dem Java-API ist ebenfalls **NICHT** zulässig!

Hinweise

Achten Sie bei allen Operationen darauf, dass beim ggf. notwendigen Erzeugen neuer Ziffernfelder immer nur so viele Elemente beschafft werden, wie nötig!

Auch wenn Java prinzipiell die Verwendung von Umlauten (und vielen anderen Sonderzeichen) erlaubt, sollten Sie im Praktikum generell darauf verzichten, um sich nicht vom sog. *encoding* abhängig zu machen. Das betrifft übrigens auch Texte in Kommentaren!

FAQ

1. **Q:** Was sind denn Abgabe- und Leistungspunkte?
   A: Wenn Sie die Praktikumsordnung gelesen haben, sollte klar sein, dass sie alle Aufgaben abgeben müssen. Die Abgabepunkte erhalten Sie dann, wenn Sie eine erkennbar ernst gemeinte Abgabe eingereicht haben. Die eigentlich in der jeweiligen Aufgabe geforderte Leistung wird dann in Abhängigkeit von der Erfüllung mit Leistungspunkten bewertet.

2. **Q:** Gibt es ein Zeitlimit für die Ausführung der **`main`**-Funktion ?
   A: Ja, das Ergebnis der nötigen Berechnungen und des Vergleichs soll (inklusive Ausgaben) in weniger als 5 Sekunden vorliegen.

3. **Q:** Wenn ich in *eclipse* eine neue Klasse anlege (siehe 1. Aufgabe) kann ich diese einem sog. *Package* zuordnen. Soll/darf ich davon Gebrauch machen?
   A: Nein! Benutzen Sie immer die Voreinstellung (*default*), weil sich ansonsten Ihr Programm nicht im aktuellen Verzeichnis des Quelltextes ausführen lässt! Da dies bei der Korrektur zu aufwändigen Anpassungen ihrer Lösung führt, wird es u.U. mit Punktabzug 'bestraft' - vermeiden Sie es daher.

4. **Q:** Wie kann ich so große Ergebnisse auf Korrektheit überprüfen - mit meinem Taschenrechner geht das nicht ?
   A: Die Unix-Utility **`bc`** kann spielend mit solchen Zahlen umgehen (lesen Sie ggf. unter **`man bc`** nach).

5. **Q:** Ist es nicht ein enorme Verschwendung von Speicherplatz, jede einzelne Ziffer in einem **`int`** zu erfassen?
   A: Durchaus! Das soll im Rahmen dieser Aufgabe aber keine Rolle spielen.

6. **Q:** Soll mein Programm tatsächlich nur diesen einen Test berechnen?
   A: In der abzugebenden Version schon. Allerdings ist es durchaus sinnvoll, dass Sie sich mit weiteren Tests davon überzeugen, dass Ihre Berechnungen die richtigen Ergebnisse unter Einhaltung der beschriebenen Randbedingungen liefern. Bei der Korrektur werden wir dies auch tun, indem wir in einer **`main`**-Methode einer (anderen) Testklasse Ihre Methoden aufrufen und deren Ergebnisse überprüfen.

7. **Q:** Wie soll die Ausgabe in **`maxDigit`** formatiert sein?
   A: Eine Zeile bestehend aus der (kleinsten) Ziffer mit größter Häufigkeit, gefolgt von Doppelpunkt und Leerzeichen, gefolgt von der absoluten Häufigkeit dieser Ziffer in der als Parameter übergebenen Zahl.
   `maxDigit(fromInt(11222333));` soll also ausgeben: **`2: 3`**

8. **Q:** Was hat es mit den Kubikzahlen in der Überschrift dieser Aufgabe auf sich ?
   A: Lesen Sie etwas über dieses höchst aktuelle Ergebnis unter https://www.sciencealert.com/the-sum-of-three-cubes-problem-has-been-solved-for-42 ? Sie können es nun in Ihrem Java-Programm zumindest überprüfen! Für die Entdeckung waren allerdings über eine Million Stunden CPU-Zeit auf einem weltweit verteilten System von über 500.000 PCs (der sog. Charity Engine) erforderlich!

9. **Q:** Das vorgegebene Quelltextfragment ist ja gar nicht fehlerfrei übersetzbar !?
   A: Leere Funktionskörper sind nun mal nur für **`void`**-Funktionen erlaubt. Sie könnten die (noch) nicht implementierten Funktionen zunächst auskommentieren ODER mit einer dummy-Implementation

versehen: alle **`boolean`**-Funktionen: **`{return true;}`** , alle **`int[]`**-Funktionen: **`{return Null(); }`** - dazu sollte die einfachste Funktion **`Null`** natürlich schon mal (idealerweise korrekt) implementiert werden.

ACHTUNG

Um die Korrektur der Aufgaben durch die Tutoren zu erleichtern, gelten wie immer (d.h. auch bei allen weiteren Aufgaben) die folgenden Regeln:

- Eingesandte Java-Programme, die nicht vom Java-(Referenz-)Compiler akzeptiert werden (d.h. Fehler bei der Übersetzung liefern) werden ohne weitere Inspektion mit 0 (Leistungs-) Punkten bewertet! Ob Sie die 5 Abgabepunkte erhalten, entscheiden wir in Abhängigkeit davon, ob Ihre Einsendung genügend mit der Aufgabe zu tun hat ;-)
- Halten Sie sich bei der Benennung von Klassen, Funktionen und Dateien immer an die Vorgaben und achten Sie darauf, dass Sie keine Packages verwenden (s.o.). Alle Abweichungen von diesen Regeln führen zu zusätzlichem Aufwand bei der Bewertung und daher auch zu Punktabzug.
- Die vom implementierten Programm erzeugten Ausgaben müssen das geforderte Format exakt einhalten (incl. Zeilenstruktur und Leerzeichen). Soll und Ist werden automatisch verglichen. Sofern Abweichungen festgestellt werden, gibt es Punktabzug, über dessen Höhe dann die Tutoren entscheiden! Die einzig zulässige Abweichung (eine die man bei der Textansicht des Programms und seiner Ausgaben u.U. gar nicht feststellen kann) besteht in der Codierung der Zeilenenden, die unter Windows mit zwei Zeichen (carriage return [/x0d] + line feed [/x0a]) unter Unix dagegen nur mit einem Zeichen (newline [/x0a]) dargestellt werden.