Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Institute of Information Security and Dependability (KASTEL)
https://sdq.kastel.kit.edu/wiki/Programmieren
programmieren-vorlesung@cs.kit.edu

Programmieren – Wintersemester 2025/26

Übungsblatt 2
Version 1.0

20 Punkte

Ausgabe:	19.11.2025, ca. 12:00 Uhr
Abgabestart:	26.11.2025, 12:00 Uhr
Abgabefrist:	04.12.2025, 06:00 Uhr

Plagiarismus

Es werden nur selbstständig angefertigte Lösungen akzeptiert. Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen, ist ein Täuschungsversuch und führt jederzeit (auch nachträglich) zur Bewertung „nicht bestanden“. Ausdrücklich ausgenommen hiervon sind Quelltextschnipsel von den Vorlesungsfolien und aus den Lösungsvorschlägen des Übungsbetriebes in diesem Semester. Alle benutzten Hilfsmittel müssen vollständig und genau angegeben werden. Alles, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, muss deutlich kenntlich gemacht werden. Beachten Sie darüber hinaus die Richtlinien der Fakultät zum Verwenden von Generativer KI [1].

Studierende, die den ordnungsgemäßen Ablauf einer Erfolgskontrolle stören, können von der Erbringung der Erfolgskontrolle ausgeschlossen werden. Ebenso stellt unter anderem die Weitergabe von Teilen von Testfällen oder Lösungen bereits eine Störung des ordnungsgemäßen Ablaufs dar. Auch diese Art von Störungen können ausdrücklich jederzeit zum Ausschluss der Erfolgskontrolle führen. Dies bedeutet ausdrücklich, dass auch nachträglich die Punktzahl reduziert werden kann.

Kommunikation und aktuelle Informationen

In unseren *FAQs*[2] finden Sie einen Überblick über häufig gestellte Fragen und die entsprechenden Antworten zum Modul „Programmieren“. Bitte lesen Sie diese sorgfältig durch, noch bevor Sie Fragen stellen, und überprüfen Sie diese regelmäßig und eigenverantwortlich auf Änderungen. Beachten Sie zudem die Hinweise im Wiki[3].

In den *ILIAS-Foren* oder auf *Artemis* veröffentlichen wir gelegentlich wichtige Neuigkeiten. Eventuelle Korrekturen von Aufgabenstellungen werden ebenso auf diesem Weg bekannt gemacht. Das aktive Beobachten der Foren wird daher vorausgesetzt.

[1] https://www.informatik.kit.edu/faq-wiki/doku.php?id=generative_ki
[2] https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ
[3] https://sdq.kastel.kit.edu/programmieren/

Überprüfen Sie das Postfach Ihrer *KIT-Mailadresse* regelmäßig auf neue E-Mails. Sie erhalten unter anderem eine Zusammenfassung der Korrektur per E-Mail an diese Adresse. Alle Anmerkungen können Sie anschließend im Online-Einreichungssystem[4] einsehen.

Bearbeitungshinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der verpflichtenden Tests für eine erfolgreiche Abgabe von Übungsblatt 2 notwendig ist. Ihre Abgabe wird automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Sie müssen zuerst die verpflichtenden Tests bestehen, bevor die anderen Tests ausgewertet werden können. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie ausschließlich *Java SE 21*.
- Sofern in einer Aufgabe nicht ausdrücklich anders angegeben, verwenden Sie keine Elemente der Java-Bibliotheken. Ausgenommen ist die Klasse `java.util.Scanner` und alle Elemente aus den folgenden Paketen: `java.lang`, `java.io` und `java.math.BigInteger`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 140 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.
- `System.exit()`, `Runtime.exit()` oder ähnliches dürfen nicht verwendet werden.

Diese folgenden Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe. Dennoch wird Ihre Abgabe durch das Abgabesystem *nicht* automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Orientieren Sie sich zudem an den Bewertungskriterien im Wiki.

- Fügen Sie außer Ihrem u-Kürzel keine weiteren persönlichen Daten zu Ihren Abgaben hinzu.
- Beachten Sie, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im Wiki ein.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.
- Die Kommentare sollen einheitlich in englischer oder deutscher Sprache verfasst werden.
- Geben Sie im Javadoc-Autoren-Tag nur Ihr u-Kürzel an.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

[4] `https://artemis.cs.kit.edu/`

Checkstyle

Das Online-Einreichungssystem überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen das Online-Einreichungssystem die Abgabe mit null Punkten bewertet, da diese Regeln verpflichtend einzuhalten sind. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki beschreibt, wie Checkstyle verwendet werden kann.

Abgabehinweise

Die Abgabe im Online-Einreichungssystem wird am 26.11.2025, 12:00 Uhr, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Einreichungssystem bei der richtigen Aufgabe vor Ablauf der Abgabefrist am 04.12.2025, 06:00 Uhr, hochzuladen. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären. Falls Sie mit Git abgeben, *muss immer* auf den `main`-Branch gepusht werden.

- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe A in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.
- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe B in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.
- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe C in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.

Wiederverwendung von Lösungen

Falls Sie für die Bearbeitung der Abschlussaufgaben oder Übungsblätter Beispiellösungen aus diesem Semester wiederverwenden, *müssen* Sie in die entsprechenden Klassen "Programmieren-Team" ins Autor-Tag eintragen. Dies ist nötig, um die Checkstyle-Kriterien zu erfüllen.

Aufgabe A: String-Utilities (6 Punkte)

Implementieren Sie die folgenden sechs öffentlichen Methoden in einer Klasse `StringUtilities`. Diese Klasse darf keine `main`-Methode enthalten. Stellen Sie zudem sicher, dass alle Methoden statisch sind und die Klasse keine Instanzattribute beinhaltet.

Sie können davon ausgehen, dass die als Methodenparameter übergebenen Zeichenketten immer mindestens ein gültiges Zeichen enthalten. Beachten Sie bei diesen Zeichenketten sowohl die Groß- und Kleinschreibung als auch Sonder- und Zahlzeichen. Verwenden Sie selber für diese Aufgabe *keine* weiteren Klassen direkt aus der Java-API, mit Ausnahme von `String` und `Character`.

1. Schreiben Sie eine Methode `String caesarEncryption(String input, int key)`, die eine Zeichenkette mittels der Caesar-Verschlüsselung verschlüsselt. Die Verschlüsselung verschiebt jedes Zeichen um eine bestimmte Anzahl von Positionen im Alphabet nach rechts, basierend auf dem gegebenen Schlüssel.Die Methode gibt die verschlüsselte Zeichenkette zurück. Sie dürfen davon ausgehen, dass die Eingabe nur Buchstaben enthält.

   Beispiel: `caesarEncryption("Prog",5)` gibt `"Uwtl"` zuürck.

2. Schreiben Sie eine Methode `String caesarDecryption(String input, int key)`, die eine mit der Caesar-Verschlüsselung verschlüsselte Zeichenkette entschlüsselt. Die Methode verwendet den gleichen Schlüssel wie die Verschlüsselung und gibt die ursprüngliche Zeichenkette zurück. Sie dürfen davon ausgehen, dass die Eingabe nur Buchstaben enthält.

   Beispiel: `caesarDecryption("Uwtl",5)` gibt `"Prog"` zuürck.

3. Schreiben Sie eine Methode `int countCharacter(String word, char character)`, welche für ihr gegebenen Zeichenkette die Anzahl des Auftretens des ihr gegeben einzelnen Zeichens bestimmt. Diese Methode gibt die Anzahl des Zeichens in der Zeichenkette zurück. Groß- und Kleinschreibung sollen ignoriert werden.

   Beispiel: `countCharacter("Aa abbc","a")` gibt `3` zurück.

4. Schreiben Sie eine Methode `String alternateCase(String input)`, die jeden Buchstaben der Zeichenkette abwechselnd in Groß- und Kleinschreibung umwandelt, beginnend mit Großbuchstaben. Nicht-buchstabige Zeichen (z. B. Leerzeichen, Ziffern, Satzzeichen) bleiben unverändert und beeinflussen nicht den Wechsel der Groß-/Kleinschreibung.

   Beispiel: `alternateCase("hello world!")` gibt `"HeLlO wOrLd!"` zurück.

5. Schreiben Sie eine Methode `String reverse(String word)`, welche die ihr gegebenen Zeichenkette umkehrt. Diese Methode gibt die Zeichen in umgekehrter Reihenfolge wieder als eine neue Zeichenkette zurück.

   Beispiel: `reverse("EineHordeBedroheNie")` gibt `"eiNehordeBedroHeniE"` zurück.

6. Schreiben Sie eine Methode `boolean isRotation(String word, String rotation)`, die überprüft, ob die Zeichenkette *rotation* eine Drehung der Zeichenkette *word* ist. Die Methode gibt *true* zurück, wenn dies der Fall ist, andernfalls *false*.

   Beispiel: `isRotation("waterbottle", "erbottlewat")` gibt `true` zurück.

Aufgabe B: IBAN (6 Punkte)

In dieser Aufgabe soll ein Programm geschrieben werden, das vereinfacht für eine gegebene Länderkennung, Bankleitzahl und Kontonummer die zugehörige IBAN berechnet. IBAN steht für *International Bank Account Number* und ist eine internationale, standardisierte Notation für Kontonummern. *Gehen Sie in dieser Aufgabe davon aus, dass alle IBANs unabhängig von ihrer Länderkennung immer gleich aufgebaut sind.* Das heißt eine IBAN setzt sich immer aus exakt 22 Stellen zusammen:

1 2	3 4	5 6 7 8 9 10 11 12	13 14 15 16 17 18 19 20 21 22
Länderkennung	Prüfsumme	Bankleitzahl	Kontonummer

Die Länge des primitiven Datentyps für ganzzahlige Werte `long` reicht für die Berechnungen der Aufgabe nicht aus. Für diese Aufgabe muss daher mit der `BigInteger`-Klasse[5] des *java.math*-Pakets gearbeitet werden. Mithilfe der Klasse `BigInteger` ist es möglich, beliebig genaue ganzzahlige Zahlen anzulegen, zu verwalten und Berechnungen durchzuführen.

Die 2-stellige Länderkennung besteht immer aus Großbuchstaben und wird als erster Kommandozeilenparameter Ihrem Programm übergeben. Die Bankleitzahl besteht immer aus positiven ganzzahligen Ziffernwerten mit genau acht Stellen und wird als zweiter Kommandozeilenparameter übergeben. Hierbei liegt die Bankleitzahl im abgeschlossenen Intervall von 10000000 bis 99999999. Die Kontonummer besteht immer aus positiven ganzzahligen Ziffernwerten und wird als dritter und letzter Kommandozeilenparameter übergeben. Hierbei liegt die Kontonummer im abgeschlossenen Intervall von 1 bis 9999999999. Gehen Sie davon aus, dass alle übergebenen Parameter den beschriebenen Vorgaben entsprechen.

Vor der Erstellung der IBAN muss die Prüfsumme berechnet werden. Die stets zweistellige Prüfsumme befindet sich bei jeder IBAN an der dritten und vierten Stelle und wird aus genau zwei positiven ganzzahligen Ziffernwerten gebildet. Hierzu wird die zweistellige Länderkennung, die achtstellige Bankleitzahl und die (zehnstellige) Kontonummer benötigt.

Die Berechnung der Prüfsumme erfolgt in mehreren Schritten. Zuerst wird aus der Kontonummer und der Bankleitzahl eine neue Zahl generiert. Sofern die Kontonummer aus weniger als 10 Ziffern besteht, wird diese zunächst auf zehn Stellen mit führenden Nullen aufgefüllt. Beispiel: Wenn als Kommandozeilenparameter für die Kontonummer die Zahl 99 übergeben wird, wird die Kontonummer entsprechend um 8 führende Nullen zu 0000000099 ergänzt. Anschließend wird die zehnstellige Kontonummer mit der Bankleitzahl verknüpft, indem die Kontonummer an die Bankleitzahl angehängt wird. Zum Beispiel, die Bankleitzahl 12345678 und Kontonummer 1234567890 ergeben 123456781234567890, die Bankleitzahl 12345678 und Kontonummer 12345 ergeben 123456780000012345.

Im nächsten Schritt werden die beiden Großbuchstaben der Länderkennung in eine positive Ganzzahl umgewandelt. Die Grundlage für die Zahlen, die aus den Buchstaben gebildet werden, bildet die Position des jeweiligen Zeichens im lateinischen Alphabet. Zu diesem Zahlenwert wird 9 addiert. Die Summe ergibt die Zahl, die den jeweiligen Buchstaben ersetzen soll. Dementsprechend steht für A (Position 1+9) die Zahl 10, für D (Position 4+9) die 13, für E (Position 5+9) die 14 und für

[5] `https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/math/BigInteger.html`

Z (Position 26+9) die 35. Anschließend wird die Zahl für die Länderkennung am Ende um zwei Nullen ergänzt. Zum Beispiel, die Länderkennung DE für Deutschland entspricht der Zahl 131400 und die Länderkennung AT für Österreich entspricht der Zahl 102900.

Im nächsten Schritt wird diese erweiterte Zahl für die Länderkennung, an die zusammengefügte Kontonummer und Bankleitzahl, angehängt. Zum Beispiel aus der Eingabe von DE 12345678 12345 ergibt sich 123456780000012345131400. Diese Zahl wird Modulo 97 genommen, das heißt, es wird der Rest berechnet, der sich bei der Teilung der 24-stelligen Zahl durch 97 ergibt. Das Resultat wird von der Zahl 98 subtrahiert. Ist die Differenz kleiner Zehn, so wird ihr eine Null vorangestellt, sodass sich wieder ein zweistelliger Wert ergibt. Die auf diese Weise errechnete zweistellige Zahl wird als Prüfnummer verwendet. Aus der zweistelligen Länderkennung, der zweistelligen Prüfsumme, der achtstelligen Bankleitzahl und der zehnstelligen Kontonummer wird die IBAN generiert. Zum Beispiel aus der Eingabe von DE 12345678 12345 ergibt sich DE21123456800000012345.

Geben Sie am Ende Ihrer Berechnung die IBAN in einem für Menschen gut lesbaren Format aus. Hierzu soll die IBAN fünfmal in eine Vierergruppe und einmal in eine Zweiergruppe geteilt werden. Hierbei wird die IBAN von links beginnend in Vierergruppen durch ein Leerzeichen gruppiert. Zum Beispiel ergibt sich aus der IBAN DE21123456780000012345 die Ausgabe DE21 1234 5678 0000 0123 45. Die Ausgabe auf die Kommandozeile soll mit der Methode `System.out.println()` erfolgen.

Zur besseren Veranschaulichung das Ganze noch einmal anhand zweier Beispiele zusammengefasst:

Länderkennung	CH
Bankleitzahl	12341234
Kontonummer	1234123412
Bankleitzahl verbunden mit Kontonummer	123412341234123412
numerische Länderkennung ergänzt um 00	121700 (C = 12, H = 17)
Bankleitzahl, Kontonummer und Länderkennung	123412341234123412121700
Modulo 97	91
Subtrahieren von 98	7 (98 - 91)
Prüfsumme	07
Länderkennung, Prüfsumme, Bankleitzahl, Kontonummer = IBAN	CH0712341234123412341 2
Ausgabe	CH07 1234 1234 1234 1234 12

Länderkennung	FR
Bankleitzahl	76543210
Kontonummer	4321
Bankleitzahl verbunden mit Kontonummer	765432100000004321
numerische Länderkennung ergänzt um 00	152700 (F = 15, R = 27)
Bankleitzahl, Kontonummer und Länderkennung	765432100000004321152700
Modulo 97	93
Subtrahieren von 98	5 (98 - 93)
Prüfsumme	05
Länderkennung, Prüfsumme, Bankleitzahl, Kontonummer = IBAN	FR0576543210000000 4321
Ausgabe	FR05 7654 3210 0000 0043 21

Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung für die gegebene Beispielinteraktion. In Beispielinteraktionen stellt das Symbol `%>` (Prozent-Zeichen und Größer-Zeichen gefolgt von einem Leerzeichen) die Kommandozeile dar.

Beispielinteraktion

```
1 %> java Main DE 12345678 12345
2 DE21 1234 5678 0000 0123 45
3 %> java Main CH 12341234 1234123412
4 CH07 1234 1234 1234 1234 12
5 %> java Main FR 76543210 4321
6 FR05 7654 3210 0000 0043 21
```

Aufgabe C: Magische Quadrate (8 Punkte)

Ein *magisches Quadrat* der Ordnung n ist eine quadratische Anordnung der Zahlen 1 bis n^2, sodass alle Zeilen- und Spaltensummen sowie die Summe der Zahlen auf den beiden Diagonalen gleich sind. Der Wert dieser Summen wird auch als *magische Zahl* S_n bezeichnet [6]. Tabelle C.1 zeigt ein solches magisches Quadrat der Ordnung 3. Es gilt folgendes:

- Magische Zahl $S_n = \frac{1}{n} \cdot \sum_{x=1}^{n^2} x = \frac{n^3+n}{2}$.
- Ein *semi-magisches Quadrat* hat dieselben Eigenschaften wie ein magisches Quadrat, mit dem Unterschied, dass mindestens eine der Summen über die Diagonalen ungleich der magischen Zahl ist.
- Um das *Komplement* eines magischen Quadrates zu erhalten, wird jeder Eintrag zunächst mit -1 multipliziert, danach wird auf jeden Eintrag $n^2 + 1$ addiert.

4	**9**	**2**	→ 15
3	**5**	**7**	→ 15
8	**1**	**6**	→ 15
15 ↙	↓ 15	↓ 15	↓ 15 ↘ 15

Tabelle C.1: Ein magisches Quadrat der Ordnung 3 bei dem alle Spalten-, Zeilen- und Diagonalsummen die magische Zahl 15 ergeben.

In dieser Aufgabe soll ein Programm geschrieben werden, das einige Operationen zu magischen Quadraten unterstützt. Legen Sie hierfür eine Klasse `MagicSquare` mit einem öffentlichen Konstruktor `MagicSquare(int[][] square)` an. Folgende Methoden sollen in der Klasse `MagicSquare` implementiert werden:

- `showMagicNumbers(int k)` - Diese Methode gibt eine Zeichenkette zurück, die alle magischen Zahlen zu $n = 1$ bis $n = k$ hintereinander, jeweils durch ein Komma getrennt, enthält. Diese Methode ist eine Klassenmethode.
- `isMagicSquare()` - Diese Methode gibt einen booleschen Wert zurück. Dieser ist `true`, wenn das Quadrat ein magisches Quadrat ist, und `false`, wenn nicht.
- `isSemimagicSquare()` - Diese Methode gibt einen booleschen Wert zurück. Dieser ist `true`, wenn das Quadrat ein semi-magisches Quadrat ist, und `false`, wenn nicht.
- `complement()` - Diese Methode wandelt das Quadrat in sein komplementäres magisches Quadrat um.
- `toString()` - Diese Methode gibt eine textuelle Repräsentation des Zahlenquadrates zurück. Das Zahlenquadrat wird als durch Zeilenumbrüche separierte Zeilen dargestellt. Jede Zeile besteht aus Ganzzahlen, die durch jeweils ein Leerzeichen separiert werden.

[6] `https://de.wikipedia.org/wiki/Magisches_Quadrat`

Vermeiden Sie Code-Kopien und verwenden Sie stattdessen geeignete Hilfsmethoden.

Zusätzlich erstellen Sie eine Klasse `Main`, die eine `main()`-Methode enthält, in der die Ein- und Ausgabe ihres Programms stattfindet. Folgende Operationen soll ihr Programm unterstützen:

- `showMagicNumbers` - Gibt alle magischen Zahlen zu $n = 1$ bis $n = k$ (wobei k als Parameter übergeben wird) hintereinander, jeweils durch ein Komma getrennt, auf der Konsole aus.
- `isMagicSquare?` - Gibt `magic square` auf der Konsole aus, wenn das Quadrat magisch ist, `semimagic square`, wenn das Quadrat ein semi-magisches Quadrat ist und `not magical`, wenn das Quadrat weder ein magisches noch ein semi-magisches Quadrat ist.
- `complement` - Gibt das Komplement des magischen Quadrates auf der Konsole aus. Das Komplement ist dabei wie in der `toString()`-Methode beschrieben formatiert.

Die Eingabe findet via Kommandozeilenargumente (Command Line Arguments) statt. Die Kommandozeilenargumente können Sie aus dem *args*-Parameter der `main()`-Methode auslesen. Als erstes Argument wird die Operation angegeben, danach folgt der Parameter. Bei `showMagicNumbers` ist dieser eine 32-Bit-Ganzzahl. Bei allen anderen Operationen ist der Parameter ein Zahlenquadrat repräsentiert durch eine Zeichenkette. Ein Zahlenquadrat wird dabei als Liste von Zeilen dargestellt. Jede Zeile beginnt mit einem Kleiner-als-Zeichen `<` und endet mit einem Größer-als-Zeichen `>`. Eine Zeile besteht aus 32-Bit-Ganzzahlen, die jeweils durch einen kurzen Bindestrich `-` (wie Sie ihn auf Ihrer Tastatur finden) separiert werden. Sie können davon ausgehen, dass alle Eingaben syntaktisch und semantisch korrekt sind. Das Programm soll daraufhin das Ergebnis der Operation ausgeben.

Zur besseren Veranschaulichung hier eine Beispielinteraktion: Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung für die gegebene Beispielinteraktion. In Beispielinteraktionen stellt das Symbol `%>` (Prozent-Zeichen und Größer-Zeichen gefolgt von einem Leerzeichen) die Kommandozeile dar.

Beispielinteraktion

```
1  %> java Main showMagicNumbers 6
2  1,5,15,34,65,111
3  %> java Main isMagicSquare? <4-9-2><3-5-7><8-1-6>
4  magic square
5  %> java Main isMagicSquare? <4-4-2><3-5-7><8-1-6>
6  not magical
7  %> java Main isMagicSquare? <9-2-4><5-7-3><1-6-8>
8  semimagic square
9  %> java Main isMagicSquare? <1-2-3><4-5-6><7-8-9>
10 not magical
11 %> java Main complement <4-9-2><3-5-7><8-1-6>
12 6 1 8
13 7 5 3
14 2 9 4
```