Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **nicht** die Klassen `StringBuffer` bzw. `StringBuilder` verwenden.
- Sie dürfen **keine** Lambdas, Streams oder Methodenreferenzen verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `length`, `substring`.
- Aus der Klasse `Math` dürfen alle Methoden verwendet werden. Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Implementieren Sie nur folgende statische Methoden:

- **`int countDivisors(int x, int y)`** liefert die Anzahl der Zahlen im Intervall [`x`,`y`] zurück, die durch 4 teilbar sind, aber *nicht* durch 6.
  Annahme(n): `x > 0`, `y > 0`, `x <= y`.
- **`int findDoubles(String text)`** zählt, wie oft der Buchstabe 'a' innerhalb des Strings `text` zweimal direkt hintereinander vorkommt, und liefert diesen Wert zurück.
  Annahme(n): `text.length() > 0`.
- **`String reverseInsert(String text, char character)`** fügt zwischen je zwei Zeichen von `text` den Buchstaben `character` ein und gibt diesen String in umgekehrter Reihenfolge zurück.
  Annahme(n): `text.length() > 1`.
- **`void printPattern(int n, char character)`** gibt auf der Konsole `n` Zeilen aus. Ist die aktuelle Zeilennummer gerade, wird 2·`n`-mal das Zeichen `character` ausgegeben. Ist die aktuelle Zeilennummer ungerade, wird `n`-mal zuerst das Zeichen `character` gefolgt von einem Punkt '.' ausgegeben. Am Ende jeder Zeile wird die Zeilennummer ausgegeben. Die Zählung der Zeilennummern beginnt mit 1.
  Annahme(n): `n > 0`.

Implementieren Sie folgende Punkte in main:

- Deklarieren Sie eine short-Variable `result` und eine String-Variable `test` mit dem Inhalt `"Teststring_Einstufungstest"`.
- `result` soll das Ergebnis des Aufrufs `countDivisors(299, 305)` ohne Compiler-Fehler zugewiesen werden.

Testen Sie alle Methoden in `main` mit zumindest folgenden Aufrufen, und erzeugen Sie mit deren Hilfe die gezeigten Ausgaben. Außer `printPattern` darf keine der implementierten Methoden eine Ausgabe erzeugen. Alle anderen Ausgaben müssen in `main` erfolgen.

Aufruf	**Ausgabe in main auf der Konsole**
`countDivisors(1, 28)`	`5`
`countDivisors(101, 2001)`	`317`
`countDivisors(8, 8)`	`1`
`findDoubles(test)`	`0`
`findDoubles("Haarspangenaal")`	`2`
`findDoubles("The Black Beast of Aaaaargh!")`	`3`
`findDoubles("Schokoladenkuchen")`	`0`
`reverseInsert(test, '.')`	`t.s.e.t.s.g.n.u.f.u.t.s.n.i.E._.g.n.i.r.t.s.t.s.e.T`
`reverseInsert("qwerty", '-')`	`y-t-r-e-w-q`
`reverseInsert("Pinkie Pie", '!')`	`e!i!P! !e!i!k!n!i!P`

`printPattern(4, '!')`	`!.!.!.!.1`
	`!!!!!!!!2`
	`!.!.!.!.3`
	`!!!!!!!!4`

`printPattern(5, '*')`	`*.*.*.*.*.1`
	`**********2`
	`*.*.*.*.*.3`
	`**********4`
	`*.*.*.*.*.5`

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`countDivisors` - 5 Punkte	Korrekte Schleife	2
	Korrekte Berechnung	2
	Korrekte Rückgabe	1
`findDoubles` - 4 Punkte	Korrekte Schleife	2
	Korrekte Verzweigung	1
	Korrekte Rückgabe	1
`reverseInsert` - 6 Punkte	Korrekte Schleife(n)	2
	Korrekte Verzweigung	2
	Randfälle (Position text.length()) korrekt behandelt	1
	Korrekte Rückgabe	1
`printPattern` - 6 Punkte	Richtige Anzahl an Zeilen	2
	Richtige Anzahl an Zeichen pro Zeile	2
	Korrekte Zeilennummern	2
`main` - 4 Punkte	Deklarationen	1
	Korrekte Zuweisung (result)	1
	Korrekte Aufrufe	1
	Ausgaben in der Konsole	1
Gesamt		25

Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **nicht** die Klassen `StringBuffer` bzw. `StringBuilder` verwenden.
- Sie dürfen **keine** Lambdas, Streams oder Methodenreferenzen verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `length`, `substring`.
- Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Implementieren Sie nur folgende **statische** Methoden:

- **`int getIntegerRoot(int k)`** bestimmt, ob `k` das Quadrat einer natürlichen Zahl ist - d.h. ob eine natürliche Zahl `i` existiert, sodass $i \cdot i = k$ ist. Ist dies der Fall, wird `i` zurückgegeben, ansonsten wird `-1` zurückgegeben.
  Annahme(n): `k > 0`.
  Hinweis: Die Berechnung muss nicht effizient implementiert werden. Bedenken Sie, dass für `i` nur Zahlen im Intervall [1,`k`] in Frage kommen. Beachten Sie auch, dass Sie gemäß den Einschränkungen keine Methoden der Klasse `Math` verwenden dürfen.
- **`String getThird(String a, String b, String c)`** vergleicht den Inhalt der Parameter `a`, `b` und `c` und gibt davon abhängig einen String zurück. Sind alle drei Parameter gleich, ist der Rückgabewert `"alle gleich"`. Sind alle drei Parameter unterschiedlich, ist der Rückgabewert `"alle unterschiedlich"`. Sind genau zwei Parameter gleich, wird der dritte (zu den anderen unterschiedliche) zurückgegeben.
  Annahme(n): `a != null`, `b != null`, `c != null`
- **`String replaceA(String s)`** gibt eine Kopie von `s` zurück, bei der das erste Vorkommen des Zeichens 'a' durch die Zahl `1`, das zweite Vorkommen von 'a' durch die Zahl `2` usw. ersetzt wird.
  Annahme(n): `s != null`
- **`void printBars(int i)`** gibt die Werte im Intervall [1,`i`] aufsteigend in zwei Zeilen aus. Die erste Zeile enthält alle Werte, die nicht durch 3 teilbar sind. Die zweite Zeile enthält alle Werte, die durch 3 teilbar sind. Nach jedem Wert wird abwechselnd das Zeichen '-' bzw. '+' ausgegeben. Die Trennzeichen der ersten Zeile beginnen mit '-', die der zweiten Zeile mit '+'.

Implementieren Sie folgende Punkte in main:

- Deklarieren Sie eine short-Variable `result` und eine String-Variable `test` mit dem Inhalt `"Blaukraut"`.
- `result` soll das Ergebnis des Aufrufs `getIntegerRoot(25)` ohne Compiler-Fehler zugewiesen werden.

Testen Sie alle Methoden in `main` mit zumindest folgenden Aufrufen, und erzeugen Sie mit deren Hilfe die gezeigten Ausgaben. Außer `printBars` darf keine der implementierten Methoden eine Ausgabe erzeugen. Alle anderen Ausgaben müssen in `main` erfolgen.

Aufruf	**Ausgabe in main auf der Konsole**
`getIntegerRoot(144)`	`12`
`getIntegerRoot(13)`	`-1`
`getIntegerRoot(1)`	`1`
`getThird("toss", "a", "coin")`	`alle unterschiedlich`
`getThird("Blaukraut", "bleibt", test)`	`bleibt`
`getThird("badger", "badger", "badger")`	`alle gleich`
`replaceA("TU Wien")`	`TU Wien`
`replaceA("Hubba bubba!")`	`Hubb1 bubb2!`
`replaceA("aaaa")`	`1234`
`printBars(2)`	`1-2+`
`printBars(19)`	`1-2+4-5+7-8+10-11+13-14+16-17+19-` `3+6-9+12-15+18-`
`printBars(20)`	`1-2+4-5+7-8+10-11+13-14+16-17+19-20+` `3+6-9+12-15+18-`
`printBars(21)`	`1-2+4-5+7-8+10-11+13-14+16-17+19-20+` `3+6-9+12-15+18-21+`

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`getIntegerRoot` - 5 Punkte	Korrekte Schleife	2
	Korrekte Berechnung	2
	Korrekte Rückgabe	1
`getThird` - 6 Punkte	Korrekter Vergleich	2
	Korrekte Verzweigungen	2
	Korrekte Rückgabe	2
`replaceA` - 5 Punkte	Korrekte Schleife	2
	Korrekte Verzweigung	2
	Korrekte Rückgabe	1
`printBars` - 5 Punkte	Richtige Anzahl an Zeilen	1
	Richtige Anzahl an Zeichen pro Zeile	2
	Korrekter Zeileninhalt	2
`main` - 4 Punkte	Deklarationen	1
	Korrekte Zuweisung (result)	1
	Korrekte Aufrufe	1
	Ausgaben in der Konsole	1
Gesamt		25

Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **nicht** die Klassen `StringBuffer` bzw. `StringBuilder` verwenden.
- Sie dürfen **keine** Lambdas, Streams oder Methodenreferenzen verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `length`, `substring`.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Character` verwenden: `isDigit`.
- Aus der Klasse `Math` dürfen alle Methoden verwendet werden. Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Implementieren Sie nur folgende statische Methoden:

- **`int sumUp(int d, int s, int t)`** erhöht `s`, bis `s` nicht mehr kleiner als `t` ist. Dazu wird wiederholt der Rest der Division `s/d` zu `s` addiert. Ist der Rest 0, dann wird statt dessen 1 addiert. Gibt zurück, wie oft der Rest der Division 0 war.
  Annahme(n): $0 < d, 0 < s < t$
- **`String addMark(String a, String pattern, int pos)`** überprüft, ob die Zeichenkette in `pattern` an bestimmten Stellen in `a` auftritt, und gibt davon abhängig einen neuen String zurück:
  - Ist `pattern` ein Teilstring von `a` an Index `pos`, wird der Inhalt von `a` ab inklusive Index `pos` zurückgegeben.
  - Ist dies nicht der Fall, aber beginnt `a` mit `pattern`, so enthält der Rückgabe-String `"--"` gefolgt von `a`.
  - Ist auch dies nicht der Fall, aber endet `a` mit `pattern`, so enthält der Rückgabe-String `a` gefolgt von `"--"`.
  - Sonst ist der Rückgabestring `"--"`.

  Annahme(n): $a \neq null, pos \geq 0, pattern.length() > 0, pos + pattern.length() \leq a.length()$
- **`String digitsToDistance(String text)`** gibt eine Kopie von `text` zurück, in der jede Ziffer durch die Anzahl jener Zeichen ersetzt wird, welche in `text` zwischen der Ziffer und der vorangehenden Ziffer in `text` stehen. Die erste Ziffer wird durch die Anzahl aller Zeichen, welche vor ihr in `text` vorkommen, ersetzt. *Hinweis:* Ziffern können durch mehrstellige Zahlen ersetzt werden.
  Annahme(n): $text \neq null$
- **`void printPattern(int lineLength, int patternLength)`** gibt auf der Konsole eine oder mehrere Zeilen der Länge `lineLength` aus. Jede Zeile enthält `patternLength` mal das Zeichen '?' direkt hintereinander. In der ersten Zeile stehen diese am Beginn der Zeile, in jeder weiteren Zeile wandern sie um jeweils `patternLength` viele Zeichen nach rechts. Die Zeilen werden rechts der '?' mit dem Buchstaben 'y' aufgefüllt, und ab der zweiten Zeile links der '?' mit dem Buchstaben 'x'.
  Es gibt genau so viele Zeilen, dass sich auch in der letzten Zeile noch alle `patternLength` viele Zeichen '?' ausgehen.
  Annahme(n): $0 < patternLength \leq lineLength$

Implementieren Sie folgende Punkte **in `main`**:

- Deklarieren Sie eine short-Variable `result` und eine String-Variable `test` mit dem Inhalt `"all: hallo hall"`.
- `result` soll das Ergebnis des Aufrufs `sumUp(4, 9, 320_300)` ohne Compiler-Fehler zugewiesen werden.

Testen Sie alle Methoden in `main` mit zumindest folgenden Aufrufen, und erzeugen Sie mit deren Hilfe die gezeigten Ausgaben. Außer `printPattern` darf keine der implementierten Methoden eine Ausgabe erzeugen. Alle anderen Ausgaben müssen in `main` erfolgen.

Aufruf	**Ausgabe in main auf der Konsole**
`sumUp(2, 5, 11)`	`3`
`sumUp(8, 8, 40)`	`4`
`sumUp(5, 1, 2000)`	`0`
`addMark(test, "allo", 6)`	`allo hall`
`addMark(test, "all", 3)`	`--all: hallo hall`
`addMark(test, "hall", 3)`	`all: hallo hall--`
`addMark(test, "@all", 3)`	`--`
`digitsToDistance("12oder34")`	`00oder40`
`digitsToDistance("Ich bin 1 Berliner!11")`	`Ich bin 8 Berliner!100`
`digitsToDistance("Heute ist der 4.März 2022")`	`Heute ist der 14.März 6000`
`digitsToDistance("Vier*mal*vier_=_0")`	`Vier*mal*vier_=_16`

`printPattern(6, 2)`	`??yyyy`
	`xx??yy`
	`xxxx??`

`printPattern(7, 3)`	`???yyyy` `xxx???y`
`printPattern(1, 1)`	`?`
`printPattern(8, 5)`	`?????yyy`

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`sumUp` - 5 Punkte	Korrekte Schleife(n)	1
	Korrekter Test	1
	Korrekte Berechnung und Rückgabe	3
`addMark` - 6 Punkte	Substrings an richtigen Stellen gesucht	2
	Korrekte Suche	2
	Korrekte Rückgabe	2
`digitsToDistance` - 4 Punkte	Schleife(n) und korrekter Test auf Ziffern	2
	Ersetzungen richtig berechnet, eingefügt, zurückgegeben	2
`printPattern` - 6 Punkte	Korrekte Programmstruktur	1
	Korrekte Anzahl an Zeilen	1
	Struktur der Zeilen korrekt	3
	Korrekte Entwicklung des Musters	1
`main` - 4 Punkte	Deklarationen	1
	Korrekte Zuweisung (result)	1
	Korrekte Aufrufe	1
	Ausgaben in der Konsole	1
Gesamt		25