Beispiel Angabe

Kürzel: asciishop-A06-PP
Name: AsciiShop, Runde#6
Kette: Asciishop PP
Kategorie: Bildverarbeitung

Mitgelieferte Datei(en): asciishop-A06-bonus1.i1, asciishop-A06-bonus1.o1
Abzugebende Datei(en): AsciiImage.java, AsciiShop.java, AsciiPoint.java, AsciiStack.java
Optional abzugebende Datei(en):
Ausführbar: AsciiShop

Die Klasse *AsciiShop* ist zu erstellen und soll eine ausführbare Klasse sein und muss daher die `public static void main(String[] args)` Methode beinhalten. Ihr Programm wird automatisch auf Korrektheit überprüft. Die Überprüfung erfolgt durch die Ausführung der als ausführbar bezeichneten Klasse (*AsciiShop*).

Kurzbeschreibung:

Das Programm erstellt im ersten Schritt ein leeres Bild, auf dem dann unterschiedliche Operationen ausgeführt werden können. Neben den in der Vorrunde implementierten Operationen, gibt es noch zwei weitere Befehle, mit denen man Flächen eines bestimmten Zeichens wachsen lassen und deren Schwerpunkt bestimmen kann. Zusätzlich ist es nun möglich, eine beliebige Anzahl von Befehlen rückgängig zu machen.

Lernziele:

- Umgang mit Arrays
- Umgang mit Collections
- Implementieren einer eigenen Collection
- Kapselung von Funktionalität

Aufgabenstellung | Klassen und Methoden | Ein- und Ausgabedaten | Bewertung und Kriterien
Hinweise | FAQ | Fehlerbehandlung | Testen

Aufgabenstellung:

Mit zunehmender Funktionalität wird es erforderlich, diese sinnvoll zu kapseln. So sollen zwei neue Klassen eingeführt werden: Eine stellt die Funktionalität eines Stacks zur Verfügung, auf den ASCII-Bilder gelegt werden kÃ¶nnen, um so Operationen am Bild rückgängig machen zu können, die andere kapselt die Koordinaten eines Punktes in einem Ascii-Bild. Außerdem soll eine Hilfsmethode implementiert werden, die zur Übersichtlichkeit im Code beiträgt und die Code-Wiederverwendung unterstützt.

Erzeugen des Bildes

Das Erzeugen eines neuen Bildes erfolgt, wie in Runde 5 spezifiziert, mit Hilfe des Befehls **`create`**, dessen Parameter die Breite und Höhe eines neu zu erzeugenden Bildes spezifizieren. Dieser muss der erste eingegebene Befehl sein. Sind Breite oder Höhe kleiner oder gleich null, so soll "`INPUT MISMATCH`" ausgegeben werden.

Das leere Bild kann mit Daten gefüllt werden. Dazu stehen verschiedene Befehle zur Verfügung, darunter der Befehl load. Dieser Befehl (und damit das Einlesen von Bildern von der Standardeingabe) ist für ein gültiges Bild nicht obligatorisch.

Befehle und Operationen am Bild

Die folgenden Befehle sind zulässig, neue Befehle sind farblich hervorgehoben:

- **`centroid`** `c` bestimmt den Schwerpunkt aller Pixel, die dem als Parameter definierten Zeichen (`c`) entsprechen und gibt das Ergebnis in der Form '`(x,y)`' in einer eigenen Zeile aus. Sollte es kein Vorkommen des angegeben Zeichens geben, so ist '`null`' auszugeben.
- **`clear`** löscht den gesamten Bildinhalt, alle Pixel des Bildes werden auf '`.`' gesetzt.
- **`grow`** `c` vergrößert den Bereich der Pixel mit dem Zeichen `c`. Dies bewirkt, dass alle Pixel, die nicht gesetzt sind (also das Zeichen '`.`' sind), und als Nachbarn einen Pixel mit dem Zeichen `c` haben, auf das Zeichen `c` gesetzt werden.
- **`line`** x_0 y_0 x_1 y_1 `c` zeichnet eine Linie, wobei die ersten beiden Parameter (x_0, y_0) die Koordinaten/Indizes des Startpunktes angeben, der dritte und vierte Parameter (x_1, y_1) die Koordinaten/Indizes der Endposition. Der letzte Parameter spezifiziert das zu verwendende Zeichen. Das zugrunde liegende Koordinatensystem hat den Ursprung links oben im Bild, nach rechts verläuft die x-Achse, nach unten die y-Achse.
- **`load`** `eof` liest ein Bild zeilenweise ein und speichert es in das anfangs mit **`create`** erzeugte Bild. Um das Ende der Eingabe zu

erkennen, wird als Parameter eine Zeichenkette (`eof`) angegeben, die das Ende der Bildeingabe kennzeichnet. Entspricht die Höhe und Breite des eingelesenen Bildes nicht exakt jener des mit **`create`** erzeugten Bildes, so soll "`INPUT MISMATCH`" ausgegeben werden.

- **`print`** gibt das ASCII-Bild gefolgt von einer Leerzeile aus. Im Gegensatz zu früheren Runden, erfolgt die Ausgabe nicht mehr automatisch am Ende des Programms, sondern nur bei Eingabe dieses Befehls.
- **`replace`** `oldChar newChar` ersetzt alle Vorkommen eines bestimmten Zeichens (oldChar) im Bild durch ein anderes Zeichen (newChar).
- **`transpose`** transponiert das Bild.
- **`fill`** `x y c` führt ausgehend von der Position `(x,y)` ein Floodfill (siehe Spezifikation zu Runde 3) mit dem Zeichen `c` durch.
- **`undo`** macht einen Befehl rückgängig. Gibt es keinen weiteren Befehl, der rückgängig gemacht werden kann, soll "`STACK EMPTY`" ausgegeben werde. Ansonsten soll die Belegung des Stacks nach dem Entfernen des obersten Elements in der Form '`STACK USAGE genutzterPlatz/verfügbarerPlatz`' angezeigt werden.

Die genannten Befehle **`centroid`**, **`clear`**, **`grow`**, **`line`**, **`replace`**, **`transpose`** und **`fill`** haben entsprechende Methoden in der Klasse **`AsciiImage`**. Alle hier aufgeführten Befehle können in beliebiger Reihenfolge auftreten. So kann zum Beispiel nach **`create`** erst **`print`** und danach **`load`** folgen.

Code-Wiederverwendung

Ein wichtiges Ziel beim Programmieren ist es, doppelten Code zu vermeiden und durch geschickte Nutzung bestehenden Codes neuen Code einfacher zu schreiben, bzw. bestehenden Code zu überarbeiten. Überladen Sie die Methoden `getPixel` und `setPixel`, sodass sie Koordinatenangaben sowohl als zwei einzelne Werte als auch als `AsciiPoint` akzeptieren. Implementieren Sie diese Methoden möglichst effizient, sprich stellen Sie sicher, dass kein Code dupliziert wird. Die neu zu implementierende Methode `getPointList` bietet die Funktionalität alle Punkte einer Farbe zu erhalten. Identifizieren Sie neue aber auch bestehende Methoden, die diese Funktionalität ebenfalls benötigen und schreiben bzw. verändern Sie den Code so, dass dort die getPointList Methode verwendet wird.

Übungsmöglichkeiten

Sie können zu Übungszwecken in `AsciiImage` auch die Methoden `getUniqueChars` und `flipV` aus Runde 3 entsprechend der Spezifikation implementieren und diesen Methoden die entsprechenden Befehle **`uniqueChars`** und **`flip-v`** zuordnen. Dies wird jedoch nicht getestet. Falls Sie **`uniqueChars`** implementieren, geben Sie bei Eingabe dieses Befehls das entsprechende Ergebnis in einer eigenen Zeile aus

Bonusaufgaben

Bonusaufgaben werden nicht im Online-System getestet, die Beurteilung erfolgt erst während des Abgabegesprächs.

Flächen glätten *(+1,0 Punkte)*

In AsciiImage soll die Methode `public void straightenRegion(char c)` implementiert werden. Diese soll das Bild durch Entfernen dünner Linien bzw. einzelner Störpixel glätten. Dafür werden alle Pixel des Zeichens `c` entfernt (sprich auf '.' gesetzt), die einen oder keinen Nachbarpixel mit dem Zeichen `c` besitzen. Hierbei werden, wie beim Befehl **`grow`**, wieder vier Nachbarpixel betrachtet. Das Verfahren soll solange iteriert werden, bis keine Pixel mehr entfernt werden können. Dies führt dazu, dass allein im Bild liegende oder an Flächen angrenzende einzelne Punkte und Linien mit einer Breite von einem Pixel entfernt werden. Ordnen Sie dieser Methode den Befehl **`straighten c`** zu. Nutzen Sie zum Testen Ihrer Implementierung den in der Downloadversion mitgelieferten Testdatensatz `asciishop-A-05-bonus1.i1`.

Klassen und Methoden:

Die folgende Aufzählung umfasst geforderte Methoden, neue Methoden sind farblich hervorgehoben. Sie können nach Bedarf Hilfsmethoden und Methoden für freiwillige Aufgaben (Bonusaufgaben, Übungsaufgaben) hinzufügen. Achten Sie auf die korrekte Datenkapselung. Insbesondere sollen Sie sinnvolle Zugriffsmodifikatoren für Variablen (und Methoden) verwenden.

AsciiShop

Diese Klasse ist ausführbar und beinhaltet daher die `main`-Methode. Sie verarbeitet die Eingaben, erzeugt das `AsciiImage` und gibt das Ergebnis aus. Methoden dieser Klasse lesen direkt von `System.in` ein und geben direkt auf `System.out` aus.

```
public static void main(String[] args)
```

liest die Daten und Befehle ein und gibt das Ergebnis aus.

AsciiImage

Diese Klasse repräsentiert ein ASCII-Bild, es speichert die Zeichen des Bildes und bietet entsprechende Methoden zur Modifikation und zur Abfrage von Eigenschaften, wie beispielsweise Höhe und Breite. Methoden dieser Klasse lesen weder direkt von `System.in` ein, noch geben sie direkt auf `System.out` aus.

```
public AsciiImage(int width, int height)
```

erzeugt ein ASCII-Bild der spezifizierten Größe. Anfangs sind alle Pixel auf den Wert '.' gesetzt. Sie dürfen an dieser Stelle davon

ausgehen, dass Breite und Höhe beide größer 0 sind. Überprüfen Sie diese Bedienung an geeigneter Stelle in der Klasse `AsciiShop`.

```
public AsciiImage(AsciiImage img)
```

ist ein Kopierkonstruktor. Er erzeugt ein neues AsciiImage mit dem gleichen Inhalt, wie das übergebene Bild.

```
public void clear()
```

setzt alle Pixel des Bildes auf das Zeichen '.'.

```
public void drawLine(int x0, int y0, int x1, int y1, char c)
```

zeichnet eine Linie zwischen den Koordinaten (x_0, y_0) und (x_1, y_1). Anfangs- und Endpunkt sind dabei inkludiert. `c` spezifiziert das zu verwendende Zeichen. Sie können davon ausgehen, dass nur gültige Koordinaten übergeben werden.

```
public AsciiPoint getCentroid(char c)
```

bestimmt den Schwerpunkt aller Pixel mit dem Zeichen `c` und gibt diesen als `AsciiPoint` zurück. Kommt das Zeichen nicht vor, so wird `null` zurückgegeben.

```
public int getHeight()
```

gibt die Höhe des Bildes (die Anzahl der Zeilen) zurück.

```
public char getPixel(int x, int y)
```

gibt das an den übergebenen Koordinaten/Indizes gespeicherte Zeichen zurück. Sie dürfen an dieser Stelle davon ausgehen, dass die `x` und `y` gültige Koordinaten sind.

```
public char getPixel(AsciiPoint p)
```

gibt, analog zur Methode `public char getPixel(int x, int y)`, das Zeichen, an der durch `p` spezifizierten Stelle, zurück.

```
public ArrayList<AsciiPoint> getPointList(char c)
```

gibt eine ArrayList aller Pixel eines bestimmten Zeichens zurück. In dieser ArrayList sind Objekte vom Typ AsciiPoint, sollte es keine Punkte mit dem angegebenen Zeichen geben, so soll eine leere Liste zurückgegeben werden. Verwenden Sie diese Methode überall dort, wo sie alle Pixel mit einem bestimmten Zeichen benötigen.

```
public int getWidth()
```

gibt die Breite des Bildes (die Länge der Zeilen) zurück.

```
public void growRegion(char c)
```

vergrößert die die Flächen aller Pixel des Zeichens `c`, in dem es an diese Pixel angrenzende Hintergrundpixel (also Pixel mit dem Zeichen '.') auf das Zeichen `c` setzt.

`public void replace(char oldChar, char newChar)`

ersetzt alle Vorkommen eines bestimmten Zeichens `oldChar` im Bild durch ein anderes Zeichen `newChar`.

`public void setPixel(int x, int y, char c)`

speichert an den übergebenen Koordinaten/Indizes das übergebene Zeichen. Sie dürfen an dieser Stelle davon ausgehen, dass `x` und `y` gültige Koordinaten sind.

`public void setPixel(AsciiPoint p, char c)`

speichert, analog zur Methode `public char setPixel(int x, int y, char c)`, das übergebene Zeichen an der durch den `p` spezifizierten Stelle.

`public String toString()`

gibt eine lesbare Darstellung des ASCII-Bildes zurück. Die einzelnen Zeilen sollen dabei durch Zeilenumbrüche '\n' getrennt werden.

`public void transpose()`

vertauscht Zeilen und Spalten des Bildes, sprich aus der ersten Zeile im Bild wird die erste Spalte usw. Dabei ändern sich Höhe und Breite des Bildes (vgl. Matrix_(Mathematik)). Je nach Implementierung muss diese Methode sicherstellen, dass abhängige Eigenschaften des Bildes (Höhe, Breite) aktualisiert werden. Nutzen Sie für diese Methode wenn möglich ihren Code früheren Runden.

`public void fill(int x, int y, char c)`

Ersetzt das Zeichen an der Position *(x,y)* mit dem Zeichen *c* und ruft sich ggfs. selbst rekursiv auf (mit neuen Werten von *(x,y)* die den Nachbarpositionen entsprechen). Die Methode implementiert den rekursiven Floodfill Algorithmus (siehe Runde 3). Nutzen Sie für diese Methode ihren Code früheren Runden.

AsciiPoint

Diese Klasse repräsentiert einen Punkt, spezifiziert durch zwei ganzzahlige Koordinaten. Diese Klasse ist unveränderlich (immutable), sprich die Koordinaten sollen nachträglich nicht mehr veränderbar sein. Stellen Sie dies durch den Einsatz geeigneter Modifier sicher.

`public AsciiPoint(int x, int y)`

erzeugt einen Punkt mit den angegebenen Koordinaten.

`public int getX()`

gibt die x-Koordinate des Punktes zurück.

`public int getY()`

gibt die y-Koordinate des Punktes zurück.

`public String toString()`

gibt eine lesbare Darstellung des Punktes in der Form `(x,y)` zurück.

AsciiStack

Diese Klasse implementiert einen Stack (vgl. Stapelspeicher), der seine Größe dynamisch anpasst. Er kann eine beliebige Anzahl an AsciiImage-Objekten speichern, wobei der Zugriff immer nur auf das oberste Element möglich ist. Diese Implementierung nutzt intern ein Array zum Speichern der Elemente.

`public AsciiStack(int increment)`

erzeugt einen Stack, der initial `increment` Elemente speichern kann. Sie dürfen davon ausgehen, dass `increment` größer 0 ist. Das Parameter `increment` gibt auch an, um wieviel der Stack bei Bedarf vergrößert bzw. verkleinert werden soll.

`public int capacity()`

gibt die Anzahl der Stack bereit stehenden Plätze zurück (sprich wie groß das zu Grunde liegende Array ist). Aufgrund der Vorgehensweise bei Vergößerung und Verkleinerung, ist das Ergebnis dieser Methode immer ein Vielfaches von `increment`.

`public boolean empty()`

überprüft, ob der Stack leer ist. In diesem Fall liefert die Methode `true`. Wenn mindestens ein Element am Stack liegt liefert die Methode `false`.

`public AsciiImage pop()`

gibt das oberste Element am Stack zurück und entfernt dieses. Liegt kein Element am Stack, so soll `null` zurückgegeben werden. Sind nach dem Entfernen mehr als `increment` Plätze leer, so soll der Stack um `increment` verkleinert werden. Jedoch soll der Stack nie kleiner als `increment` sein, sprich wenn alle Elemente entfernt wurden, soll die Kapazität des Stacks gleich `increment` sein.

`public AsciiImage peek()`

gibt das oberste Element am Stack zurück ohne es zu entfernen. Liegt nichts am Stack, so soll `null` zurückgegeben werden.

`public void push(AsciiImage img)`

legt ein AsciiImage oben auf den Stack. Ist der Stack zu diesem Zeitpunkt voll, so soll der Stack um `increment` vergrößert werden um so das Bild speichern zu können.

`public int size()`

gibt die Anzahl der im Stack belegten Plätze zurück.

Hinweise:

Beachten Sie die allgemeinen Hinweise zur Installation und zur Ein-/Ausgabe, sowie zur Abgabe und zur Beurteilung in den FAQ.

Schwerpunkt berechnen

Die Methode `getCentroid` soll den Schwerpunkt aller Pixel eines bestimmten Zeichens bestimmen. Dieser ist durch das arithmetische Mittel der Koordinaten all dieser Punkte definiert. Um dieses zu berechnen, werden von allen Pixel mit dem entsprechenden Zeichen die x und y Koordinaten aufsummiert und dann durch die Anzahl dividiert. Beachten Sie, dass das Ergebnis mathematisch exakt auf ganze Zahlen gerundet werden soll. Zum Runden verwenden Sie `Math.round`, für notwendige Typumwandlungen im Rahmen der Berechnungen, nutzen Sie doppelte Genauigkeit (`double`). Der Schwerpunkt soll als AsciiPoint zurückgegeben werden.

Flächen vergrößern

Um die Fläche eines bestimmten Zeichens zu vergrößern, durchlaufen Sie alle Punkte mit dem entsprechenden Zeichen. Überprüfen Sie für jeden Nachbarn, ob er die Hintergrundfarbe (‘.’) besitzt. Ist dies der Fall, so setzen Sie den entsprechenden Nachbarpunkt auf das Zeichen der Region. Als Nachbarschaft wird eine sogenannte Vierer-Nachbarschaft (vgl. Vierer-Nachbarschaft) vorausgesetzt, sprich jeder Pixel hat vier Nachbarn: die über und unter dem Pixel, sowie die links und rechts vom Pixel.

Undo-Funktionalität

Implementieren Sie die Klasse AsciiStack mittels eines Arrays, verwenden Sie keine Collection (wie beispielsweise java.util.ArrayList oder java.util.Stack). Nutzen Sie diese Klasse dann im AsciiShop mit einem `increment` von 3. Um das Rückgängigmachen von Operationen zu ermöglichen, gehen Sie wie folgt vor: Speichern Sie in einer Variable `img` (in `AsciiShop`) immer das aktuelle Bild. Legen Sie, bevor Sie eine Operation, die das Bild verändert, durchführen (das sind insbesondere **`grow`**, **`clear`**, **`replace`**, **`line`**, **`transpose`** und **`load`**), eine Kopie des Bildes auf den Stack. Nutzen Sie dazu den Kopierkonstruktor der Klasse AsciiImage. Die Bildoperationen können Sie dann auf `img` ausführen.

Entfernen Sie nach Einlesen des Befehl **undo** jeweils das oberste Element vom Stack, speichern Sie es in die lokale Variable `img` (in `AsciiShop`) und geben Sie Größe und Kapazität des Stacks aus. Geben Sie "`STACK EMPTY`" aus, falls keine weiteren Schritte rückgängig gemacht werden können. Beachten Sie, dass der **`create`** Befehl nicht rückgängig gemacht werden kann.

	size	capacity
`AsciiStack(3)`	0	3
`.push(...)`	1	3
`.push(...)`	2	3
`.push(...)`	3	3
`.pop()`	2	3
`.push(...)`	3	3
`.push(...)`	4	6
`.pop()`	3	6
`.pop()`	2	3
Undo-Stack Zustand nach den entsprechenden Aufrufen		

Kopier-Konstruktor

Stellen Sie sicher, dass der Kopierkonstruktor der Klasse AsciiImage die Daten tief kopiert, sprich es muss der Arrayinhalt und nicht nur die Referenz darauf kopiert werden.

Wenn Sie Fragen zur Implementierung oder auch zu Java haben, können Sie das Informatik-Forum nutzen. Im Rahmen der wöchentlichen Laborien stehen Tutoren fÃŒr Fragen zur Verfügung.

Informatik-Forum Laborien

Eingabedaten:

Der erste Befehl muss **`create`**, gefolgt von Breite und Höhe des Bildes sein. Danach können in beliebiger Reihenfolge beliebige viele der oben definierten Befehle folgen. Beachten Sie jedoch, dass der **`create`** Befehl nur einmal (nämlich als erster Befehl) auftreten darf. Sie dürfen davon ausgehen, dass die mittels **`load`** eingelesenen Bilddaten keine Leerzeichen enthalten. Sie können weiters davon ausgehen, dass die gesamte Eingabe nicht leer ist.

Ausgabedaten:

Bei jedem Aufruf von **`print`** soll das Bild korrekt formatiert und von einer Leerzeile gefolgt ausgegeben werden. Wird **`centroid`** eingegeben, so wird der Schwerpunkt in der Form `(x,y)` in einer eigenen Zeile ausgegeben. Wird **`undo`** angegeben, so wird das Ergebnis dieses Befehls, also entweder "`STACK EMPTY`" oder "`STACK USAGE genutzterPlatz/verfügbarerPlatz`" ausgegeben.

Fehlerbehandlung:

Geben Sie "`INPUT MISMATCH`" aus und brechen Sie die weitere Verarbeitung ab, falls einer der folgenden Fehler auftritt:

- Der erste Befehl ist nicht **`create`** gefolgt von zwei Zahlen größer 0
- Zwischen dem Befehl **`load`** und der End-Zeichenkette sind mehr oder weniger Zeilen, als das Bild hat.
- Die Länge einer eingelesenen Bildzeile ist ungleich der Breite des Bildes.
- Einer der Parameter eines Befehls ist ungültig, im Speziellen: fehlende Parameter, Parameter vom falschen Typ

Geben Sie "`UNKNOWN COMMAND`" aus und brechen Sie die weitere Verarbeitung ab, falls einer der folgenden Fehler auftritt:

- Nach dem Befehl **`create`** folgt (später) ein unbekannter Befehl.
- Nach dem Befehl **`create`** folgt (später) erneut der Befehl **`create`**.

Geben Sie "`OPERATION FAILED`" aus und brechen Sie die weitere Verarbeitung ab, falls der folgende Fehler auftritt:

- Die bei **`fill`** angegebene Position liegt ausserhalb des Bildbereichs

Testen:

```
create 40 24
```

In

```
load !
.....................................
.....................................MM...
.................................MMMMMM,...
.............................,MMMMMMMMMM....
.....................,MMMMMMMMMMMMM.....
...XXX.........MMMMMMMMMMMMMMMMMM.....
...XXX.........MMMMMMMMMMMMMMMM......
...XXX.........MMMMMMMMMMMMMMMM......
...XXX.........MMMMMMMMMMMMMMMM,.....
...XXX.........+MMMMMMMMMMMMMMMM.....
...XXX.........MMMMMMMMMMMMMMMM......
.............MMMMMMMMMMMMMMMMMM......
.............MMMMMMMMMMMMMM.MMMM,.....
.........MMMMMMMMMMMMMM+....MM.....
.....+MMMMMMMMMMMMM.................
.....MMMMMMMMMMMMM...................
.....MMMMMMMMMMMMMM..................
....MMMMMMMMMMMMM.....................
..MMMMMMMMMMMMM+.....................
..MMMMMMMMMMMM.......................
....MMMMMMMM.........................
.....MMMMMM..........................
.....MMM.............................
.....................................
!
centroid M
centroid X
```

Out

```
(19,11)
(4,8)
```

Beschreibung Berechnung des Schwerpunkts der FlÃ€che des Zeichens 'M' und des Zeichens 'X'.

In

```
create 41 10
clear
line 0 0 40 8 B
clear
print
undo
```

```
print
```

```
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
```

Out

```
STACK USAGE 2/3
BBB.....................................
...BBBBB................................
........BBBBB...........................
............BBBBB.......................
................BBBBB...................
....................BBBBB...............
........................BBBBB...........
............................BBBBB.......
................................BBBBB...
....................................BBB
........................................
```

Beschreibung Einfacher Test des Undo-Stacks. Eine Linie wird gelÃ¶scht, dies wird mittels "undo" rÃŒckgÃ€ngig gemacht.

In

```
create 13 8
line 5 0 0 2 #
line 7 0 12 2 #
line 3 4 3 7 #
line 9 7 9 4 #
line 0 4 3 4 #
line 12 4 9 4 #
line 6 4 6 4 *
print
grow *
print
grow *
grow *
```

```
grow *
grow *
grow *
grow *
replace # .
print
```

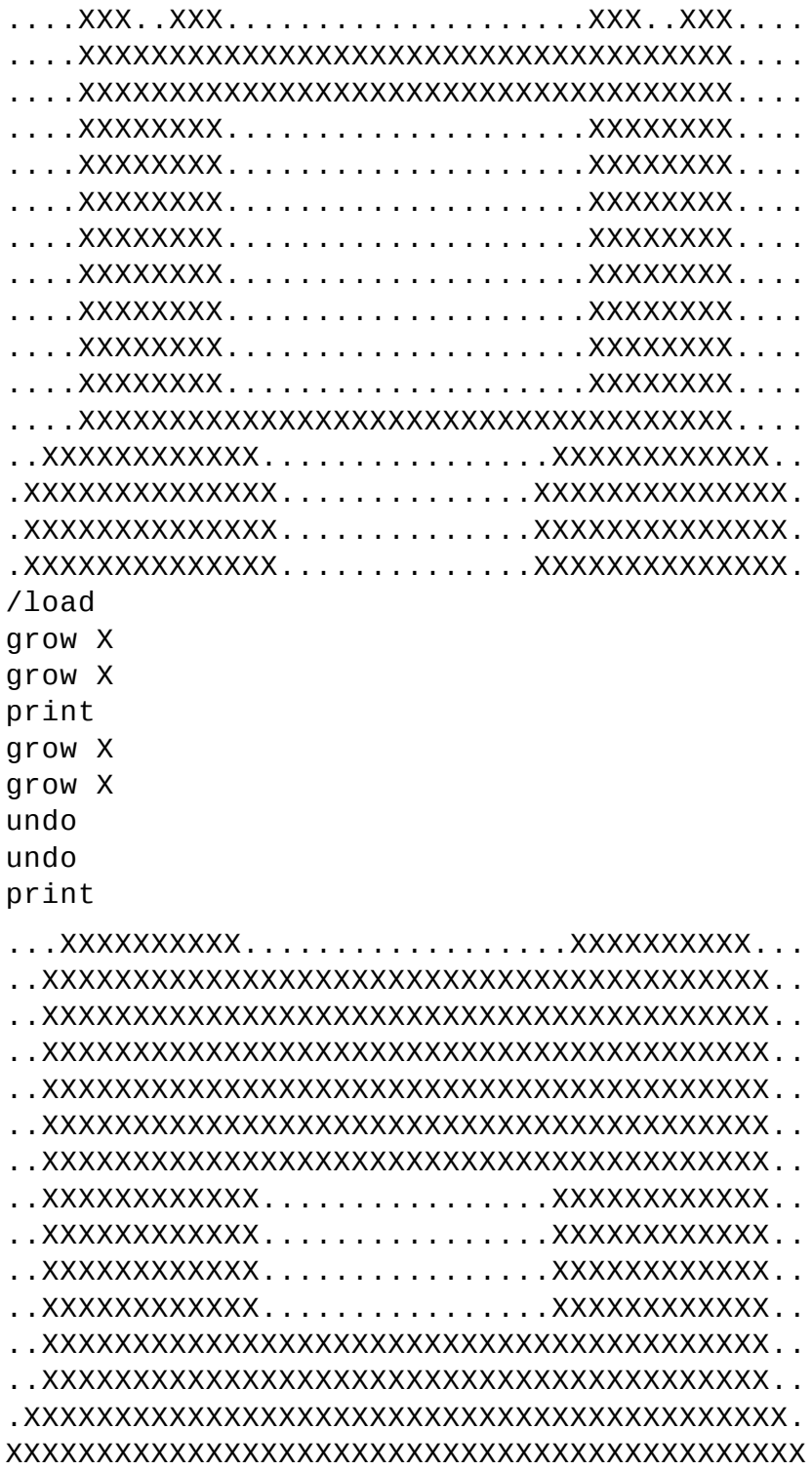
Out

```
....##.##....
..##.....##..
##.........##
.............
####..*..####
...#.....#...
...#.....#...
...#.....#...

....##.##....
..##.....##..
##.........##
......*......
####.***.####
...#..*..#...
...#.....#...
...#.....#...

......*......
....*****....
..*********..
*************
....*****....
....*****....
....*****....
....*****....
```

Beschreibung Mit Hilfe einiger Linien und mehrfacher Anwendung des grow Befehls entsteht ein Pfeil.

```
create 44 18
load /load
.....X....X......................X....X.....
....XXX..XXX....................XXX..XXX....
```

In



```
/load
grow X
grow X
print
grow X
grow X
undo
undo
print
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX.........XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX.........XXXXXXXXXXXXXXXX
```

Out

```
STACK USAGE 4/6
STACK USAGE 3/6
...XXXXXXXXXX..................XXXXXXXXXX...
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
..XXXXXXXXXXXXX................XXXXXXXXXXXXX..
..XXXXXXXXXXXXX................XXXXXXXXXXXXX..
..XXXXXXXXXXXXX................XXXXXXXXXXXXX..
..XXXXXXXXXXXXX................XXXXXXXXXXXXX..
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
.XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXX.........XXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXX.........XXXXXXXXXXXXXXXXX
```

Beschreibung Durch mehrfaches AusfÃŒhren von grow, muss der Undo-Stack vergrÃ¶Ãert werden.

```
create 30 27
load END
============,.V+.;============
===========;.X##..;===========
===========..####..===========
==========,.##M###.:==========
=========;.R##WW##=.;=========
=========..##MWMW##,.=========
========:.I##WWMWW#W.,========
=======;.:##WWWWWW##I.;=======
=======:.W########M##..=======
=======.:############V.;======
```

In

```
=======.####...,.,M###,.======
======,.###:..+++;,B..#.;=====
=====;.#;.#i+YXVVY:#V.#..:====
====:..#..##.RYYIt##...#=.,===
==;..+#;...#######W....##W..:=
:..t###......iVV=....,.+#iM#..
.B#i,#..,,.............,..#,..#I
V#...#..,,,,,.,,,,:,,,,.R#..V#
;#+.i#..,,,,,,,,,.,,,,,.iX.X#,
.i#=MV..,,,,.,,,,,,.,,,..###..
,.t##..,,,,,,,,,,,,,:,..V##.;
..M#M..,,,,.,.,,,,,,,:,,.=##..
i###M..,.........,,:,,..###.
.####........=II;.....,,.;##R.
,...#....X####MW###B.....Wi..:
==;,##,###;........##i...#;,==
===,.IVi..,;======:..B#B#t.;==
END
clear
undo
centroid W
clear
clear
clear
clear
undo
undo
undo
centroid W
undo
centroid W
undo
undo
```

Out

```
STACK USAGE 1/3
(16,9)
STACK USAGE 4/6
STACK USAGE 3/6
STACK USAGE 2/3
null
STACK USAGE 1/3
```

	(16,9) STACK USAGE 0/3 STACK EMPTY
Beschreibung	Durch wiederholtes AusfÃŒhren von Befehlen und dem wiederholten RÃŒckgÃ€ngigmachen, muss zuerst der Undo-Stack vergrÃ¶Ãert werden und anschlieÃend wieder verkleinert werden.

Bemerkung: Diese Beispiele dienen nur zur Verdeutlichung der Spezifikation und müssen nicht korrekt formatiert sein. Die korrekte Formatierung entnehmen Sie bitte dem mitgelieferten Outputfile. Zum Testen Ihrer Lösung können Sie aus den mitgelieferten Eingabedaten wie folgt eine Ausgabedatei erzeugen:
`java AsciiShop < asciishop-A06-PP.i1 > asciishop-A06-PP.out1`

Das erzeugte File asciishop-A06-PP.out1 können Sie dann mit dem mitgelieferten Outputfile asciishop-A06-PP.o1 vergleichen.