P08 Ascii Art

Overview

In this assignment, you are going to draw some ASCII art pictures. You will have a canvas consisting of a grid of characters. In order to maintain a more useful drawing program, you are going to define and use different stacks to enable UNDO and REDO functions. Fig. 1 shows a sample run of the driver application of this assignment.

Learning Objectives

The goals of this assignment include implementing, using, and testing an implementation of the generic interface StackADT using linked nodes.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P8 pre-assignment quiz is accessible through Canvas before having access to this specification by 9:59PM on Sunday 11/10/2019. Access to the pre-assignment quiz will be unavailable passing its deadline.
20 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should run additional tests of your own.
15 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.
10 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.

```
======== MENU ========
[1] Create a new canvas
[2] Draw a character
[3] Undo drawing
[4] Redo drawing
[5] Show current canvas
[6] Show drawing history
[7] Exit
> 1
Width > 4
Height > 4
======== MENU ========
[1] Create a new canvas
[2] Draw a character
[3] Undo drawing
[4] Redo drawing
[5] Show current canvas
[6] Show drawing history
[7] Exit
> 2
Row > 0
Col > 0
Character > L
======== MENU ========
[1] Create a new canvas
[2] Draw a character
[3] Undo drawing
[4] Redo drawing
[5] Show current canvas
[6] Show drawing history
[7] Exit
> 2
Row > 0
Col > 3
Character > R
======== MENU ========
[1] Create a new canvas
[2] Draw a character
[3] Undo drawing
[4] Redo drawing
[5] Show current canvas
[6] Show drawing history
[7] Exit
> 5
L□□R
□□□□
□□□□
□□□□
======== MENU ========
[1] Create a new canvas
[2] Draw a character
[3] Undo drawing
[4] Redo drawing
[5] Show current canvas
[6] Show drawing history
[7] Exit
> 3
```

```
======== MENU ========
[1] Create a new canvas
[2] Draw a character
[3] Undo drawing
[4] Redo drawing
[5] Show current canvas
[6] Show drawing history
[7] Exit
> 5
L□□□
□□□□
□□□□
□□□□
======== MENU ========
[1] Create a new canvas
[2] Draw a character
[3] Undo drawing
[4] Redo drawing
[5] Show current canvas
[6] Show drawing history
[7] Exit
> 4
======== MENU ========
[1] Create a new canvas
[2] Draw a character
[3] Undo drawing
[4] Redo drawing
[5] Show current canvas
[6] Show drawing history
[7] Exit
> 5
L□□R
□□□□
□□□□
□□□□
======== MENU ========
[1] Create a new canvas
[2] Draw a character
[3] Undo drawing
[4] Redo drawing
[5] Show current canvas
[6] Show drawing history
[7] Exit
> 6
2. draw 'R' on (0,3)
1. draw 'L' on (0,0)
======== MENU ========
[1] Create a new canvas
[2] Draw a character
[3] Undo drawing
[4] Redo drawing
[5] Show current canvas
[6] Show drawing history
[7] Exit
> 7
Bye!
```

Figure 1: Sample output of this ASCII Art Application - note that squares represent spaces

Additional Assignment Requirements and Notes

- You MUST NOT add any fields either instance or static, and any public methods either static or instance to your `DrawingChange`, `DrawingStack`, `DrawingStackIterator`, and `Canvas` classes, other than those defined in this write-up.
- DO NOT make any change to the provided source files `DrawingChange.java`, `DrawingStack.java`, `DrawingStackIterator.java`, `Canvas.java`, and `AsciiArtTester.java`. In addition, **DO NOT submit any of those provided files on gradescope**.
- All your test methods should be defined and implemented in your `AsciiArtTester.java`.
- You CAN define local variables that you may need to implement the methods defined in this program.
- You CAN define private methods to help implement the different public methods defined in this write-up, if needed.
- In addition to the required test methods, we HIGHLY recommend (not require) that you develop your own unit tests (public static methods that return a boolean) to convince yourself of the correctness of every behavior implemented in your `DrawingStack`, `DrawingStackIterator`, and `Canvas` classes with respect to the specification provided in this write-up. Make sure to design the test scenarios for every method before starting its implementation. Make sure also to test all the special cases.
- All implemented methods including the overridden ones MUST have their own javadoc-style method headers, according to the CS300 Course Style Guide.

1 Getting Started

Start by creating a new Java Project in eclipse called P08 Ascii Art, for instance. You have to ensure that your new project uses Java 8, by setting the "Use an execution environment JRE:" drop down setting to "JavaSE-1.8" within the new Java Project dialog box. Then, add these provided StackADT.java, and LinkedNode.java. The provided `StackADT` interface represents the generic abstract data type stack. This interface will be implemented by two non-generic stack data structures in the next steps of this assignment. The `LinkedNode` class implements a linked node that can be used to implement any signly-list based data structure.

2 Create the DrawingChange Class

Now, create a class called `DrawingChange`. This class records the details of a single change made to your canvas. It MUST have the following fields.

```
public final int row; // row (y-coordinate) for this DrawingChange
public final int col; // col (x-coordinate) for this DrawingChange
public final char prevChar; // previous character in the (row,col) position
public final char newChar; // new character in the (row,col) position
```

Notice that all four fields for this class are `public final`. You can make these fields public because they are final, and because each DrawingChange object represents nothing more than a collection of these fields. In addition, the DrawingChange class must include a single constructor with exactly the following signature, but has no other methods.

```
public DrawingChange(int row, int col, char prevChar, char newChar) {}
```

3 DrawingStack Implementation and Testing

In the next steps of this programming assignment, you are going to create two stacks of DrawingChanges to implement UNDO and REDO operations related to this Ascii Art application. With this in mind, you have to create your own stack implementation that stores elements of type DrawingChange. To do so, create two class named `DrawingStack`, and `DrawingStackIterator`.

3.1 Notes related to the DrawingStack class

- Your DrawingStack class must implement both our provided `StackADT` interface and java.lang.Iterable interface. Notice that this StackADT is similar (but not identical!) to the version we discussed in class.
- Notice that `StackADT` is generic. But, `DrawingStack` must **NOT BE GENERIC**.
- Use the provided `LinkedNode` class to implement the stack using a chain-of-linked-nodes approach. **DO NOT create an array or ArrayList based stack implementation.**
- A `DrawingStack` MUST contain only elements of type `DrawingChange`. It defines only one private instance field called `top` of type $LinkedNode < DrawingChange >$ which refers to the top (meaning head) of the linked stack, and one private instance field called `size` of type int which keeps track of the total number of DrawingChange objects stored in the stack.
- Recall that you DO NOT need to submit the `StackADT` interface or the LinkedNode class; we will substitute our own copy of this code.
- Your DrawingStack's `iterator()` method should return a new DrawingStackIterator that starts at the top of the stack of DrawingChanges.

3.2 Notes related to the DrawingStackIterator class

- Your `DrawingStackIterator` class must implement $Iterator < DrawingChange >$ and must iterate directly over your stack.
- Your `DrawingStackIterator` class must define a private instance field of type $LinkedNode < DrawingChange >$ to keep track of the next element in the iteration.
- The constructor of your iterator should take a $LinkedNode < DrawingChange >$ as input parameter (which represents the top of the stack).
- DrawingStackIterator's next() method should throw NoSuchElementException with a descriptive error message if the stack is exhausted and the current element in the iteration does not have a next item.
- Recall that when called for the first time, next() method defined in DrawingStackIterator class must return the element at the top or head of the stack (if it is not empty).

3.3 Design and implement your unit tests first!

We highly recommend the use of the Test Driven Development process where the tests come first. Create the class `AsciiArtTester`. Make sure to design and implement a set of unit tests to check the correctness of the methods that you are going to implement. Among others, your `AsciiArtTester` has to include the following test method with exactly the following signature.

```
public static boolean testStackPushPeek(){ }
```

This test checks for the correctness of both `DrawingStack.push()` and `DrawingStack.peek()` methods. Make sure to vary your test scenarios considered in that test method. For instance, create an empty DrawingStack object. You can consider first the case to call peek method on an empty stack. Then, push a `DrawingChange` onto the stack, and then use peek to verify that the correct item is at the top of the stack. You can further consider pushing other elements onto the stack. Then, each call of peek() method should return the correct DrawingChange object that should be at the top of the stack.

4 Create the Canvas class

Now, create a class called `Canvas.java`. This class represents an ASCII-art image. It must have the following fields (you will not need any others).

```
private final int width; // width of the canvas
private final int height; // height of the canvas
private char [][] drawingArray; // 2D character array to store the drawing
private final DrawingStack undoStack; // store previous changes for undo
```

```
private final DrawingStack redoStack; // store undone changes for redo
```

In addition, your `Canvas` class must implement all the following public methods.

```
// Constructor creates a new canvas which is initially blank (use the default value
// for char type or you can use spaces)
public Canvas(int width, int height) {
  // Throws an IllegalArgumentException with a descriptive error message
  // if width or height is 0 or negative.
}
// Draw a character at the given position drawingArray[row][col]
public void draw(int row, int col, char c) {
  // This method should throw an IllegalArgumentException if the drawing
  // position is outside the canvas
  // If that position is already marked with a different character, overwrite it.
  // After making a new change, add a matching DrawingChange to the undoStack
  // so that we can undo if needed.
  // After making a new change, the redoStack should be empty (meaning that
  // you should clear the redoStack if it is not already empty).
}
// Undo the most recent drawing change.
public boolean undo() {
   // Return true if successful. False otherwise.
   // An undone DrawingChange should be popped off the undoStack.
   // An undone DrawingChange should be added to the redoStack so that
   // we can redo if needed.
   // The content of the drawingArray should be updated accordingly to this change.
}
// Redo the most recent undone drawing change.
public boolean redo() {
   // Return true if successful. False otherwise.
   // A redone DrawingChange should be popped off the redoStack.
   // A redone DrawingChange should be added (back) to the undoStack so that
   // we can undo again if needed.
   // The content of the drawingArray should be updated accordingly to this change.
}
// Return a printable string version of the Canvas.
public String toString() {
/* Format example:
 * [_ is blank. Use System.lineSeparator() to put a newline character between rows]
 * X___X
 * _X_X_
 * __X__
 * _X_X_
 * X___X
 */
```

```
}
```

In addition to the methods above, you may create the following additional public methods:

- Accessors and/or mutators for the private fields listed above.
- `public void printDrawing()` which prints the Canvas's string representation to `System.out`.
- If you create this method, your driver class may call it. Otherwise, your driver will need to use your Canvas's `toString()` method to display a drawing.
- `public void printHistory()` which prints a record of the changes that are stored on the undoStack. We have no specific requirements for this method, but you may find it helpful for understanding what your program is doing. We included some calls to this method in the sample run at the top of this page (Fig. 1.

5 Expanding The AsciiArtTester Class

Before going on to run the driver for your game, expand your `AsciiArtTester` class. Any testing methods you add should be **public static boolean** and take no parameters, returning true when they detect correct behavior of your program and false otherwise. You may add additional private helper methods as necessary. You may also add a main method to your testing class. In particular, you must add the following test method:

```
public static boolean runAsciiArtTestSuite() { }
```

Your `runAsciiArtTestSuite()` test method should run multiple other test methods. It **should return false if any of its component tests fail, and true if they all succeed**. Note that we will run this specific test method against several `DrawingStack` or `Canvas` implementations (some working, some broken) in our automated tests on Gradescope.

6 Driver Application AsciiArt Class

As the final step of this assignment, download and use this driver class called AsciiArtDriver.java to launch and run the Ascii Art Canvas Drawing application. Running the main method in this class must result in an interactive session like the one demonstrated in Fig. 1.You do not need to submit or make any change to the provided class `AsciiArtDriver`.

Here are some details on how this interactive session works:

- Launching the AsciiArt class's main method should immediately begin a driver loop and present the user with a menu similar to the one at the top of this writeup.

- From the user's perspective, the top left corner of the Canvas is located at row 0, column 0. When prompting the user for drawing coordinates, ask for the row and then the column on separate lines. Notice that Scanner.nextLine() is used to read user input.
- The menu should be stable. Bad user input should result in a message indicating the bad input and should prompt the user again.
- Entering drawing coordinates is stable. Your Canvas.draw() method throws an IllegalArgumentException with a descriptive error message if given coordinates are outside the drawing area. Notice that the driver avoids throwing this exception by handling it using a try-catch block.

7 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through gradescope.com. The only 3 files that you must submit include: `DrawingChange.java`, `DrawingStack.java`, `DrawingStackIterator.java`, `Canvas.java`, and `AsciiArtTester.java`. Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline of Due: **9:59PM on November 13**th. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline. Finally, the third portion of your grade for your submission will be determined by humans looking for organization, clarity, commenting, and adherence to the CS300 Course Style Guide.

Extra Challenges

Here are some suggestions for interesting ways to extend this memory game, after you have completed, backed up, and submitted the graded portion of this assignment. **No extra credit will be awarded for implementing these features**, but they should provide you with some valuable practice and experience. DO NOT submit such extensions via gradescope.

1. **Suggestion 1** – Add an option to save a Canvas and create a stack of Canvases to hold these saved Canvases. Add an option to "animate" your ASCII art by printing each of these saved Canvases in turn. (If you set this up correctly, you can get a flipbook-like animation effect.) Notice that this requires you to make a stack of a different type–you could either make a CanvasStack implementation, or modify your program to use a generic Stack implementation instead.
2. **Suggestion 2** – Extend the `DrawingChange` class in some way to allow changes beyond individual character shifts. Can you maintain backwards-compatibility so that your Driver and Canvas can work with both new and old DrawingChanges?