Assignment 1

CS 1027
Computer Science Fundamentals II

Due date: February 6 at 11:55 pm

Learning Outcomes

In this assignment, you will get practice with:

- Loops, 1D and 2D arrays (char and int), conditionals
- Creating and using Strings and constants
- Creating and using classes and objects
- Implementing constructor, toString methods, and other methods
- Overloading a method
- Implementing pseudocode

Introduction

There are several casual games that use a grid of tiles with the goal to make matching lines of the same tile by swapping adjacent tiles. Some notable examples are Bejeweled, Candy Crush, and Royal Match. When a line of matching tiles is created, it is removed and replaced often by tiles "falling" from above. For example, in the 5x5 grid below (the tiles are capital letters instead of jewels or candies as in Bejeweled or Candy Crush), the highlighted C can be swapped with the highlighted D which would then form a row of three Cs. In the games, this would cause these C tiles to disappear, and the tiles above to replace the hole left by the removal. Whenever tiles are removed a cascade of removals could take place since the new arrangement may also have lines that need to be removed (see Figures 2 and 3).

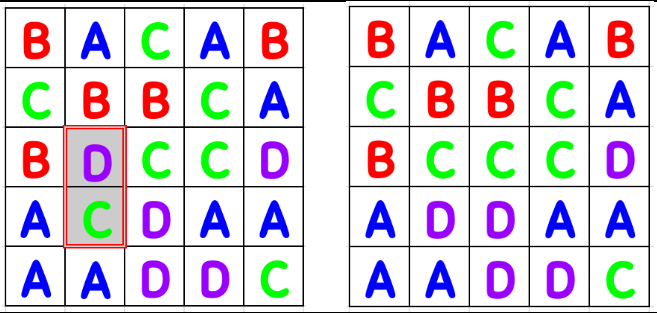


Fig 1. Swapping the highlighted D and C creates a line of three Cs

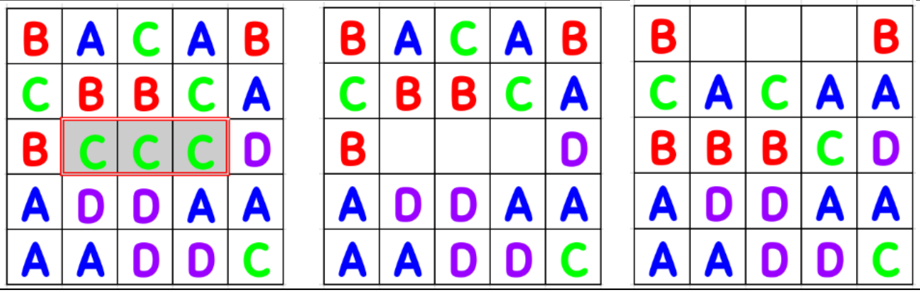


Fig 2. When the highlighted line of Cs is removed, tiles from above "fall" into their place.

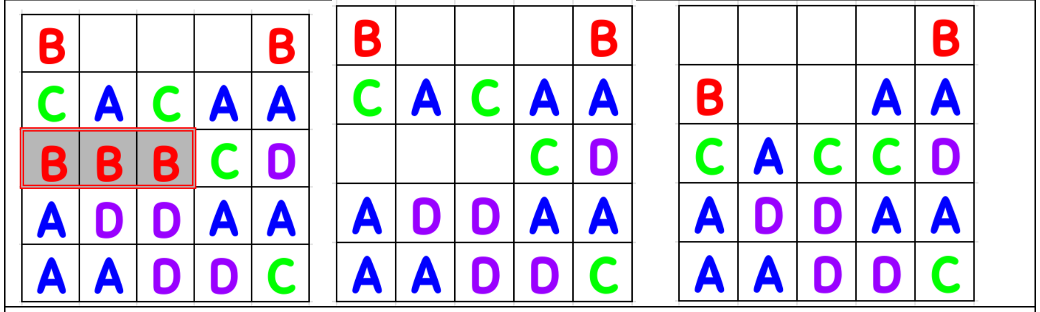


Fig 3. A new line of three Bs is created that will also be removed and consequently be filled in with the above tiles.

In this assignment, we will focus on an algorithm for cascading removal of tile lines. The gameplay and interface will not be part of the assignment.

Provided Files

Two files are provided to **help** check that your java Classes are implemented correctly. A similar file will be incorporated into Gradescope's auto-grader. **Passing all the tests within these files does not necessarily mean that your code is correct.**

- LineTest.java
- LetterCrushTest.java

Classes to Implement

For this assignment, you must implement two Java classes: Line and LetterCrush. Follow the guidelines for each one below.

In both these classes, you can implement more private (helper) methods, if you want to, but you may **not** implement more public methods except for a public static void main for testing. You **may not** add instance variables other than the ones specified below nor change the variable types or accessibility (i.e. making a variable public when it should be private). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

Line.java

This class is used to represent a horizontal or vertical line in an array. It stores the rows and columns of the endpoints of the line.

The class must have these private instance variables:

- private int[] *start*, *end*

The class must have the following public methods:

- public Line(int *row*, int *col*, boolean *horizontal*, int *length*) [this is the constructor]
  - Constructs a line starting at *row*, *col* that is horizontal if parameter *horizontal* is true or vertical if the parameter is false; the length of the line is specified by the last parameter (an int greater than 0). Notes: The parameters of this method specify the row and column of the beginning of the line, but you must compute the row and column of the end of the line. *start*[0] should be set to the row of the beginning of the line and *start*[1] should be set to its column; *end*[0] should be set to the row of the end of the line and *end*[1] should be set to its column.
- public int[] getStart()
  - Returns a new int[] with the same data as the *start* array. Note that you need to create and return a new array and copy in it the information stored in array *start*.
- public int length()
  - Returns the length of the line (a positive number) which indicates the number of entries in the line (this value should be equal to the value of the argument length from the constructor)
- public boolean isHorizontal()
  - Returns true if the end and start of this line are on in the same row, i.e. if the line is horizontal
  - Returns false, otherwise
- public boolean inLine(int *row*, int *col*)
  - Returns true if the position of the grid at the row and column indicated by the parameters is contained in this Line; explicitly: is *row* and *col* on or between the start and end of the line? For example, consider a line that starts at row 2 of the grid and column 1 and it ends at row 2 and column 5; then the invocation inLine(2,4) must return the value true as position (2,4) of the grid is part of the line, however the invocation inLine(3,2) must return the value false.
  - Returns false, otherwise
- public String toString()
  - Returns a String with the endpoints, that **exactly** matches the format of given below:
  - (new Line(1,1,true,2)).toString() returns `"Line:[1,1]->[1,2]"`

LetterCrush.java

This class is used to represent a tile grid board.

The class must have the following private instance variable:

- private char[][] *grid*;

  - Stores the current arrangement of characters in the grid

Also, this class must contain the following constant:

- public static final char EMPTY
  - This constant is set to a space (`' '`)

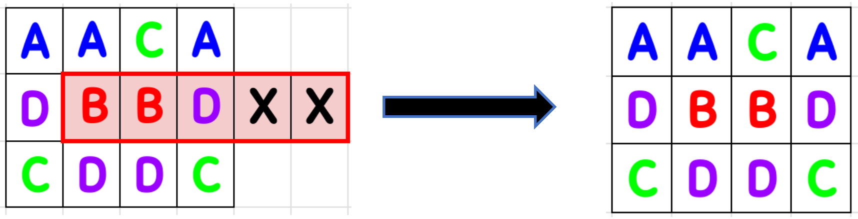
The class must have the following public methods:

- public LetterCrush(int *width*, int *height*, String *initial*) [constructor]
  - The parameters *width* and *height* determine the grid's dimensions, and the parameter *initial* is a String of capital letters used to initialize the two-dimensional array *grid*.
  - Specifically, the number of rows in *grid* is given by the second parameter, and the number of columns is given by the first parameter.
  - The first "width" characters of the String *initial* must be stored in the first row of the grid, the second "width" characters of *initial* must be stored in the second row of the grid, etc. If there are fewer characters in the String *initial* than the number of entries in the grid, the remaining grid entries are set to EMPTY. If there are more characters in *initial* than the number of entries in the grid, the extra characters in *initial* are ignored. For example, if *width* = 4, *height* = 3, and *initial* = "ABAABBAABA", then the first row of *grid* will store the characters A, B, A, and A; the second row of *grid* will store B, B, A, and A, and the last row of *grid* will store B, A, and two spaces.
- public String toString()
  - Returns a String representing the characters stored in *grid* in a format that is easy to read, that **exactly** matches the format given below.
  - (new LetterCrush(4,3,"AACADBBDCD")).toString() must return

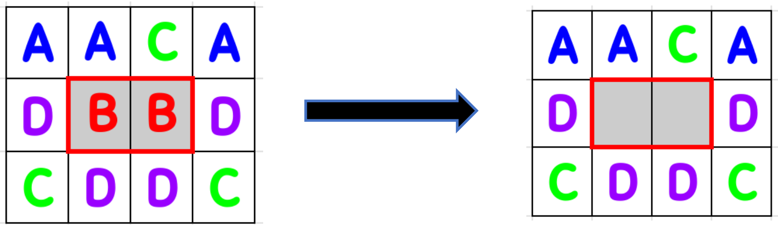
    ```
    "LetterCrush
    |AACA|0
    |DBBD|1
    |CD  |2
    +0123+"
    ```
  - Note the row number at the end of the 2nd to 4th lines, and the column numbers in the last row. The column indices will always be one-digit values (grids with widths greater than nine will not be tested).
- public boolean isStable()
  - Returns false if there is any position in *grid* where a non-EMPTY character is above an EMPTY character (a *grid* entry with a smaller row than a *grid* entry with a larger row in the same column —thus, for the example given above in the description of the toString() method, isStable() would return false).
  - Otherwise, returns true.
- public void applyGravity()
  - This method simulates a simple step for gravity pulling tiles down by replacing each EMPTY cell in *grid* with the one above it (and setting the one above to

EMPTY). It starts at the bottom row of *grid* and processes up to the top. Note: this replacement cannot happen for the top row of *grid* since there is not a row above the top row.

- public boolean remove(Line *theLine*)
  - Returns false if *theLine* is not a valid line, i.e. if the line's start or end points are not within the grid. For example, if grid has 3 rows and 4 columns, *theLine* starts at row 1, column 1 and it ends at row 1, column 5 then method remove must return false as part of *theLine* is not within the array grid. Recall that in a two-dimensional array the first row has index 0 and the first column also has index 0.



  - If *theLine* is valid, this method replaces with EMPTY the entries in *grid* corresponding to *theLine*. And then it returns true. For example, if *grid* has 3 rows and 4 columns and *theLine* starts at row 1, column 1, and it ends at row 1, column 2, then the entries of *grid* in row 1, columns 1 and 2 are replaced with EMPTY and the method returns true.



- public String toString(Line *theLine*)
  - Returns a String similar to the String returned by the above toString() method (note that the above toString method does not have any parameters) except that any letter corresponding to a *grid* location within *theLine* will be shown as a lowercase letter (hint: you can use Character.toLowerCase(ch) to convert char ch to lowercase). The String returned by this method must **exactly** match the format given below:
  - For example, (new LetterCrush(4,3,"AACADBBDCD")).toString(new Line(0,0,false,2)) returns

```
"CrushLine
|aACA|0
|dBBD|1
|CD  |2
+0123+"
```

- **public Line longestLine()**
  - Returns null if there does not exist in *grid* a line of at least 3 adjacent matching letters.
  - Otherwise, returns a *uniquely defined* longest line in *grid* that contains adjacent matching letters. To make this method unambiguous, preferences must be specified to select among equally long lines of adjacent matching letters. Use these preferences for this method:
    - A horizontal line has preference over a vertical one of the same length.
    - If two horizontal lines have the longest length, then we give preference to the line appearing in the lowest row (largest index); if both lines are in the same row, we prefer the line that appears to the left.
    - If two vertical lines have the longest length, we give preference to the line that appears in the leftmost column; if both lines appear in the same column, we prefer the line the appears at the bottom.
  - For example, if there are three lines with the maximum length and none of them are horizontal, and among these lines two of them are in the same column but to the left of the third line, then the line in the left column that appears in the lowest row will be the one that is returned.
  - See the pseudocode below for details.
- public void cascade()
  - This method iteratively removes the longest line of adjacent matching letters, of length at least 3 and repeatedly invokes the applyGravity method to shift letters from *grid* to the empty spaces until no more shifts are possible (you can check this by invoking method isStable). The process is repeated until the longest line of adjacent matching letters has a length less than three.

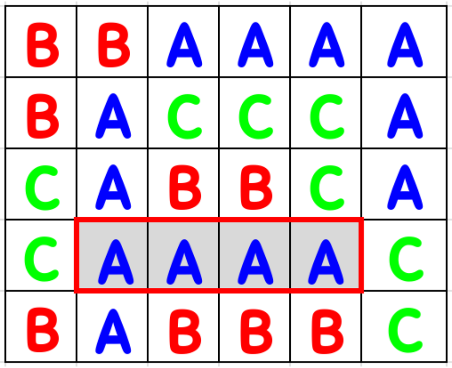
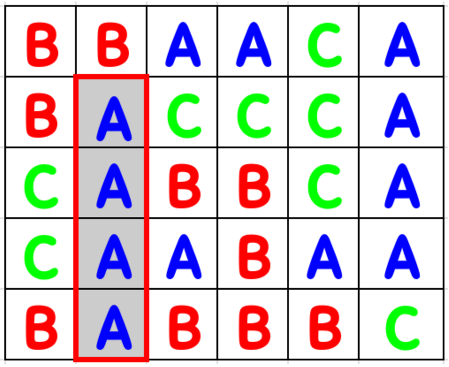
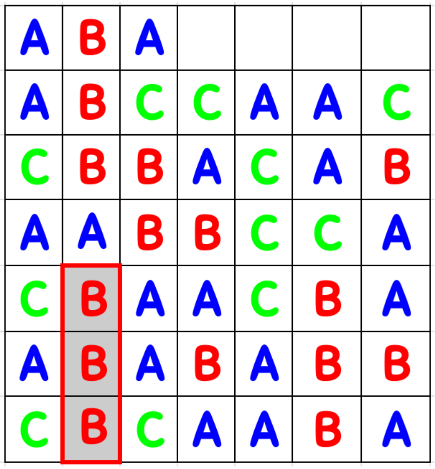
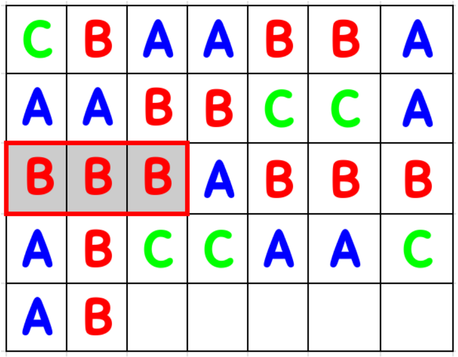
Pseudocode for Algorithm longestLine()

```
Algorithm longestLine()
// Scan first the rows of the grid from bottom to top
Create an object of the class Line representing a horizontal line starting at the first row and first column
of grid and of length 1. Store the address of this object in a variable called longLine.
largest = 0
for each row i of the grid starting at the bottom and moving to the top do {
    letter = letter at row i and leftmost column of the grid
    adjacent = 1
    for each column j of the grid starting at the second one and moving to the right do {
        if letter at row i and column j of the grid is equal to letter and letter ≠ blank space then {
            increase adjacent
            if adjacent is bigger than largest then {
                largest = adjacent
                longLine = new object of the class Line representing a horizontal line
                starting at row i and column j – adjacent + 1    (Why this column?)
                of length adjacent
            }
        }
        else {
            letter = letter at row i and column j of the grid
            adjacent = 1
        }
}
// Now scan the columns from left to right; each column is scanned from the bottom to the top
for each column j of the grid starting at the leftmost one and moving to the right do {
    letter = letter at the bottom row of the grid and column j
    adjacent = 1
    for each row i of the grid from the second row from the bottom and moving to the top do {
        if letter at row i and column j of the grid is equal to letter and letter ≠ blank space then {
            increase adjacent
            if adjacent is bigger than largest then {
                largest = adjacent
                longline = new object of the class Line representing a vertical line
                starting at row i and column j of length adjacent
            }
        }
        else {
            letter = letter at row i and column j of the grid
            adjacent = 1
        }
}
if the length of longline is larger than 2 then return longline else return null
```

Assignment 1

CS 1027
Computer Science Fundamentals II

The following figures show examples of the longest lines that the algorithm must select.

Marking

Functional Specifications

- Does the program behave according to specifications?
- Does it produce the correct output and pass all tests?
- Are the classes implemented properly?
- Does the code run properly on Gradescope (even if it runs on Eclipse, it is **up to you** to ensure it works on Gradescope to get the test marks)
- Does the program produce compilation or run-time errors on Gradescope?
- Does the program fail to follow the instructions (i.e. changing variable types, etc.)

Non-Functional Specifications

- Are there comments throughout the code (Javadocs or other comments)?
- Are the variables and methods given appropriate, meaningful names?
- Is the code clean and readable with proper indenting?

- Is the code consistent regarding formatting and naming conventions?
- Submission errors (i.e. missing files, too many files, etc.) will receive a penalty of 5%
- **Including a "package" line at the top of a file will receive a penalty of 5%**

Remember you must do all the work on your own. Do not copy or even look at the work of another student. All submitted code will be run through similarity-detection software.

Submission (due Tuesday, February 6 at 11:55 pm)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

Rules

- Please only submit the files specified below.
- Do not attach other files that were given to you as part of the assignment.
- **Do not only upload the .class files! Penalties will be applied for this.**
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope. **If your code runs on Eclipse but not on Gradescope, you will NOT get the marks! Make sure it works on Gradescope to get these marks.**
- You are expected to perform additional testing (create your own test harness class to do this). We are providing you with some tests but we may use additional tests that you haven't seen before for marking.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular assignment deadline will receive a penalty.

Files to submit

- Line.java
- LetterCrush.java

Grading Criteria

Total Marks: [20]

Assignment 1

CS 1027
Computer Science Fundamentals II

Functional Specifications:

[1] Line.java

[3] LetterCrush.java

[13] Passing Tests (some additional, hidden tests will be run on Gradescope)

Non-Functional Specifications:

[1] Meaningful variable names, private instance variables

[1] Code readability and indentation

[1] Code comments