

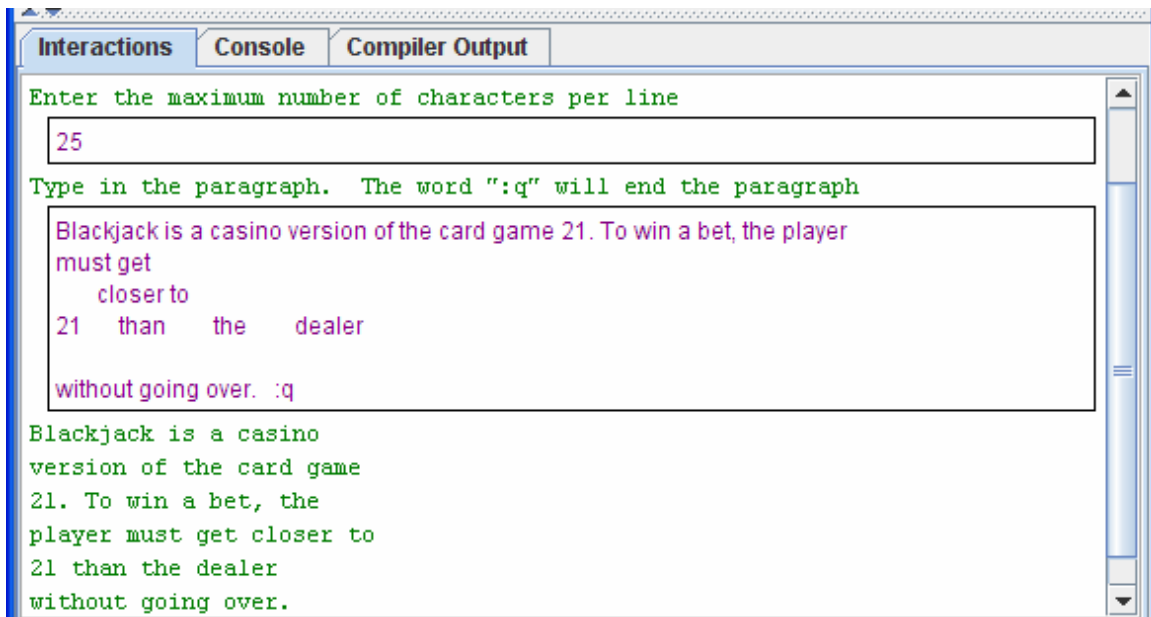
Question 1: Pretty Printer (20%)

Write a program called `PrettyPrinter.java` which will nicely format a paragraph of text that the user enters.

Regardless of how the user enters the input, each word the user types should appear separated by only a single space character. As well, no line of output should be longer than a specified amount. If the next word would exceed this amount, it is moved to the start of the next line.

At the beginning of the program, the user inputs the line length, and then begins to enter the paragraph. The user specifies the end of the paragraph by the special 2 character word “:q”. This word should not actually appear in the output.

Here is a sample run of the program in Dr. Java:



```
Interactions Console Compiler Output
Enter the maximum number of characters per line
25
Type in the paragraph. The word ":q" will end the paragraph
Blackjack is a casino version of the card game 21. To win a bet, the player
must get
    closer to
21 than the dealer
without going over. :q
Blackjack is a casino
version of the card game
21. To win a bet, the
player must get closer to
21 than the dealer
without going over.
```

You may assume that the user never enters a word whose length is greater than the maximum line length

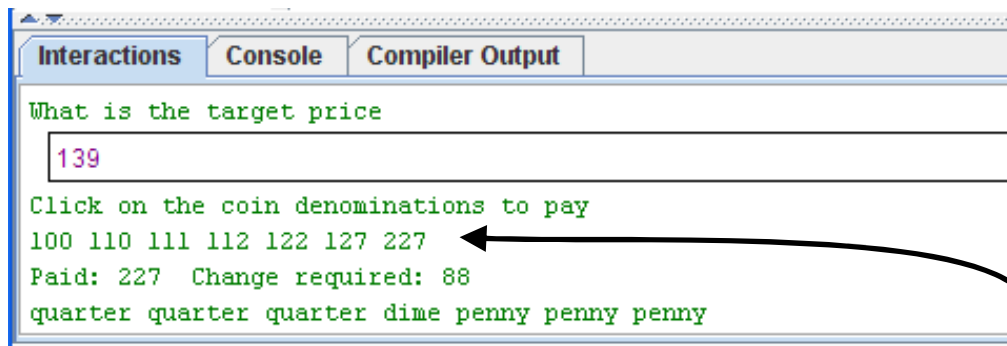
Files to Submit: `PrettyPrinter.java`

Question 2: Vending Machine (25%)

Write a program called `CoinCounter.java` that will create a 1-D board with five squares representing the five smallest coins: **penny** (1 cent), **nickel** (5 cents), **dime** (10 cents), **quarter** (25 cents) and **loonie** (100 cents). The program will then read in a target price, and the user will repeatedly click on the squares for the coins (to simulate coins going into a vending machine). As each coin is selected, your program prints the total amount entered so far.

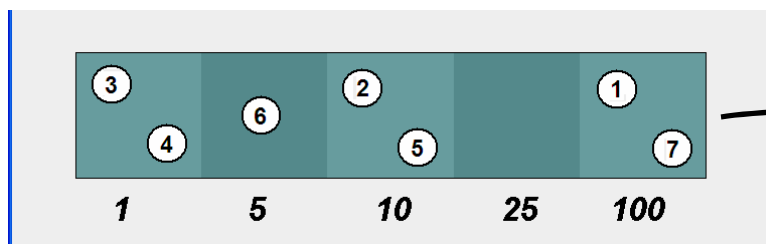
When the user has entered enough money for the purchase, the machine will proceed to give change back to the user. When giving change the machine selects as many of the largest coin denomination, then as many of the next largest denomination and so forth. As each coin is given back, your program prints it to the screen.

Here is a sample run of the program. Your program should replicate the output exactly.



```
Interactions Console Compiler Output
What is the target price
139
Click on the coin denominations to pay
100 110 111 112 122 127 227
Paid: 227 Change required: 88
quarter quarter quarter dime penny penny penny
```

Here is the order and location that the user clicked on the board:



You may assume the user always enters a non-negative starting price.

Files to Submit: `CoinCounter.java`

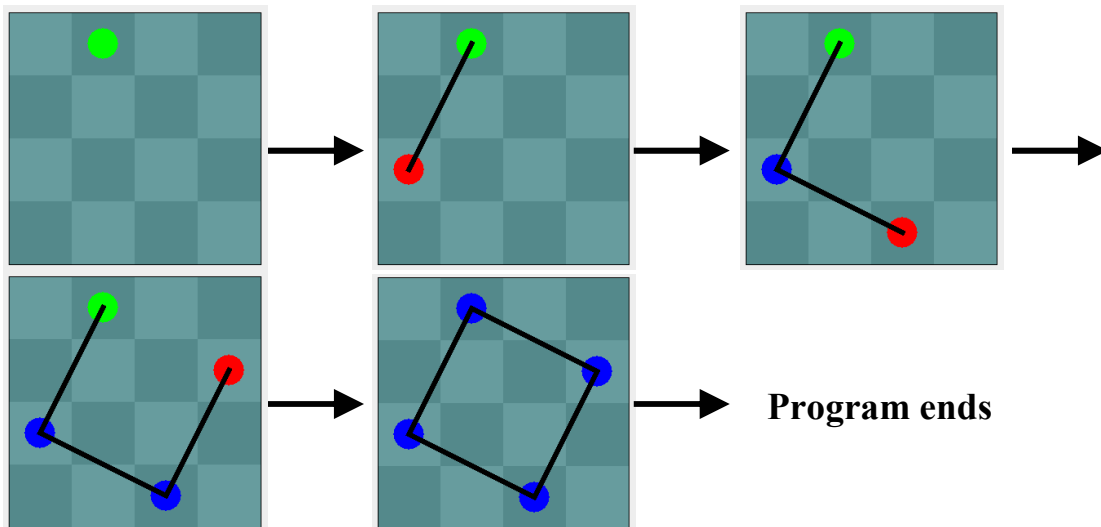
Question 3: Line Art (25%)

Write a program called `LineDraw.java` which creates an 8x8 board and allows the user to repeatedly click on squares on the board. The program will draw a line segment between the previous pair of squares. The final line segment drawn occurs if the user clicks back on the starting square.

The program will also draw pegs in the following manner:

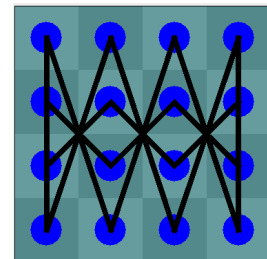
- The first square is colored green (so the user knows where the start is)
- The most recent square is colored red (so the user knows where they are)
- All other squares clicked will have a blue peg.
- The only exception is when the last line segment is placed, all pegs should be blue.

Here is a sample run on a 4x4 board (note your board should be 8x8).



Note, there is no limit to the number of squares in the shape, and the user can click on squares multiple times.

Files to Submit: `LineDraw.java`



Question 4: Treasure Huntin' Robot (30%)

Arrgh! There be treasure in these parts, and we are going to send a robot to try and recover it. Write a program called `Treasure.java` which creates a board representing the map.

Your program will read in the following information:

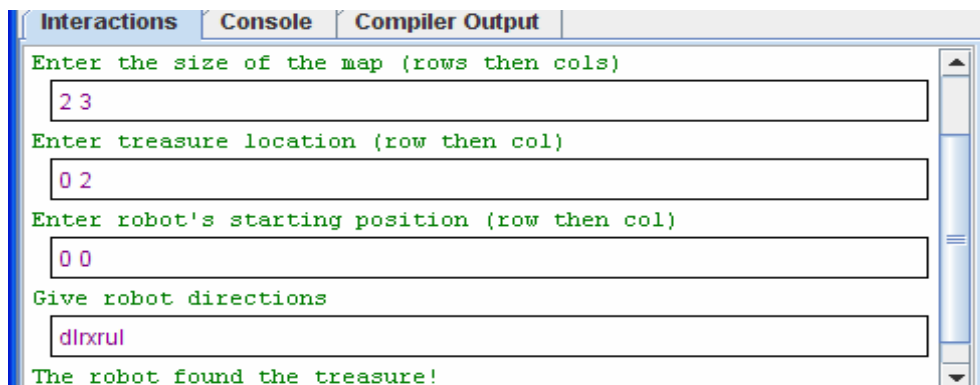
- The width and height of the map
- The row and column for the treasure
- The row and column for the starting position of the robot
- A string containing directions for the robot to follow

The program then draws the map, and places the treasure (a yellow peg) and the robot (a green peg). At this point each time the user clicks anywhere on the board, the robot will make one step according to the directions.

The directions are simply a series of letters indicating that the robot should travel (**u**)p, (**d**)own, (**l**)eft, or (**r**)ight. The peg for the robot should be **moved** one step in that direction. If the direction would move the robot off the board, or if the next direction is something other than one of the four lower-case letters given above, the robot stays in the current square (although it turns red in anger for that turn).

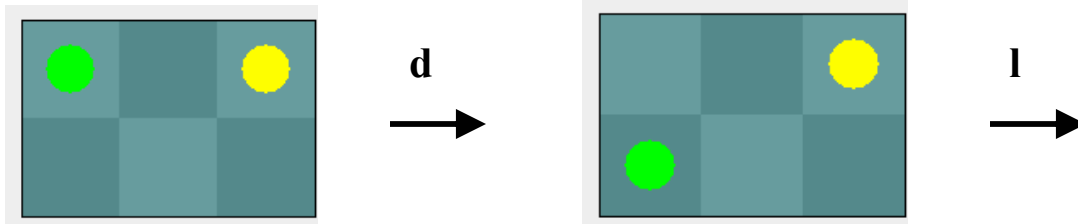
When the robot reaches the end of the directions the program prints a message indicating if the robot found the treasure at some point in the journey and terminates.

Here is a sample run of the program:

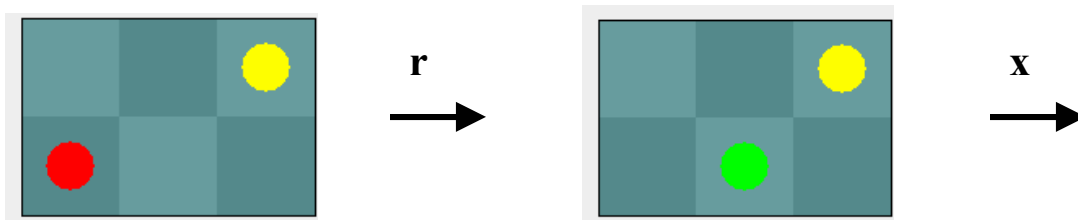


```
Interactions | Console | Compiler Output
Enter the size of the map (rows then cols)
2 3
Enter treasure location (row then col)
0 2
Enter robot's starting position (row then col)
0 0
Give robot directions
dlrxrul
The robot found the treasure!
```

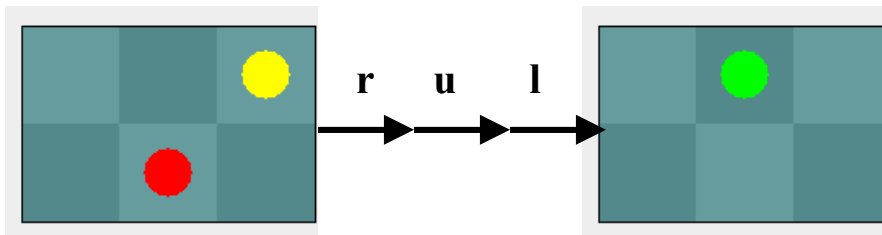
Here is the initial position:



After clicking on the board once, the robot follows the instruction for “d”. After another click, the robot tries to follow the instruction for “l”, but cannot so it turns red.



After another click the robot follows the instruction for “r”. After another click the robot does not understand the instruction for “x” and stays put.



After three more clicks the robot follows the last three instructions and the board is as above right. Note the treasure was picked up.

You may assume the size of the map is valid, and the positions of the treasure and robot exist on the board. If the robot does not find the treasure, the final output message should be “*The robot did not find the treasure.*”

Files to Submit: `Treasure.java`