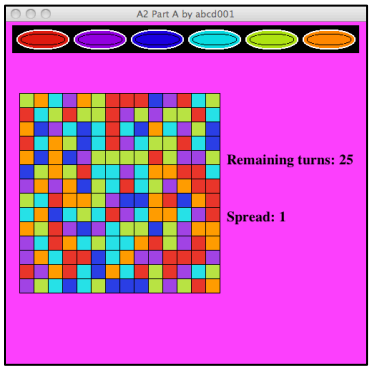
CompSci 105 S2 C
- ASSIGNMENT TWO -

The work done on this assignment must be your own work. Think carefully about any problems you come across, and try to solve them yourself before you ask anyone else for help. Under no circumstances should you use code written by another person in your assignment solution.

Assignment Two

Due 4.30pm , Thursday 27th September, 2012
Worth 8% of your final mark



Aim of the assignment

- practice with recursion
- practice with 2D arrays
- practice writing to a text file
- practice with Stacks and Queues

There are two parts for this assignment. Each part is worth 50% of the assignment.

For part A you need to submit SIX Java source files:

- `A2PartA.java`
- `A2Constants.java`
- `MyColourButton.java`
- `A2JPanel.java`
- `Cell.java`
- `BlockOfCells.java`

For part B you need to submit the Java source file, `Calculator.java`. Please do not change any of the other classes in the resource files for part B.

You also need to submit your feedback on the assignment (`A2.txt` - see later in this document)

NOTE: For part A of this assignment you are required to make one change to the `A2PartA` class, to complete the `BlockOfCells` class and to complete the `storeGame()` method in the `JPanel` class. The other classes are complete.

Part A The `ColourMe` Game (50 marks)

Task 1 (3 Marks)

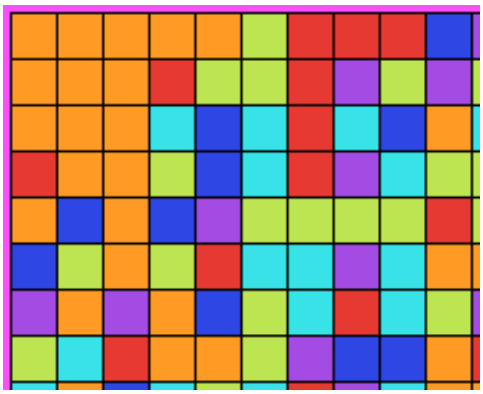
Your UPI should be displayed inside the title bar of the window. To do this, open the `A2PartA.java` application file and insert your UPI, e.g.,

```
JFrame colourMe = new A2JFrame("A2 Part A by abcd001", ... );
```

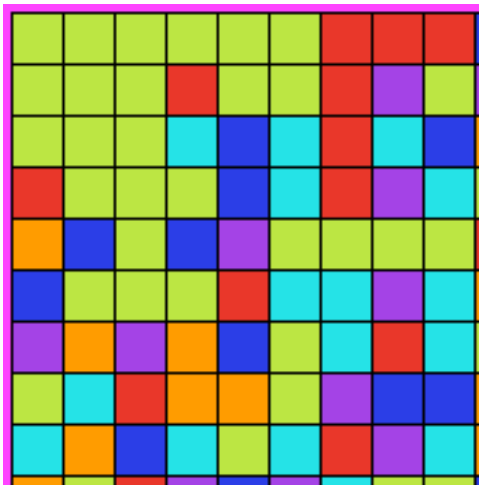
Task 2. The `ColourMe` game (37 Marks)

The aim of this game is to fill the whole game board with the same colour in less than 25 moves. The board is made up of six colour buttons at the top of the `JPanel` and a grid of cells and, initially, each cell is assigned a random colour (there are 6 possible colours). Each cell has **4** adjacent 'connected' neighbours (above, to the right, below, to the left) except for the border cells. The user selects a colour by pressing one of the colour buttons and, starting from the top-left cell, all the cells adjacent to the top-left cell with the same colour as the top-left cell, change to the new colour selected by user.

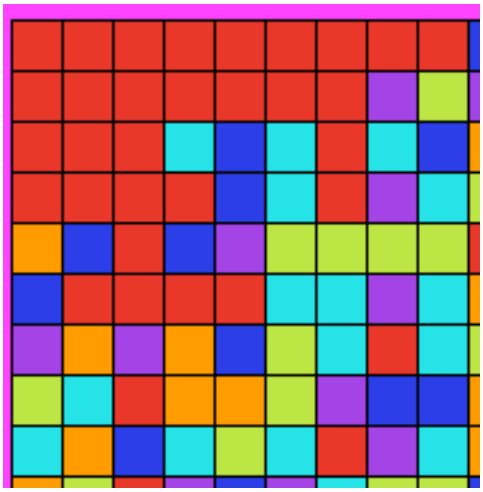
Slowly by selecting different colours the user changes the whole board to one single colour. For example, given the following section of the `ColourMe` board:



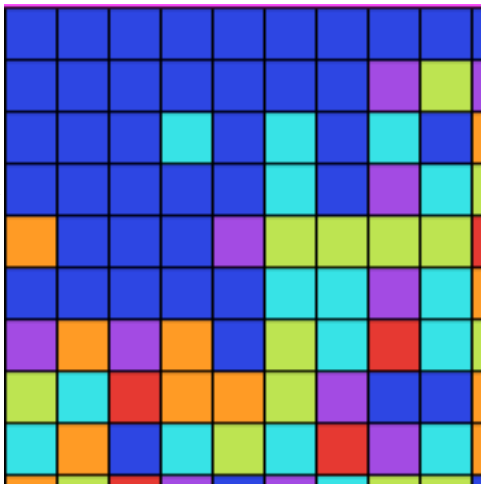
If the user now presses the pale green button all the **touching** orange cells change to pale green.



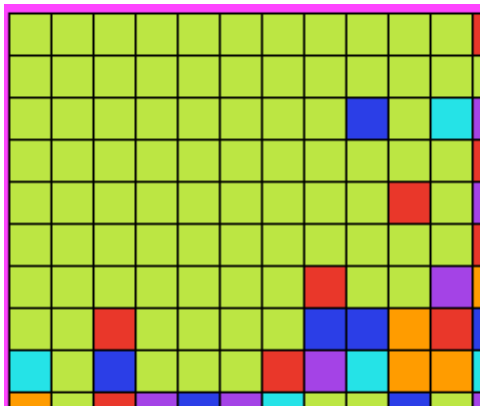
If the user now presses the red button all the **touching** pale green cells change to red.



If the user now presses the blue button all the **touching** red cells change to blue.



and so on until the board becomes more and more filled with the same colour:



Task 2 (37 Marks)

Task 2 has been broken up into five stages. The classes in this program have been completed except for the `BlockOfCells` class (and part of the `JPanel` class – see Task 3) which you need to complete. The `BlockOfCells` class represents the whole grid of coloured cells and handles all the actions to do with managing the individual cells. The skeleton of the `BlockOfCells` class is shown on the next page:

```
public class BlockOfCells {
	public static final int NUMBER_OF_ROWS = ...
	public static final int NUMBER_OF_COLS = ...
	public static final int CELL_SIZE = A2Constants.CELL_SIZE;
	public static final Rectangle GAME_AREA = ...

	private Cell[][] cellBlock;

	public BlockOfCells() {    ... }
		cellBlock = new Cell[NUMBER_OF_ROWS][NUMBER_OF_COLS];
		createTheCells(cellBlock);
	}

	private void createTheCells(Cell[][] cellBlock) {
		int x = GAME_AREA.x;
		int y = GAME_AREA.y;

		for(int i = 0; i < cellBlock.length; i++) {
			x = GAME_AREA.x;
			for(int j = 0; j < cellBlock.length; j++) {
				cellBlock[i][j] = new Cell(x, y);
				x = x + CELL_SIZE;
			}

			y = y + CELL_SIZE;
		}
	}

//---------------------------------------------------------
// Stage 2 (4 marks) Methods to do with the colour index of
// each cell
//---------------------------------------------------------
	public int getCellColourIndex(int row, int col) { ... }
	public void setCellColourIndex(int row, int col, int colourIndex) { ...
}

//---------------------------------------------------------
// Stage 3 (4 marks) Reset hasBeenVisited for all cells
//---------------------------------------------------------
	private void resetAllCellHasBeenVisited() {    ... }

//---------------------------------------------------------
// Stage 5  (10 marks) Recursive method which returns the number of
// cells connected (i.e., with the same colour index as the
// top left cell (0, 0))
//---------------------------------------------------------
	public int getNumberOfConnectedCells() {
		int colourIndex = getCellColourIndex(0, 0);
		resetAllCellHasBeenVisited();
		return getNumberOfCellsInUserBlock(0, 0, colourIndex);
	}

	private int getNumberOfCellsInUserBlock( ... ) { ... }

//---------------------------------------------------------
// Stage 4 (10 marks) Recursively updates the colour of all cells
// connected (i.e., with same colour index as the top left
// cell (position 0, 0))
//---------------------------------------------------------
	public void updateConnectedCells(int userColourIndex) {
		int colourToChangeIndex = getCellColourIndex(0, 0);
		resetCellHasBeenVisited();
		updateUserAreaColours(0, 0, getCellColourIndex,
                                                colourToChangeIndex);
	}

	private void updateUserAreaColours( ... ) { ... }

//---------------------------------------------------------
// Stage 6 (4 marks) returns a String with all the colour
// indexes of the cells concatenated, row by row
//---------------------------------------------------------
	public String colourIndexesToString() { ... }
```

```
//-------------------------------------------------------
// Stage 1 (5 marks) Draw the 2D array of coloured cells
//-------------------------------------------------------
      public void drawCells(Graphics g) { ... }
}
```

Notes on Task 2

Stage 1 `drawCells`() – 5 marks
The `BlockOfCells` class has one instance variable:

```
private Cell[][] cellBlock;
```

which stores the grid of `Cell` objects. This method draws all the `Cell` objects in the grid.

Once you have completed stage 1, the user should see all the coloured cells. The cells are randomly assigned colours. Please note that the user can play a new game by pressing the 'n' (or 'N') key hence you can see a new randomly coloured grid by pressing the 'n' key.

Stages 2 and 3 see the skeleton code above - 8 marks

Stage 4 `updateUserAreaColours()` – 10 marks
This instance method is called whenever the user chooses a new colour by pressing one of the Colour buttons. This method should update the colour of all the cells connected to the top left cell (0, 0) to the new colour selected by the user. Note that each cell has **4** adjacent (i.e., connected) neighbours (except for the border cells). Note that each cell stores the index of its fill colour (index of one of the colours in the COLOURS array). To complete this method you should use a recursive algorithm.

Once you have completed stage 4, the user should be able to fill the grid with colour by pressing the Colour buttons.

Stage 5 `getNumberOfCellsInUserBlock()` – 10 marks
This instance method is called after the colour in the cells has been changed (i.e., after the user has pressed one of the Colour buttons). This method returns the number of cells which have the same colour as the top left cell (0, 0) and are connected to the top left cell. Note that each cell has **4** adjacent (i.e., connected) neighbours (except for the border cells). Note that each cell stores the index of its colour (index of one of the colours in the COLOURS array). To complete this method you should use a recursive algorithm.

Once you have completed stage 5, you should see the current number of connected cells in the JPanel on the right of the grid of cells. This number should change depending on how many connected cells there are.

Stage 6 see the skeleton code above - 4 marks

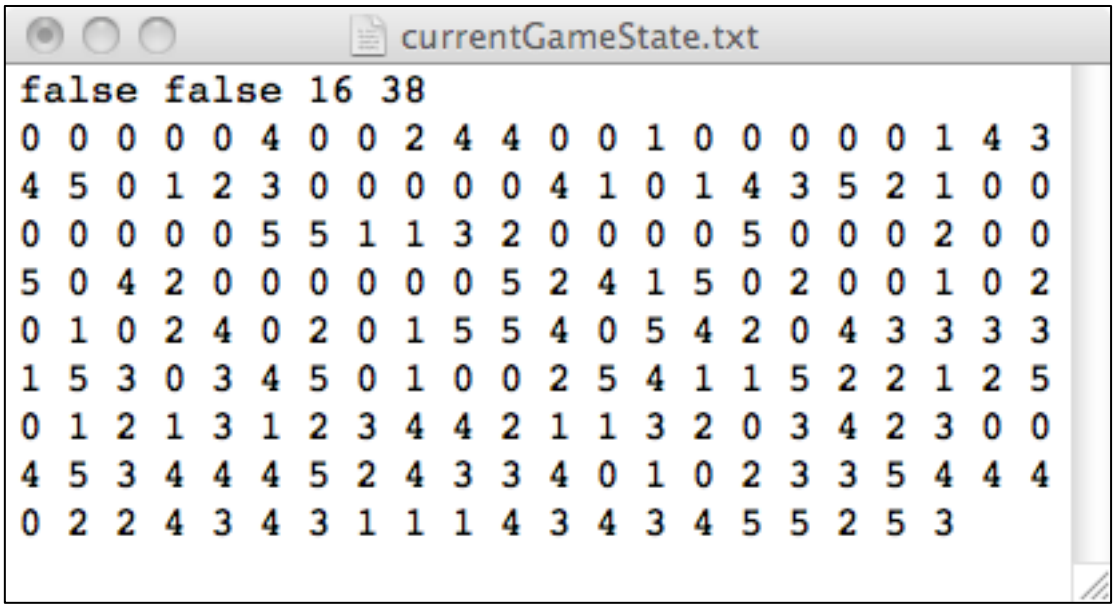
Task 3 (5 Marks)

In the `A2JPanel` class complete the `storeGame()` method. This method writes the current state of the `ColourMe` game to the file (the name of the file where the game is stored is passed as a parameter to the `storeGame()` method). The format of the file is as shown in the screenshot below.

The first line of the text file contains the values of the four instance variables:

```
      gameHasEnded (boolean),
      userHasWon (boolean),
      turnsRemaining (int)
and
      numberConnected (int)
```

A new line follows and then the indexes of all the colours of each cell (taken row by row) are stored in the file (the `colourIndexesToString()` method of the `BlockOfCells` class is useful here).



Please Note

Now that you have completed task 3, the user is able to (see the code in the `keyPressed()` method in the `JPanel` class):

save the current game by pressing the 's' (or 'S') key,
load the most recently saved game by pressing the 'l' (or 'L') key,
and
reload the current game from the beginning by pressing the 'r' (or 'R') key.

Note that the initial state of the current game is stored in the text file, "restartGame.txt", and the last saved game is stored in the text file, "currentGameState.txt".

You have now completed part A of this assignment.

Part B – A Simple Calculator Program (50 marks)

Design and implement a simple mathematical expression calculator using the Stack data structure. The mathematical expressions used in this assignment are made up of:

the operators '+', '-', '*' and '/',
parentheses '(' and ')',
and
operands which consist of integer numbers (single or multi-digit).

The mathematical expressions are in infix format, and the operators '*', '/' have higher precedence than the operators '+', '-'. The mathematical expressions contain no unary operators and no spaces.

For this question, you need to write a program called `Calculator`, which

1) checks the syntax of the expression, and converts the infix form into its equivalent postfix form (using spaces to separate the numbers and operators);
2) calculates the value of the mathematical expression (after it has been converted into postfix form) and prints the result. Your program must compute the value of the expression correctly.

The mathematical expression in infix form is passed to the program as a command line argument. Your program should print the infix expression, the postfix expression and the calculated value. You are required to use the stack data structure to solve this problem. Two example outputs using the `Calculator` program are shown below:

```
c:\105\A2>java Calculator (8+12)*3/(14-9)
A simple calculator by abdc001.
Evaluating ...
Infix Expression: (8+12)*3/(14-9)
Postfix Expression: 8 12 + 3 * 14 9 - /
Result: 12.0
```

Note that syntax errors in the user input expressions should be detected, e.g.,

```
c:\105\A2>java Calculator 5+(2-4/5*6
A simple calculator by abdc001.
Evaluating ...
Infix Expression: 5+(2-4/5*6
Syntax error: brackets mismatch -> too many open brackets
```

Hints

- Use a Stack structure to convert an infix expression to its postfix form, and then use a Stack structure to compute the value of the postfix expression (algorithms are taught in the lectures).
- In the case of checking syntax errors of an infix mathematical expression, there are five major types of errors that you should consider:
  - invalid inputs, e.g., non-digit or other characters. Note: a constant indicating the valid characters might be useful in your program, e.g.,
    ```
    final String ALL_VALID = "0123456789+-/*()";
    ```
  - brackets mismatch, e.g., too many open bracket, too many closing bracket;
  - missing operand, e.g., two consecutive operators, '(' followed by operator, operator followed by ')', expression end with '(', empty bracket '( )', expression begin with ')', expression begin with operators, expression end with operators;
  - missing operator, e.g., digit followed by '(', ')' followed by digit, ')' followed by '(';
  - division by zero, e.g., '/' followed by 0;

Notes For Part B

- Your program must use the `StackReferenceBased` class to solve the problem.
- For this question, you should submit the `Calculator.java` class.
  Your UPI must appear in the output, as shown in the example above.

Submission Instructions

You should submit all of the following files for this assignment through the web-based Assignment Dropbox.

REQUIRED FILES for part A of the assignment

For this part of the assignment you need to submit FOUR Java source files and ONE text file:

- `A2PartA.java`
- `A2Constants.java`
- `MyColourButton.java`
- `A2JPanel.java`
- `Cell.java`
- `BlockOfCells.java`

REQUIRED FILES for part 2 of the assignment

For this part of the assignment you need to submit ONE Java source files:

- `Calculator.java`

FEEDBACK FILE for the assignment

- `A2.txt` – a text file containing your feedback on the assignment (see below).

MAKING MORE THAN ONE SUBMISSION

You can make more than one submission - every submission that you make *replaces* your previous submission. **Submit ALL your files in every submission**. Only your very latest submission will be marked.

NEVER SUBMIT SOMEONE ELSE'S WORK:

- If you submit an assignment you are claiming that you did the work. Do not submit work done by others.
- Do not *under any circumstances* copy anyone else's work – this will be penalized heavily.
- Do not *under any circumstances* give a copy of your work to someone else.
- We uses copy detection tools on the files you submit. If you copy from someone else, or allow someone else to copy from you, this copying will be detected and disciplinary action will be taken.

What you should include in the A2.txt file

You **must** include a text file named A2.txt in your submission. There will be a 5 mark penalty for not doing so. This text file must contain the following information:

Your full name
Your login name and ID number
How much time did the assignment take overall?
What areas of the assignment did you find easy?
What areas of the assignment did you find difficult?
Any other comments you would like to make.

Marking Schedule for part A

Part A - Style Comment at the top of the BlockOfCells class with name, date and a comment. **(1)** Correct indentation. **(2)** Code is easy to read and understand. **(2)**		**/5**
Part A - Student's UPI is in the title bar of the window.		**/3**
Part A - Task 2 – the BlockOfCells class **Part A - Task 3** – the storeGame() method in the A2JPanel class		**/37** **/5**
Total for part 1		**/50**

Marking Schedule for part B

Part 2 - Style Appropriate comments, structures and variable names		**/5**
Part 2 - Task 1 – Check the syntax of the mathematical expression and, if the expression is syntactically correct, convert the infix form into its equivalent postfix form. **Part 2 - Task 2** – Compute the value of the postfix expression.		**/30** **/15**
Total for part 2		**/50**

Penalties

A2.txt information was not submitted - penalty of 5 marks