Recitation 7

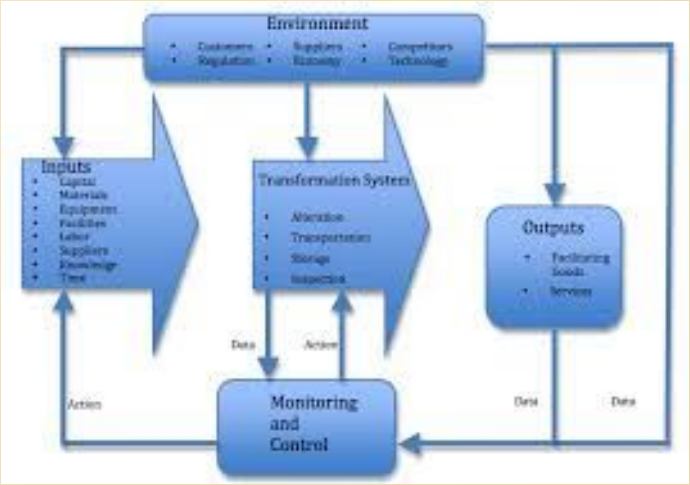
Standard IO

- STD libraries
  - In
  - StdOut
  - StdDraw
  - StdAudio
  - StdRandom

- The Color class
- PPM Format
- Image Processing

Recitation 7

Input/Output



Standard IO

- StdOut – standard output

public class StdOut	
void print(String s)	*print s*
void println(String s)	*print s, followed by newline*
void println()	*print a new line*
void printf(String f, ...)	*formatted print*

API for our library of static methods for standard output

- In – Input
  - In in = new In (); // reads from terminal
  - In in = new In (string filePath); // reads from txt file
  - in.readLine(); // reads the next line
  - in.readInt(); // reads the next int
  - in.readString(); // reads the next string

Standard IO

- StdDraw – standard drawing

public class StdDraw *(basic control commands)*	
void setXscale(double x0, double x1)	*reset x range to* (x_0, x_1)
void setYscale(double y0, double y1)	*reset y range to* (y_0, y_1)
void setPenRadius(double r)	*set pen radius to r*

Note: Methods with the same names but no arguments reset to default values.

public class StdDraw *(shapes)*
void circle(double x, double y, double r)
void filledCircle(double x, double y, double r)
void square(double x, double y, double r)
void filledSquare(double x, double y, double r)
void polygon(double[] x, double[] y)
void filledPolygon(double[] x, double[] y)

Question 1 – printAll numbers + Solution

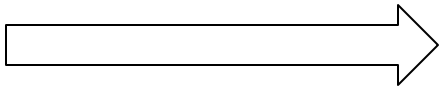
- The function in.isEmpty() checks if there are any more unread characters in the file.

```
public static void main(String[] args) {
      In in = new In(”./data.txt");
      while (!in.isEmpty()){
            StdOut.println(in.readInt());
      }
}
```

- Note: if a file with the name “data.txt” does not exist in the same folder as the program, an exception will be thrown.

data.txt

```
4
2 5
1 9
7
```

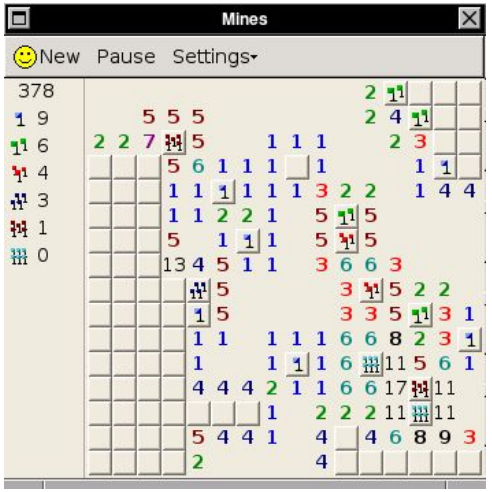


output

```
4
2
5
1
9
7
```

Question 2 – Mine Sweeper

- In the popular game Mine Sweeper, a player has to guess where mines are located in a board.
- If a square in the board does not contain a mine, then it lists the number of mines in all adjacent squares.
- We will build a program to design a Mine Sweeper board.
- Design a program which does the following:
  - Receives three command line arguments from the user:
    - Two integers N and M which will be the size of the board.
    - A double p, which will be the probability that a square contains a mine.
  - Prints the completed board, using '*' to denote a mine.



Question 2 - Mine Sweeper

```
% java MineSweeper 5 5 0.4
     2 * * 2 *
     * 5 5 5 3
     3 * * * *
     3 * * * 3
     * 3 3 2 1

% java MineSweeper 3 6 0.2
     * 1 1 * * 1
     2 2 2 2 3 2
     1 * 1 0 1 *
```

Question 2 – Solution structure

```java
public class MineSweeper {
    /**
     * Initializes a Minesweeper board with randomly placed mines based on the given probability.
     *
     * @param N           Number of rows for the board (excluding padding).
     * @param M           Number of columns for the board (excluding padding).
     * @param probability Probability of a cell containing a mine (value between 0 and 1).
     * @return A 2D array representing the board, where 1 indicates a mine, and 0 indicates no mine.
     */
    public static int[][] initializeMines(int N, int M, double probability);
    /**
     * Calculates the number of adjacent mines for each cell in the board.
     *
     * @param N     Number of rows for the board (excluding padding).
     * @param M     Number of columns for the board (excluding padding).
     * @param mines A 2D array indicating the placement of mines.
     * @return A 2D array representing the board, where each cell contains the count of adjacent mines.
     */
     public static int[][] calculateBoard(int N, int M, int[][] mines);

    /**
     * Prints the Minesweeper board, displaying '*' for mines and numbers for adjacent mine counts.
     *
     * @param N     Number of rows for the board (excluding padding).
     * @param M     Number of columns for the board (excluding padding).
     * @param mines A 2D array indicating the placement of mines.
     * @param board A 2D array representing the board with adjacent mine counts.
     */
     public static void printBoard(int N, int M, int[][] mines, int[][] board);
}
```

Question 2 – Solution initializeMines

```java
/**
 * Initializes a Minesweeper board with randomly placed mines based on the given probability.
 *
 * @param N           Number of rows for the board (excluding padding).
 * @param M           Number of columns for the board (excluding padding).
 * @param probability Probability of a cell containing a mine (value between 0 and 1).
 * @return A 2D array representing the board, where 1 indicates a mine, and 0 indicates no mine.
 */
public static int[][] initializeMines(int N, int M, double probability) {
    int[][] mines = new int[N + 2][M + 2]; // adds padding for mines calculation
    for (int i = 1; i < N + 1; i++) {
        for (int j = 1; j < M + 1; j++) {
            if (Math.random() < probability) {
                mines[i][j] = 1;
            }
        }
    }
    return mines;
}
```

Question 2 – Solution calculateBoard

```java
/**
 * Calculates the number of adjacent mines for each cell in the board.
 *
 * @param N     Number of rows for the board (excluding padding).
 * @param M     Number of columns for the board (excluding padding).
 * @param mines A 2D array indicating the placement of mines.
 * @return A 2D array representing the board, where each cell contains the count of adjacent mines.
 */
public static int[][] calculateBoard(int N, int M, int[][] mines) {
    int[][] board = new int[N + 2][M + 2];
    for (int i = 1; i < N + 1; i++) {
        for (int j = 1; j < M + 1; j++) {
            board[i][j] = mines[i - 1][j] + mines[i - 1][j - 1] +
                          mines[i - 1][j + 1] + mines[i][j - 1] +
                          mines[i][j + 1] + mines[i + 1][j] +
                          mines[i + 1][j + 1] + mines[i + 1][j - 1];
        }
    }
    return board;
}
```

Question 2 – Solution printBoard

```java
/**
 * Prints the Minesweeper board, displaying '*' for mines and numbers for adjacent mine counts.
 *
 * @param N     Number of rows for the board (excluding padding).
 * @param M     Number of columns for the board (excluding padding).
 * @param mines A 2D array indicating the placement of mines.
 * @param board A 2D array representing the board with adjacent mines counts.
 */
public static void printBoard(int N, int M, int[][] mines, int[][] board) {
    for (int i = 1; i < N + 1; i++) {
        for (int j = 1; j < M + 1; j++) {
            String cur = mines[i][j] == 0 ? board[i][j] + "" : "*";
            System.out.print(cur + " ");
        }
        System.out.println();
    }
}
```

Question 2 – Solution Play + main

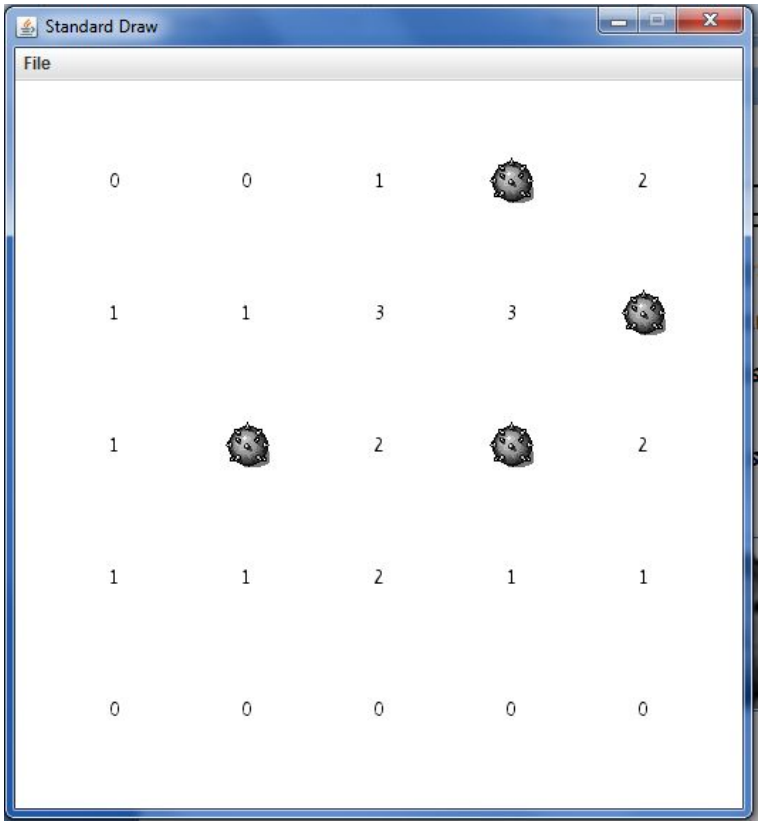
```
public class MineSweeper {

    public static void main(String[] args) {
      play();
    }
    public static void play() {
        // Create an In instance to read from standard input
        In in = new In();

        // Take input for number of rows (N)
        System.out.println("Enter the number of rows (N): ");
        int N = in.readInt();

        // Take input for number of columns (M)
        System.out.println("Enter the number of columns (M): ");
        int M = in.readInt();

        // Take input for mine probability
        System.out.println("Enter the probability of a mine (0 to 1): ");
        double probability = in.readDouble();

        // Initialize the game
        int[][] mines = initializeMines(N, M, probability);
        int[][] board = calculateBoard(N, M, mines);

        // Print the game board
        printBoard(N, M, mines, board);
    }
}
```

Question 2, Expansion 1– MineSweeper.

- Now that we know how to generate a board, we can make it more presentable.
- Design a program which does the following:
  - Receives N and M (the size of the board) as command line arguments.
  - Reads from the standard input a minesweeper board of the appropriate size (assume the board size is legal).
  - Prints the board using StdDraw. If a square contains a bomb, use the following icon named "mine.png".
    - StdDraw.picture(x, y , filename) draws the picture in filename centered at location (x, y)
    - So StdDraw.picture(0, 0, "mine.png") will put the mine icon centered at the bottom left corner.
    - StdDraw.text(x, y, text) prints the string text centered at location (x, y).

Question 2, Expansion 1– MineSweeper.

```
% java MineSweeper 5 5 0.2
```



Question 2, Expansion 1 – Solution

```
public class DrawMineSweeper {
      public static void main (String[] args) {
            int N = Integer.parseInt(args[0]);
            int M = Integer.parseInt(args[1]);
            //sets the scale of the drawing board.
            StdDraw.setXscale(0, N);
            StdDraw.setYscale(0, M);
            In in = new In();
            //at each cell put a number or a bomb
            for (int i = 0; i < N; i++) {
                  for (int j = 0; j < M; j++) {
                        String s = in.readString();
                              if (s.equals("*")) {
                                    StdDraw.picture(i + 0.5, j + 0.5,"mine.png");
                              } else {
                                    StdDraw.text(i + 0.5, j + 0.5, s);
                              }
                  }
            }
      }
}
```

Question 3 – Battleships

- Battleships is a classic strategy game that tests your tactical skills, memory, and intuition. Played between two opponents, the game revolves around a grid-based battlefield where players secretly position their fleets of ships. The objective is to locate and destroy the opponent's fleet before they destroy yours.
- Each player has their own, for placing their ships and tracking hits/misses on the opponent's grid. The fleet typically includes ships of varying sizes, such as a carrier, battleship, cruiser, submarine, and destroyer. Players take turns calling out coordinates (e.g., 0,1) to launch "attacks". If a ship occupies the attacked coordinate, it's a "hit", otherwise it's a "miss". Ships are sunk when all their corresponding segments are hit.The first player to sink all of their opponent's ships wins.

- Build your own single player Battleship game, the fleet contains ships in size 1X1 only. The game should start by selecting the grid size (N > 0) by input given by the user. Then it puts N battleships in N different locations.
- Then After initializing the game you are required it is now time to play. In every round, you are asked to select coordinates, if they are out of bounds, lets you know, and repeat, the same as found/not found a battleship once they are done. The game let you know

Question 3 - Battleships

```
% java Battleship
```

What size game would you like to play?
2
Enter your guess:
0 0
Nothing here, only water.

Enter your guess:
1 1
You sunk my battleship, good job.

Enter your guess:
3 0
Illegal coordinates, try again.

Enter your guess:
0 1
You sunk my battleship, good job.

VICTORY!!!!!!

Question 3 – Solution

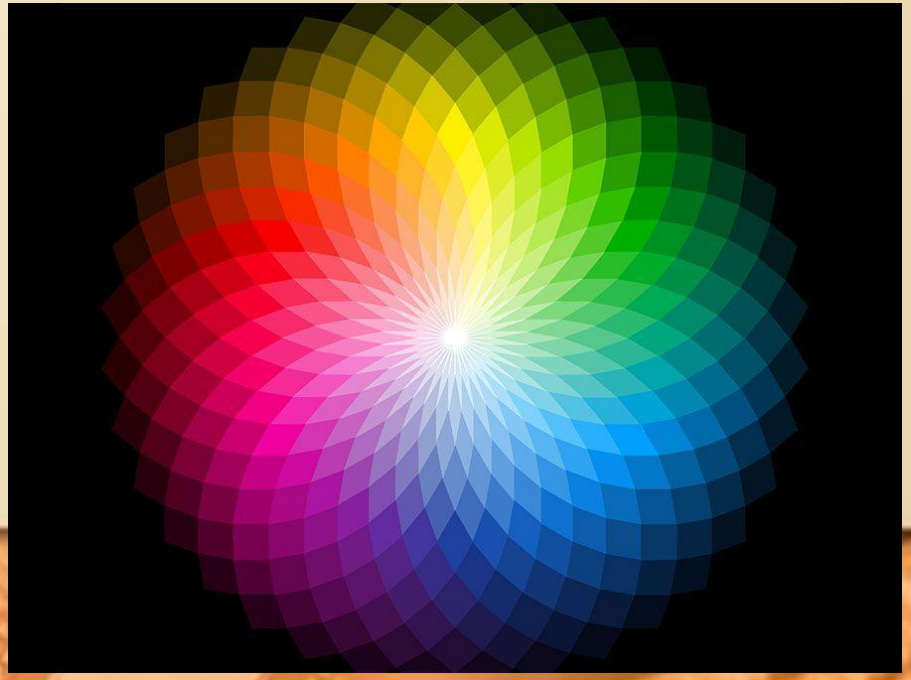
```
public class Battleships {
      public static void main(String[] args){
            StdOut.println("What size game would you like to play?");
            In in = new In(); // Input from Terminal
            int N = in.readInt();

            //create an N X N board
            boolean[][] subs = new boolean[N][N];

            //Choose N random locations for battleships.
            for (int i = 0; i < N; i++){
                  int x = (int) (Math.random() * N);
                  int y = (int) (Math.random() * N);

                  //if current value has battleship.
                  while(subs[x][y]){
                        x = (int) (Math.random() * N);
                        y = (int) (Math.random() * N);
                  }
                  subs[x][y] = true;
            }


            ...
```

Question 3 – Solution (Continued)

```
        // ...
        int userX = 0;
        int userY = 0;
        int sunk = 0;
        //Keep playing the game until the number of sunk battleship reaches N
        while (sunk < N){
              StdOut.println("Enter your guess:");
              userX = in.readInt();
              userY = in.readInt();
              //check if the coordinates are within the board
              if (userX >= N || userX < 0 || userY >= N || userY < 0){
              StdOut.println("Illegal coordinates, try again.");
              StdOut.println();
              continue; // moves to next iteration
              }
              //if a battleship is in the coordinates update the data
              if (subs[userX][userY]){
                    StdOut.println("You sunk my battleship, good job.");
                    sunk++;
                    subs[userX][userY] = false;
              } else {
                    StdOut.println("Nothing here, only water.");
              }
              StdOut.println();
        }
        StdOut.println("VICTORY!!!!!");
    }
}
```

Recitation 6

The Color Class



RGB Color System

- Every color noticeable to the human eyes can be decomposed into its red, green and blue component.
- The RGB system uses that fact to represent each color as 3 integers in the range [0,255]. The first integer corresponds to the intensity of red, the second corresponds to the intensity of green, and the third to blue.
- Examples:
  - {255, 0, 0} is Red.
  - {0, 255, 0} is Green.
  - {0, 0 ,255} is Blue.
  - {199, 67, 117} is Pink Fuchsia.

- An interesting note: this system is unique for humans. For example, the mantis shrimp can decompose colors into 16(!!) different base colors.

The Color class

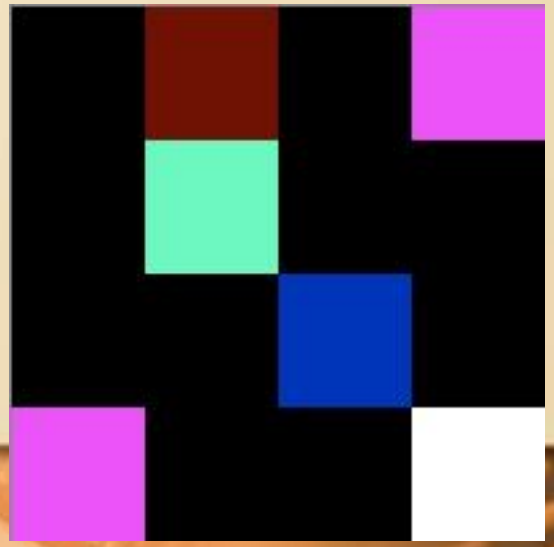
- While we can represent everything using 3-dimensional arrays, it can get cumbersome….
- Fortunately, Java has a Color data type which makes handling images more straightforward.
- Essentially a variable of type Color, is an array of 3 integers with values between 0 and 255. (Color is an advanced type)
- To use this variable in your program you need to add "import java.awt.Color;" as the first statement.
- Initialization is done exactly like arrays.
  - Color c = new Color(255, 0, 0);
- To check the r,g,b values we can use the functions getRed(), getGreen(), getBlue().
  - That is, c.getRed() will be 255.
- https://docs.oracle.com/javase/7/docs/api/java/awt/Color.html

Example 1 – Color

```
import java.awt.Color;

public class Colors {
      public static void main(String[] args) {
                  Color red = new Color(255, 0, 0);
                  Color green = new Color(0, 255, 0);
                  Color blue = new Color(0, 0, 255);
                  Color beccaPurple = new Color(102, 51, 153);
                  System.out.println(red.getRed());     // 255
                  System.out.println(red.getGreen());   // 0
                  System.out.println(green.getGreen()); // 255

                  Color[][] array = new Color[2][2];
                  array[0][0] = red;
                  array[0][1] = blue;
                  array[1][0] = beccaPurple;
                  array[1][1] = new Color(100, 100, 100);

                  System.out.println(array[1][1].getGreen()); // 100

       }

}
```

Recitation 6

PPM Format & Image Processing

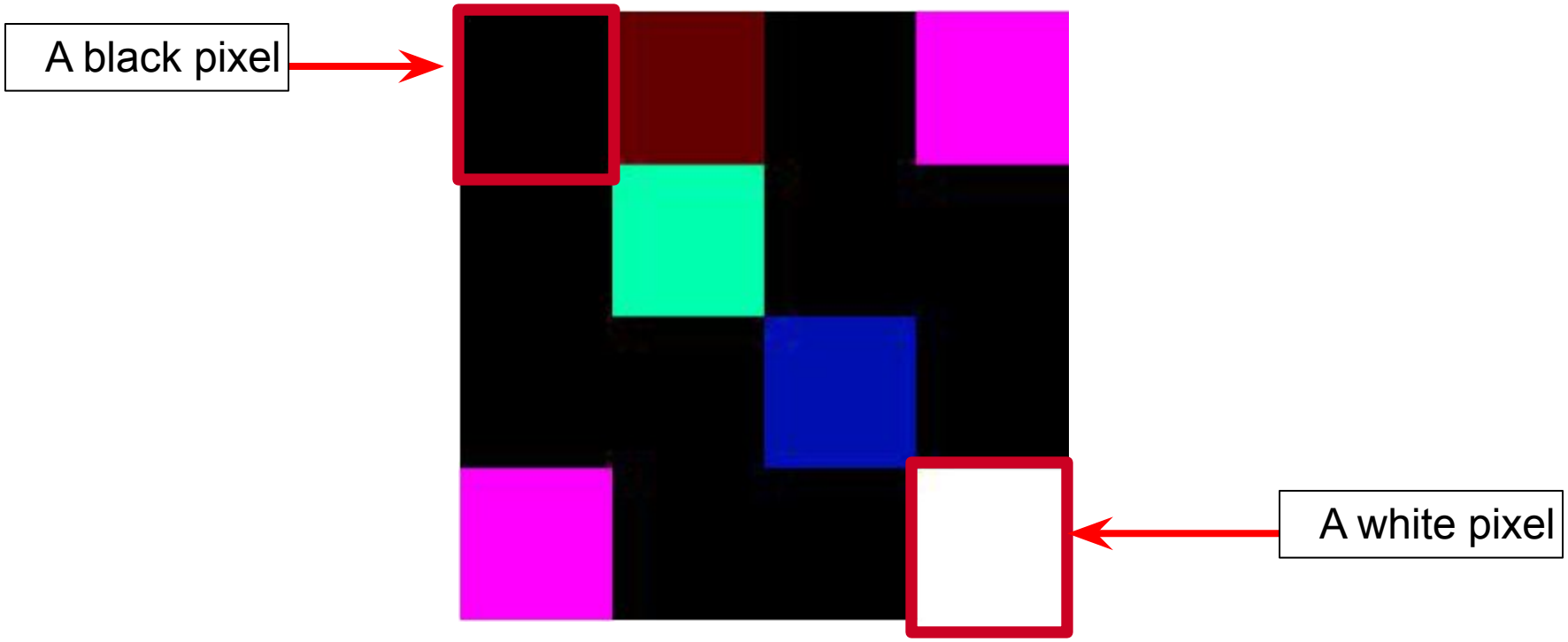


Digital Images – PPM format

- PPM, is an uncompressed image format, and so is not widely used for real applications.
- However, it's simple enough to fit most basic exercises, which makes it ideal for learning how to deal with digital images.
- Before explaining how the PPM format is organized, we give a brief explanation of some fundamental concepts of digital Images.

Pixels

- Pixels are the 'atoms' of an image.
- Each image is composed of many pixels.
- A pixel has two characteristics
  - Location
  - Color



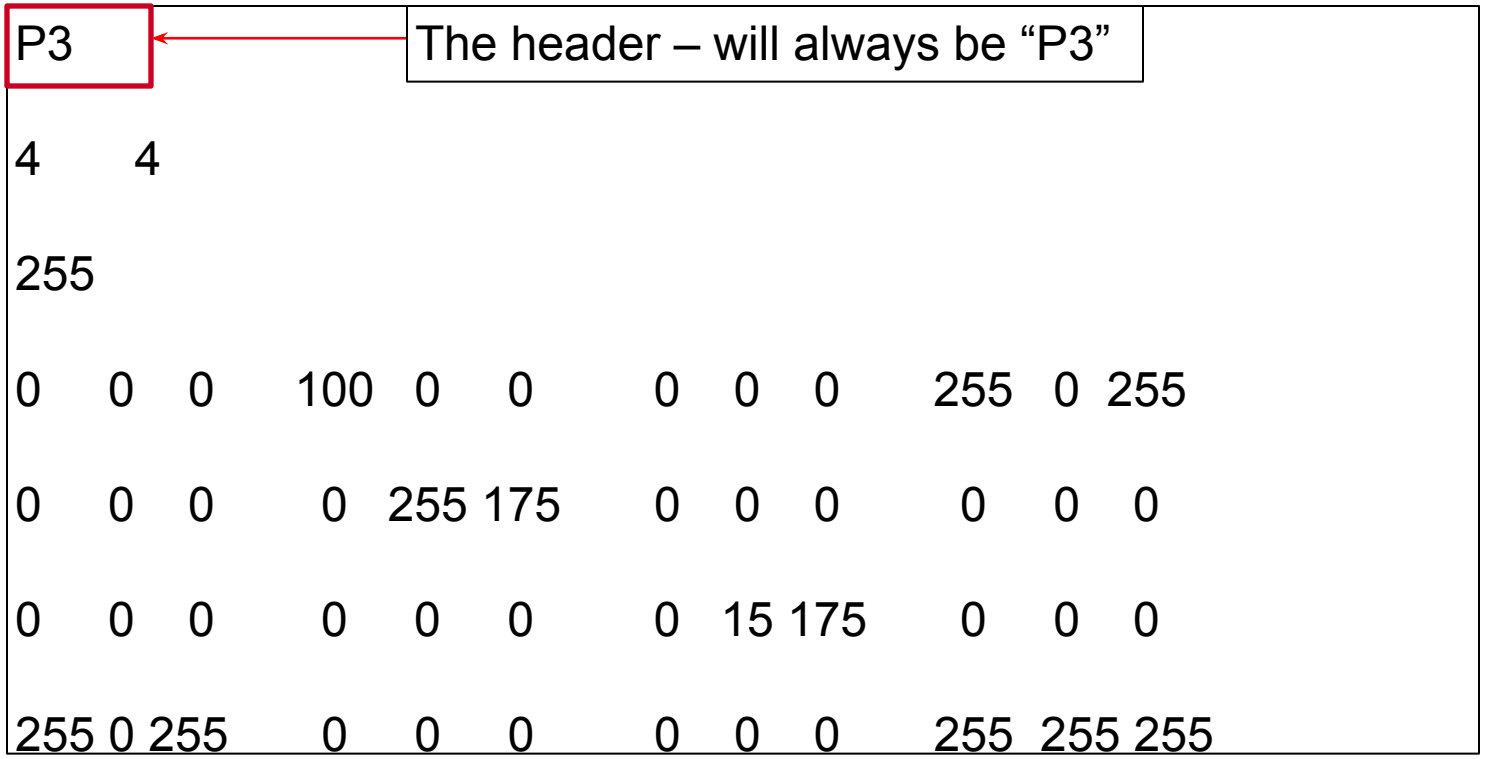
PPM

tinypic.ppm

```
P3
4  4
255
0   0   0    100  0   0    0   0   0    255  0  255
0   0   0    0   255 175   0   0   0    0   0   0
0   0   0    0   0   0    0  15 175    0   0   0
255 0 255    0   0   0    0   0   0    255 255 255
```

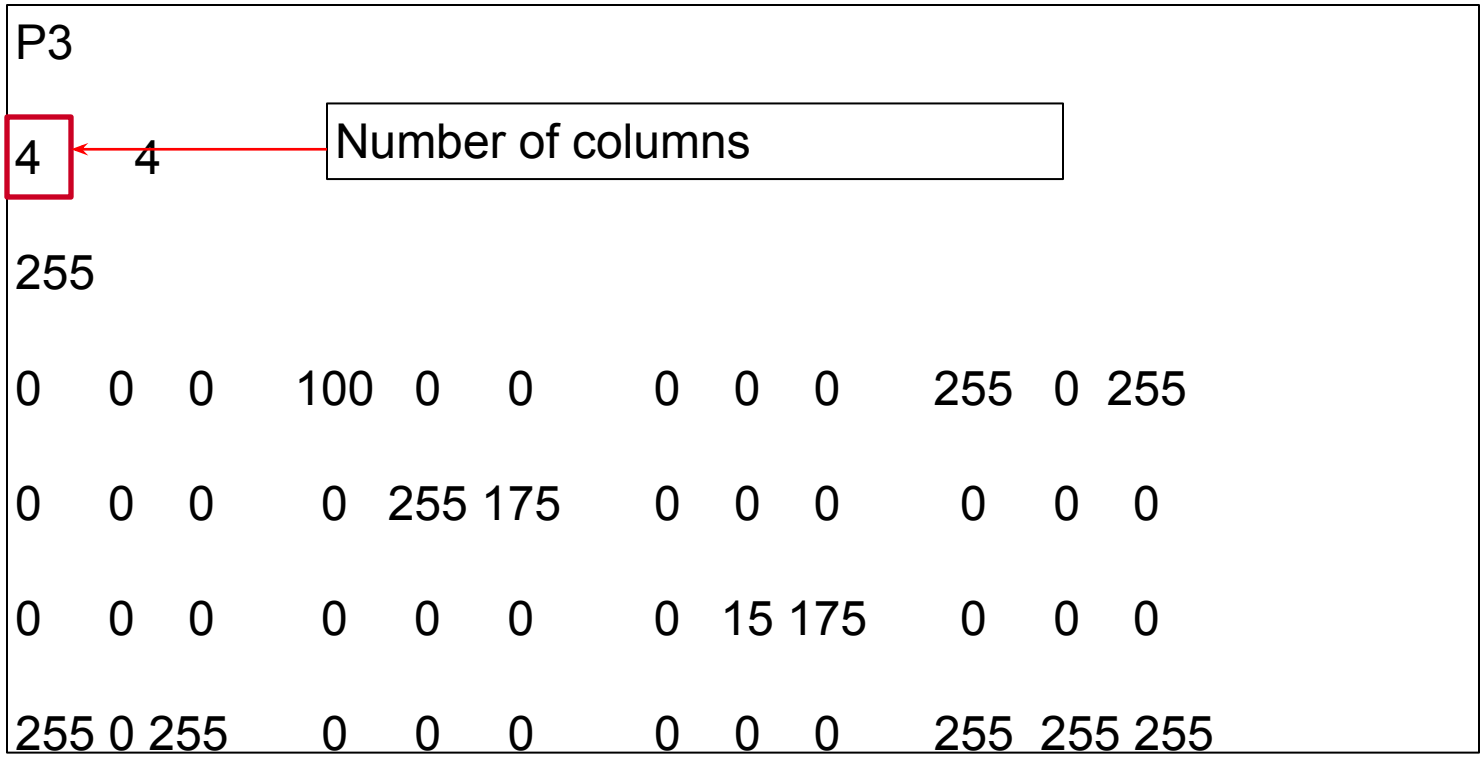
PPM

tinypic.ppm



PPM

tinypic.ppm



PPM

tinypic.ppm

```
P3
4       4
255
0   0   0     100  0   0      0   0   0     255  0  255
0   0   0       0  255 175    0   0   0       0  0    0
0   0   0       0  0   0      0   15  175     0  0    0
255 0   255     0  0   0      0   0   0     255  255  255
```

Number of rows

PPM

tinypic.ppm

```
P3

4   4

255

0   0   0     100   0   0     0   0   0     255   0   255

0   0   0     0   255 175     0   0   0     0     0   0

0   0   0     0   0   0     0   15  175     0     0   0

255 0   255     0   0   0     0   0   0     255   255 255
```

Maximum intensity – will always be 255

PPM

tinypic.ppm

```
P3
4  4
255
0   0   0    100  0   0     0   0   0    255  0  255
0   0   0      0  255 175   0   0   0      0  0   0
0   0   0      0  0   0     0   15  175    0  0   0
255 0   255    0  0   0     0   0   0    255  255 255
```

Each three numbers define RGB values for one pixel, this is the first pixel, it's Black.

PPM

tinypic.ppm

```
P3
4  4
255
0   0   0     100  0   0      0   0   0     255  0  255
0   0   0       0  255 175    0   0   0       0  0   0
0   0   0       0  0   0      0   15  175     0  0   0
255 0   255     0  0   0      0   0   0     255  255 255
```

Second pixel happens to be Bordeaux.

PPM

tinypic.ppm

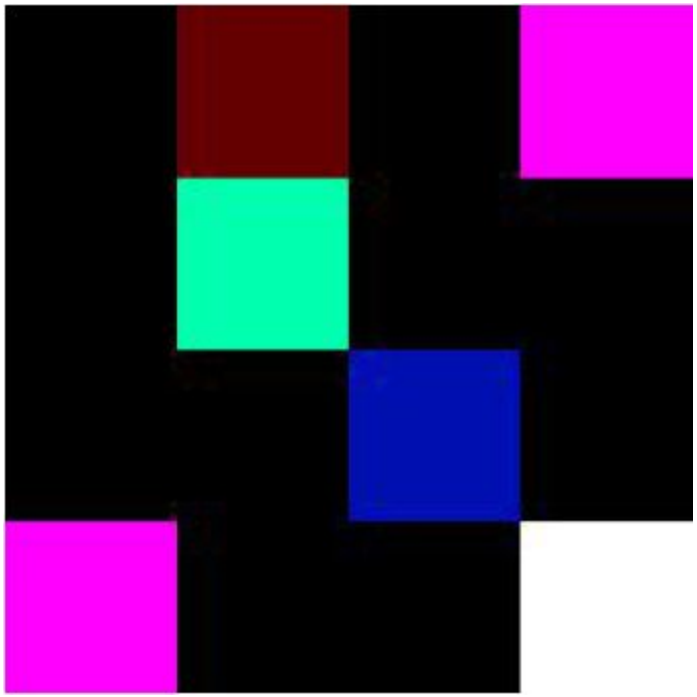
```
P3
4  4
255
0   0   0     100  0   0     0   0   0     255  0  255
0   0   0     0   255 175    0   0   0     0    0   0
0   0   0     0    0   0     0   15 175     0    0   0
255 0 255     0    0   0     0   0   0     255  255 255
```

Overall, there are 4 X 4 pixels. Pixels are arranged according to their location in the image.

PPM

tinypic.ppm

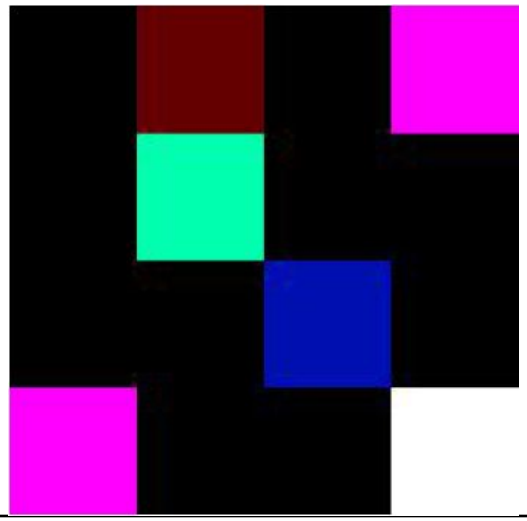
```
P3
4  4
255
0   0   0     100  0   0     0   0   0     255  0  255
0   0   0       0  255 175   0   0   0       0  0    0
0   0   0       0    0   0     0  15 175     0  0    0
255 0 255       0    0   0     0   0   0     255 255 255
```



Reading PPM files

- In order to operate on digital images we must first represent them inside our program.
- A very natural way is to store the image's data inside a 3 - dimensional array.
- The first two dimensions correspond to one pixel's locations.
- The third dimension will be of size 3, and correspond to the pixel's RGB values.

```
{{{0   , 0  , 0   }, {100, 0   , 0   }, {0, 0  , 0   }, {255, 0   , 255 }},
 {{0   , 0  , 0   }, {0   , 255, 175}, {0, 0  , 0   }, {0   , 0   , 0   }},
 {{0   , 0  , 0   }, {0   , 0   , 0   }, {0, 15 , 175}, {0   , 0   , 0   }},
 {{255, 0  , 255}, {0   , 0   , 0   }, {0, 0  , 0   }, {255, 255, 255 }}}
```

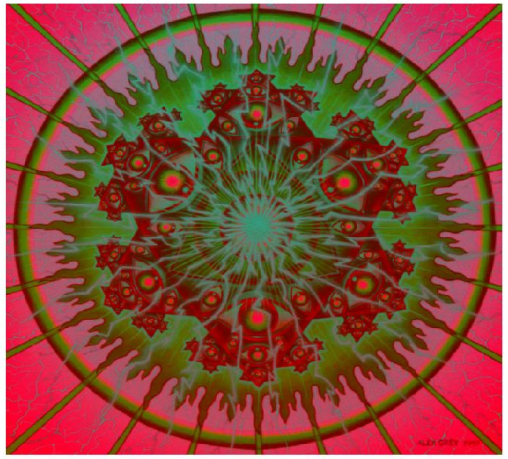


Question 5 - Color Inverting

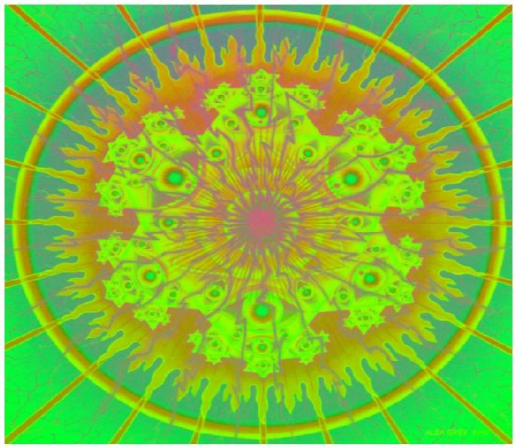
- Using the RGB system allows us to invert colors.
- Inversion of a color is done by subtracting it from 255.
- So, the red inverse of {10, 20, 30} is {245, 20, 30}.
- We can invert more than one color, if we invert all of the colors, the previous pixel becomes {245, 235, 225}.
- We will use the following image as an example:

Inversions

Red Inversion

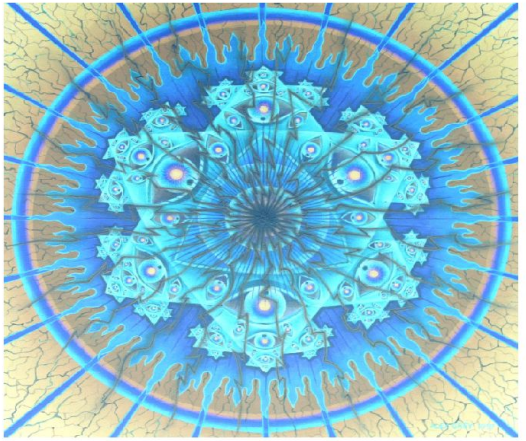


Green Inversion



Blue Inversion

Total



Question 5 – Solution

```
import java.awt.Color;
public class ColorInverter {
	public static final int MAX_INTENSITY = 255;
	public static final int RED_IDX = 0;
		public static final int GREEN_IDX = 1;
		public static final int BLUE_IDX = 2;

	/**
	* Inverts all the colors of an image
	* @param pic – the image
	* @return the inverted image
	*/
	public static Color[][] invertAll(Color[][] pic){
		pic = invertColor(pic, RED_IDX);
		pic = invertColor(pic, GREEN_IDX);
		pic = invertColor(pic, BLUE_IDX);
		return pic;
	}

	public static Color[][] invertColor(Color[][] pic, int colorIdx){
		...
	}

	public static Color invertPixel(Color pixel, int colorIdx){
		...
	}

}
```

Question 5 – Solution (Continued)

```
/**
* inverts one color of an image.
* @param pic - the picture to be inverted
* @param colorIdx - the index of the color to be inverted
* @return the inverted image
*/
public static Color[][] invertColor(Color[][] pic, int colorIdx){
      Color[][] ans = new Color[pic.length][pic[0].length];

      for (int i = 0; i < ans.length; i++) {
            for (int j = 0; j < ans[0].length; j++) {
                  ans[i][j] = invertPixelColor(pic[i][j], colorIdx);
            }
      }
      return ans;
}

/**
* invert one color of a given pixel.
* @param pixel - the color array to be inverted
* @param colorIdx - the index of the color to be inverted
* @return - the inverted pixel
*/
public static Color invertPixelColor(Color pixel, int colorIdx){
      int[] newPixel = {pixel.getRed(), pixel.getGreen(), pixel.getBlue()};
      newPixel[colorIdx] = MAX_INTENSITY - newPixel[colorIdx];
      return new Color(newPixel[RED_IDX], newPixel[GREEN_IDX], newPixel[BLUE_IDX]);
}
```