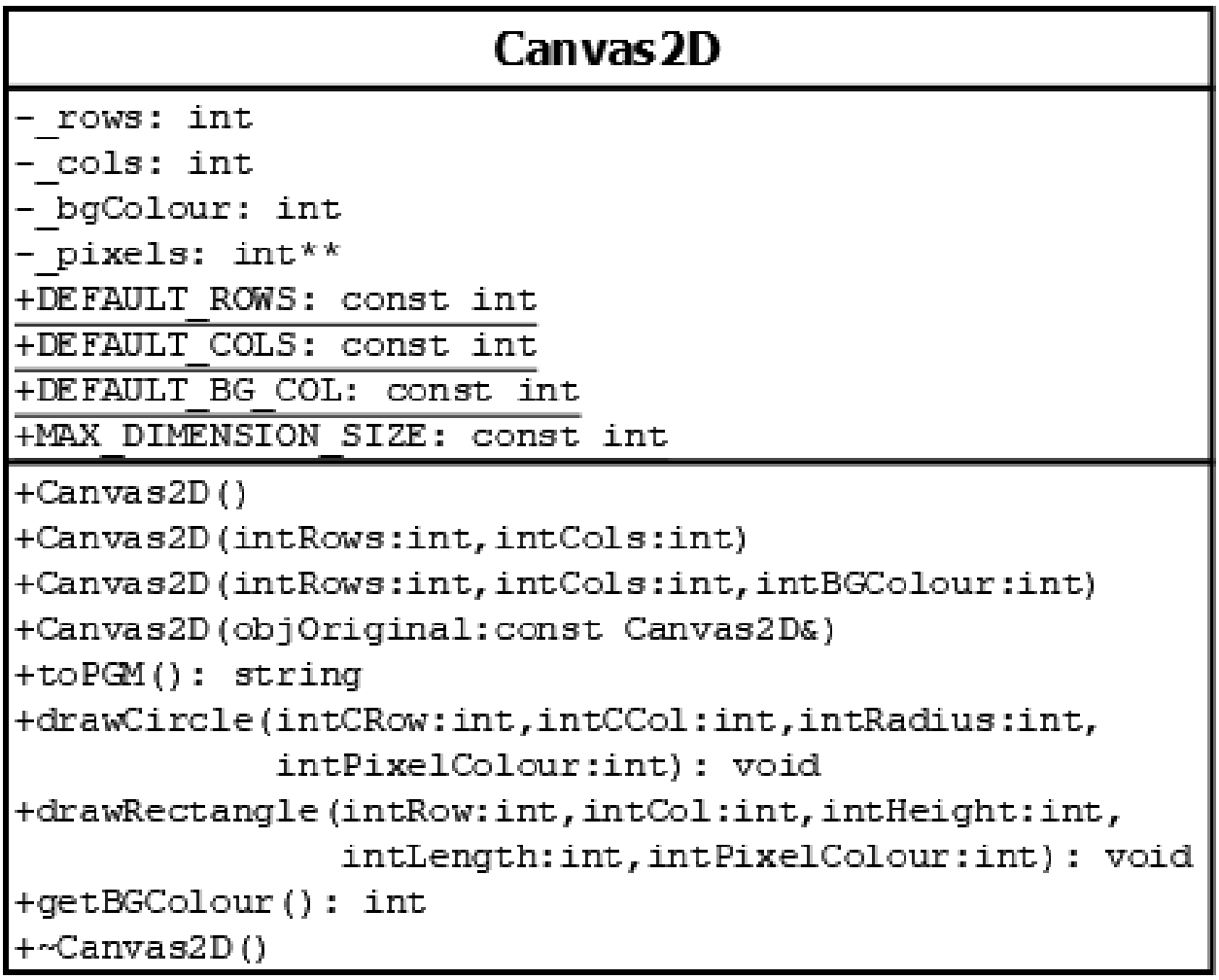
SP SITHUNGU

200000000

PRACTICAL 2 CLASS DIAGRAM



```cpp
#ifndef CANVAS2D_H
#define CANVAS2D_H

#include <string>
using namespace std;

namespace CanvasSpace{
    enum ERROR_CODE{
        SUCCESS,
        ERROR_RANGE
    };
    /*
     * The required data structure to manage a 2D integer array.
     */
    class Canvas2D{
    public:
        // Constructor for initialising the data structure.
        Canvas2D();
        Canvas2D(int intRows, int intCols, int intBGColour);
        Canvas2D(const Canvas2D& objOriginal);
        // A member function for creating and returning at P2 PGM string from the pixel array.
        string toPGM() const;
        // A member function for drawing a circle.
        void drawCircle(int intCRow, int intCCol, int inRadius, int intPixel);
        // A member function for drawing a rectangle.
        void drawRectangle(int intRow, int intCol, int intHeight, int intLength, int
intPixel);
        // A member function that returns the background colour.
        int getBGColour() const;
        // Destructor for deallocating the pixel array.
        ~Canvas2D();
        // Class constants.
        static const int DEFAULT_ROWS = 500;
        static const int DEFAULT_COLS = 500;
        static const int DEFAULT_BG_COL = 132;
        static const int MAX_DIMENSION_SIZE = 100000;
    private:
        // Utility functions.
        void alloc(int intRows, int intCols, int intBGColour);
        void dealloc();
        void clone(const Canvas2D& objOriginal);
        void enforceRange(int intArg, int intMin, int intMax) const;
        double distance(int intX1, int intX2, int intY1, int intY2) const;
        // Member variables.
        int _rows;
        int _cols;
        int _bgColour;
        int** _pixels;
    };
}

#endif // CANVAS2D_H

```

```cpp
#include "Canvas2D.h"

#include <sstream>
#include <cmath>
#include <iostream>

namespace CanvasSpace{
    Canvas2D::Canvas2D() : Canvas2D(DEFAULT_ROWS, DEFAULT_COLS, DEFAULT_BG_COL)
    {
        /*
         * Nothing to do since the parameterised constructor will do the work
         * via constructo chaining.
         */
    }
    // Parameterised constructor.
    Canvas2D::Canvas2D(int intRows, int intCols, int intBGColour){
        alloc(intRows, intCols, intBGColour);
    }

    Canvas2D::Canvas2D(const Canvas2D& objOriginal){
        // Allocate memory for the new object.
        alloc(objOriginal._rows, objOriginal._cols, DEFAULT_BG_COL);
        // Clone the pixel array to the new object.
        clone(objOriginal);
    }

    string Canvas2D::toPGM() const{
        stringstream ssPPM;
        // P2 for PGM.
        ssPPM << "P2" << endl
              << _cols << ' ' << _rows << endl
              << 255 << endl;
        for(int r = 0; r < _rows; r++){
            for(int c = 0; c < _cols; c++){
                ssPPM << _pixels[r][c] << ' ';
            }
            ssPPM << endl;
        }
        return ssPPM.str();
    }

    void Canvas2D::drawCircle(int intCRow, int intCCol, int intRadius, int intPixel){
        for(int r = 0; r < _rows; r++){
            for(int c = 0; c < _cols; c++){
                if(distance(r, intCRow, c, intCCol) <= intRadius){
                    _pixels[r][c] = intPixel;
                }
            }
        }
    }

    void Canvas2D::drawRectangle(int intRow, int intCol, int intHeight, int intLength, int
intPixel){
        for(int r = 0; r < _rows; r++){
            for(int c = 0; c < _cols; c++){
                if(r >= intRow && r <= intRow + intHeight){
                    if(c >= intCol && c <= (intCol + intLength))
                        _pixels[r][c] = intPixel;
                }
            }
        }
    }

    int Canvas2D::getBGColour() const{
        return _bgColour;
    }

    Canvas2D::~Canvas2D(){
        dealloc();
    }

    void Canvas2D::alloc(int intRows, int intCols, int intBGColour){
        _rows = intRows;
        _cols = intCols;
        _bgColour = intBGColour;
        _pixels = new int*[_rows];
        for(int r = 0; r < _rows; r++){
            _pixels[r] = new int[_cols];
            for(int c = 0; c < _cols; c++){
```

```
79                  // Set all the pixels to the background colour.
80                  _pixels[r][c] = _bgColour;
81              }
82          }
83      }
84
85      void Canvas2D::dealloc(){
86          for(int r = 0; r < _rows; r++){
87              delete [] _pixels[r];
88          }
89          delete [] _pixels;
90      }
91
92      void Canvas2D::clone(const Canvas2D& objOriginal){
93          for(int r = 0; r < _rows; r++){
94              for(int c = 0; c < _cols; c++){
95                  // Deep copy.
96                  _pixels[r][c] = objOriginal._pixels[r][c];
97              }
98          }
99      }
100
101     void Canvas2D::enforceRange(int intArg, int intMin, int intMax) const{
102         if(intArg < intMin || intArg >intMax){
103             cerr << intArg << " must be in [" << intMin << ", " << intMax << "]" << endl;
104             exit(ERROR_RANGE);
105         }
106     }
107
108     double Canvas2D::distance(int intX1, int intX2, int intY1, int intY2) const{
109         return sqrt(pow(intX1 - intX2, 2) + pow(intY1 - intY2, 2));
110     }
111 }
112
```

```cpp
#include "Canvas2D.h"

#include <ctime>
#include <iostream>

using namespace CanvasSpace;

int main()
{
    // Create a Canvas2D object.
    Canvas2D objCanvas(700, 700, 255);
    // Draw a circle for the face.
    objCanvas.drawCircle(350, 350, 300, objCanvas.getBGColour() - 100);
    // Draw the left eye.
    objCanvas.drawCircle(225, 225, 50, 0);
    // Draw the right eye.
    objCanvas.drawCircle(225, 475, 50, 0);
    // Draw the "nose".
    objCanvas.drawRectangle(325, 325, 50, 50, 0);
    // Draw the "mouth".
    objCanvas.drawRectangle(450, 250, 50, 200, 0);
    // Insert the PGM string to the output stream (cout) using the stream insertion operator.
    cout << objCanvas.toPGM() << endl;
    return SUCCESS;
}

```