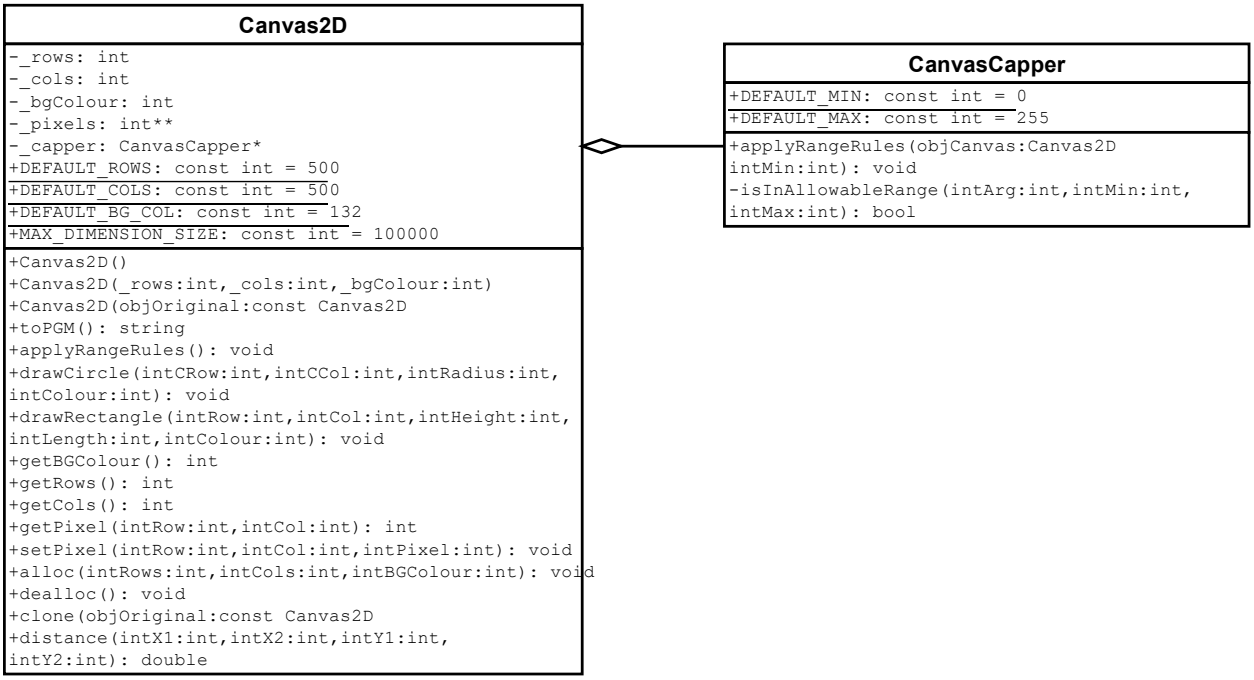
SP SITHUNGU
200000000
PRACTICAL 3 CLASS DIAGRAM



```cpp
#ifndef CANVAS2D_H
#define CANVAS2D_H

#include <string>
using namespace std;

/*
 * The forward declaration of CanvasCapper must be made
 * outside the CanvasSpace namespace declaration.
 */
class CanvasCapper;

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;
        // The following function will delegate its task to the contained object.
        void applyRangeRules();
        // 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);
        // Accessor methods.
        int getBGColour() const;
        int getRows() const;
        int getCols() const;
        int getPixel(int intRow, int intCol) const;
        // Mutator method.
        void setPixel(int intRow, int intCol, int intPixel);
        // 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;
        CanvasCapper* _capper;
        int** _pixels;
    };
}

#endif // CANVAS2D_H
```

```cpp
#include "Canvas2D.h"
#include "CanvasCapper.h"

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

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::applyRangeRules(){
        _capper->applyRangeRules(*this, 187, 55);
    }

    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;
                }
            }
        }
```

```
67      }
68
69      int Canvas2D::getBGColour() const{
70          return _bgColour;
71      }
72
73      int Canvas2D::getRows() const{
74          return _rows;
75      }
76
77      int Canvas2D::getCols() const{
78          return _cols;
79      }
80
81      int Canvas2D::getPixel(int intRow, int intCol) const{
82          enforceRange(intRow, 0, _rows - 1);
83          enforceRange(intRow, 0, _cols - 1);
84          return _pixels[intRow][intCol];
85      }
86
87      void Canvas2D::setPixel(int intRow, int intCol, int intPixel){
88          enforceRange(intRow, 0, _rows - 1);
89          enforceRange(intRow, 0, _cols - 1);
90          _pixels[intRow][intCol] = intPixel;
91      }
92
93      Canvas2D::~Canvas2D(){
94          dealloc();
95      }
96
97      void Canvas2D::alloc(int intRows, int intCols, int intBGColour){
98          _rows = intRows;
99          _cols = intCols;
100         _bgColour = intBGColour;
101         _capper = new CanvasCapper;
102         _pixels = new int*[_rows];
103         for(int r = 0; r < _rows; r++){
104             _pixels[r] = new int[_cols];
105             for(int c = 0; c < _cols; c++){
106                 // Set all the pixels to the background colour.
107                 _pixels[r][c] = _bgColour;
108             }
109         }
110     }
111
112     void Canvas2D::dealloc(){
113         assert(_pixels != nullptr);
114         assert(_capper != nullptr);
115         for(int r = 0; r < _rows; r++){
116             delete [] _pixels[r];
117         }
118         delete [] _pixels;
119         delete _capper;
120     }
121
122     void Canvas2D::clone(const Canvas2D& objOriginal){
123         for(int r = 0; r < _rows; r++){
124             for(int c = 0; c < _cols; c++){
125                 // Deep copy.
126                 _pixels[r][c] = objOriginal._pixels[r][c];
127             }
128         }
129     }
130
131     void Canvas2D::enforceRange(int intArg, int intMin, int intMax) const{
132         if(intArg < intMin || intArg >intMax){
```

```
            cerr << intArg << " must be in [" << intMin << ", " << intMax << "]" << endl;
            exit(ERROR_RANGE);
        }
    }

    double Canvas2D::distance(int intX1, int intX2, int intY1, int intY2) const{
        return sqrt(pow(intX1 - intX2, 2) + pow(intY1 - intY2, 2));
    }
}
```

```cpp
#ifndef CANVASCAPPER_H_INCLUDED
#define CANVASCAPPER_H_INCLUDED

/*
 * We need to #include Canvas2D.h in this header file because it needs to know about
 * The CanvasSpace namespace. Just making a forward declaration of Canvas2D will
 * result in a compilation error.
 */
#include "Canvas2D.h"

class CanvasCapper{
public:
    void applyRangeRules(CanvasSpace::Canvas2D& objCanvas, int intMax, int intMin) const;
    static const int DEFAULT_MIN = 0;
    static const int DEFAULT_MAX = 255;
private:
    bool isInAllowableRange(int intArg, int intMin, int intMax) const;
};

#endif // CANVASCAPPER_H_INCLUDED
```

```cpp
#include "CanvasCapper.h"

#include <iostream>
using namespace std;
using namespace CanvasSpace;

void CanvasCapper::applyRangeRules(Canvas2D& objCanvas, int intMax, int intMin) const{
    if(!isInAllowableRange(intMin, DEFAULT_MIN, DEFAULT_MAX) || !isInAllowableRange(intMax, DEFAULT_MIN,
DEFAULT_MAX)){
        cerr << "The specified min/max range must be in [0, 255]." << endl;
        exit(ERROR_RANGE);
    }

    for(int r = 0; r < objCanvas.getRows(); r++){
        for(int c = 0; c < objCanvas.getCols(); c++){
            if(objCanvas.getPixel(r, c) < intMin){
                objCanvas.setPixel(r, c, intMin);
            }else if(objCanvas.getPixel(r, c) > intMax){
                objCanvas.setPixel(r, c, intMax);
            }
        }
    }
}

bool CanvasCapper::isInAllowableRange(int intArg, int intMin, int intMax) const{
    if(intArg < intMin || intArg > intMax){
        return false;
    }
    return true;
}
```

```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);
    // Test the capping of pixels using applyRangeRules().
    objCanvas.applyRangeRules();
    // Insert the PGM string to the output stream (cout) using the stream insertion operator.
    cout << objCanvas.toPGM() << endl;
    return SUCCESS;
}
```