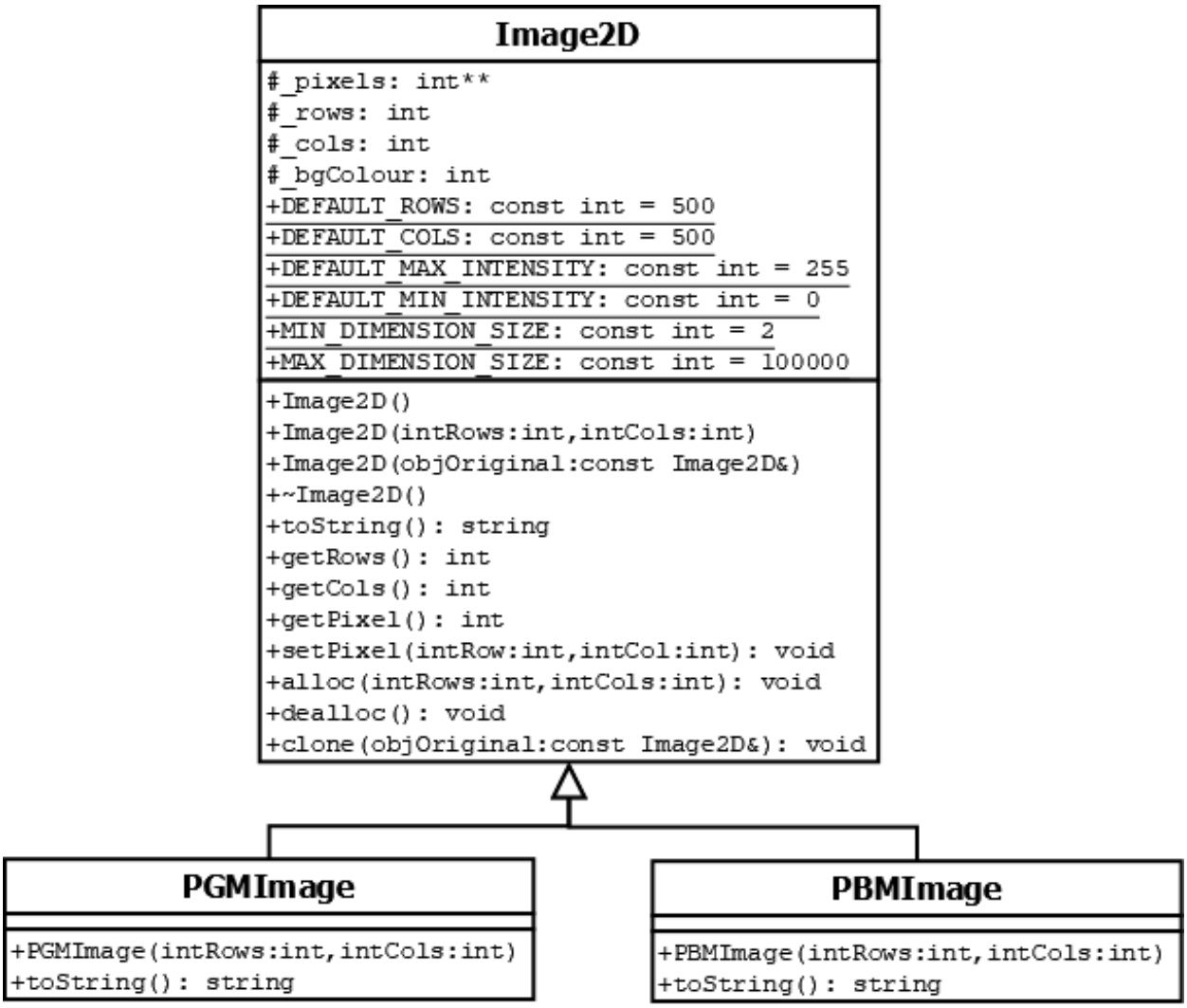
SP SITHUNGU
200000000
PRACTICAL 4 CLASS DIAGRAM



```cpp
#ifndef COMMONLIB_H
#define COMMONLIB_H

#include <string>

namespace CommonSpace{
    enum ERROR_CODE{
        SUCCESS,
        ERROR_ARGS,
        ERROR_RANGE
    };

    int CInt(std::string arg);
    void enforceRange(int intArg, int intMin, int intMax);
    int rangedRandom(int intLBound, int intUBound);
}

#endif // COMMONLIB_H
```

```cpp
#include "CommonLib.h"

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

namespace CommonSpace{
    int CInt(string arg){
        stringstream ssConv(arg);
        int intResult = 0;
        ssConv >> intResult;
        return intResult;
    }

    void enforceRange(int intArg, int intMin, int intMax){
        if(intArg < intMin || intArg >intMax){
            cerr << intArg << " must be in [" << intMin
                 << ", " << intMax << "]" << endl;
            exit(ERROR_RANGE);
        }
    }

    int rangedRandom(int intLBound, int intUBound){
        return rand() % (intUBound - intLBound + 1) + intLBound;
    }
}

```

```cpp
#ifndef IMAGE2D_H
#define IMAGE2D_H

#include <string>

class Image2D{
public:
    Image2D();
    Image2D(int intRows, int intCols);
    Image2D(const Image2D& objOriginal);

    virtual std::string toString() const = 0;

    // Accessor methods.
    int getRows() const;
    int getCols() const;
    int getPixel(int intRow, int intCol) const;
    // Mutator method.
    void setPixel(int intRow, int intCol, int intPixel);

    static const int DEFAULT_ROWS = 500;
    static const int DEFAULT_COLS = 500;
    static const int DEFAULT_MAX_INTENSITY = 255;
    static const int DEFAULT_MIN_INTENSITY = 0;
    static const int MIN_DIMENSION_SIZE = 2;
    static const int MAX_DIMENSION_SIZE = 100000;

    ~Image2D();
protected:
    void alloc(int intRows, int intCols);
    void dealloc();
    void clone(const Image2D& objOriginal);
    int _bgColour;
    int _rows;
    int _cols;
    int** _pixels;
};

#endif // IMAGE2D_H
```

```cpp
#include "CommonLib.h"
#include "Image2D.h"

#include <cassert>
#include <iostream>

using namespace std;
using namespace CommonSpace;

Image2D::Image2D() : Image2D(DEFAULT_ROWS, DEFAULT_COLS){}

Image2D::Image2D(int intRows, int intCols){
    alloc(intRows, intCols);
}

Image2D::Image2D(const Image2D& objOriginal) : Image2D(objOriginal._rows, objOriginal._cols){
    clone(objOriginal);
}

int Image2D::getRows() const{
    return _rows;
}

int Image2D::getCols() const{
    return _cols;
}

int Image2D::getPixel(int intRow, int intCol) const{
    enforceRange(intRow, 0, _rows - 1);
    enforceRange(intCol, 0, _cols - 1);
    return _pixels[intRow][intCol];
}

void Image2D::setPixel(int intRow, int intCol, int intPixel){
    enforceRange(intRow, 0, _rows - 1);
    enforceRange(intCol, 0, _cols - 1);
    _pixels[intRow][intCol] = intPixel;
}

void Image2D::alloc(int intRows, int intCols){
    enforceRange(intRows, MIN_DIMENSION_SIZE, MAX_DIMENSION_SIZE);
    enforceRange(intCols, MIN_DIMENSION_SIZE, MAX_DIMENSION_SIZE);
    _rows = intRows;
    _cols = intCols;
    _pixels = new int*[_rows];
    for(int r = 0; r < _rows; r++){
        _pixels[r] = new int[_cols];
        for(int c = 0; c < _cols; c++){
            _pixels[r][c] = rangedRandom(DEFAULT_MIN_INTENSITY, DEFAULT_MAX_INTENSITY);

        }
    }
}

void Image2D::dealloc(){
    assert(_pixels != nullptr);
    for(int r = 0; r < _rows; r++){
        delete [] _pixels[r];
    }
    delete [] _pixels;
}

void Image2D::clone(const Image2D& objOriginal){
    for(int r = 0; r < _rows; r++){
        for(int c = 0; c < _cols; c++){
```

```
65              _pixels[r][c] = objOriginal._pixels[r][c];
66          }
67      }
68  }
69
70  Image2D::~Image2D(){
71      dealloc();
72  }
```

```
#ifndef PGMIMAGE_H
#define PGMIMAGE_H

#include "Image2D.h"

class PGMImage : public Image2D{
public:
    PGMImage(int intRows, int intCols);
    virtual std::string toString() const;
};

#endif // PGMIMAGE_H
```

```cpp
#include "PGMImage.h"

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

PGMImage::PGMImage(int intRows, int intCols)
        : Image2D(intRows, intCols){}

string PGMImage::toString() const{
    stringstream ssPPM;
    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();
}
```

```cpp
#ifndef PBMIMAGE_H
#define PBMIMAGE_H

#include "Image2D.h"

class PBMImage : public Image2D{
public:
    PBMImage(int intRows, int intCols);
    virtual std::string toString() const;
};

#endif // PBMIMAGE_H
```

```cpp
#include "PBMImage.h"

#include <sstream>
using namespace std;

PBMImage::PBMImage(int intRows, int intCols){

}

string PBMImage::toString() const{
    stringstream ssPPM;
    ssPPM << "P1" << endl
          << _cols << ' ' << _rows << endl;
    for(int r = 0; r < _rows; r++){
        for(int c = 0; c < _cols; c++){
            _pixels[r][c] > 0 ? ssPPM << 1 : ssPPM << _pixels[r][c];
            ssPPM << ' ';
        }
        ssPPM << endl;
    }
    return ssPPM.str();
}
```

```cpp
#include "CommonLib.h"
#include "PBMImage.h"
#include "PGMImage.h"

#include <iostream>

using namespace std;
using namespace CommonSpace;

void printImage(Image2D& objImage);

int main(int argc, char** argv)
{
    if(argc != 4){
        cerr << "Usage: " << argv[0]
             << " NUM_ROWS NUM_COLS IMAGE_TYPE"
             << endl;
        exit(ERROR_ARGS);
    }
    int intRows = CInt(argv[1]);
    int intCols = CInt(argv[2]);
    int intImageType = CInt(argv[3]);
    enforceRange(intImageType, 0, 1);

    if(intImageType == 0)
        printImage(*(new PGMImage(intRows, intCols)));
    else
        printImage(*(new PBMImage(intRows, intCols)));

    return SUCCESS;
}

void printImage(Image2D& objImage){
    cout << objImage.toString();
}
```