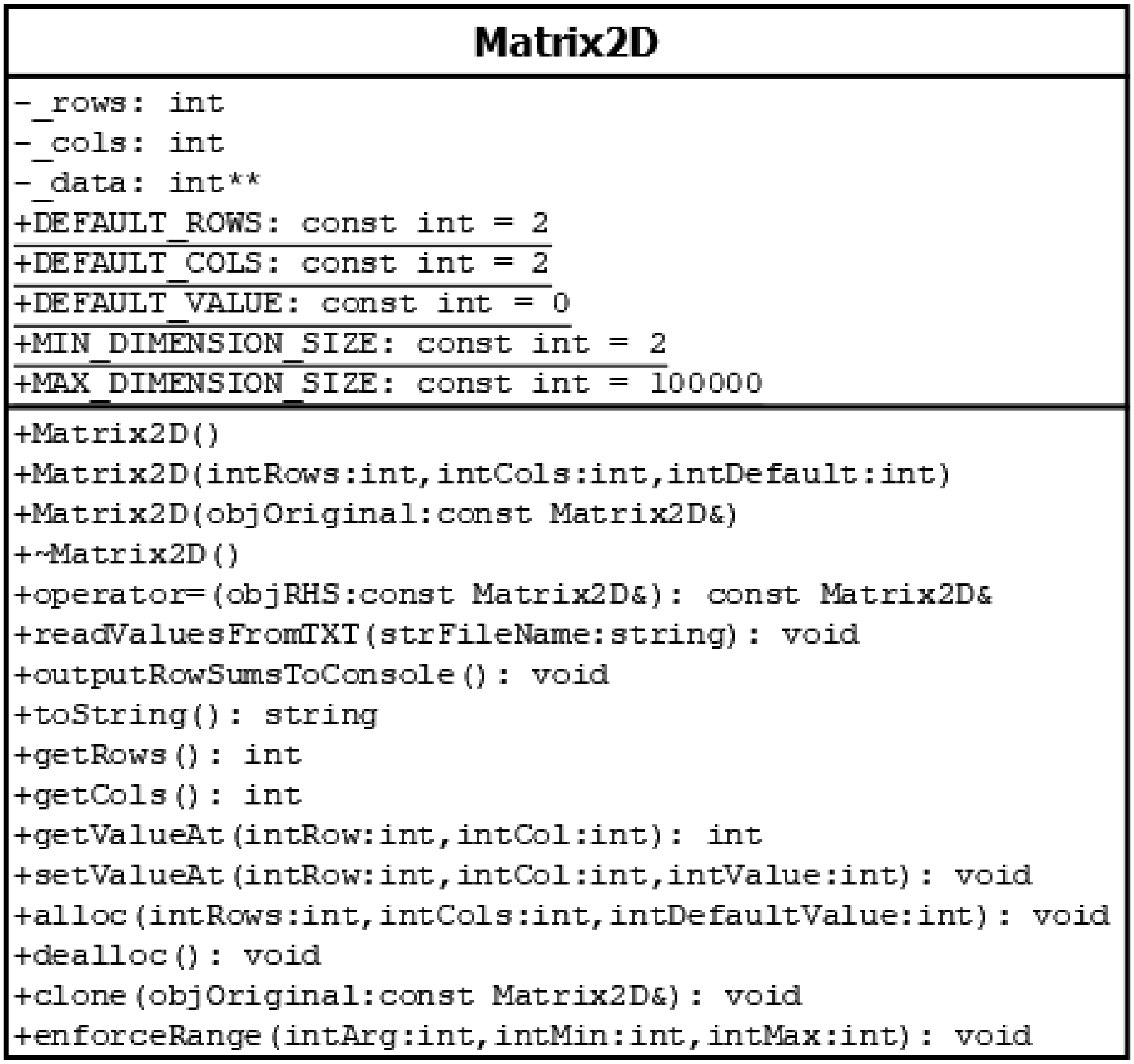
SP SITHUNGU

200000000

PRACTICAL 8 DESIGN



Matrix2D

```
-_rows: int
-_cols: int
-_data: int**
+DEFAULT_ROWS: const int = 2
+DEFAULT_COLS: const int = 2
+DEFAULT_VALUE: const int = 0
+MIN_DIMENSION_SIZE: const int = 2
+MAX_DIMENSION_SIZE: const int = 100000
```

```
+Matrix2D()
+Matrix2D(intRows:int,intCols:int,intDefault:int)
+Matrix2D(objOriginal:const Matrix2D&)
+~Matrix2D()
+operator=(objRHS:const Matrix2D&): const Matrix2D&
+readValuesFromTXT(strFileName:string): void
+outputRowSumsToConsole(): void
+toString(): string
+getRows(): int
+getCols(): int
+getValueAt(intRow:int,intCol:int): int
+setValueAt(intRow:int,intCol:int,intValue:int): void
+alloc(intRows:int,intCols:int,intDefaultValue:int): void
+dealloc(): void
+clone(objOriginal:const Matrix2D&): void
+enforceRange(intArg:int,intMin:int,intMax:int): void
```

```cpp
#ifndef EXCEPTIONS_H
#define EXCEPTIONS_H

#include <string>

class Exception{};

class FileException : public Exception{};

class RangeException : public Exception{};

#endif // EXCEPTIONS_H
```

```cpp
#ifndef MATRIX2D_H
#define MATRIX2D_H

#include <iostream>
#include <string>

enum ERROR_CODE{
    SUCCESS,
    ERROR_ARGS,
    ERROR_RANGE
};

class Matrix2D{
public:
    Matrix2D();
    Matrix2D(int intRows, int intCols, int intDefault);
    Matrix2D(const Matrix2D& objOriginal);

    const Matrix2D& operator=(const Matrix2D& objRHS);

    void readValuesFromTXT(std::string strFileName);
    //void savePixelsToTXT(std::string strFileName) const;
    void outputRowSumsToConsole() const;

    std::string toString() const;

    int getRows() const;
    int getCols() const;
    int getValueAt(int intRow, int intCol) const;

    void setValueAt(int intRow, int intCol, int intValue);

    static const int DEFAULT_ROWS = 2;
    static const int DEFAULT_COLS = 2;
    static const int DEFAULT_VALUE = 0;
    static const int MIN_DIMENSION_SIZE = 2;
    static const int MAX_DIMENSION_SIZE = 100000;

    ~Matrix2D();
private:
    void alloc(int intRows, int intCols, int intDefaultValue);
    void dealloc();
    void clone(const Matrix2D& objOriginal);
    void enforceRange(int intArg, int intMin, int intMax) const;
    int** _data;
    int _rows;
    int _cols;
};

#endif // MATRIX2D_H
```

```cpp
#include "Exceptions.h"
#include "Matrix2D.h"

#include <cassert>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

Matrix2D::Matrix2D() : Matrix2D(DEFAULT_ROWS, DEFAULT_COLS, DEFAULT_VALUE){}

Matrix2D::Matrix2D(int intRows, int intCols, int intDefaultValue){
    alloc(intRows, intCols, intDefaultValue);
}

Matrix2D::Matrix2D(const Matrix2D& objOriginal) : Matrix2D(objOriginal._rows,
objOriginal._cols, DEFAULT_VALUE){
    clone(objOriginal);
}

const Matrix2D& Matrix2D::operator=(const Matrix2D& objRHS){
    if(this != &objRHS){ // Check for self-assignment.
        dealloc();
        alloc(objRHS._rows, objRHS._cols, DEFAULT_VALUE);
        clone(objRHS);
    }
    return *this;
}

void Matrix2D::readValuesFromTXT(std::string strFileName){
    ifstream ifReader(strFileName);
    if(ifReader.fail()){
        throw FileException();
    }
    int intRow = 0;
    int intCol = 0;
    int intValue = 0;
    while(ifReader >> intRow >> intCol >> intValue){
        try{
            enforceRange(intRow, 0, _rows - 1);
            enforceRange(intCol, 0, _cols - 1);
        }catch(RangeException& re){
            throw; // Re-throwing the exception so that it is handled by the calling
function instead.
        }
        _data[intRow][intCol] = intValue;
    }
    ifReader.close();
}

/*void Matrix2D::savePixelsToTXT(std::string strFileName) const{
    ofstream ofWriter(strFileName);
    if(ofWriter.fail()){
        throw FileException();
    }
    for(int r = 0; r < _rows; r++){
        for(int c = 0; c < _cols; c++){
            ofWriter << r << ' '
                     << c << ' '
                     << _data[r][c] << endl;
        }
    }
    ofWriter.close();
}*/
```

```cpp

void Matrix2D::outputRowSumsToConsole() const{
    for(int r = 0; r < _rows; r++){
        int intSum = 0;
        for(int c = 0; c < _cols; c++){
            intSum += _data[r][c];
        }
        cout << intSum << ' ';
    }
}

string Matrix2D::toString() const{
    stringstream ssReturn;
    for(int r = 0; r < _rows; r++){
        for(int c = 0; c < _cols; c++){
            ssReturn << _data[r][c] << ' ';
        }
        ssReturn << endl;
    }
    return ssReturn.str();
}

int Matrix2D::getRows() const{
    return _rows;
}

int Matrix2D::getCols() const{
    return _cols;
}

int Matrix2D::getValueAt(int intRow, int intCol) const{
    enforceRange(intRow, 0, _rows - 1);
    enforceRange(intCol, 0, _cols - 1);
    return _data[intRow][intCol];
}

void Matrix2D::setValueAt(int intRow, int intCol, int intValue){
    enforceRange(intRow, 0, _rows - 1);
    enforceRange(intCol, 0, _cols - 1);
    _data[intRow][intCol] = intValue;
}

void Matrix2D::alloc(int intRows, int intCols, int intDefaultValue){
    enforceRange(intRows, MIN_DIMENSION_SIZE, MAX_DIMENSION_SIZE);
    enforceRange(intCols, MIN_DIMENSION_SIZE, MAX_DIMENSION_SIZE);
    _rows = intRows;
    _cols = intCols;
    _data = new int*[_rows];
    for(int r = 0; r < _rows; r++){
        _data[r] = new int[_cols];
        for(int c = 0; c < _cols; c++){
            _data[r][c] = intDefaultValue;
        }
    }
}

void Matrix2D::dealloc(){
    assert(_data != nullptr);
    for(int r = 0; r < _rows; r++){
        delete [] _data[r];
    }
    delete [] _data;
}

void Matrix2D::clone(const Matrix2D& objOriginal){
    for(int r = 0; r < _rows; r++){
        for(int c = 0; c < _cols; c++){
            _data[r][c] = objOriginal._data[r][c];
        }
    }
}

void Matrix2D::enforceRange(int intArg, int intMin, int intMax) const{
        if(intArg < intMin || intArg >intMax){
            cerr << intArg << " must be in [" << intMin
                 << ", " << intMax << "]" << endl;
            exit(ERROR_RANGE);
        }
}

Matrix2D::~Matrix2D(){
    dealloc();
}
```

```cpp
#include "Exceptions.h"
#include "Matrix2D.h"

#include <iostream>

using namespace std;

int main()
{
    // Testing the fully parameterised constructor.
    Matrix2D objMatrix(10, 10, 1);
    // Testing the copy constructor.
    Matrix2D objCopy = objMatrix;
    // Testing the overloaded assignment operator.
    Matrix2D objSmallerSizedMatrix(2, 2, 5);
    objSmallerSizedMatrix = objCopy;
    try{
        objSmallerSizedMatrix.readValuesFromTXT("data/array_values.txt");
        objSmallerSizedMatrix.outputRowSumsToConsole();
    }catch(FileException& fe){
        cerr << "A FileException Occurred." << endl;
    }catch(RangeException& re){
        cerr << "A RangeException Occurred." << endl;
    }catch(Exception& e){
        cerr << "An Exception Occurred." << endl;
    }catch(...){
        cerr << "An unknown error occurred." << endl;
    }
    return 0;
}
```