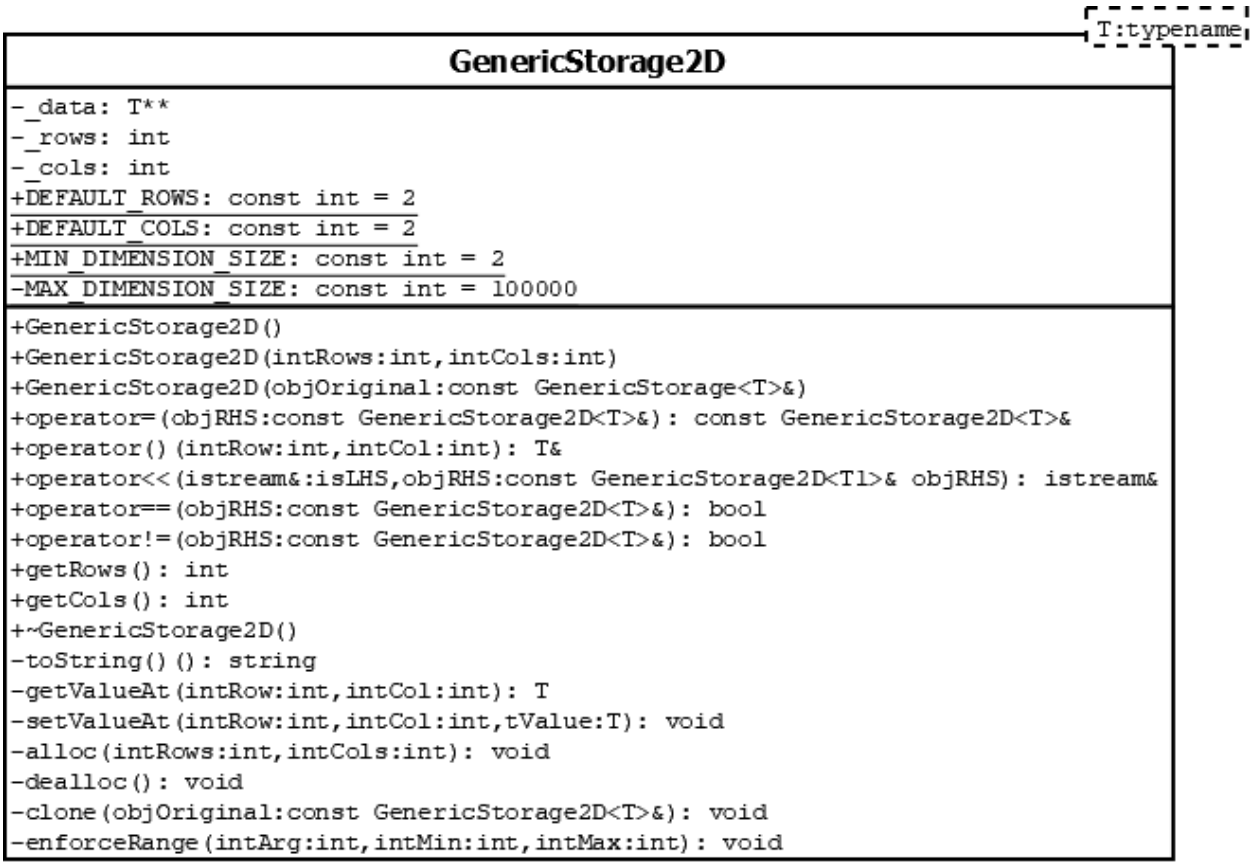
SP SITHUNGU

200000000

PRACTICAL 7 DESIGN



```
#ifndef GenericStorage2D_H
#define GenericStorage2D_H

#include <iostream>
#include <string>

enum ERROR_CODE{
    SUCCESS,
    ERROR_ARGS,
    ERROR_RANGE
};

template <typename T>
class GenericStorage2D{
public:
    GenericStorage2D();
    GenericStorage2D(int intRows, int intCols);
    GenericStorage2D(const GenericStorage2D<T>& objOriginal);

    const GenericStorage2D& operator=(const GenericStorage2D<T>& objRHS);
    T& operator()(int intRow, int intCol);
    template<typename T1>
    friend std::ostream& operator<<(std::ostream& osLHS, const GenericStorage2D<T1>&
objRHS);
    template<typename T1>
    friend std::istream& operator<<(std::istream& osLHS, const GenericStorage2D<T1>&
objRHS);
    bool operator==(const GenericStorage2D<T>& objRHS);
    bool operator!=(const GenericStorage2D<T>& objRHS);

    int getRows() const;
    int getCols() const;

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

    ~GenericStorage2D();
private:
    std::string toString() const;
    int getValueAt(int intRow, int intCol) const;
    void setValueAt(int intRow, int intCol, T tValue);
    void alloc(int intRows, int intCols);
    void dealloc();
    void clone(const GenericStorage2D<T>& objOriginal);
    void enforceRange(int intArg, int intMin, int intMax) const;
    T** _data;
    int _rows;
    int _cols;
};

#include "GenericStorage2D.imp"

#endif // GenericStorage2D_H
```

```cpp
#include <cassert>
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

template <typename T>
GenericStorage2D<T>::GenericStorage2D() : GenericStorage2D(DEFAULT_ROWS,
DEFAULT_COLS){}

template <typename T>
GenericStorage2D<T>::GenericStorage2D(int intRows, int intCols){
    alloc(intRows, intCols);
}

template <typename T>
GenericStorage2D<T>::GenericStorage2D(const GenericStorage2D& objOriginal)
                    : GenericStorage2D(objOriginal._rows, objOriginal._cols){
    clone(objOriginal);
}

template <typename T>
const GenericStorage2D<T>& GenericStorage2D<T>::operator=(const GenericStorage2D<T>&
objRHS){
    if(this != &objRHS){ // Check for self-assignment.
        dealloc();
        alloc(objRHS._rows, objRHS._cols);
        clone(objRHS);
    }
    return *this;
}

template <typename T>
T& GenericStorage2D<T>::operator()(int intRow, int intCol){
    enforceRange(intRow, 0, _rows - 1);
    enforceRange(intCol, 0, _cols - 1);
    return _data[intRow][intCol];
}

template <typename T1>
ostream& operator<<(ostream& osLHS, const GenericStorage2D<T1>& objRHS){
     /*
      * We can use the already existing
      * toString member function from P5.
      */
    osLHS << objRHS.toString() << endl;
    return osLHS;
}

template <typename T1>
istream& operator<<(istream& isLHS, const GenericStorage2D<T1>& objRHS){
    for(int r = 0; r < objRHS._rows; r++){
        for(int c = 0; c < objRHS._cols; c++){
            isLHS >> objRHS._data[r][c];
        }
    }
    return isLHS;
}

template <typename T>
bool GenericStorage2D<T>::operator==(const GenericStorage2D<T>& objRHS){
    if(_rows != objRHS._rows || _cols != objRHS._cols)
        return false;

    for(int r = 0; r < _rows; r++){
```

```
65          for(int c = 0; c < _cols; c++){
66              if(_data[r][c] != objRHS._data[r][c])
67                  return false;
68          }
69      }
70      return true;
71  }
72
73  template <typename T>
74  bool GenericStorage2D<T>::operator!=(const GenericStorage2D<T>& objRHS){
75      if(_rows != objRHS._rows || _cols != objRHS._cols)
76          return true;
77
78      for(int r = 0; r < _rows; r++){
79          for(int c = 0; c < _cols; c++){
80              if(_data[r][c] != objRHS._data[r][c])
81                  return true;
82          }
83      }
84      return false;
85  }
86
87  template <typename T>
88  string GenericStorage2D<T>::toString() const{
89      stringstream ssReturn;
90      for(int r = 0; r < _rows; r++){
91          for(int c = 0; c < _cols; c++){
92              ssReturn << _data[r][c] << ' ';
93          }
94          ssReturn << endl;
95      }
96      return ssReturn.str();
97  }
98
99  template <typename T>
100 int GenericStorage2D<T>::getRows() const{
101     return _rows;
102 }
103
104 template <typename T>
105 int GenericStorage2D<T>::getCols() const{
106     return _cols;
107 }
108
109 template <typename T>
110 int GenericStorage2D<T>::getValueAt(int intRow, int intCol) const{
111     enforceRange(intRow, 0, _rows - 1);
112     enforceRange(intCol, 0, _cols - 1);
113     return _data[intRow][intCol];
114 }
115
116 template <typename T>
117 void GenericStorage2D<T>::setValueAt(int intRow, int intCol, T tValue){
118     enforceRange(intRow, 0, _rows - 1);
119     enforceRange(intCol, 0, _cols - 1);
120     _data[intRow][intCol] = tValue;
121 }
122
123 template <typename T>
124 void GenericStorage2D<T>::alloc(int intRows, int intCols){
125     enforceRange(intRows, MIN_DIMENSION_SIZE, MAX_DIMENSION_SIZE);
126     enforceRange(intCols, MIN_DIMENSION_SIZE, MAX_DIMENSION_SIZE);
127     _rows = intRows;
128     _cols = intCols;
129     _data = new int*[_rows];
130     for(int r = 0; r < _rows; r++){
```

```
131         _data[r] = new int[_cols];
132     }
133 }
134 
135 template <typename T>
136 void GenericStorage2D<T>::dealloc(){
137     assert(_data != nullptr);
138     for(int r = 0; r < _rows; r++){
139         delete [] _data[r];
140     }
141     delete [] _data;
142 }
143 
144 template <typename T>
145 void GenericStorage2D<T>::clone(const GenericStorage2D& objOriginal){
146     for(int r = 0; r < _rows; r++){
147         for(int c = 0; c < _cols; c++){
148             _data[r][c] = objOriginal._data[r][c];
149         }
150     }
151 }
152 
153 template <typename T>
154 void GenericStorage2D<T>::enforceRange(int intArg, int intMin, int intMax) const{
155         if(intArg < intMin || intArg >intMax){
156             cerr << intArg << " must be in [" << intMin
157                  << ", " << intMax << "]" << endl;
158             exit(ERROR_RANGE);
159         }
160 }
161 
162 template <typename T>
163 GenericStorage2D<T>::~GenericStorage2D(){
164     dealloc();
165 }
```

```cpp
#include "GenericStorage2D.h"

#include <iostream>

using namespace std;

int rangedRandom(int intLB, int intUB);

int main()
{
    // Testing the fully parameterised constructor.
    GenericStorage2D<int> objStorage(10, 10);
    // Testing the copy constructor.
    GenericStorage2D<int> objCopy = objStorage;
    cout << "objCopy == objStorage? ";
    cout << (objCopy == objStorage ? "Yes" : "No") << endl;
    //Testing the indexing operator.
    for(int r = 0; r < objCopy.getRows(); r++){
        for(int c = 0; c < objCopy.getCols(); c++){
            objCopy(r, c) = rangedRandom(0, 8);
        }
    }
    // Testing the overloaded assignment operator.
    GenericStorage2D<int> objSmallerSizedStorage(2, 2);
    objSmallerSizedStorage = objCopy;
    cout << "objSmallerSizedStorage != objCopy? ";
    cout << (objSmallerSizedStorage != objCopy ? "Yes" : "No") << endl;
    // Testing the stream insertion operator.
    cout << objSmallerSizedStorage << "Done!" << endl;
    return 0;
}

int rangedRandom(int intLB, int intUB){
    return rand() % (intUB - intLB + 1) + intLB;
}
```