CSC 122 001 Computer Science I
Julius Ranoa

Chapter 16 Programming Challenge 9 *SortableVector* Class Template
Write a class template named *SortableVector*. The class should be derived from the *Simple-Vector* class presented in Chapter 16. It should have a member function that sorts the array elements in ascending order. (Use the sorting algorithm of your choice.) Test the template in a driver program.

Screenshots of runtime.

```
Vector of random integers:
55  93  678  684  676  32  225  424  748  201

... sorting the vector ...

The vector after sorting:
32  55  93  201  225  424  676  678  684  748
```

Files Included:

(1) main.cpp
(2) SortableVector.h
(3) SimpleVector.h *(This is an example from the book. Only one line, highlighted below, is modified)*

main.cpp

```cpp
#include <iostream>
#include "SortableVector.h"

int main() {
    SortableVector<int> s(10);
    srand(time(NULL));
    for (int i = 0; i < s.size(); i++) {
        s[i] = rand() % 1000 + 1;
    }
    cout << "Vector of random integers: \n";
    s.print();

    cout << "\n";
    cout << "... sorting the vector ... \n";
    cout << "\n";
    s.sort();

    cout << "The vector after sorting: \n";
    s.print();
    return 0;
}
```

SortableVector.h

```
#ifndef CH16_PR9_SORTABLEVECTOR_CLASS_TEMPLATE_SORTABLEVECTOR_H
#define CH16_PR9_SORTABLEVECTOR_CLASS_TEMPLATE_SORTABLEVECTOR_H

// Header file copied from book example, per requirement.
#include "SimpleVector.h"

template <class T>
class SortableVector : public SimpleVector<T> {

public:
    SortableVector(int size) : SimpleVector<T>(size) {} // Constructor
    SortableVector(const SortableVector &v) : SimpleVector<T>(v) {}
    void sort();

};

template <class T>
void SortableVector<T>::sort() {
    T min_val;
    int min_idx;

    for (int i = 0; i < this->arraySize; i++) {
        min_idx = i;
        min_val = this->aptr[i];
        // Getting smallest value.
        for (int j = i + 1; j < this->arraySize; j++) {
            if (this->aptr[j] < min_val) {
                min_idx = j;
                min_val = this->aptr[j];
            }
        }
        // Swapping if smallest value is not
        // in the beginning.
        if (min_val != (this->aptr)[i]) {
            std::swap(this->aptr[min_idx], this->aptr[i]);
        }
    }
}

#endif //CH16_PR9_SORTABLEVECTOR_CLASS_TEMPLATE_SORTABLEVECTOR_H
```

SimpleVector.h

```
#include <iostream>
#include <cstdlib>
#include <memory>
using namespace std;

// Exception for index out of range
struct IndexOutOfRangeException
{
    const int index;
    IndexOutOfRangeException(int ix) : index(ix) {}
};

template <class T>
class SimpleVector
{
protected: // CHANGED THIS FROM PRIVATE
    unique_ptr<T []> aptr;
    int arraySize;
public:
    SimpleVector(int);                    // Constructor
    SimpleVector(const SimpleVector &);   // Copy constructor

    int size() const { return arraySize; }
    T &operator[](int);                   // Overloaded [] operator
    void print() const;                   // outputs the array elements
};

//*********************************************************
// Constructor for SimpleVector class. Sets the size     *
// of the array and allocates memory for it.             *
//*********************************************************
template <class T>
SimpleVector<T>::SimpleVector(int s)
{
    arraySize = s;
    aptr = make_unique<T[]>(s);
    for (int count = 0; count < arraySize; count++)
        aptr[count] = T();
}
//*********************************************************
// Copy Constructor for SimpleVector class.              *
//*********************************************************
template <class T>
SimpleVector<T>::SimpleVector(const SimpleVector &obj)
{
    arraySize = obj.arraySize;
    aptr = make_unique<T[]>(obj.arraySize);
    for (int count = 0; count < arraySize; count++)
        aptr[count] = obj[count];
}


//*********************************************************
// Overloaded [] operator. The argument is a subscript. *
```

```cpp
// This function returns a reference to the element     *
// in the array indexed by the subscript.               *
//******************************************************
template <class T>
T &SimpleVector<T>::operator[](int sub)
{
    if (sub < 0 || sub >= arraySize)
        throw IndexOutOfRangeException(sub);
    return aptr[sub];
}
//******************************************************
// prints all the entries is the array.                 *
//******************************************************
template <class T>
void SimpleVector<T>::print() const
{
    for (int k = 0; k < arraySize; k++)
        cout << aptr[k] << "  ";
    cout << endl;
}
```