**UML**

**Square.h**

#include "Transform.h"

class Square: public Transform

{

public:

Square() = default;

//Override

int transform(int value);

};

int Square::transform(int value)

{

return value\*value;

}

**Sucessor.h**

#include "Transform.h"

class Sucessor: public Transform

{

public:

Sucessor() = default;

//Override

int transform(int value);

};

int Sucessor::transform(int value) {

return value + 1;

}

**Transform.h**

#ifndef \_\_TRANSFORM\_H\_\_

#define \_\_TRANSFORM\_H\_\_

class Transform

{

public:

virtual int transform(int value)=0;

};

#endif

Table.h

#include "Transform.h"

#include <vector>

class Table{

private:

std::vector<int> \_vector; // vector of integers

int \_size;

public:

Table(int size);

virtual ~Table();

int getSize() const;

void setValue(int position, int value);

void setAll(int value);

virtual void print(Transform \*t) const;

};

**Table.cpp**

#include "Table.h"

#include "Sucessor.h"

#include "Square.h"

#include <iostream>

Table::Table(int size) {\_vector.reserve(size); \_size=size;}

Table::~Table(){};

// int Table::getSize() const { return \_size; };

void Table::setValue(int position, int value)

{

\_vector[position] = value;

}

int Table::getSize() const { return \_size; }

void Table::setAll(int value)

{

for(int position=0;position<getSize();position++){

\_vector[position] = value;

}

}

void Table::print(Transform \*t) const

{

int position;

std::cout << "<";

for(position = 0; position<getSize()-1;position++)

std::cout << t->transform(\_vector[position]) << ", ";

std::cout << t->transform(\_vector[position]) << ">" << std::endl;

}

int main() {

Table table(10);

table.setAll(10);

table.setValue(9,9);

Sucessor s1;

Square s2;

table.print(&s1);

table.print(&s2);

}