# Exercise 1: Generic Point

Up till now our Point class has been based on “int”. Make a generic Point class that could be based on anything … ints, floats, or even strings. Test-drive a few types with a main function.

Make the methods of Point virtual for future expansion.

# Exercise 2: Point Decoration

Create a function “doublePoint” that takes a Point<int> reference. This function should use the Point “getters” and “setters” to double the X,Y values.

Create a Point<int> decorator that limits the “setter” values of X,Y to 5,5.

* Make the PointLimit class that inherits from Point<int>
* Add a “Point<int>& target” private member and a constructor to set it
* Add all Point<int> methods and have them relay to the target
* Modify the setters to limit X and Y to 5 and 5

#include <iostream>

#include <string>

#include <vector>

#include <map>

using namespace std;

class ArrayInt {

int\* data;

int size;

public:

ArrayInt() {

data = 0;

size = 0;

}

~ArrayInt() {

delete [] data;

}

int getSize() {

return size;

}

int getValue(int index) {

return data[index];

}

void add(int value) {

int\* newData = new int[size+1];

for(int x=0;x<size;++x) {

newData[x] = data[x];

}

delete [] data;

data = newData;

data[size] = value;

size = size+1;

}

void remove(int index) {

int\* newData = new int[size-1];

for(int x=0;x<index;++x) {

newData[x] = data[x];

}

for(int x=index+1;x<size;++x) {

newData[x-1] = data[x];

}

delete [] data;

data = newData;

size = size - 1;

}

};

template <class MY>

class Array {

MY\* data;

int size;

public:

Array() {

data = 0;

size = 0;

}

~Array() {

delete [] data;

}

int getSize() {

return size;

}

MY getValue(int index) {

return data[index];

}

void add(MY value) {

MY\* newData = new MY[size+1];

for(int x=0;x<size;++x) {

newData[x] = data[x];

}

delete [] data;

data = newData;

data[size] = value;

size = size+1;

}

void remove(int index) {

MY\* newData = new MY[size-1];

for(int x=0;x<index;++x) {

newData[x] = data[x];

}

for(int x=index+1;x<size;++x) {

newData[x-1] = data[x];

}

delete [] data;

data = newData;

size = size - 1;

}

};

class Driveable {

public:

virtual ~Driveable() {}

virtual void start() = 0;

virtual void stop() = 0;

virtual void moveTo(int x, int y) = 0;

};

class Car : public Driveable {

public:

virtual void start() {

cout << "Car starting" << endl;

}

virtual void stop() {

cout << "Car stoping" << endl;

}

virtual void moveTo(int x, int y) {

cout << "Car movingTo " << x << "," << y << endl;

}

};

class Decorator : public Driveable {

Driveable& target;

public:

Decorator(Driveable& \_target) : target(\_target) {

}

virtual void start() {

cout << "CHECK IT OUT:";

target.start();

}

virtual void stop() {

cout << "CHECK IT OUT:";

target.stop();

}

virtual void moveTo(int x, int y) {

cout << "CHECK IT OUT:";

target.moveTo(x,y);

}

};

void park(Driveable& drive) {

drive.start();

drive.moveTo(5,6);

drive.stop();

}

template <class TYPE>

class Point {

TYPE x;

TYPE y;

public:

Point() {}

Point(TYPE \_x, TYPE \_y) : x(\_x), y(\_y) {}

virtual ~Point() {}

virtual TYPE getX() {return x;}

virtual TYPE getY() {return y;}

virtual void setXY(TYPE ix, TYPE iy) {x=ix;y=iy;}

virtual TYPE getSpecial() {return x+y;}

};

class LimitPoint : public Point<int> {

Point<int>& target;

public:

LimitPoint(Point<int>& it) : target(it) {}

virtual int getX() {return target.getX();}

virtual int getY() {return target.getY();}

virtual void setXY(int x, int y) {

if(x>5) {x=5;}

if(y>5) {y=5;}

target.setXY(x,y);

}

virtual int getSpecial() {return target.getSpecial();}

};

void movePoint(Point<int>& p) {

p.setXY(20,40);

cout << "Changed to " << p.getX() << "," << p.getY() << endl;

}

void main() {

Point<string> stringPoint("First", "Last");

cout << stringPoint.getX() << " " << stringPoint.getY() << " "

<< stringPoint.getSpecial() << endl;

Point<int> a(1,1);

movePoint(a);

Point<int> b(1,1);

movePoint(LimitPoint(b));

cout << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

Car car;

park(car);

park(Decorator(car));

cout << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

ArrayInt ar;

ar.add(100);

ar.add(101);

ar.add(102);

ar.remove(1);

Array<int> art;

art.add(200);

art.add(201);

art.add(202);

art.remove(1);

Array<string> sar;

sar.add("Hello");

sar.add("There");

sar.add("World");

sar.remove(1);

cout << sar.getSize() << " " << sar.getValue(1) << endl;

cout << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

vector<string> vstr;

vstr.push\_back("Hello");

vstr.push\_back("There");

vstr.push\_back("World");

vector<string>::const\_iterator it;

it = vstr.begin();

++it;

vstr.erase(it);

for(it=vstr.begin();it!=vstr.end();++it) {

cout << \*it << endl;

}

cout << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

map<string,string> myMap;

myMap["Chris"] = "1234";

myMap["Jane"] = "4444";

myMap["Bob"] = "9876";

cout << myMap["Jane"] << endl;

map<string,string>::iterator mit = myMap.find("Jill");

if(mit==myMap.end()) {

cout << "Jill not found" << endl;

}

}