# Exercise 1: Valet Parking

Use the Vehicle base class given. Add a Truck that blows its horn (cout) before it moves. Add a Convoy that is a group of four Trucks. The start/stop/move methods should be relayed to all members of the Convoy at the same time.

Create a valetParking function that takes a Vehicle and moves it into the parking deck (coordinates 5,5). Be sure to start the vehicle before moving it and stop it afterwards. Add couts to the various methods and constructors/destructors to see what is going on.

#include <iostream>

using namespace std;

class Animal {

int legs;

public:

Animal(int numLegs) {

legs = numLegs;

}

void speak() {

cout << "Plain animal" << endl;

}

int getNumberOfLegs() {

return legs;

}

};

class Dog : public Animal

{

public:

Dog() : Animal(4) {}

void speak() {

cout << "This is a dog: woof woof!" << endl;

}

};

class Cat : public Animal

{

public:

Cat() : Animal(3) {}

void speak() {

cout << "The cat says meeeooowwwww!" << endl;

}

};

class Snake : public Animal

{

public:

Snake() : Animal(0) {}

void speak() {

cout << "This is a snake: o~~~~~~~" << endl;

}

};

void talk(Animal& animal) {

animal.speak();

cout << "It has " << animal.getNumberOfLegs() << " legs." << endl;

animal.speak();

cout << endl;

}

class Point {

private:

int x;

int y;

public:

Point(int \_x, int \_y) {

x = \_x;

y = \_y;

}

virtual ~Point() {}

virtual int getX() {

return x;

}

virtual int getY() {

return y;

}

virtual float getLength() {

float a = getX(); // x;

float b = getY(); // y;

return sqrt(a\*a + b\*b);

}

};

class RandomPoint : public Point

{

char\* useless;

public:

RandomPoint() : Point(0,0) {

useless = new char[1024];

}

virtual ~RandomPoint() {

delete [] useless;

}

virtual int getX() {

return rand()%10 + 1;

}

virtual int getY() {

return rand()%10 + 1;

}

};

class Vehicle {

protected: // Protected ... people who derive have access too

int x; // In our system, most forseeable vehicles have X,Y ...

int y; // ... more convienient to specify here

public:

Vehicle() {

cout << "Vehicle constructor" << endl;

}

virtual ~Vehicle() {

cout << "Vehicle destructor" << endl;

}

// Getters and setters ... virutal? How do you see this being used

virtual int getX() {return x;}

virtual int getY() {return y;}

virtual void setX(int \_x) {x=\_x;}

virtual void setY(int \_y) {y=\_y;}

// Virtual functions. No implementations in base class;

virtual void start() = 0;

virtual void stop() = 0;

virtual void moveTo(int x, int y) {

x = x;

y = y;

}

// NOT virtual

void startMoveStop(int x, int y) {

start();

moveTo(x,y);

stop();

}

};

class Car : public Vehicle {

private:

string make;

public:

virtual void start() {

cout << "Car starting" << endl;

}

virtual void stop() {

cout << "Car stopping" << endl;

}

virtual void setX(int \_x) {

if(x==\_x) {

cout << "X didn't change" << endl;

return;

}

//x = \_x;

Vehicle::setX(\_x);

}

virtual string getMake() {

// Return reference? Pointer?

return make;

}

};

class Plane : public Vehicle {

protected:

int z;

public:

virtual void start() {

cout << "Plane starting" << endl;

}

virtual void stop() {

cout << "Plane stopping" << endl;

}

virtual void moveTo(int \_x, int \_y) {

cout << "Taking to the sky!" << endl;

Vehicle::moveTo(\_x, \_y);

}

};

class WrongWay : public Plane {

public:

virtual void moveTo(int \_x, int \_y) {

\_x = rand()%10 +1;

\_y = rand()%10 +1;

Vehicle::moveTo(\_x, \_y);

}

};

class Truck : public Vehicle {

public:

virtual void start() {

cout << "Truck starting" << endl;

}

virtual void stop() {

cout << "Truck stopping" << endl;

}

virtual void moveTo(int \_x, int \_y) {

cout << "Toot Toot! East bound and down, good-buddy!" << endl;

Vehicle::moveTo(\_x, \_y);

}

};

class Convoy : public Vehicle {

private:

Truck truckA;

Truck truckB;

Truck truckC;

Truck truckD;

public:

Convoy() {

cout << "Making a Convoy!";

}

virtual void start() {

truckA.start();

truckB.start();

truckC.start();

truckD.start();

}

virtual void stop() {

truckA.stop();

truckB.stop();

truckC.stop();

truckD.stop();

}

virtual void moveTo(int \_x, int \_y) {

truckA.moveTo(\_x,\_y);

truckB.moveTo(\_x,\_y);

truckC.moveTo(\_x,\_y);

truckD.moveTo(\_x,\_y);

}

};

void valetPark(Vehicle& vehicle) {

cout << "Parking a vehicle." << endl;

vehicle.start();

vehicle.moveTo(5,5);

vehicle.stop();

cout << "Vehicle parked." << endl;

}

void main() {

Car car;

Plane plane;

//Truck truck;

Convoy convoy;

valetPark(car);

valetPark(plane);

valetPark(convoy);

RandomPoint rp;

cout << rp.getX() << " " << rp.getX() << " " << rp.getX() << endl;

cout << rp.getLength() << endl;

// Notice that x,y are useless for point. Users don't care about the guts. They want the interface.

// You the designer may want to pick up the structure. Trade off.

// Make X and Y public ... speed in a GUI framework. Might be better to make "getLength" in RandomPoint.

// Add other interfaces ... pure virtual ... multiple inheritance

}