COM S 213 – Fall 2004

Assignment #4

The Used Car Dealership – Expands!

Due September 23, 2004

The main purpose of this assignment is to experience *polymorphism*—the ability to associate many meanings to one function name by means of virtual member functions.

Our favorite used car dealership is going to expand. Selling cars isn't nearly gratifying enough, this dealership will be selling Bicycles and Boats as well. This dealership would like to keep an inventory of its vehicles (Bikes, Boats and Cars), and would like to store the following information about each:

Bicycles	**Boats**	**Automobiles**
Make	Make	Make
Model	Model	Model
Year	Year	Year
Price	Price	Price
Brake Type	Engine Type	Mileage
Tire Size	Length	Has Antilock Breaks?
# of Speeds	Registration Number	Has Four Wheel Drive?

Simply, you are to write a program which allows these items to be stored in a Dealership class in the form of a dynamic array (similar to Assignment #3) which will allow the inventory to be stored and printed on demand.

Tasks

You will need to abstract out the common fields associated with each of the vehicles above into a base class called `Vehicle`. You will have three classes derived from the `Vehicle` class: `Boat`, `Bicycle` and `Automobile`. Finally, you will need a `Dealership` class similar to the class you wrote in Assignment #3.

For this assignment I will spell out how I'd like these classes designed—in future assignments you will need to do this part yourself!

Here are the four classes:

Class `Vehicle`	
MEMBER FUNCTIONS	
Vehicle()	*Constructor*, allows member fields below to be initialized when the object is created by taking parameters.

printVehicleType()	*Pure virtual*, prints out (through implementation in each of the derived classes) where the item is a "Boat", "Bicycle" or "Automobile"
printDescription()	*Virtual,* used to print out make, model, year and price on one line
MEMBER VARIABLES	
Make	What car manufacturer builds this vehicle?
Model	What model of car is this vehicle?
Year	What year was this vehicle produced for?
Price	What is the asking price of this vehicle?

Class `Boat`	
MEMBER FUNCTIONS	
Boat()	*Constructor*—takes initial values for *all* member variables (including member variables from the base class) and puts those initial values in the right member variables.
printVehicleType()	Prints out "Boat"
printDescription()	Calls Vehicle::printDescription to print out make, model, year, and cost. Then prints out specific information about the boat (length, engine type and registration number)
MEMBER VARIABLES	
Length	How long is this boat?
Engine	What type of engine does the boat have?
Registration	What is the boat's call numbers/letters?

Class `Bicycle`	
MEMBER FUNCTIONS	
Bicycle()	*Constructor*—takes initial values for *all* member variables (including member variables from the base class) and puts those initial values in the right member variables.
printVehicleType()	Prints out "Bicycle"
printDescription()	Calls Vehicle::printDescription to print out make, model, year, and cost. Then prints out specific information about the bicycle (# of speeds, brake type and tire size)
MEMBER VARIABLES	
Speeds	How many speeds does this bike have?
mBrakeType	What kind of brakes does this bike have?
TireSize	How big are the tires?

Class `Automobile`	
MEMBER FUNCTIONS	
Automobile()	*Constructor*—takes initial values for *all* member variables (including member variables from the base class) and puts those initial values in the right member variables.
printVehicleType()	Prints out "Automobile"
printDescription()	Calls Vehicle::printDescription to print out make, model, year, and cost. Then prints out specific information about the car mileage and whether or not the car has antilock brakes and whether or not the car has four wheel drive)
MEMBER VARIABLES	
Mileage	How many miles are on the car
ABS	A `bool` value (either true or false) which states if the vehicle has antilock brakes!
4WD	A `bool` value (either true or false) which states if the vehicle has four wheel drive.

Finally, we have the Dealership class:

Class `Dealership`	
MEMBER FUNCTIONS	
Dealership(size)	*Constructor*—takes one parameter which is the size of inventory to be allocated. *You do not need a default constructor*.
addToInventory()	Takes a Vehicle pointer as sole argument, adds it to the next open slot.
displayInventory()	Displays entire inventory
freeInventory()	Frees all dynamically allocated objects.
MEMBER VARIABLES	
Vehicles	An array of *pointers* to the Vehicle class.
numVehicles	The maximum number of vehicles in our inventory.
VehicleIndex	The number of vehicles added to the inventory.

You are to construct the classes above such that your main() program can consist of this logic:

```
int main (int argc, char * const argv[]) {
    // Declare a dealership
    Dealership ronsVehicles(10);

    // Add vehicles
    ronsVehicles.addToInventory(new
```

```
Automobile("Honda","Accord","1984",6999.99,150000,false,false));
    ronsVehicles.addToInventory(new
Automobile("Dodge","Caravan","1990",7999.99,124001,false,false));
    ronsVehicles.addToInventory(new Automobile("Ford","F-
150","1994",4999.99,89934,false,true));
    ronsVehicles.addToInventory(new
Automobile("Mercury","Mountaineer","2003",40999.04,10554,true,true));
    ronsVehicles.addToInventory(new
Bicycle("Huffy","BMXcellent","2003",69.99,10,"top pull",26));
    ronsVehicles.addToInventory(new
Bicycle("Schwinn","Varsity","1982",129.99,10,"side pull",27));
    ronsVehicles.addToInventory(new
Bicycle("NoBrand","MyFirstBike","2002",29.99,1,"coasting",16));
    ronsVehicles.addToInventory(new Boat("Sunfish","Sail-
1","2000",1299.99,15,"none","NY S45311"));
    ronsVehicles.addToInventory(new Boat("Jeep Boats","Fisherman's
Dream","1998",10999,29,"Binford XL2","NY JU71123"));


    // Display inventory
    ronsVehicles.displayInventory();

    ronsVehicles.freeInventory();
    return 0;
}
```

and the output will look something like this:

```
DEALER INVENTORY
================================================
1. AUTOMOBILE
1984 Honda Accord $6999.99
150000 miles

2. AUTOMOBILE
1990 Dodge Caravan $7999.99
124001 miles

3. AUTOMOBILE
1994 Ford F-150 $4999.99
89934 miles , 4 Wheel Drive

4. AUTOMOBILE
2003 Mercury Mountaineer $40999
10554 miles , Antilock Brakes, 4 Wheel Drive

5. BICYCLE
2003 Huffy BMXcellent $69.99
10 speed, top pull brakes, 26 inch tires

6. BICYCLE
1982 Schwinn Varsity $129.99
10 speed, side pull brakes, 27 inch tires
```

```
7. BICYCLE
2002 NoBrand MyFirstBike $29.99
1 speed, coasting brakes, 16 inch tires

8. BOAT
2000 Sunfish Sail-1 $1299.99
15 feet, none engine, registration: NY S45311

9. BOAT
1998 Jeep Boats Fisherman's Dream $10999
29 feet, Binford XL2 engine, registration: NY JU71123
```

Hints

There's some trickiness in the `Dealership` class while the other classes should be fairly straight forward.

First, you need to store an array of objects which represent the inventory, but you will be putting an instance of one of three derived classes in each array element. Remember that an array must be declared to be of *one* type only. So, to do this, you will need to create an array of *pointers to the base class* (Vehicle) to store the inventory. To declare an array of pointers, you would use the following expression:

```
Vehicle **arrayOfPointers;
```

And to allocate memory to it (dynamically), you would use the following expression:

```
arrayOfPointers = new (Vehicle *)[size];
```

where `size` is an integer representing the size of the array you wish to dynamically allocate.

The second hint is that you will be using a constructor to do the dynamic memory allocation in the `Dealership` class. Since a constructor can never be called directly once the object is created, you won't need logic to check to see if there was previously allocated memory in this constructor. This makes the constructor much, much simpler. Since a constructor cannot return an error code, it also means that you cannot check to see if the memory allocation failed and do anything to let the rest of your program know that a problem occurred. Because of this you must make sure that you always check to see if the dynamic array pointer is NULL before accessing it elsewhere in your program. Since the allocation is supposed to occur when the `Dealership` class is created, you can terminate your program if the dynamic array is ever found to be a NULL pointer.

The third hint is that your `freeInventory()` method must remember to free *all* memory that was dynamically allocated. Remember that the pointers to objects stored in the inventory array were dynamically allocated as well as the array itself.

This assignment is obviously the most complex assignment we've done, but it also the most important. The concept of polymorphism is one of the main reasons for having an object oriented language so it is important that you understand it!

Please come to office hours or make an appointment if you need me to clarify anything in this assignment!