EXERCISE 5, INHERITANCE – TOWARDS POLYMORPHISM

# Exercise 5, Inheritance – towards polymorphism

## Objective

The primary objective for this lab is to enable you to derive new types and to override functionality.

## Overview

Read the instructions below and critically evaluate each code sample. Trust nothing and make comments that will be reviewed as a class.

### Part 1 – Testing Racing Cars

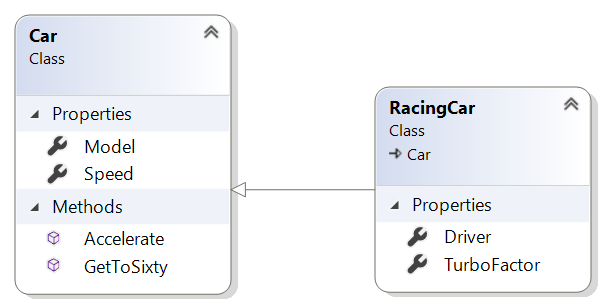
### Scenario

You are going to work with a project which consists of a Car class, a RacingCar class and a Program (test) class that fills a Car[] array with Cars and RacingCars objects.   
  
The Program class will then process each Car, setting their initial speeds to 60MPH (this is done using a blatant cheat that normal cars don’t usually possess – a GetToSixty() method). The test will then continue by making each ‘Car’ Accelerate for 2 seconds before writing their model and speed to the Console.   
  
If the car is a RacingCar we will need to add some additional code to the Program class so that a little extra information is written out.

### Step by step

1. Open the Labs project
2. Create a new package called **lab05**
3. Add a class called **Program** with a **Main()** method to the lab05 package.
4. Add a class called Car to the new package.
5. Add another class called RacingCar. This class extends Car
6. The Car and RacingCar classes will have the fields and methods as displayed in the class diagram below.
7. You will also need to add constructors for the two class.

Please study the diagram below before writing code.



1. Add code to the Car class.

Create property methods for the **model** and **speed** fields.  
The **GetToSixty()** method just sets the **speed** field to 60.

The accelerate method should look like **void** **Accelerate**(**int** *seconds*)  
and increase the speed by *5 \* seconds*.

Create a suitable constructor for the Car class.

1. Add extra code to the RacingCar class.

Add a **string** **Driver** property method.

Add an **int** TurboFactor property method.

1. You'll have to create a suitable constructor for the RacingCar class.
2. The **Accelerate()** method of the RacingCar will invoke the base class (Car) Accelerate method and then multiply the **speed** by the **TurboFactor**.

Tip: use the **base**.**Accelerate()** method.

1. The **Main()** method should create an array of Cars comprising of a few cars and racing cars.
2. The Main() method should then pass the **cars** array to a method called **ProcessCars**.   
     
   The ProcessCars method currently:
3. Gets each ‘Car’ up to 60MPH as a start point
4. Then accelerates each Car for 2 seconds
5. And displays details of each car.
6. You'll also display the driver's name but only if a Car in the array is a RacingCar.

* You'll need to examine the type of car in the array and if it is a RacigCar, cast it to a RacingCar in order to get the driver's name.

**Tip:** use the **is** keyword: if (**c** **is** *RacingCar*) {….}  
 where '**c'** is a car element in the array.

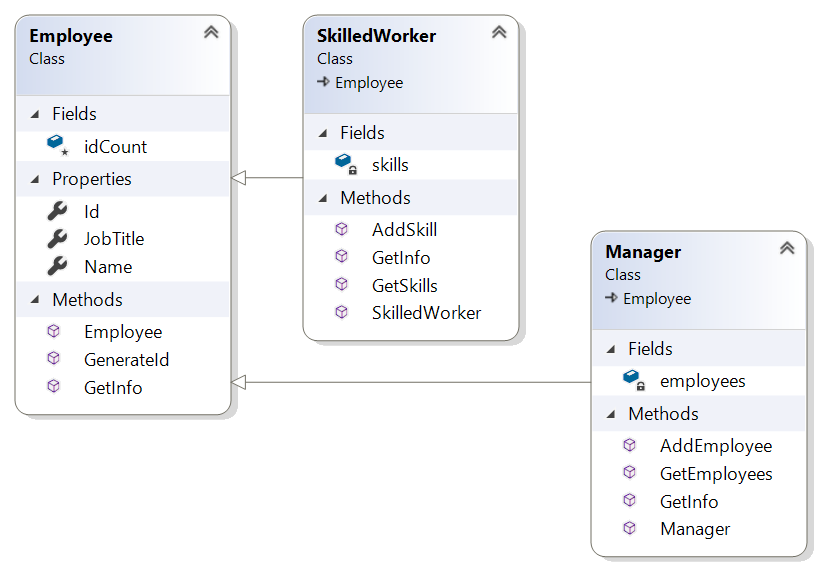
### Part 2 – Employee Hierarchy

### Scenario

In these labs, you will design and create a class hierarchy for an employee tracking system.

### Designing the hierarchy

1. Create the following classes. The employee class is provided below. Just copy and paste. We'll need you to concentrate on the other two classes.   
   **Please note** that any method shown below with the same name as the class is the **constructor** for that class.



1. The **Manager** class holds a **List<Employee>** called **employees**.  
   A Manager is also an Employee.
2. The **AddEmployee(Employee emp)** method adds the **emp** object to the **employees** List.
3. The **GetInfo()** method of the Manager should first gather the manager's details (using base.GetInfo()) and then use a for-loop to go through the **employees** List in order to call their GetInfo() method. It should then return the resulting string.
4. The **SkilledWorker** has a **List** <String> called **skills** which holds the names of skills possessed by a SkilledWorker instance.
5. The **AddSkill(String skill)** method adds the **skill** String to the **skills** ArrayList.
6. In the main() method
7. Create a Manager instance
8. Create a few regular Employee instances and add them to the manager's employees ArrayList
9. Create a SkilledWorker object with a few skills and then add the instance to the Manager's employee ArrayList.
10. Call the manager's GetInfo() method and display the resulting String.

**Question:** Can we add a manager instance to a manager's employees?

\*\*\* END \*\*\*

To get you started here is the code for the Employee class:

**public** **class** Employee {

**protected** **static** **int** *idCount*;

**public** string Name { get; set; }

**public** string JobTitle { get; set; }

**public** int ID { get; **private** set; }

**public** Employee(string name, string jobTitle ) {

ID = ++idCount \* 10;

Name = name;

JobTitle = jobTitle;

}

**public** **virtual** **String** getInfo() {

string info = "\n\*\*\*\* \*\*\*\*\*\n";

info += "Name: " + Name + "\n";

info += "Job Title: "+ JobTitle + "\n";

info += "Employee ID: "+ ID + "\n";

return info;

}

}