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 – Racing Cars

### Scenario

You are going to work with a project which consists of a Car class, a RacingCar class. Your code will then fills a Car[] array with Car and RacingCar objects.   
  
Your code 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).   
  
Your code will then continue by making each ‘Car’ Accelerate for 2 seconds before writing their model and speed to the Console Window.   
  
If the car is a RacingCar we will need to add some additional code so that a little extra information is written out.

### Step by step

1. Open the **Labs** Console project
2. Create a new static method in the Program class, called **Lab8()**
3. Call this method from the Main() method, commenting out any other calls.
4. Add a class called **Car** to the Console app project.
5. Add another class called **RacingCar**. This class extends (inherits from) 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.  
Please also note that any method shown below with the same name as the class is the constructor for that class.

Diagram

Description automatically generated

1. Add code to the **Car** class.

Create getters and setters for the **model** and **speed** fields.  
The **GetToSixty()** method just sets the **speed** to 60.

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

1. Create a suitable constructor for the Car class to set the model.
2. Add a method called GetDetails() to return the speed and the model of the field values.
3. In the RacingCar class.
   1. Add a new string property **procedure** called **Driver**
   2. Add a new public integer property procedure as **TurboFactor**..
4. You'll have to create a suitable constructor for the RacingCar class to set the driver’s name as well as the model.
5. Create a method in the RacingCar class called Race() which just prints a message like “Racing!”
6. The **Accelerate()** method will invoke the base class (Car) Accelerate method and then multiply the **speed** by the **turboFactor**.

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

1. Override the GetDetails of the super class (Car) to return the name of the driver as well.
2. The **Lab8()** method should create an array of Cars comprising of a few cars and racing cars.
3. The Lab8() method should then pass the cars array to a method called **ProcessCars**.   
     
   The ProcessCars method currently:

* Gets each ‘Car’ up to 60MPH as a start point
* Then Accelerates each Car for 2 seconds
* And displays details of each car.
* You'll also display the driver's name but only if a Car is of type RacingCar.

You'll need to examine the type of car in the array and if it is a RacigCar, cast it to a RacingCar to get the driver's name.  
  
**Tip:** use **is operator** like: if (car **is** *RacingCar*) {….}  
 where '**car'** is a car element in the array of Cars.

1. Replace the **GetDetails()** method with an override of the Object class’s **ToString()** method. All Objects have a ToString() method which you can use to get details of an object.

## 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.

Graphical user interface, diagram, application

Description automatically generated

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

{

private int id;

protected static int idCount;

public int Id { get; set; }

public string Name { get; set; }

public string JobTitle { get; set; }

public Employee(string name, string jobTitle)

{

Id = ++Employee.idCount \* 10;

Name = name;

JobTitle = jobTitle;

}

public String GetDetails()

{

return $"\n\*\*\*\* Name: {Name}\nJob Title: {JobTitle}\nEmployee ID: {Id}\n";

}

}

The Manager class holds a List<Employee> called **employees**.

1. The **AddEmployee(Employee emp)** method adds the **emp** object to the **employees** List.
2. The **GetDetails()** method of the Manager should first gather the manager's details (using **base**.GetDetails()) and then use a for-loop to go through the **employees** List in order to call their GetDetails() method.  
   It should then return the resulting string.
3. The **SkilledWorker** has an **List**<string> called **skills** which holds the names of skills possessed by a SkilledWorker instance.
4. The **AddSkill(String skill)** method adds the **skill** String to **skills**.
5. In the **Lab8()** method
   1. Create a Manager instance
   2. Create a few regular Employee instances and add them to the manager's employees List
   3. Create a SkilledWorker object with a few skills and then add the instance to the Manager's employee List.
   4. Call the manager's GetDetails() method and display the result.

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

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