LAB 11, COLLECTIONS AND GENERICS

FUNDAMENTALS

Lab 11, C# - Collections and Generics

Objective

In this lab you will practice using generic List, Queues and in a separate task, you'll work with a Dictionary.

Part 1 – Use a List<Shape>

1. Open the bouncing shape exercise that you did in the previous lab.
2. Change the array of Shapes to List<Shape>.
   Tip: you'll need to use the List's Add() method in the Form Load event to add the three shapes;

Part 2 - Using a Queue and a Stack

1. Create a new Console application called Lab11.
2. Add a class called ShoppingBasket with following fields
   string productName.
   int quantity.
   double price
3. Create a constructor for the ShoppingBasket class to set all the above fields.
4. Create a method called DisplayDetails() to display the fields' values.
5. Add another class called QAShop to simulate an ecommerce site.
6. In the QAShop class, create a private Queue of ShoppingBasket called baskets (as a field).
7. Create a method in QAShop called Buy()as:

   ```
   public void Buy(ShoppingBasket basket)
   ```

   This method adds the basket reference to the baskets queue.

8. Call the Buy() method from within Main() two or three times.
9. To process the baskts you'll write another method as:
   `public void ProcessBaskets()`

   This method removes each item from the baskets queue and then calls the item's DisplayDetails() method.

   This method simulates the processing of shopping baskets for payment and shipping. For simplicity we just investigate the queue actions.
10. Call the ProcessBaskets() method in Main().
11. Run and test your code.

Part 3 Using Dictionary<K,V>

Scenario: A Zoo has a number of current animals and is expecting new arrivals soon. They wish to keep track of which animal types they have and record the count of each animal type.

You will create Dictionary<string, int>. The key of String to store animal type and a value of Integer for the count of occurrences (Lion 3, Zebra 2 etc).

Step by step instructions.

1. Create a class called Zoo with a default constructor.
2. Create an instance of Zoo in Main(). This will kick start the Zoo's constructor.
3. Declared and initialise the following fields in the Zoo class.

   The following string arrays contain the names of the existing animals and the new animals we will add to our zoo. The Dictionary will keep track of the animals and their count.

   ```
   Dictionary<String, int> animalMap = null;
   string[] originalAnimals = {"Zebra", "Lion", "Buffalo"};
   string[] newAnimals = {"Zebra", "Gazelle", "Buffalo", "Zebra"};
   ```

4. Instantiate the *animalMap* Dictionary reference (new it) in the constructor. (This is just good practice)
5. Create a method in Zoo as
   void addAnimals(string[] animals)
6. Call `addAnimals()` twice from the *constructor* and pass it the *originalAnimals* and then *newAnimals*.
7. addAnimals() must iterates over the `string[]` (method parameter) and add each String to the *animalMap*.

   If the animal's name is already in *animalMap* then its count must increase by 1, otherwise you should set its count to

   Tips:
   a) Use the *animalMap*.`Add()` method to add a new item.

   b) Use *animalMap*.`ContainsKey()` to test if an animal exists in dictionary.

   c) To get the count of an animal type:
   ```
   int count = animalMap[<an animal name>];
   ```

   d) To increase the count of an animal by 1, type:
   ```
   animalMap[<an animal name>]++;
   ```

8. Create a new method in the Zoo class called **`displayAnimals`**(). Call this method from the Zoo's *constructor*.

   Tip: Use a foreach loop. Each item will have a Key and Value.

```
foreach (var animal in animalMap) {
    Console.WriteLine($"{animal.Key}{animal.Value}");
}
```

   This method displays key/value pairs in 2 columns like:

Zebra	3
Lion	2
Buffalo	5

9. Run and test your code

** End **