LAB 8, C# TYPES I - GETTING STARTED

Lab 8, C# Types I – Getting started

Objective

The primary objective for this lab is as follows:

- To create and use references.
- To consolidate on passing by value and passing by reference

Part 1 – Creating and using reference types

Step by step

1. Create a new Console application called Lab08

   Please refer to lab1's instructions if you need help.

2. Create another class called Account with following private fields:

   ```
   int id;
   string owner;
   double balance;
   ```

   and methods:

   ```
   void Deposit(double amount) { }
   void Withdraw(double amount) { }
   ```

3. Create a read only property for the id and owner fields. For example fir the ID fields you can type:

   ```
   public int ID
   {
         get { return id; }
         set { id = value; }
   }
   ```

4. Make sure no one can deposit negative amount of money!

5. Also make sure no one can withdraw money they don't have.

6. Create a constructor for the account class to set its state.

7. Create a method called GetDetails() to return details of the account to console.

8. In the `Main()` method create an instance of the *Account* class

9. Invoke its Deposit() and Withdraw() methods. Make sure you test a few valid and invalid values.

10. Call the account's GetDetails() method and print the result to make sure your code has had the expected effect.

Part 2 – Passing reference types to a method

In this part you'll examine passing reference types between methods. This is an important topic which has security implication.

Step by step

1. In the Account class, create a new method as:
   ```
   public void AddInterest()
   ```
   Write code in this method to add 2.5% interest to the balance.
2. Add code in Main() to create an account called *myAccount*, with a balance of £100.
3. Call the *AddInterest* method of the *myAccount* instance.
4. After the call, display the details of myAccount (using its GetDetails()).
   Did the balance change?
5. Create another Account reference called partnerAccount and set it to the myAccount instance as:
   ```
   Account  partnerAccount = myAccount;
   ```
6. Call the partnerAccount.AddInterest() method
7. Call myAccount.GetDetails() method. Did the balance change?

Let's do another experiment

8. Create a method in the **Program** class (just under the Main())
   ```
   static void ProcessAccount(Account acc){
           acc.AddInterest();
   }
   ```
9. Back in Main(), create an instance of Account called myAccount and call the ProcessAccount method, passing myAccount as parameter.
10. Call myAccount.getDetails() method.
    Did the balance change?

11. Let's do another important experiment.
12. Create a method under the Main() method as:
    ```
    private static void IncInt(int x) {
            x++;
    }
    ```
13. Add code to the Main() method to create an integer like
    int k=100;
14. Call the **IncInt**() method and pass it the value held by **k**.

15. After the call, print the value of k.
    Did the value of **k** change after the call?

Explanation: All primitive data types (built into the language) are **Passed by Copy** (also known as **passed by value**). Therefore, changing x did not change k and changing the name of k to x will not alter anything!

All Reference types (like myAccount) are also passed by copy but what is passed is their address in memory. Therefore, when you call a method like **ProcessAccount**(Account acc) and pass it an account reference (such as myAccount) you are actually passing its address! The acc reference will be pointing to (referencing) myAccount. Any change made by acc will directly affect myAccount.

** End **