# CAB201 Programming Principles

## Worksheet 9 – Debugging C# programs by hand & mind

## Learning Objectives

1. Manually debugging
2. Become familiar with the *Microsoft Visual Studio .NET* debugging tools.

## Assumptions

That you are already familiar with;

* Lectures 1 – 9
* Worksheets 1 - 5

## Activity Overview

There are 5 activities in this worksheet:

1. Review of last Problem Solving Exercise
2. Finding errors in a method
3. Introduction to linking Components together
4. Creating a subclass
5. Creating a subclass to an existing subclass

**Activity 1: Last Problem Solving Exercise**

*A group of aeroplanes is based on an island located on the equator. The fuel tank of each plane holds just enough fuel to take it halfway around the world.*

*Any desired amount of fuel can be transferred from the tank of one plane to another plane while the planes are in flight. The only source of fuel is on the island, and for the purposes of this problem it is assumed that there is no time lost in refuelling either in the air or on the ground.*

*Determine the number of planes required which will ensure the flight of one plane around the world, assuming that the planes have the same constant ground speed and rate of fuel consumption and that all planes return safely to the island base?*

**Activity 2: Finding errors in a method manually**

In an exam students were ask to write a method, **int FirstPairOfDuplicates (int[] anArray)** that returns the index of the first elementof the first pair of duplicate numbers in **anArray**, where the two numbers in the pair are adjacent to each other.

For example, if the array contained the values {1, 9, 1, 9, 9, 5, 7, 7, 4}, then the method would return the value 3, which is the index of the first element of the <9, 9> pair which is the first pair of duplicate numbers in the array. The method returns -1, if there are no adjacent pairs of duplicate numbers.

Each of the following methods is a student’s answer to that question and each one contains one or more logic errors and/or runtime errors which would have been detected with a reasonable designed Test plan.

1. For each, identify what is the error (or errors). Be prepared to discuss each answer with your tutor. There are no compiler errors in any of these answers.

Test Cases:

1. Empty array – [] = -1
2. One element – [4] = -1
3. No duplicates – [1, 2, 3, 4] = -1
4. All same – [3, 3, 3, 3] = 0
5. Duplicates at end – [1, 2, 3, 3] = 2
6. Write your own solution to this question of finding the first pair of duplicate numbers in an array. Are you sure that your answer is correct?
7. **static int FirstPairOfDuplicates (int[] an Array){**

**int matchIndex = -1;**

**if (anArray.Length > 1){**

**bool match = false;**

**int index = 1;**

**do {**

**if (index < anArray.Length){**

**if (anArray[index] == anArray[index-1]){**

**matchIndex = index – 1;**

**match = true;**

**}**

**}**

**index++;**

**} while (!match);**

**}**

**return matchIndex;**

**}**

**static int FirstPairOfDuplicates (int[] an Array){**

**for (int i = 0; i < anArray.Length; i++){**

**if(anArray[i] == anArray[i+1]){**

**return i;**

**} else {**

**return -1;**

**}**

**}**

}



**static int FirstPairOfDuplicates (int[] an Array){**

**for (int i = 0; i < anArray.Length; i++){**

**for (int j = i+1; j < anArray.Length; j++){**

**if (anArray[i] == anArray[j]) {**

**return i;**

**}**

}

}

**return -1;**

**}**



**static int FirstPairOfDuplicates (int[] an Array){**

**int value = -1;**

**for (int i = 0; i < anArray.Length-1; i++){**

**if (anArray[i] == anArray[i+1]){**

**value = i;**

**}**

**}**

**return value;**

**}**

### Activity 3 Introduction to linking Components together

From the file **Components.**zip, extract the three project folders which will be used in this activity, **Employee Class Library, PayRoll System and Payroll System Test Driver.**

Open each folder in turn and open their respective Solution file; **Employee Class Library.sln, PayRoll System.sln and Payroll System Test Driver.sln.** At the moment these three separate classes are not connected. We need to add references to **PayRoll System** so that it can access the classes contained within the **Employee Class Library** as well addingreferences in **Payroll System Test Driver** to the other two classes.

Open **Solution Explorer** windowof **PayRoll System,** select the project name, right-click it and select **Add Reference.** A window is displayed that enables you to use the **Browse** tab to locate the DLL (Dynamic Link Library) file of **Employee Class Library**, **Employee Class Library.dll**, which is located within the **Debug** directory of **bin** directory within **Employee Class Library**. Select the ddl file, click **Add**, click **OK**.

Within the **Solution Explorer** window, under **References** you will see that the reference to **Employee Class Library** has been added. Now need to add a **using** directive for the **Employee Class Library namespace** with the other **using** directives of **PayRoll System.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Employee\_Class\_Library;

Save **PayRoll System.cs** and compile it using **BUILD**. If you have done this correctly, the Build will be successfully.

We now need to repeat the above process for **Payroll System Test Driver.** Open the **Solution Explorer** windowof **Payroll System Test Driver** and add a reference to **PayRoll System,** then add a reference to **Employee Class Library.** Now add two **using** directivesto **Payroll System Test Driver**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

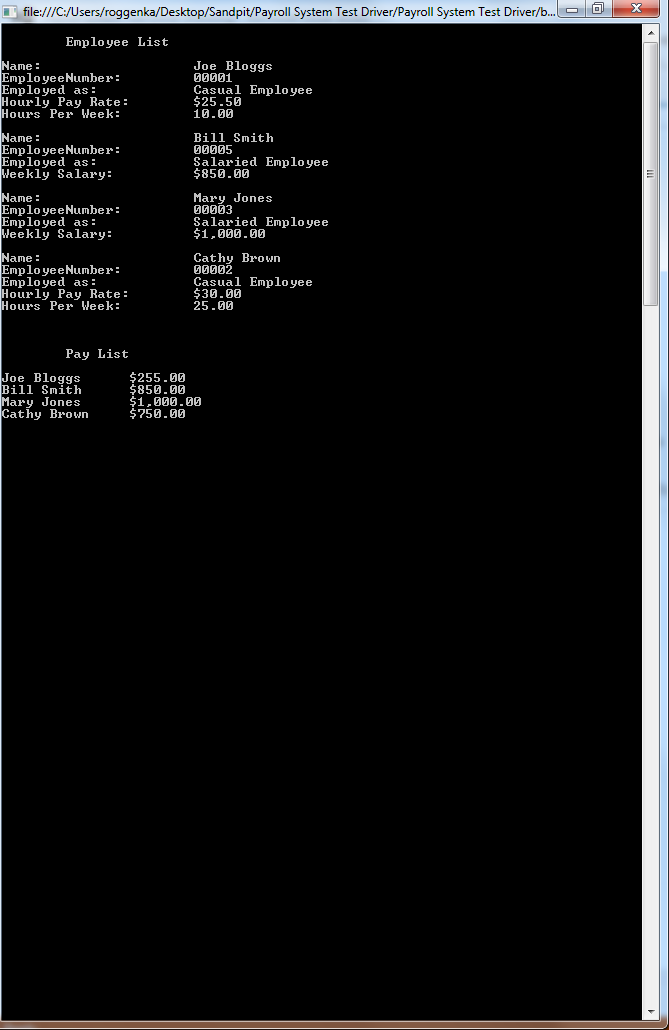
using System.Threading.Tasks;

using Employee\_Class\_Library;

using PayRoll\_System;

Save and Build **Payroll System Test Driver.cs**, you should now be able to run **Payroll System Test Driver** which will produce output identical to the following screen shot.

Leave these projects open.



**Activity 4 Preamble**

Open the Solution file of **Employee Class Library** and look over the code of **Employee.cs** which is an abstract class which implements the common attributes and behaviours of an employee. **Salaried Employee** and **Casual Employee** are subclasses of **Employee.** Look at the code of each of these subclasses.

The **PayRoll System** class models a simple Payroll system which is used in **Payroll System Test Driver** to test the functionality provided by both **PayRoll System** and **Employee Class Library.**

Look at the code of **PayRoll System.cs** especially the methods **OpenPayRoll, ClosePayRoll** and **ReadnextEmployee** which models sequential access to the **Employee** data maintained in **PayRoll System** and how this functionality is used in **Payroll System Test Driver** methods **PrintEmployeeList** and **PrintPayList**.

### Activity 4 Creating a Subclass

Open the Solution file of **Employee Class Library** if it is not already opened. In the **Solution Explorer** window, select the project name, right-click it and select **Add**, in its window select **Class**. Change the default name **Class1.cs**, to **Commissioned Employee.cs**, click **Add**.

Make the class **public** and a subclass of **Employee**

public class Commission\_Employee : Employee {

Then add to the body of the class using the following UML class diagram.

|  |
| --- |
| **Commission\_Employee extends Employee** |
| **-commissionRate : double**  **-grossSales: double** |
| **+Commission\_Employee(string name, int num, double sales, double rate)**  **+CalculatePay:double**  **+GetDetails: string**  **+GetGrossSales:double**  **+SetGrossSales(double)** |

The above UML class diagram shows an accessor and mutator methods for the instance variable **grossSales**, you can implement these as a Property similar to what was done in **Casual Employee.cs.**  The body of **GetDetails** should be similarly to the corresponding method in **Salaried Employee.cs** and **Casual Employee.cs**

Implement each method. The **CalculatePay** method simply multiplies the **grossSales** amount by the **commissionRate.** Build when finished. Once Build is successfully, open **Payroll System Test Driver.cs** and add one or more statements to **Main** which instantiates several **Commission Eployees** similar to the following statement:

payRoll.AddEmployee(new Commission\_Employee("Kerryn Woods", 7, 1000, 0.1));

Execute **Payroll System Test Driver.cs**.

**Activity 5 Creating a Subclass of an Existing Subclass**

In **Employee Class Library** add a subclass which extends **Commissioned Employee** named **Base Plus Commission Employee.**

|  |
| --- |
| **Base\_Plus\_Commission\_Employee extends Commission\_Employee** |
| **-baseSalary: double** |
| **+Base\_Plus\_Commission\_Employee(string name, int num, double sales, double rate,**  **double base )**  **+CalculatePay:double**  **+GetDetails: string**  **+GetBaseSalary:double**  **+SetBaseSalary(double)** |

As in **Commissioned Employee** you can implement a Property in place of the accessorand mutator for **BaseSalary.** The **CalculatePay** method simply adds the **baseSalary** to the amount returned from **CalculatePay** of its parent class, **Commissioned Employee.**