# JIT121 Programming Tutorial 6

## Learning Objectives

1. Working with GUI controls
2. Practice developing an application which uses a non-static user-defined class
3. Working with arrays of objects
4. Understanding the difference between a class and an object.
5. Using UML diagrams

## Assumptions

That you are already familiar with;

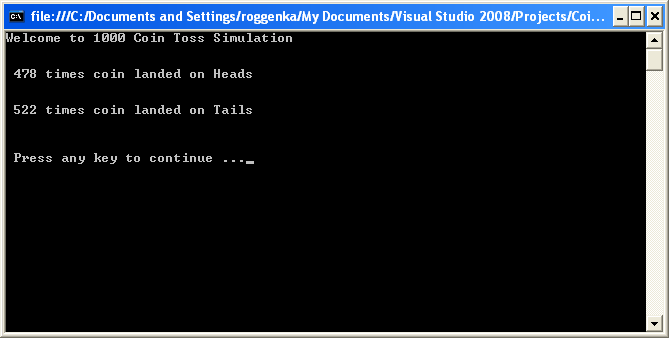
* Lectures 1 – 6 and Tutorials 1 - 5

## Activity Overview

1. Using a User-Defined Class
2. Change Direction Board Game
3. Variation on User-Defined Class using an array
4. Interpreting and implementing from a UML diagram

### Activity 1: Using a User-defined Class

### This activity gives you some initial experience in working with a simple user defined class. You are given a program shell containing an empty Main method and a user defined class Coin. You are to write a program which simulates the tossing of a coin 1000 times and keeps a count of the number of times the coin shows Heads and Tails respectively.



Download **CoinApplication.zip** and unzip it. From the **CoinApplication** folder, open the solution file, **CoinApplication.sln**.

Look at **Coin.cs,** to understand the **attributes** and **behaviours** of this class. However *do not change any part of this file*. You will not need to use the **ToString** method in your program; it is merely there for completeness.

Open the file, **CoinTossing.cs** from the **Solution Explorer** window. The body of **Main** contains an algorithm for this exercise. Implement the algorithm to produce output similar to the above screen shot.

### Activity 2: Change Direction Board Game

This two person game is a variation of the simple **Race to the End** board game introduced in Lecture 6.

The board consists of 41 squares in a row and both players start on the *middle* square. Each player rolls a die to determine the number of squares to move, initially towards the right hand end of the board. However on alternate turns the player moves in the opposite direction.

Should a player reach the left hand end of the board, the other player wins providing they have not reached the left hand end of the board on that turn. If both players reach the left hand end on the same turn then no-one wins.

If a player reaches the right hand end of the board that player wins, though it will be a tie if both players reach the right hand end on the same turn.

See Lecture 6 to familiarise yourself with the **Simple Race to the End** game. Download **L6\_RaceToTheEnd.cs** to see how it was implemented. This should provide you with a starting point to implement the **Change Direction** game.

**Activity 3: Variation on Coin Tossing Activity**

Copy the zip file from the previous coin tossing activity, **CoinApplication.zip,** rename it **CoinApplication\_ARRAY.zip** and unzip it. It does not matter that the solution file is named **CoinApplication.sln**.

This time, write a program which tosses two (2) coins, 1000 times and counts the number of times both coins are Heads, both are Tails and both are different. Use an **array** and do not declare two separate Coin variables.

This code will only be a small variation on that used in the previous activity but will give you experience using an array of Coin. (BTW this was an old exam question and students had 10 minutes to write out the body of **Main**, they were supplied a copy of the Coinclass. It was worth 4%.)

### Activity 4: Implementing a Class from a UML Diagram

The following UML diagram describes a class ShoeStock which is to be used as part of an inventory system at a shoe shop.

|  |
| --- |
| **ShoeStock** |
| -inventoryNumber : int  -shoeStyle : string  -numberAvailable : int |
| +ShoeStock(inventoryNumber: int, shoeStyle: string, numberAvailable: int)  +InventoryNumber : int {property}  +ShoeStyle : string {property}  +NumberAvailable : int {property}  +ShoeInStock() : bool  +Reorder() : bool  +ToString(): string |

A ShoeStock's inventoryNumber and shoeStyle will never change. Shoes are in stock if numberAvailable > 0. Shoes should be reordered when numberAvailable < 10. (This description of behaviours will help you write the code for the class methods.)

**Your Task**: Using this description and the UML diagram, write the complete class including the method bodies. Then write a driver class to create a **List** of ShoeStock objects and test the objects' functionality to produce the output below:

