## Weight Conversionsh

1. Create a method as   
   **public** **static void** ConvertInputToStonesPounds(**int** pounds).
2. Ask the user for a total weight in pounds in Main() and pass the result to the above new method.
3. Display the result (stones & pounds) in the new method.

Note: there are 14 pounds in a stone.   
**Tip**: Use division (/) and modulus (%)

1. Create another method as  
   **public** **static void** **ConvertKgsToStonesPounds**(**int** kg).
2. Ask the user for a weight in kilograms.
3. Convert the weight and display it in stones and pounds

**Hint:** 1 kilo = 2.20462 pounds   
**Tip**: convert the Kg to pounds and then call ConvertInputToStonesPounds(**int** kg)

1. Test your code at each stage

## How long does it take to double your money using compound interest

Assuming and initial investment of say £100, how many years does it take to grow to £200 given an interest rate of 5 percent?

### **Step by step**

1. Create a new method In Account class you created in Lab6 like  
     
   **public int** **DoubleYourMoney**(double interestRate)
2. Write code to calculate the number of years it takes to double the current balance of the account (using compound interest)
3. Create an Account instance. Put some money in the account and then display the number of years it takes to double the balance for an interest rate of say 2%

## Nested Loop Practice

Ensure you can code up nested loops understanding the full sequence in which everything runs and effectively use the outer and ‘inner’ loop variables together in a nested loop. In this part you'll produce a multiplication table.

### Step by step

1. Create a method called MultiplicationTable().
2. We want you to produce this output on the console.

1 2 3 4 5 6 7 8 9 10   
2 4 6 8 10 12 14 16 18 20   
3 6 9 12 15 18 21 24 27 30   
4 8 12 16 20 24 28 32 36 40   
5 10 15 20 25 30 35 40 45 50

6 12 18 24 30 36 42 48 54 60

7 14 21 28 35 42 49 56 63 70

8 16 24 32 40 48 56 64 72 80

9 18 27 36 45 54 63 72 81 90

10 20 30 40 50 60 70 80 90 100

**Tip:** Two nested for loops (count from 1..10) are best for this.

Also, to print the product of two variables called row and col in 5 spaces, use a statement like:

**Console.Write($"{col \* row , 5}");**

# **Arrays**

1. Create a new method and call it from Main()
2. In the new method, declare an array of integers as:

int[] numbers = { 1, 4, -5, 7, 0, 4, 6, 8 };

1. Task 1: write code to find the sum of every number in numbers
2. Task 2: Find the average of these numbers
3. Task 3: Find the minimum number in the numbers array
4. Task 4: Find the maximum number in the numbers array
5. Task 5: Find the index of number zero in numbers

# **Arrays and Strings**

**Given the following deck of cards**

String[] cards = {"Ace Spades","2 Spades","3 Spades","4 Spades","5 Spades","6 Spades","7 Spades","8 Spades","9 Spades","10 Spades","Jack Spades","Queen Spades","King Spades","Ace Hearts","2 Hearts","3 Hearts","4 Hearts","5 Hearts","6 Hearts","7 Hearts","8 Hearts","9 Hearts","10 Hearts","Jack Hearts","Queen Hearts","King Hearts","Ace Clubs","2 Clubs","3 Clubs","4 Clubs","5 Clubs","6 Clubs","7 Clubs","8 Clubs","9 Clubs","10 Clubs","Jack Clubs","Queen Clubs","KingClubs","Ace Diamonds","2 Diamonds","3 Diamonds","4 Diamonds","5 Diamonds","6 Diamonds","7 Diamonds","8 Diamonds","9 Diamonds","10 Diamonds","Jack Diamonds","Queen Diamonds","King Diamonds"};

**Draw 5 cards at random (Use the class Random).**

**Display the cards and their total value.**

**Jack, King, Queen and Ace have a value of 10. Other cards have the same value as the first character of their names imply.**

**To make this game more suitable, use Collections.shuffle(); method to shuffle the deck of cards before drawing a hand.**

**Draw two hand and see which one is the winner!**

# **21**

**After shuffling the cards, place the cards in a queue object.**

**Draw cards, one at a time. Remove the cards that are drawn from the queue.**

**Print out the value of the hand each time a card is drawn out of the queue.  
Stop if the user sticks.**

**Do the same for the House and see which one wins.**

**Your software declares the winner!**

# **Static and Single responsibility for classes**

Ball contains a few static fields which defines the world

Move these into a separate class called World.

Game contains all the Shapes. Move the List of Shapes into World.

Give the World an **Add()** method which adds a Shape to the list.

The Game class can now call the World class to move the balls and draw the balls.

This greatly simplifies the Game class which should only be responsible for the UI.

# **Working with files**

Store the state of each Shape in a file.

Create an interface for a method for two methods

**void** SaveShapes(Shape shape);

List<Shape> GetShapes();

Implement this method in the Shape abstract class

Modify your code to save all the shapes’ states (x,y,w,h,SHAPE\_TYPE) each time a Shape is added to the list of Shapes.

Modify the Main() method to load the shapes the next time application runs.

I leave the planning and coding of this app to you!

# **Implementing a Design Pattern**

In this exercise you will implement the **Composite** pattern

The World in which the Shape objects move is static and does not move!

What if the World itself was a class which was derived from the Shape class? Would that make the world itself move inside another World object?

Have a go to see if such a thing is possible.