COM6471 – Foundations of object-oriented programming
Assignment 2 – Sheepdog trials (30%)

This is the second assignment for COM6471 Foundations of Object Oriented Programming, and it is worth 30% of your final module mark.

The aim of this assignment is to test your ability to:

- Write a Java program from a specification of what it should do.
- Use loops and arrays in Java.
- Write a Java program that makes use of several different classes.

The game

The objective of this assignment is to write a Java program that implements a simple game. The game involves 4 sheep and one dog who exist in a NxN grid. It is a (greatly simplified) representation of sheep herding with a border collie (see https://youtu.be/2Z-FYXXYkBM for some more background. If you are interested, there is a famous local sheep herding competition at the Longshaw estate just outside Sheffield, and the next one will be in September 2021 – see https://www.longshawsheepdog.co.uk/ ☺).

In our game each sheep and the dog are located in one particular square for each turn of the game. The user controls the dog, and the sheep move (mostly) in response to the Dog's position. Note that in English, the word sheep is *both* singular *and* plural.

The aim of the game is to herd the sheep into a single square. The game starts with 4 sheep randomly distributed across the grid. In each turn the user enters the co-ordinates (row and column) of one of the squares, and the dog is moved to that square. The sheep should then move 1 square, according to the following rules:

- Each sheep picks a random number between 1 and 6.
- If the number is in the range 1-3, then the sheep moves away from the dog. This should be to the square directly away from the dog, unless this move would take the sheep off the edge of the grid, in which case it should stay still.
- If the number is 4 or 5, the sheep should move beside the dog — i.e., it should move to a square adjacent to the dog. Unless the dog is close to the edge of the grid there are four possible moves, and the sheep should choose one of these randomly.
- If the number is a 6 then the sheep should move to the same square as the dog
- If the dog is on the same square as the sheep at the start of the turn, then the sheep moves randomly to any adjacent square.
- Sheep move either along rows or along columns, they cannot move diagonally.

The task

Your assignment is to implement this game in Java, and you should follow the instructions below exactly:

- The program must include a `Sheep` class that models each individual sheep. This class must contain an instance variable that tracks which square the `Sheep` is on, and

a `void move(int dogRow, int dogCol)` method that takes the `Dog`'s current square as an argument and updates the `Sheep`'s square using the rules provided above.

- A random integer can be generated by importing `java.util.Random`, creating a `Random` object, and using the `nextInt(6)` method on the Random object to generate a random integer between 0 to 5 (i.e. from 0 to less than 6), and `nextInt(4)` to generate a random integer between 0 and 3.
- The grid should be of size N x N, where N is an integer between 3 and 9. You can either set N as a constant, or N can be specified by the user.
- During each turn, the program should display a grid that shows the location of each `Sheep` and the `Dog`. You do not need to match the example below exactly. Using letters and a simple grid displayed in the command window or terminal, as shown below, is fine. You can use more a imaginative display if you want to. If you would like a challenge, then you can use the updated `EasyGraphics` class from the sheffield package for a graphical display. The version of `EasyGraphics.java` on the Assignment2 section on Blackboard has updated capability for displaying colours. However, you should not use other third-party libraries for a graphical display.
- The game should be run from a class called `SheepHerding.java` which should have a main method. At the beginning of the game, this class should place the 4 `Sheep` randomly on the grid, and then prompt the user for the co-ordinates of the `Dog` (you can use `EasyReader` for this. The `move(dogRow, dogCol)` method should then be called for each `Sheep`, and the position of each `Sheep` should be updated. The grid should then be re-drawn and the user prompted for the next turn.
- The game should end if (i) all the `Sheep` are in the same square, or (ii) the user enters a number 0. It should print a message to say either "All sheep in one square!" or "User quit" to indicate the event that caused the program to end.
- You should create other classes and methods as you choose to form a structured, Object-Oriented system. Only create additional classes if they are needed. It is possible to write an acceptable solution with only 3 classes, `SheepHerding`, `Sheep` and `DisplayGame`.

Example output

```
Please enter new dog row: 3
Please enter new dog column (0 to quit): 2
Moving sheep
      1    2    3
   +--------------+
 1 |    |    |S S |
   |    |    |    |
   +--------------+
 2 | S  |    |    |
   |    |    |    |
   +--------------+
 3 |    |   S|    |
   |    |  D |    |
   +--------------+

Please enter new dog row:
```

In this example the grid has size 3; sheep are indicated by 'S', and the dog by 'D'. The example below shows the final turn from a game where the dog succeeds in herding the sheep into a single square on a grid with size 5 (this is quite a rare occurrence!).

```
Please enter new dog row: 3
Please enter new dog column (0 to quit): 5
Moving sheep
      1    2    3    4    5
   +-------------------------+
 1 |    |    |    |    |    |
   |    |    |    |    |    |
   +-------------------------+
 2 |    |    |    |    |    |
   |    |    |    |    |    |
   +-------------------------+
 3 |    |    |    |    |    |
   |    |    |    |    |  D |
   +-------------------------+
 4 |    |    |    |    |    |
   |    |    |    |    |    |
   +-------------------------+
 5 |SSSS|    |    |    |    |
   |    |    |    |    |    |
   +-------------------------+

All sheep in one square!
```

How to proceed

This is a more complex assignment them the first one, and you should think carefully before beginning to write any code. The following points will help you to think through the design.

- Consider what each `Sheep` object needs to know (instance variables), and what it needs to do (the instance methods). How many get and set methods does each `Sheep` need?
- Is it necessary to have a separate `Dog` class?
- Decide how to represent the grid. Will you need a 2D array, or do you just need to store the row and column co-ordinates for each sheep and the dog?
- What processing can you delegate to static methods in the `SheepHerding` class?
- How will you implement the rules for moving sheep, and how will you make sure they stay within the grid? I recommend that you think very carefully through the logic, and construct if-else statements accordingly.

Once you have some design ideas, write some test code to check that these ideas work. Don't immediately try to write your final submission. Instead test out ideas for implementing the movement rules, displaying the grid, and other parts of the program. Make use of `System.out.println()` to inspect the values taken on by your variables, and test whether they are correct.

Submission

Before you upload your code, check that it compiles on the on the command line using javac SheepHerding.java. Check that the code is not part of a package.

Your code for all 3 stages should then be submitted as a ***java files*** and uploaded to Blackboard using the upload tool before the deadline. ***Do not submit archives (e.g. zip files) or .class files***.

Assessment

Your code will be assessed for functionality, style, and flair. A mark of zero will be awarded for programs that do not compile; it is therefore important that you check that your code compiles before you submit it.

Overall, half of the marks (15/30) will be awarded for functionality. If your program compiles and works correctly, then you can expect to be awarded 10 marks for functionality.

You should bear in mind that there are different ways to code the solution to this assignment. There is no single "correct" solution. However, some solutions are better than others because they are more concise, and more readable.

Ten marks (10/30) will be therefore awarded for programming style. Code written in good style will have the following characteristics:

- Each class will have a header comment, typically specifying the program name, the author, and the date.
- The code is indented consistently throughout using four spaces and not tabs, so that the code in each method is easily identified.
- Variables and methods have meaningful names (except for loop variables, which can be a single letter, e.g. `for (int j = 0; j <= max; j++)`). Variables and methods should be in camelCase (e.g. `numberOfSheep`).
- There are sufficient lines of code to meet the requirements, and no extra or unnecessary code or comments.

Five marks (5/30) will be awarded for flair. These will be awarded for programs with evidence of elegant and readable code, for an imaginative display, for a graphical display, and for good OO design (see https://software.ac.uk/resources/guides/writing-readable-source-code for some background).

Unfair means

We take the use of unfair means very seriously. This will be covered in a tutorial, and there is a page in the MSc handbook that covers unfair means[1]. This is an individual assignment. You must work on it alone and hand in your own code. You must not share code with other students, or submit work done by another student. All submissions will be run through specialised software that detects similarity among Java programs, and action will be taken in cases where unfair means are suspected.

November 2020

[1] https://sites.google.com/sheffield.ac.uk/compgtstudenthandbook/menu/referencing-unfair-means