School of
Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

Assignment 2

Algorithms and Data Structures: Assignment 2

- Due April 9 23:30

Goals

This assignment will give you more experience in developing programs that use arrays and ArrayLists of objects.

Resources and links

- Download Zip file of necessary code and data.
- Submit your answers.
- Marks and Feedback

Summary

- Earthquake Analyser.
  ⇨ Write a program to analyse data about a collection of earthquakes.
- Domino Game.
  ⇨ Write a version of the Domino game.
- Reflection.
  ⇨ Write up your reflections on this assignment.

Overview

The assignment involves two programs to submit:

- The first program analyses data about a collection of 4335 NZ earthquakes from May 2016 to May 2017.
- The second program lets a player play a simple solitaire dominoes game.

There is one exercise to give you practice with arrays and another exercise (`TileExercise`) closely related to `DominoGame`.

Make sure that you do `TileExercise` before starting the domino game program.

To Submit:

- Your version of `DominoGame.java`
- Your version of `EarthquakeAnalyser.java`
- Your `Reflection.txt` file

You may work in pairs for the Core and Completion parts, but if you do you must include a comment at the top of your program saying who you worked with. You must do the Challenge part on your own.

You are reminded of the School's policy on plagiarism - see the COMP102 Web site for details.

Preparation

Download the zip file for assignment 2 and extract it to your home folder. It should contain templates for the Java programs you are to complete. Read through the whole assignment to see what you need to do. You can run the demo programs on the ECS lab computers.

Look again at your answer and/or the model answer to the WaveformAnalyser program from assignment 6 and go over the code examples in slides from the lectures on arrays and ArrayLists.

Earthquake Analyser

`EarthquakeAnalyser` analyses data about a collection of 4335 New Zealand earthquakes from May 2016 to May 2017. The data is from http://quakesearch.geonet.org.nz/. You are to complete the `EarthquakeAnalyser` program which does some useful analysis of this data.

Each line of the file "earthquake-data.txt" has a description of one earthquake:

- ID, time, longitude, latitude, magnitude, depth, region

eg

```
2016p385555 2016-05-22T15:43:56.619Z 175.6988525 -39.39247894 1.0 18.4 taranaki
2016p415555 2016-06-02T18:07:49.213Z 175.3636322 -39.28754807 1.0 11.7 taranaki
```

Note the file `bigearthquake-data.txt` is a subset of `earthquake-data.txt`. It has just the 421 earthquakes of magnitude 4.0 and above which may be useful for testing, since it is not as big as the full file.

The overall design of the EarthquakeAnalyser program is provided, along with the declaration of a field to keep track of the earthquakes, and the constructor. You have to complete the definitions of the methods that deals with ArrayList.

A `Earthquake` class is also provided. It contains a constructor for creating a new Earthquake object. The constructor has one parameter which must be a String containing one line of the "earthquake-data.txt" file. It has methods for computing the distance and time difference between two earthquakes, for getting a description string of the earthquake, and for accessing some of the information fields.

Core

Complete two methods:

- `doLoadData`
  - Loads the data from a file into a field containing an ArrayList of Earthquake objects.
  - Hint : to make an Earthquake object, read one line from the file and pass it as the argument to the Earthquake constructor. Make sure any previous values in the field are removed
- `doFindBigOnes`
  - Lists all the earthquakes in the ArrayList that have a magnitude 5.5 and over.
  - Hint: see the methods in the Earthquake class, especially getMagnitude and toString

Completion

Complete the `doFindPairs` method:

- Compares each Earthquake in the list with every other Earthquake and finds every pair of "close" earthquakes - earthquakes that
  - are within 1km of each other, and
  - have depths within 1km of each other, and
  - are separated by at least 24 hours days
- For each "close" pair, it should print
  - their ID's,
  - the distance between them
  - the depth difference
  - the number of days between them.

Challenge

Complete the `doFindFollowOns` method:

- Constructs a new ArrayList containing every earthquake with a magnitude that is at least 6.0. For each such earthquake on this list
    - Finds a list of all the "follow-on" earthquakes: later earthquakes within a distance of 10km and depth difference <= 10km.
    - If there are at least two follow-on earthquakes, then it prints out the full details of the big earthquake followed by ID, magnitude and days since the big one for each follow-on earthquake.

The file "example-output.txt" has sample output for the "bigearthquake-data.txt" file, which only contains earthquakes with magnitude 4 and above.

Domino Game

Dominoes are rectangular tiles that have two numbers on them, usually represented by patterns of dots (no dots represents 0). There are many games using dominoes, many of which involve a player pickup up a "hand" of dominoes, and then placing them on the "table" according to some rules.

The `DominoGame` program allows a player to play one of these games. The program displays the set of six dominoes in their hand, and the dominoes on the table. It allows the user to

- Pick up a new domino to fill an empty space in the hand;
- Select a domino in the hand (with the mouse)
- "Flip" the selected domino
- Place the selected domino on the table.
- Rearrange the order of the dominoes on their hand;

The program has eight buttons ("Pickup", "Place", "Flip", "Left", "Right", "Suggest", Restart", and "Quit") and also responds to the mouse.

The overall design of the `DominoGame` program is provided, along with declarations of all the fields and constants you need, and some of the code to respond to the buttons and mouse. You have to complete the definitions of the methods that do all the work!

A `Domino` class is also provided. It contains

- a constructor for creating a new Domino,
- a `draw(double left, double top)` method for drawing a Domino at a specified position, and
- constants specifying the size of the dominoes.
- `getTop()` and `getBottom()` methods for returning the top and bottom number of the domino
- a `flip()` method to turn the domino over

Read through the provided code to make sure you know what it does.

Program Design

The program has two fields to keep track of the dominoes:

- `hand` : an array that can hold up to 6 `Domino`es. It will contain `null` in any position that does not have a `Domino`.
- `table` : an ArrayList of the `Domino`es that have been placed on the table.

The hand should be displayed with a rectangular border and each domino drawn in its place. Empty spaces (containing `null`) should be displayed as empty.

The user can select a position on the hand using the mouse. The selected position (domino or empty space) should be highlighted in some way, eg by drawing a border around it, or by drawing some marker underneath it. The index of the selected position is stored in the `selectedPos` field.

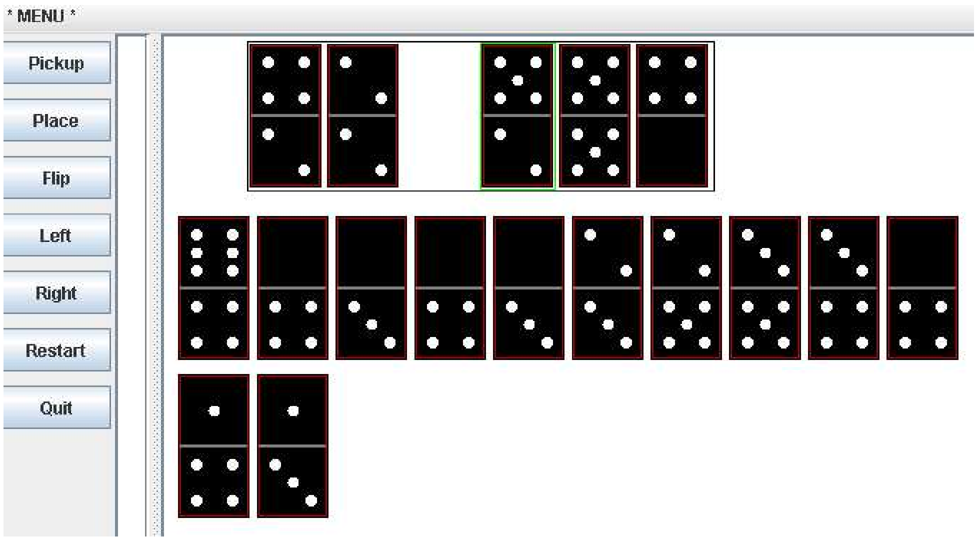
The user can use the **Left** or **Right** button to move the contents of the selected position one place to the left or the right (by swapping with the contents of the adjacent position).

If there are any empty positions on the hand, the user can use the **Pickup** button to get a new (random) domino which will be added to the hand at the leftmost empty position. If there are no empty positions, **Pickup** should do nothing.

If the selected position contains a domino, the user can use the **Place** button to move the selected domino to the table.

The **table** is represented by an `ArrayList` of `Domino` objects, which are drawn in rows underneath the hand. At the beginning of the game the table should be empty. Dominoes should be added to the end of the table.

The following is a screenshot of the layout of the "hand" and the "table", showing a hand with five dominoes, and a "table" with 12 dominoes.



Core

- Complete the Constructor so that it
  - initialises the `hand` and `table` fields
  - sets up the buttons, and the mouselistener
  - calls the `doRestart()` method
  - calls the `redraw()` method to redraw the hand and the table

- Complete the `doRestart()` method so that it
  - sets the `table` field to be empty
  - sets the `hand` field to have no dominoes in it

- Complete the `doPickup()` method so that it
  - looks for an empty position in the hand array, and if there is one, creates a new `Domino` object and puts it at that position in the hand array.

- Complete the `drawHand()` method so that it
  - draws the outline of the hand
  - draws each domino in the hand array
  - highlights the currently selected position, eg by drawing a coloured rectangle around it

- Complete the `doPlaceDomino()` method so that it moves the domino at the selected position in the hand (if there is one) to the table.

- Complete the `drawTable()` method so that it
  - draws the dominoes on the "table", (For the core, it can draw them in one long row.)

- Test your program carefully, to make sure that it works correctly in all cases:

    - when the hand is empty, when it is full, and when it is only part full.
    - when the selected position is at the ends and when it is not at the ends.

Completion

- Complete the `doFlipDomino()` method so that it flips over the domino at the selected position in the hand (if there is one). (Hint: use the methods provided in the `Domino` class.)

- Complete the `doMoveLeft()` and `doMoveRight()` methods so that they move the contents of the currently selected position on the hand to the left or the right, swapping the tile (or space) with the one next to it. (Note that they should not attempt to move something off either end of the hand.)

- Complete the `drawTable()` method so that it can draw the dominoes on the table in multiple rows, so that they are all visible.

- Complete the `doSuggestDomino()` method, which finds a domino in the user's hand which can be placed into a position on the board. The basic rule of Dominoes is that
    - If there are no dominoes on the table yet, only a double (a domino with the same number top and bottom) can be placed on the table.
    - A domino can only be placed next to another domino if they have matching numbers (one of the numbers on one domino is the same as one of the numbers on the other domino). This method should look for a domino in the hand that can be added to the end of the dominoes on the table. If there is such a domino, it should tell the user which domino it is; if there is no domino that matches, it should inform the user that there are no possible moves.

Challenge

There are lots of ways in which this program could be improved or extended to encompass more of the game. Getting 100% for the program would require at least two of the following.

- Enforce the rule that a domino can only be placed on the table if either the top number or the bottom number matches the corresponding number of the previous domino on the table. (The example diagram above satisfies this rule.) There are methods in the `Domino` class that you can use for helping with this.

- Let the user move Dominoes around on the hand by dragging them with the mouse. If a Domino is dragged to another Domino, it should be inserted by moving the other Dominoes along. If a Domino is dragged to a space, it can simply be put in the space.

- Represent a bag of dominoes from which the user can pickup new dominoes. It should initially contain exactly one of each type of domino. (You will need to make a new constructor for the `Domino` class.) Once the bag is empty, the user can't pick up any more dominoes for their hand.

Reflection

Answer the following questions in the `Reflection.txt` file, and make sure you submit it.

1. Explain why "off-by-one" errors are so easy to make when programming with arrays.
2. Both DominoGame and EarthquakeAnalyser used another class that was written for you. This saved you writing some code, but also constrained you to write the program in a particular way. Discuss advantages and disadvantages you found when having a predefined class that you had to use as part of your program.