Individual Assignment: Patience is a Virtue

Chris Loftus
Handed out: Monday 9th March 2015
Hand-in: Friday 8th May 2015, 3pm via Blackboard upload
Feedback: Friday 29th May 2015
Percentage of overall module marks: 50%

Problem

My old friend Becky taught me a form of patience (solitaire) that I would like to be able to play on the computer. It is very simple, so really needs no graphics but, for fun, I will give you classes to do that as well.

This is how it works:

- The object of the game is to end up with one pile of cards. I have never succeeded in that.
- Shuffle a deck of cards
- Lay down the first card face up
- Lay down the next one next to it, and so on
- This would lead you to 52 cards all facing upwards, except that you can put one pile on another if any of the following apply:
    - They are next to each other and the same number or suit
    - There are two piles between them and they are the same number or suit

Here is an example (H is hearts etc. The new card dealt is in **bold**):

4H
4H **2C**
4H 2C **7D**
4H 2C 7D **3D** (two diamonds in a row combine – so..
4H 2C 3D (notice now the 7D is covered we never see it again
4H 2C 3D **4S** (there are two piles between the two 4s combine – so..
4S 2C 3D
4S 2C 3D **4D** (this time two possible moves – could do either
4D 2C 3D (choose this one
4D 2C 3D **9C**
4D 2C 3D 9C **3C** (this time three things happen, first put 3C on 9C …
4D 2C 3D 3C (now put 3C on 3D …
4D 2C 3C (now can put 3C on 2C
4D 3C

Sometimes you can amalgamate piles in the middle:
3H 4D 7C 8S 3D becomes 3H 3D 7C 8S and then 3D 7C 8S (think about it)

Your game must have a command-line menu that allows you to:

- Shuffle
- Show the pack (this is so you can check that it plays properly)
- Deal a card

- Make a move, move last pile onto previous one
- Make a move, move last pile back over two piles
- Amalgamate piles in the middle (by giving their numbers)
- Play for me once (if two possible moves, makes the 'furthest' one)
- Play for me a number of times (repeats Play for me once a given number of times)
- Show low scores (see below)
- Select whether text output is also required and if so whether to the console, a file or both
- Quit

On each go, if text output is switched on then the state of the game is displayed or appended to a file.
You should save a little file of low scores – the score is the number of piles left at the end of the game. When the person is finished playing you should give them the opportunity to submit their name and score to that file.

Strong hints:
MAKE SURE YOU DO THIS A BIT AT A TIME!!!! Test each class and function as you develop it – never get too far from a working application. The first three requirements should be working before you attempt to support the playing the basic game requirements.

Your code should demonstrate use of ArrayLists, a do..while loop, catching exceptions, reading from and writing to a text file, packages, checking errors (for instance illegal moves), small methods and inheritance. It should have good Javadoc comments and appropriate algorithms. You should use the provided graphical user interface classes and gif files (see final section of this brief). However, if you wish, you may enhance and/or replace these as part of the flair marks. I have provided a zip file with the 'card' information in it. Read it in to get a 'pack'.

Hand-in

Upload two files to Blackboard:

1. A zip file (not rar or anything else) containing your Eclipse project.
    a. If you are not using Eclipse then package up the source code files and class files.
    b. I will run from the command line. Make sure your code runs from the command line and not just from your IDE.
2. A pdf (no other format will be accepted) 2000-word document that contains:
    a. A front cover page that includes your name, email, document date, and the title of this mini-assignment.
    b. Page two must provide a contents page.
    c. Page three must start with an introduction that says what is to follow.
    d. Have a section called *Requirements Analysis* that presents the requirements as a UML Use Case diagram.
    e. Have a section called *Design* that has a class diagram, and a textual description of each class and its relationships. I will be looking for

well-designed classes, method names and attribute names. Include some pseudo-code examples and a sequence diagram of the more complex interactions and algorithms.

f. Have a section called *Testing* that presents screenshots of your program running. It must also have a discussion of test data used and the results of running the tests. A table of expected results, results obtained and whether they passed or failed: see the example test table attached to this assignment on Blackboard. Note that you should be able to include test cases even if a particular function has not been implemented.

g. Have a section called *Evaluation*. Describe how you went about solving the assignment: what was difficult, what remains to be done, what did you learn, and what mark do you think you should be awarded and why.

Note: this is an "individual" assignment and must be completed as a one-person effort by the student submitting the work. This assignment is **not** marked anonymously.

When you submit your files to Blackboard, you will be shown the wording regarding the **Declaration of Originality;** these are the same as those found on the Anonymous Marking Sheet, which is available on the resources page of the Department's Intranet: http://www.aber.ac.uk/~dcswww/intranet/staff-students-internal/teaching/resources.php. When you click submit, we will assume that you have agreed to the terms of that statement. You do not need to complete a separate form regarding the Declaration of Originality.

Marking scheme

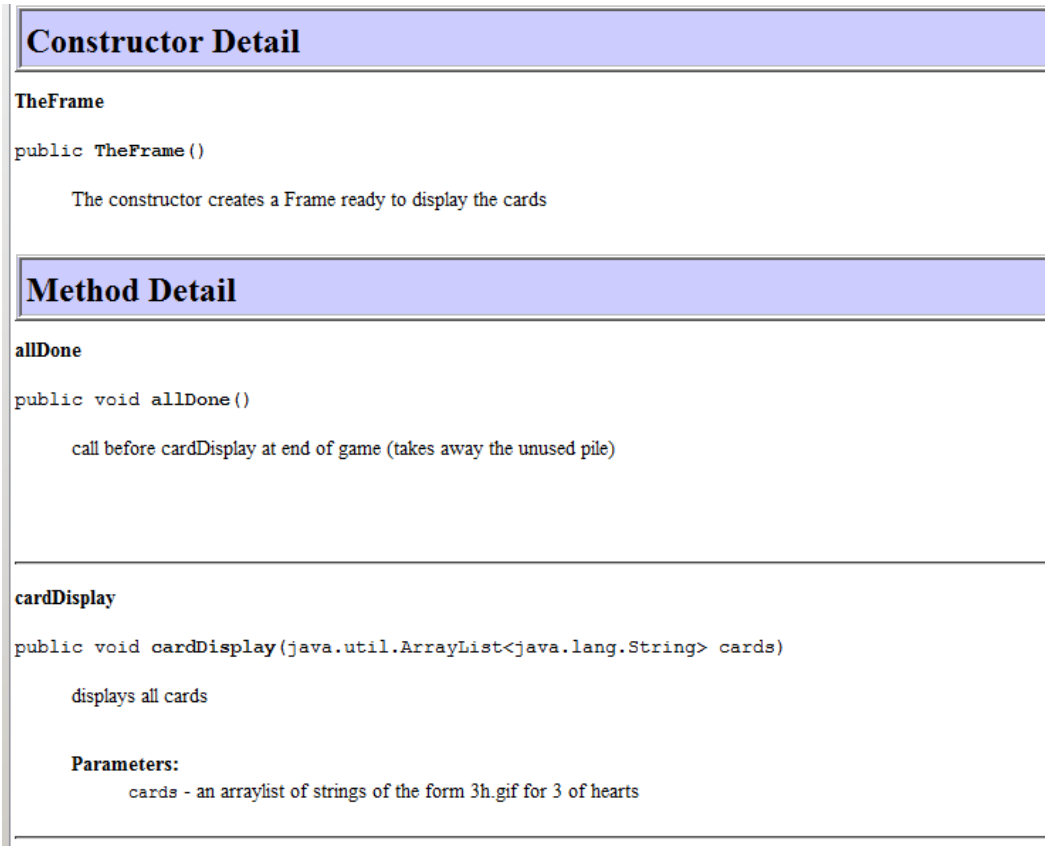
Assessment will be based on the assessment criteria described in Appendix AA of the Student Handbook: http://www.aber.ac.uk/~dcswww/Dept/Teaching/Handbook/AppendixAA.pdf. However, the following table gives you some indication of the weights associated with individual parts of the assignment. This will help you judge how much time to spend on each part. Note that I will award full marks for each category below if you fully complete the requested functionality. However, the flair marks represent work that goes beyond the requested features.

Analysis and design documentation	Does the documentation follow instructions in points *a* to *e* above and follow general upload instructions?	25%
Implementation: general	Are the required functions implemented correctly? Is the code of high quality, e.g. good identifier names, indentation, small methods, commenting, error checking and exceptions, reading and writing a file, use of ArrayLists (or similar), packages?	30%
Implementation: use of inheritance	Has inheritance been used effectively?	10%
Testing	Does the documentation follow the instructions in point *f* above?	10%

Evaluation	Does the documentation follow instructions in point *g* above?	5%
Flair	I am looking for applications that show flair, creativity or innovation. This addresses the Appendix AA 80% to100% assessment criteria that state: *"and will more than completely fulfill the functional requirements."* A good candidate here is to extend and improve the Swing graphical user interface. Ask me if you are not sure.	20%

Provided graphics code

We give you a class and some images to display the game graphically.
Here is its documentation:



Constructor Detail

TheFrame

`public TheFrame()`

The constructor creates a Frame ready to display the cards

Method Detail

allDone

`public void allDone()`

call before cardDisplay at end of game (takes away the unused pile)

---

cardDisplay

`public void cardDisplay(java.util.ArrayList<java.lang.String> cards)`

displays all cards

Parameters:

`cards` - an arraylist of strings of the form 3h.gif for 3 of hearts

---

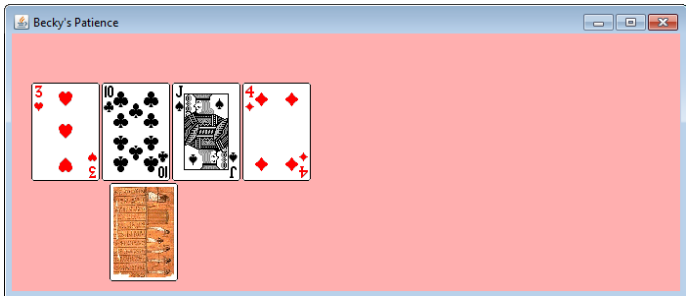
Here is an example of using the code (this is also in the zip file):

```
package uk.ac.aber.dcs.cs12320.cards;
import java.util.ArrayList;
import uk.ac.aber.dcs.cs12320.cards.gui.TheFrame;

public class TestFrame {
        public static void main(String args[]) {
                TheFrame frame = new TheFrame();
```

```
                ArrayList<String> cardStrings = new ArrayList<String>();
                cardStrings.add("3h.gif");
                cardStrings.add("tc.gif");
                cardStrings.add("js.gif");
                cardStrings.add("4d.gif");
                frame.cardDisplay(cardStrings);
        }
}
```

And the result:



There is also a file in the directory that contains the 'cards' so you can start by reading this in (cards.txt), and also one file for each card image.

TO START

- Copy assign.zip into some place in your file store.
- Unzip it - you will get a directory called 'help'.
- In Eclipse import project help.
- Run *main* from TestFrame class to see the above.
- Look at cards.txt for the card 'information'.
- Read in that information.
- Do a dummy menu:
  1 - Print the pack out (this is so you can check that it plays properly)
  2 - Shuffle
  3 - Deal a card
  4 - Make a move, move last pile onto previous one
  5 - Make a move, move last pile back over two piles
  6 - Amalgamate piles in the middle (by giving their numbers)
  7 - Play for me (if two possible moves, makes the 'furthest' move)
  8 – Show low scores
  Q - Quit
- Implement the 'print' function.
- Implement the rest of the menu items in the order they exist above being sure to always have something that works before you go on to the next thing.
- Make sure you support the "save scores" operation when Quit is requested.