Software Development 2 Coursework (SET11103)

“Get out of my swamp”

That’s one mean looking coursework!

Software Development 2

SET11103

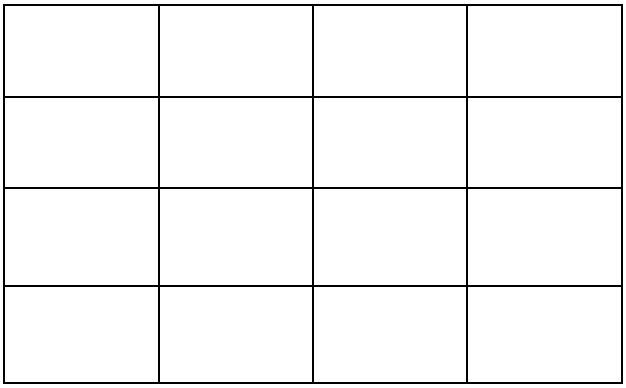
You should write a Java program to meet the specification shown below.
This assignment constitutes 100% of the module assessment.

Problem Statement

You are to create a game called “Get Out of My Swamp”. In this game an ogre, called ‘Hek’, wanders around his swamp, if he encounters an ogre enemy in his swamp he kills it. If he encounters two ogre enemies in the same place they kill the ogre and the game ends. Full details of the game are given below. There is a sample session from a game given at the end of the coursework.

The Swamp

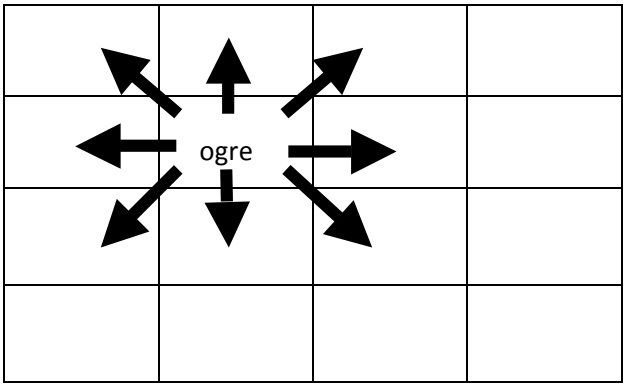
The swamp can be thought of as a four by four grid.



When the game starts the ogre is placed in a random square (i.e. part of the swamp). However he must not be placed in the top left hand corner of the swamp when the game starts, although he can move there during the game.

Ogre

When the ogre moves, he can move to any of the neighbouring squares as shown below. The ogre cannot move out of the swamp.



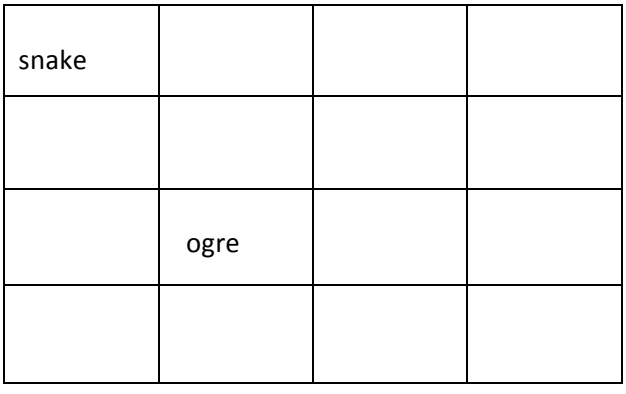
The Ogre can move to any of its neighbouring squares

The square the ogre moves to is chosen at random from all the possible squares that it can move to.

Ogre Enemies

HEK's swamp is not all sweetness and light. There are three types of ogre enemies that can inhabit HEk's swamp; a snake, a parrot and a donkey. There is a hole in the fence around Hek's swamp in the top left corner and this is where the enemies enter the swamp. Once in the swamp the enemies move in exactly the same way as the ogre; i.e. they move randomly to any of the neighbouring squares.

Every time Hek moves there is a one in three chance of an enemy entering the swamp, the type of enemy is completely random.

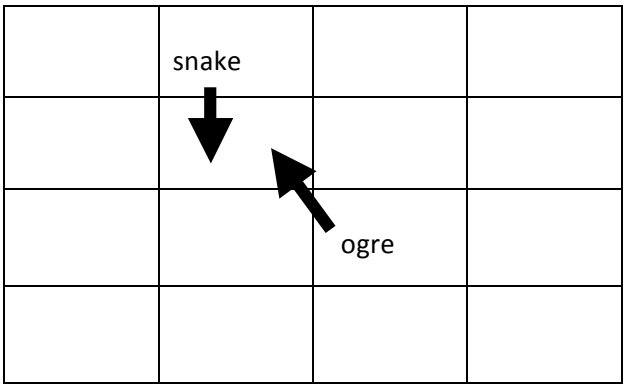


Enemies always enter the swamp in the top left corner

Although there are only three types of enemies. new types of enemy may be available in the future. You should take this into account when coding your solution.

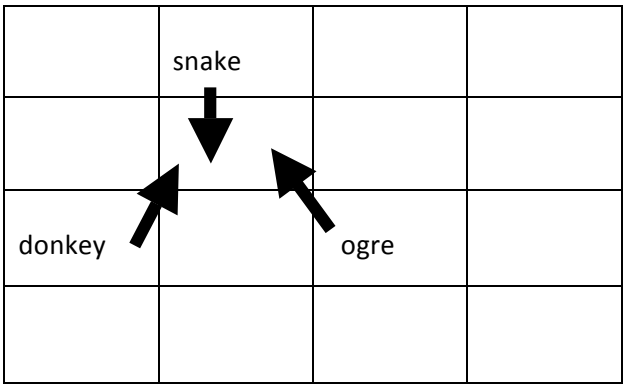
Conflict

If the ogre moves into the same square as an enemy, the ogre kills the enemy and the enemy is removed from the swamp.



An ogre can kill a lone enemy

If the ogre moves into the same square as two or more enemies, the enemies kill the ogre and the game is over.



Two or more enemies can kill the ogre and the game is over

Interface

Your program should have an interface which is completely independent of the game.

Saved Data

If the ogre is still alive and the user chooses not to make another move the game state should be saved using serialization. When the program starts the user should have the option of loading saved data or starting a new game.

JUnit Testing

You should provide comprehensive JUnit tests for each method (excluding the constructors) defined on the class Swamp.

Extensibility

It should be possible to change the size of your swamp by simply changing one integer in your program.

Demo

You must demonstrate your program working during the tutorial on **Tuesday 24th April 2012**. Full details of the demos will be posted on WebCT nearer the time. At the demonstration you will be asked questions about your work, your response to the questions forms part of the assessment. **No demo – no mark**.

Report

You should write a report, which should include details about

- How you created your swamp
- Where and why you have used polymorphic programming
- Your programming style (i.e. showing where and why you have used things like interfaces, enumerator, exceptions etc)
- How your swamp is extensible.
- Any problems encountered during the coding of your game

Marks

Marks will be allocated as follows

Programming Style	70%
JUnit test	10%
Report	10%
Response to questions	10%

Collaboration and Plagiarism

This is an individual piece of assessment and the work submitted should be entirely your own. You are not allowed to collaborate with other students or to copy the work of other students.

Your coursework will be electronically checked against all other submissions. If any plagiarism is detected your coursework will not be marked. In the event of any doubt about authorship you will be interviewed and may be asked questions about any aspect of the work.

Submission

You should demonstrate your program working during the tutorial on **Tuesday 24th April 2012.**

After the demonstration you should submit your assignment. The easiest way to do this is to put everything into your src directory of the project and compress the directory; this means you only need to e-mail one file.
You should include your UML diagram as a jpg file. You can do this very easily in Eclipse; Right click and select Export... UML Diagram... Select the diagram to export...Select jpg when given a selection of formats.

Note that you must send the e-mail from your Napier account as attachments are often removed from external accounts by the e-mail system

Your submission is entirely electronic, you should e-mail your

- .java files
- JUnit test files
- UML Diagram
- report

to j.owens@napier.ac.uk by **23.59 on Tuesday 24th April 2012.**

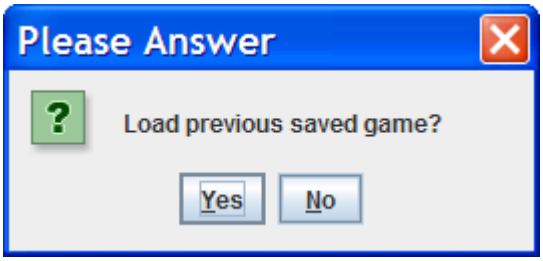
The title of your e-mail should be SD2 submission.

Sample Program

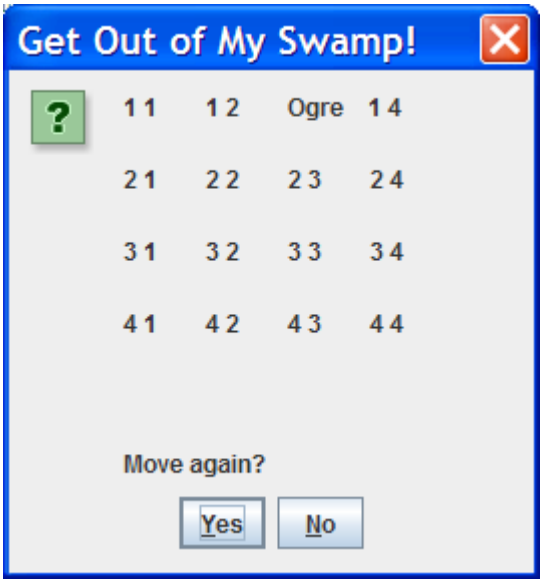
Here are some sample screen shots from a game running.
Your program may vary from this but should still include all the features mentioned in the problem statement.

You can watch videos of this game being played – the link to the videos is on WebCT. The videos give a full insight into how the game is played.

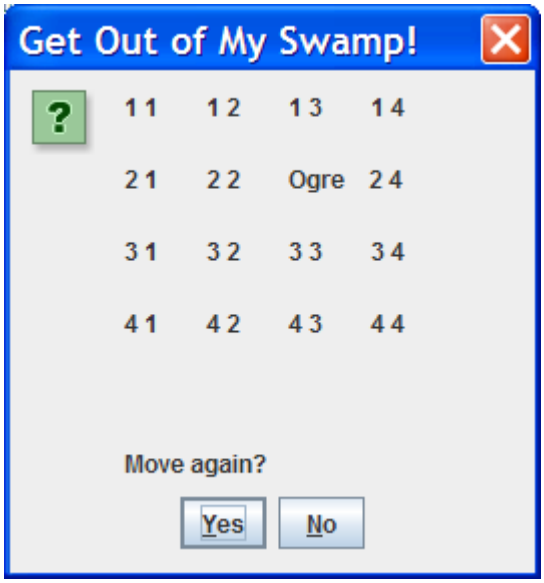
When the game starts the user has the option to load saved data.



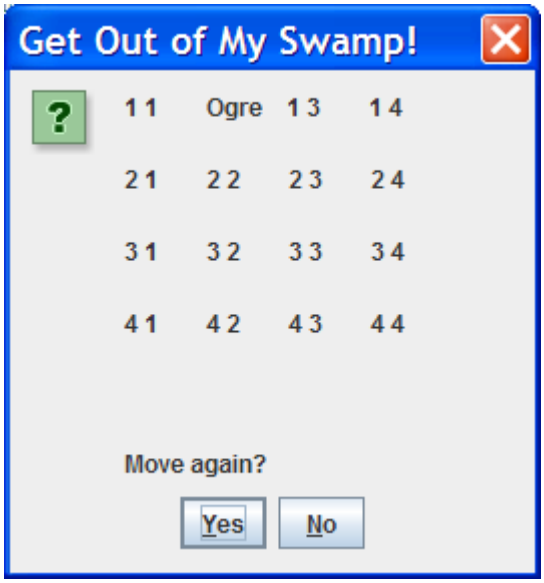
Assuming we say No we will start with a new game.



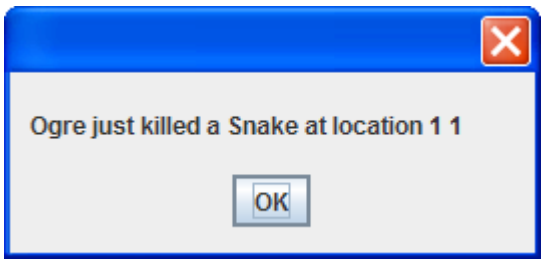
The Ogre has been randomly placed (but not in the top left hand corner) and no enemies have entered the swamp. Let's make a move.



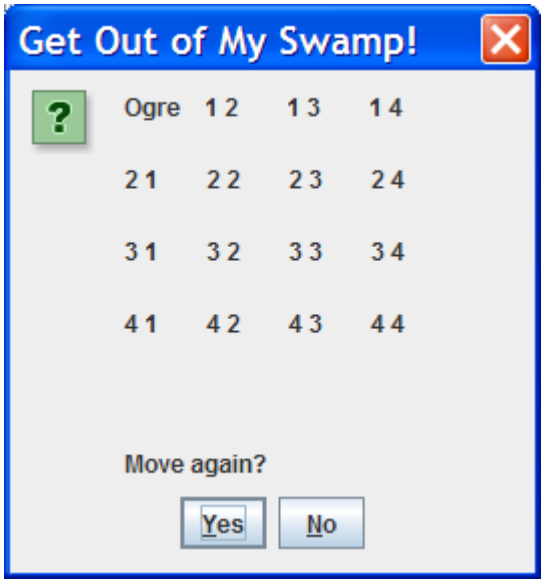
The ogre has moved and no enemies have entered the swamp. Another move...



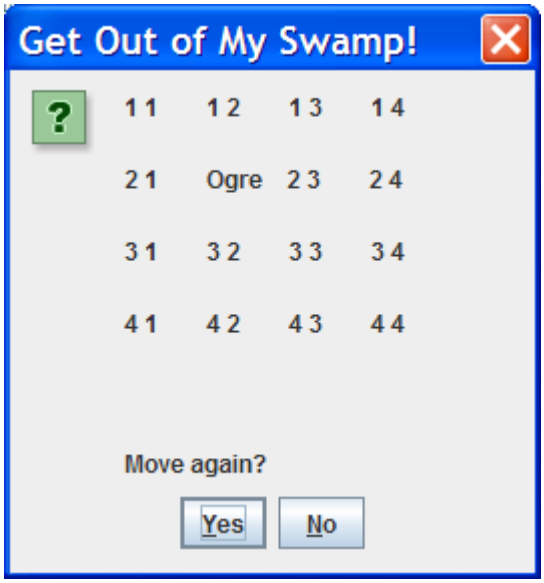
The ogre has moved again and no enemies have entered the swamp. Another move ...



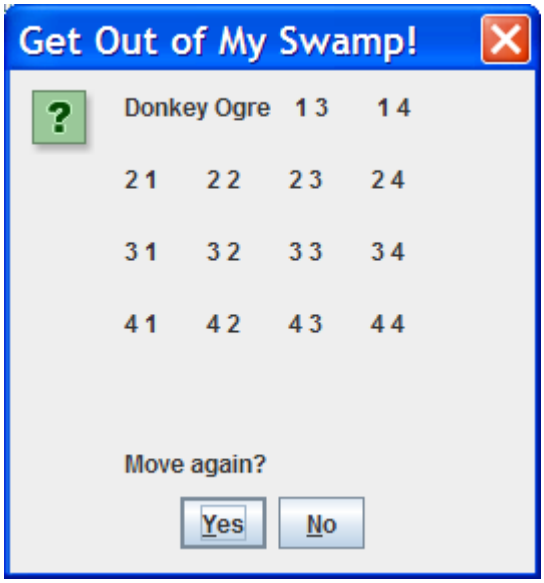
The ogre move to the top left corner just as a snake entered the swamp



Another move...



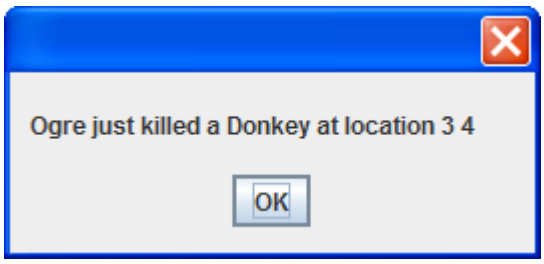
The ogre move, no enemies enter the swamp. Another move...



The ogre moves and a donkey enters the swamp.

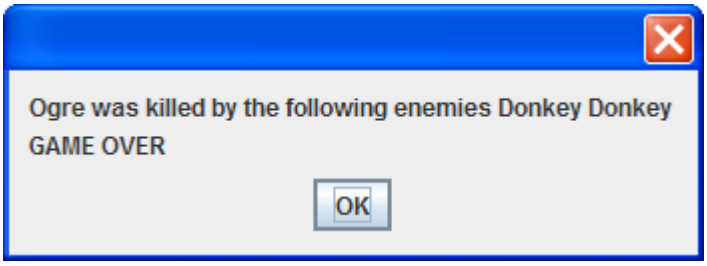
Several moves later ...

There are many enemies in the swamp.
Several moves later.



The ogre kills a donkey

And a few moves later... the ogre encounters two donkeys in the same place ...



... and the game is over.

By changing the value of only one variable you should be able to change the size of the swamp