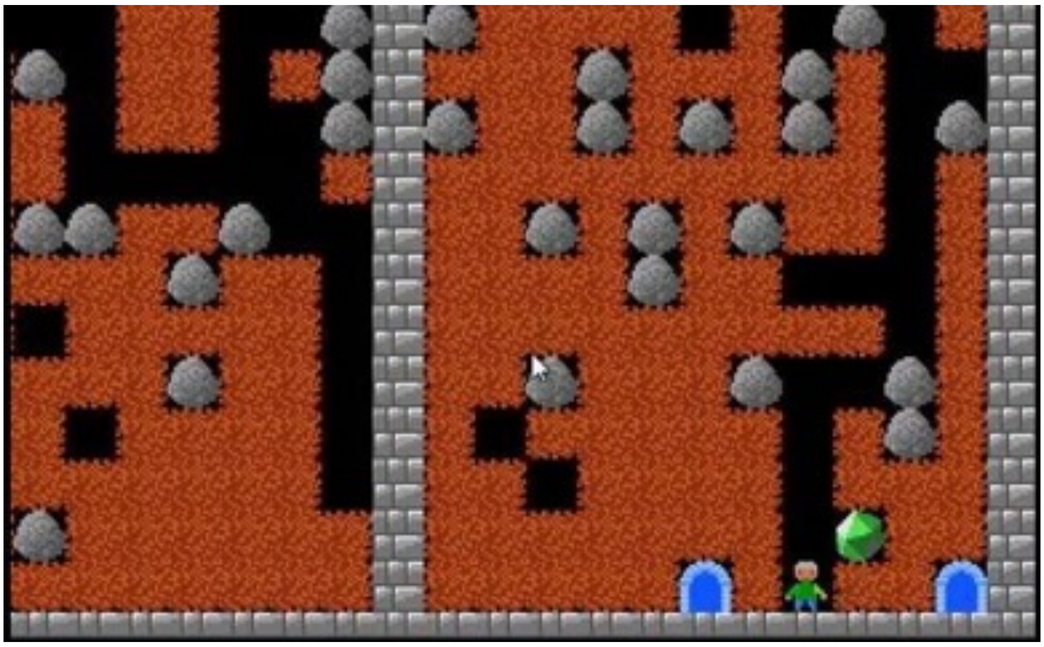
PRA2003: Programming Skills

Week 1: Intro, Modularity, Arrays and Recursion

This week, you will learn about the course, and refresh your memory of methods, arrays and recursion. Your assignment will be to start designing your game using basic Object Oriented Programming (OOP) concepts.

Task 0. Emerald Mine Intro

In this course, you will learn programming concepts that build on your knowledge from MAT1004 and/or MAT2007, and demonstrate your skills by developing a simplified version of the classic computer game Emerald Mine (*aka* Boulder Dash or Supaplex).



Emerald Mine involves a number of levels in which you must collect a number of emeralds (1 pt each) and diamonds (3 pts each), by digging through rocks, walls and other obstacles — including monsters! — before finding the exit to complete the level.

Watch this video to see Emerald Mine in action:
http://www.youtube.com/watch?v=fxaZkMW8454

Task 1. Methods

Code modularity is the practice of designing code that can be *reused* (so we are creating reusable *modules*). In Java this is achieved by constructing methods for specialised tasks and then re-using these methods as much as possible. Use the code provided in `StatKeeper.java` as a starting point.

Create a class (with your name as the class name) which includes a main method that creates a StatKeeper object and uses it to compute some basic statistics for an array of integers. Study the class to understand how this is constructed. For the purpose of this exercise we assume that this array is $\boldsymbol{L=(l_1,l_2,\ldots,l_n)}$=(72, 81, 65, 93, 88, 21, 45, 28, 89, 67), but your methods should work for any other array.

1. Create a `StatKeeper` object in your main class. Add the numbers in the above list *L* to this object.

2. Implement a method that computes the mean of *L*. Call (invoke) this method from main to output the mean of array:

$$\bar{L} = \frac{\sum_{i=1}^{n} l_i}{n}$$

3. Implement a method that uses this to compute the variance of *L*:

$$\sigma^2 = \frac{\sum_{i=1}^{n} (l_i - \bar{L})^2}{n}$$

Call your method from the main class to output the variance of *L*.

4. Implement a method that computes the standard deviation of L.

$$\sigma = \sqrt{\sigma^2}.$$

Use the Math class from the Java API if you need further functions:
http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

5. Implement a method that computes the width of a 95% confidence interval for the mean of *L*:

$$\frac{1.96\sigma}{\sqrt{n}}.$$

Your method must make use of your previous method for computing the standard deviation. Call (invoke) this method from your main class to output the width of a 95% confidence interval for the mean of *L*.

Task 2. Matrix Multiplication

An $r \times c$ matrix is a two-dimensional array of elements with *r* rows and *c* columns. Typically, in linear algebra, a matrix contains numbers. For example:

$$A = \begin{pmatrix} 3 & 1 \\ -1 & 0 \\ 4 & 2 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 2 & -3 \\ -2 & 5 & 1 \end{pmatrix}, \quad C = \begin{pmatrix} -2 & 11 & -8 \\ 0 & -2 & 3 \\ -4 & 18 & -10 \end{pmatrix}$$

In matrix multiplication, an $m \times n$ matrix (on the left) is multiplied by an $n \times p$ matrix (on the right), producing a resulting $m \times p$ matrix. For example, in the product $AB = C$, $m = 3$, $n = 2$ and $p = 3$.

Let element a_{ij} represent the value of the element of matrix *A* at row *i* and column *j* (remember that indexing in Java starts from 0). In the matrix multiplication $AB = C$, the value of the element i^{th} row and j^{th} column of the product C is defined as:

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$$

In other words, sum the individual products of the i^{th} row of the left matrix with the j^{th} column of the right matrix. For example, the first two entries at the top of *C* matrix are obtained as:

$c_{0,0} = 3 \text{ x } 0 + 1 \text{ x } (-2) = -2$ and $c_{0,1} = 3 \text{ x } 2 + 1 \text{ x } 5 = 11$.

Using the code found in `MatrixMult.java` to implement the method for multiplying matrices (notice that all methods related to matrices are implemented in a single class file). When you have a working method for the example shown above, compute the product of the following two matrices:

$$D = \begin{pmatrix} 2 & -2 & 4 \\ 0 & 1 & 3 \\ 5 & 2 & -1 \end{pmatrix}, \quad E = \begin{pmatrix} 0 & 1 & 2 \\ 3 & -2 & 4 \\ 5 & -3 & 0 \end{pmatrix}$$

Task 3. Recursion

The Fibonacci sequence is an integer sequence in which each number is the sum of the two preceding numbers:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

where:

$F_0 = 1$
$F_1 = 1$
$F_n = F_{n-1} + F_{n-2}$

Write a Java program that implements for following methods for generating the Fibonacci number for a given n. Test these over a range of values:

1. `public static long fibonacciI(final int n);`
   which generates F_n iteratively using a single `for` loop.

2. `public static long fibonacciR(final int n);`
   which generates F_n recursively.

Which approach is most suited to this problem? Do you handle possible boundary conditions (0, negatives, large numbers, etc.)? Why are the functions `static`? Why do the functions return `long` and not `int`?

Compare the execution speed of both approaches by timing how long they each take to generate the first 50 Fibonacci numbers, using `System.nanoTime()` as shown below.

```java
long startAt = System.nanoTime();
// run test
long stopAt = System.nanoTime();
double secs = (stopAt - startAt) / 1000000000.0;
```

Why is one method so much slower than the other? *Hint:* 2^{50}

***Note*:** Some sources define the Fibonacci sequence as 0, 1, 1, 2, 3, 5, … with $F_0 = 0$. Using either definition is fine.

Assignment 1 (due 23:59 Monday 5 November)

In this assignment, you will create the basic elements of your Emerald Mine game. Create two classes, `World` and `Main`, in a single Java source file.

The `World` class should have four member variables:

• The number of rows in the world,
• The number of columns in the world,
• The required number of emeralds (remaining) to be collected,
• A two-dimensional array of characters.

Each character in the array represents a game element in the world at that location:
• a dot (`.`) is empty/open space,
• a `p` is the player,
• a `e` is an emerald, and
• an `a` is an alien.

You may also add more member variables as you see fit.

The constructor should take as an argument and initialise the number of rows, columns and emeralds remaining. It should create an appropriately sized array to describe the level and initialise sets each cell to 0 (i.e. empty). It should then place *N* emeralds in *random* locations, where *N* is one more than is required to reach the target score, and place one player and one alien in random starting locations.

The `World` class must have at least the following non-static methods:
1. `public char getMove()` which gets a direction from the user (`u, d, l, r` representing up/down/left/right) by reading the character from the keyboard and returning it.
2. `public int applyMove(char move)` which moves the player in the specified direction. If the player is moving to a location with an emerald, then the player collects the emerald and the number of remaining emeralds is decreased by one. Then, the alien takes a random move (up, down, left, or right), possibly collecting an emerald which then

cannot be taken by the player. If the player runs into the alien or the alien runs into the player, the player dies and the game is lost. If the player or the alien move off the edge of the world, they die. The game is won if the player can collect all the required emeralds without dying. The method returns 0 if the game is not yet over, 1 if the game was won by the player, and 2 if the game was lost.
3. `public String toString()` returns a string representation of the world (see example below).

Question: *Why are these methods non-static?*

The `Main` class has a single main method — you should know its format! The main method creates a world, and then simulates the play of the game by repeating the following three steps:
1. read a direction from the user using `getMove()`,
2. apply the move using `applyMove(move)`, and
3. printing the resulting state of the world using `toString()`.

This is repeated until `applyMove()` returns a terminal (i.e. non-zero) result. When the game is over, print a string printed informing the player whether they have won or lost the game.

A "screenshot" of an initial 10-by-10 world, with 7 required emeralds (8 initially created), is shown below:

```
Welcome to Emerald Mine, PRA 2003 edition.
.......e..
.e........
.........e
..p...e...
.........e
..........
..e.......
.......a..
.e........
......e...
Where to?
```

Honour Policy, Coding Style and Deliverable

• Please refer to Infosheet → Paragraph 7 → Honour policy.

• You are welcome to expand your program with additional functionality but this is not mandatory. Do not deviate from the specific requirements of the assignment!

• If you "borrow" some code from online or any other source, e.g. StackOverflow, clearly state this including a link or reference to the source in your .java source code. Make sure to read the terms of the licence for the code that you are reusing. Most code that is found openly on the internet has specific licences that you should not violate.

• Through the Courses portal, find Assignments→Assignment 1, submit a zipped file (.zip) that contains your source files (.java) for the assignment.

• Please refer to the coding style guidelines uploaded to the Courses portal (under Course materials) on how to write readable programs.

• It is good to discuss ideas, approaches and algorithms with your colleagues but DO NOT COPY EACH OTHER'S CODE! This constitutes plagiarism and is easy to detect.

<u>Hard</u> deadline for submission:

23:59 Monday 5 November 2018