Knight's Tour

This is an example application for Verification and Validation course at UPM.

The application finds solutions for Knight's tour problems.

The team members are:

- Martin Lipták
- Gábor Nagy
- Andreea Oprişan

We recommend reading the documentation files on GitHub: http://github.com/martinliptak/knights-tour/ or in "md" format accessible in the attached "zip" file.

1. Building

Java SE 6 JDK is recommended.

```
bash $ javac *.java
```

2. Running

Finding solution for 5x5 chessboard using initial point 1 1 and 4 other random initial points.

```
bash $ java Main 5
```

Finding solution for 7x7 chessboard and custom initial point 2 3.

```
bash $ java Main 7 2 3
```

3. Specifications

3.1 Definitions

- Chessboard size. Dimension N of square chessboard NxN where the knight moves.
- Initial point. The point where the knight starts his tour.
- Stack. Last-in first-out data structure.
- State. Contains current position of the knight, visited fields of the chessboard and path of the knight.

- State ancestors. New states created from the original state using all possible knight moves.

3.2 Command line

Application reads command line arguments from the operating systems.

- when 1 argument is provided
  - must be integer
  - must be from 5 to 7 (closed interval; including 5 and 7)
  - interpreted as chessboard size
  - finds solutions for 5 default initial points
  - x = 1 and y = 1
  - 4 more random initial points
- when 3 arguments are provided
  - all must be integers
  - argument 1
  - must be from 5 to 7 (closed interval; including 5 and 7)
  - interpreted as chessboard size
  - arguments 2 and 3
  - must be from 1 to the chessboard size (closed interval; including 1 and chessboard size)
  - finds solutions for 1 custom initial point with x = argument 1 and y = argument 2
- prints help when other number of arguments is provided or arguments are incorrect

3.3 Output

Application writes its output to the stdout file descriptor provided by the operating system.

EBNF

Formal specification using Extended Backus-Naur Form.

```
output = size, {solution}, execution time ; size = "Size ", <number>, "x",
<number>, <new line> ; solution = initial field, knights tour ; initial
field = "Solving field ", <number>, " ", <number>, <new line> ; knights
tour = ({knights tour line} | no solution found), <new line> ; knights tour
line = {knights tour number}, <new line> ; knights tour number = <number
padded with spaces to 2 characters> ; no solution found = "null" ;
execution time = "Execution time ", <number>, "ms", <new line> ;
```

Example

``` Size 5x5 Solving field 1 1 3 12 7 16 25 6 17 4 21 8 11 2 13 24 15 18 5 22 9 20 1 10 19 14 23

Solving field 3 4 null Solving field 1 3 25 4 15 10 19 14 9 18 5 16 1 24 3 20 11 8 13 22 17 6 23 2 7 12 21

Solving field 2 1 null Solving field 2 5 null Execution time 483ms ```
```

3.4 Finding solution

- application writes chessboard size
- application writes solutions
    - for every initial point (see command line arguments)
    - application writes initial point coordinates
    - application searches for the solution
        - first state at the initial point is pushed to the stack
        - internal loop
        - at last 100,000,000 iterations are allowed
        - if stack is empty, the solution does not exist
        - a state is popped from the stack
        - if the popped state is final, we have found the solution
        - otherwise all the ancestors of the popped state are pushed to the stack
    - if solutions exists
        - application writes path of the knight
    - if solution does not exist
        - application writes "null"
- application writes execution time

4. Testing strategies

4.1 Strategies to test if the input parameters are valid:

1. Cases in which the given arguments are not numeric or not integer values:

```
bash $ java Main a

bash $ java Main 6.3 4 4

bash $ java Main 6 @ 3
```

Faults found: the programme crashed because the arguments weren't correct.

2. Cases in which a knight cannot move in the chessboard(the chessboard size is too small):

```
bash $ java Main -2

bash $ java Main 0

bash $ java Main 1

bash $ java Main 4
```

Faults found: the programme crashed because the knight couldn’t move in the chessboard (he needs a chessboard larger or equal to 4x4).

3. Cases in which the chessboard is too big and the expected counting time is over some minutes (hours):

```
bash $ java Main 8

bash $ java Main 12
```

Faults found: the waiting time is too big.

4. Cases in which the initial point given at the input exceeds the limits of the chessboard:

```
bash $ java Main 5 6 6

bash $ java Main 6 -1 0

bash $ java Main 5 7 2
```

Faults found: the programme crashed because the arguments weren’t correct.

4.2 Strategies to test the programme when the input parameters are valid:

1. The program was tested with the first input parameter having values between 5-7.

```
bash $ java Main 5

bash $ java Main 6

bash $ java Main 7
```

Faults found: sometimes there isn't any solution for some random generated points (this isn't quite a fault, it is normal that for some points there isn't solution. The output of the this kind of result is "null")

2. The programme was tested for the custom initial point with the blind strategy, taking into consideration that the initial point must be between the limits of the chessboard

```
bash $ java Main 5 2 4

bash $ java Main 6 3 3

bash $ java Main 6 2 4
```

Faults found: sometimes there isn't any solution for some random generated points (this isn't quite a fault, it is normal that for some points there isn't solution. The output of the this kind of result is "null")