CMPT225, Spring 2021
Homework Assignment 1
Due date: Friday, January 29, 2021, 23:59

You need to implement the following classes:
Part 1:
- *myiterators/ArrayIterator.java*
- *myiterators/FibonacciIterator.java*
- *myiterators/RangeIterator.java*

Part 2:
- *rushhour/RushHour.java*
- *rushhour/IllegalMoveException.java*

Note that the files for Part 1 must be in under the package (folder) **myiterator/**,
and the files for Part 2 must be under the package (folder) **rushhour/**.
You may add more classes to your solution if necessary.

Submit all your files in ***assignment1.zip*** to CourSys. Make sure your zip file can be unzipped using the command "*unzip assignment1.zip*" in CSIL.

Discussion with others: You may discuss the assignment with your classmates/tutors (or anyone else), but coding must be entirely your own.

References: You may use textbooks, wiki, stack overflow, geeksforgeeks, etc. If you do, specify the references in comments. Posting questions online asking for solutions (e.g. using chegg.com) is prohibited.

Readability: Your code should be readable using the standard Java conventions. Add comments wherever is necessary. If needed, write helper functions or add classes to improve readability.

Compilation: Your code MUST compile in CSIL using javac.
Make sure that your code compiles without warnings/errors.
If the code does not compile in CSIL the grade on that part will be 0 (zero).
Even if you can't solve a problem completely, make sure it compiles.

The assignment will be graded mostly **automatically**, with some exceptions.

Do not add main() to your solutions. The main() method will be in the test files.
Examples of such tests are provided in *TestIterators.java* and *TestRushHour.java*.

Warnings: Warnings during compilation will reduce points.
More importantly, they indicate that something is probably wrong with the code.

Testing: Test your code. Examples of tests are included. Your code will be tested using the provided tests as well as additional tests. You should create more tests to check your solution.

Good luck!

Part 1 [50 points] - Integer Iterator

IntegerIterator represents a sequence (finite or infinite) of integers a_0 ,a_1 ,a_2 ....
The interface has the following public methods:

`public boolean hasNext()` : Returns true if the sequence has more elements, and returns false otherwise. If the sequence is infinite, has Next() always returns true.

`public int getNext()`: Returns the next element in the sequence.

`public void reset()`: Resets the iterator to the initial element of the sequence, i.e., to a_0.

The interface is provided with the assignment under the package myiterators.

Your goal is to write the following 3 classes implementing this interface:

1) **ArrayIterator [15 points]**
   The class has one constructor that gets an array of ints.
   `public ArrayIterator(int[] ar)`
   The iterator iterates through the elements of the array.

2) **RangeIterator [15 points]**
   The class has three constructors:
   `1) public RangeIterator(int n)`
   The iterator iterates through the elements 0,1...n-1. If n<0, then the sequence is empty.
   `2) public RangeIterator(int a0, int n)`
   Defines the sequence a_0, a_0+1, a_0+2,...n-1. If n<a_0, then the sequence is empty.
   `3) public RangeIterator(int a0, int n, int diff)`
   Defines the sequence a_0, a_0+diff, a_0+2diff,..., a_0+k*diff, where k is the maximal integer such that a_0+k*diff<n. If n<a_0, then the sequence is empty.

3) **FibonacciIterator [20 points]**
   The *Fibonacci sequence* is defined as a_0=0,a_1=1, and $a_n = a_{n-1}+a_{n-2}$ for all n>1.
   More generally, given the initial values of a_0 and a_1 we define the sequence as $a_n = a_{n-1}+a_{n-2}$ for all n>1.

   The class implements the infinite Fibonacci sequence. It has two constructors:
   `1) public FibonacciIterator()`
   Defines the Fibonacci sequence with a_0=0,a_1=1.
   `2) public FibonacciIterator(int a0, int a1)`
   Defines the more general Fibonacci sequence with a_0 and a_1 given as arguments to the constructor.

Part 2 [50 points] - Rush Hour

RushHour is a class representing the game RushHour.
See wiki page for the description https://en.wikipedia.org/wiki/Rush_Hour_(puzzle)
Find the game online and play it.

Your goal in this part is to read an initial board from a file, and move the cars on the board.

The file is expected to have 6 rows with 6 chars in each row.

- A dot represents an empty cell.
- Each car is represented by a letter.
- Each car can go either only horizontally or only vertically. The allowed direction of a car is defined by the longer side of the car.
- Vehicles can only be moved along a straight line on the grid; rotation is forbidden
- The red car (the one we need to get to the exit) is denoted by X.
- The goal is to bring the red car, denoted by X to the right edge of the board.
- You may assume that the file always represents a legal board.

See board1.txt for an example.

Write the class RushHour with the following constructor and three public methods:

```
1) public RushHour(String fileName) throws FileNotFoundException
```

The constructor gets a name of a file as an argument. It reads the file and initializes the board.

```
2) public void makeMove(char carName, int direction, int length)
                throws IllegalMoveException
```

The method moves the car in the car for length steps in the given direction.
If the move is illegal the method throws IllegalMoveException. The move can be illegal because the direction is incorrect or because some cells on the way are occupied.

The class IllegalMoveException is provided with the assignment.
It is very simple, it just inherits from Exception.

```
3) public boolean isSolved()
```

Returns true if and only if the car X touches the right edge of the board.

Remarks:
The provided class RushHour has several constants: UP, DOWN, LEFT, RIGHT, and SIZE.
They are provided for your convenience. You may use them , but you don't have to.

You may add additional private methods if needed. You may also add new classes if needed.
For example, you may want to consider creating a class Car to represent a car on the board.