Given: Monday March 17, 2014
Due: Monday April 7th, 2014 by Noon (No late assignments will be accepted)

Suggested work deadlines

- Friday March 22st: Inheritance set up and all parsing done (with no error checking)
- Monday March 31st:
  - Moving user and all other characters is done. Dead NPC are removed as game runs.
  - Error checking in parsing done. (Exception handling)
- Friday April 4th: Battle system is done. Complete game is finished and documented

Weight: 7%

The assignment is to be completed individually, in accordance with the policies and expectations in the course outline.

If you are unable to complete all aspects of the assignment, included a description of what you did not complete and any known bugs in the header of the class that contains the main method

Objectives:

- Very important: Object Oriented Design (a large part of your marks)
- To gain experience developing a custom class with instance variables.
- To learn about the very important concept of "encapsulation".
- To practice thinking about objects as having both *state* (via properties) and *behavior* (via public operations).
- To gain experience working with multiple objects ("instances") of a class, each with its own distinct identity and state.
- Use of aggregate or dependency relationships between objects
- JavaDoc documentation (for headers on files and for important methods)
- Using code that has been provided by someone else
- Java packages
- Java Interfaces, Inheritance and polymorphism
- Exceptions

Associated textbook sections: before starting, students should be familiar with most of **Chapter 1,4,7,8, 9, 10,11**

Note: You can use ArrayLists for this assignment

Short summary

You will be making a simple text based game called "**THE WALKING CODE: ZOMBIE ESCAPE EDITION**" that will employ interfaces, inheritance, polymorphism and exception handling.

Assignment restrictions:

- **You must use the provided BlueJ project as a starting point for the assignment. It includes a starting package structure that cannot be changed. You can create new packages to organize the code you create.**
- **You must use the UserInteraction class for all input and output to the program. You can not modify this class in any way. You can only have a single instance of the UserInteraction class for the entire program. The reference should be passed around as needed.**
- **You must use the GameRandomNumber class or all random number generation in the program. You can not modify this class in any way.**

Note: You should be trying to implement a better object-oriented design for this assignment based upon the feedback you received on assignment 1,2,3 and 4 Your overall design will affect your final grade.

Detailed description

You will be making a program that will allow the user to control a game character that has to move through a grid style world and avoid Zombies. The zombies will be moving at the same time and may get a chance to battle the user. The goal for the user is to get to the end point in the world alive. You will have to add onto the code that has already been provided.

The Game World

In the game the user will see a 2D grid style world. In the world they will be presented with walls, rats and zombies; the player cannot walk over any of these entities. The goal for the user is to get around the walls and rats while avoiding the zombies as much as possible to get to the end point with the fewest number of actions taken. They may have to battle a zombie or a group of zombies if they come too close to each other.

All movement is done through open locations that indicate that they can be moved onto. Below is an example world

```
 ------------
|#   F    E#|
|#      ####|
|# Z       #|
|#  R   S  #|
|#         #|
|# U       #|
 ------------
```

```
    # = Represents a wall - a block/location the user cannot move onto

Space = Open location in the world a bloc/location the user can move onto

    E = the end location of the game that the user must move onto to win the game

    U = the current position of the user
    R = example of a rat in the world

Z,F,S = example of zombies in the world

    | = the left and right boundaries of the World/Maze

    - = the top and bottom boundaries of the World/Maze
```

Note: The user is not allowed to move outside the boundaries of the world. They are only allowed to move around open locations (i.e. the ones represented by a space character).

Move Actions the user can make

In this game the user is allowed to move 1 position, each turn, in one of the following directions: NORTH, NORTH EAST, EAST, SOUTH EAST, SOUTH, SOUTH WEST, WEST, NORTH WEST. So for example if we had the following displayed:

```
------------
|#   F    E#|
|#      ####|
|# Z        #|
|#  R   S  #|
|#         #|
|# U       #|
------------
```

And the user decided they wanted to move the game character to the NORTH then the updated world with the new user's location would look like the following.

```
------------
|#   F    E#|
|#      ####|
|# Z        #|
|#  R   S  #|
|# U       #|
|#         #|
------------
```

If the new location the user wants to move onto is outside the boundaries of the world or is not an open location, the game character is not moved at all. In the world only one object can occupy a cell in the grid at a time. So this means that rats and zombies cannot walk over each other.

All objects in the world have the following properties

- A row and column location of where they are currently located in the world
- A property if another object can temporarily occupy the location the current object is on.
- A letter that is used to display the object

In the world you have the following elements:

- **Walls**
    - They do not move and only occupy a single cell In the world
    - They cannot be moved onto
    - Display Character: #
- **Game Characters** - including the user
    - All have an energy/health level (when the energy level reaches zero or below they are considered dead). There is no maximum on the energy level.
    - All have name that is assigned in the input file
    - **Game Character Type:**
        - **The User (The single good guy)**
            - Has a strength factor (values 1 to 100)
            - Has a scared factor (1 to 100)
                - This gets incremented by 1 every time the user battles an enemy and wins
            - Can hold a single weapon. If no weapon is assigned the weapon should default to the gold old "FIST" weapon.
            - Display Character = U
        - **Rats**
            - Picks a single random direction to move one step each time it is allowed to move.
            - Cannot be attacked or killed by any other character (by decree of Lord Kidney)
            - Each time a rat makes a move they lose 10 energy points. If a rat dies they are removed from the world
            - Display Character = R
        - **Zombies**
            - Can hold a single weapon. If no weapon is assigned it should default to the old "FIST" weapon.
            - **Zombie types**
                - **Dumb Zombie** – picks a single random direction to move one step in each time it is allowed to move.
                    - Each time it moves it loses 5 energy points. If the dumb zombie dies they are removed from the world. In this case dies = falls apart and cannot move.
                    - Display Character **= Z**
                - **Fast Zombie** – picks a random direction each time it is allowed to move and takes 2 steps in that direction
                    - Is allowed to hold a second weapon (defaults to FIST If not given)
                    - Does not loose energy as they move
                    - Display Character = F
                - **Fast Stumble Zombie**
                    - Has a stumble factor (between 1 to 3)
                    - Does not loose energy as they move

      - Each time it is allowed to move it picks a random direction and moves one step and repeats this process based upon the stumble factor (i.e if the stumble factor is 2 then a random direction is picked and moved to followed by a second direction is picked to move to next)
      - Display Character = S

- **Weapons**
  - All weapons will be able to calculate "attack power" that is specific to each weapon.
    - The attack power is used during a battle
    - If a attack power can never be below zero
  - **Weapon types**
    - **Fist --** the good old trusty fist to beat your target up with
      - attack power: always 1 no matter how much it is used
    - **CrowBar**
      - Given a starting Power value
      - attack power = Power value
    - **ChainSaw**– "Just a little off the top"
      - Given a starting power value
      - attack power = (2*Power Value)
    - **BaseballBat**
      - Given a starting power value and a weight
      - attack power = (weight*Power Value)
    - **The Shot Gun** – "I call this my BOOOOM stick"
      - Given a starting power value
      - attack power = (666*Power Value)

General flow of the program

1. Ask the user for the game file
   a. This file will contain the startup information for the current game
      i. The size of the world
      ii. The location of the end point
      iii. The starting location of the user
      iv. The users starting weapon
      v. The location and types of zombies, walls and rats
         1. Plus possible weapons or zombies

2. Loop until user is dead, quits or they have reached the end point and won the game
   a. Print the world
   b. Ask the user what direction they want to move in
   c. Move the user
   d. Check if user is now on end point (exit game with win message if they are)
   e. Move all zombies and rats
   f. Remove any dead zombies or rats

    i. Note: a debug message should be printed to indicate what dead entities have been removed. See provided output example for how you should format these messages.

  g. Check for possible battle situation

    i. If the user has one or more zombies within a radius of 1 step around them they must battle and defeat all zombies before they can move on.

Battle Situation Algorithm

Steps for a battle (Note: see provided output example for formatting details)

1) All battle actions happen automatically and are executed by the game
   a) No need for user input.
   b) All steps of the battle should be reported to the user/console in the same format as shown in examples.
   c) During this phase no other zombies or rats in the world will move until the battle is over.
2) You must loop through all zombies around the user to battle each enemy individually until either the enemy is dead or the user is dead.
   a) Steps for single battle
   b) Single battle between a zombie and the user
      (1) The user gets a chance to attack the zombie
         (a) Generate a random number (Using GameRandomNumber in the utility package). If the number is even then the users' attack is successful and the zombie looses the following amount of energy
            (i) the users' weapon attackpower + (strength - scared factor)
      (2) The Zombie gets a chance to attack the user
            (i) Generate a random number (Using GameRandomNumber in the utility package). If the number is odd the attack is considered to be successful and the user loose's the following amount of energy
            (ii) The attack power of the zombies' weapon (or sum of attack power or each weapon)
      (3) If the user is dead exit the game.
      (4) If the zombie is dead and there are still more enemies move on to the next enemy otherwise continue on with the normal game flow.
         (a) The users scared factor should be now incremented by one.
      (5) If both are still alive continue the battle and got o step 1.

Example battle output (Your output should try to match the shown format)

User move count: 18

```
 --------------------
|#      E            |
|# S  # #            |
|#                   |
|     Z              |
|Z      Z            |
|                    |
|                    |
|F                   |
|                    |
|    ZU              |
|                    |
|        RR          |
|       ## # ##      |
|                  R |
|                    |
|      Z             |
|                    |
|                    |
|                    |
|                    |
  --------------------
====== BATTLE ======
Number of enemies: 1
Enemy: row= 9, col= 4, type=Z, name=zombie4     , energy level= 210, Weapon1=BASEBALLBAT
[attackPower=  48]
=== User vs zombie4
User state: row= 9, col= 5, type=U, name=User        , energy level= 100,
Weapon1=CHAINSAW [attackPower=  10], strength= 300, scared=  0
Attack   1 (User Energy= 100, Enemy Energy= 210)
 User: attack is unsuccessful.
Enemy: attack is unsuccessful.
Attack   2 (User Energy= 100, Enemy Energy= 210)
 User: attack is successful. Damage=310
user has won
====================
<hit enter to continue>
```

Input File Description (Examples in the input directory)

- First line is the row and column size of the world.: row , col
- Second line, (row, col) of the end point
- Third line is user information: start row, star col, starting energy, strength
- Fourth line is the users starting weapon. (see weapon description below)
- The rest of the file contains information for all objects in the world (note there could be extra spaces between pieces of data, your parsing code should deal with this).
- Each object can be any of the following
  - Wall : row, col, WALL
  - Rat: row,col, RAT, name,start energy
  - Dumb Zombie: row, col, DZ, name,start energy
    - The next line after this will be the weapon description
  - Fast Zombie: row, col, FZ , name,start energy, number of weapons (1 or 2)
    - The next line(s) after this will be the weapon(s) description
  - Fast Stumble Zombie : row, col, FSZ, name, start energy, stumble factor
    - The next line after this will be the weapon description
- A weapon can be any of the following
  - FIST
  - CROWBAR , power
  - CHAINSAW , power
  - BASEBALLBAT , power , weight
  - THESHOTGUN, power

With the example world shown below

```
 ---------
|# Z    E|
|#R   # #|
|#FU   S |
 ---------
```

	0	1	2	3	4	5	6	7
0	#		Z					E
1	#	R				#		#
2	#	F	U				S	

File:

```
3,8
0,7
2,2,100,50
CROWBAR,2
0,0,WALL
1,0,WALL
2,0,WALL
1,5,WALL
1,7,WALL
2,1,FZ,zombie1,200,2
CHAINSAW,5
FIST
0,2,DZ,zombie2,100
BASEBALLBAT,16,3
2,6,FSZ,zombie3,500,3
THESHOTGUN,10
```

Exception handling

1) Must properly use exception handling to make sure the file opens properly
2) While reading in the world information you must handle in some fashion any issues where the format of the file is not correct. For example, when reading a line where an integer is expected but a string has been entered. Do not go too over board with this; try to keep it simple on yourself. If an error is found just display a message to the user and let them exit the program.
3) If you are reading and expecting data and nothing is left in the file. General I/O exceptions.
4) If you place an object in the world and there is already one at the given location
   a) In this case you must make your own new exception class (IE inherit from Exception) and throw an exception of this new type

Working with the provided code

You have three main methods you will need to complete in the TheWalkingCodeGame class to properly interface your code with the provided system:

- private boolean loadGameFile(String fileName)
  - deals with loading in the file to the system
- private void runUserMoveAction(String direction)
  - running a specific action from the user (basically moving the user)
- private void listGameState()
  - listing out the current state of all entities in the world

Example output for listGameSate using larger.txt

Below is an example output for the listGameState for the file larger.txt when the program first starts up. You should try to match this style of formatting.

```
>list
row= 0, col= 7, type=E
row= 5, col=19, type=U, name=User, energy level= 100, Weapon1=CHAINSAW [attackPower=  10], strength= 300, scared=  0
row= 0, col= 0, type=#
row= 1, col= 0, type=#
row= 2, col= 0, type=#
row= 1, col= 5, type=#
row= 1, col= 7, type=#
row= 2, col= 1, type=F, name=fastzombie1 , energy level= 200, Weapon1=FIST [attackPower=1], Weapon2=FIST [attackPower=1]
row= 0, col= 2, type=Z, name=zombie1     , energy level= 100, Weapon1=BASEBALLBAT [attackPower=  48]
row=10, col= 2, type=Z, name=zombie2     , energy level= 100, Weapon1= CROWBAR [attackPower=  16]
row= 2, col= 6, type=S, name=stumbler1   , energy level= 500, Weapon1= SHOTGUN [attackPower=66600]
row=10, col=10, type=R, name=rat1        , energy level= 100
row=11, col=10, type=R, name=rat2        , energy level= 200
row=12, col=11, type=R, name=rat3        , energy level= 300
row=19, col=19, type=R, name=rat4        , energy level= 400
<hit enter to continue>
```

Coding Style and Documentation Standards

- **You must use java doc for your headers on each file and on important methods**
    - **Look to the java doc template examples at the end of this document**
- Choose self-documenting identifiers (e.g. for variables, methods, etc.)
- explicit initialization of variables (where appropriate)
- consistent and correct use of white space, including:
    - proper indentation and use of white space to highlight the logical structure of an algorithm
- marking all methods as either public or private, as appropriate (private for helper and public for service methods)
- marking all instance data as private
- making variables local, unless they are for genuine per-object properties
- avoiding overly long/complex methods by instead delegating key subtasks to other methods (this includes avoidance of code duplication)
- using inline comments, sparingly, to clarity especially tricky or less-obvious segments of code
- avoiding hard-coded magic literals in favour of named constants
- Choose good classes and service methods based upon the practice you have had in class.

Submission Instructions

Before attempting the steps below, ensure your solution compiles without errors or warnings, ensure all existing automated tests pass (if there are any), and ensure your program works as expected. Correct any problems before continuing. **Code that does not compile and can't be tested will be heavily penalized.**

Important reminder: this assignment is to be completed *individually*. Students are strictly cautioned against submitting work they didn't do themselves.

Please follow these submission instructions **exactly**.

1. Rename your BlueJ project folder (if you haven't already) as: <LastName>_<FirstName>_***Asg5***
2. **If you were unable to compete all aspects of the assignment, included a description of what you did not complete and any known bugs in the header of the class that contains the main method**
3. Submit the entire BlueJ project folder (with all its contents) to the "submit" folder, which appears under the "I:" drive each time you log in to a university PC. If you are submitting from a non-university computer (e.g. a home PC), you can access the submit folder via secure.mtroyal.ca, into which you can upload a compressed (e.g. ".zip") version of your main folder.

Javadoc Templates

File header

```
/**
 *  <include description of the class here>
 * @author <your name>
 * @version 1.0
 * Last Modified: <date> - <action> <who made the change>
 */
```

Method with no parameters and no return type

```
/**
 *  <a description of what the method does>
 */
```

Method with no parameters and has a return type

```
/**
 *  <a description of what the method does>
 *  @return <a description of what is returned, including if errors are returned>
 */
```

Method with parameters and no return type

```
/**
 *  <a description of what the method does>
 *  @param <parame1name> <a one line description of the parameter>
 *  @param <parame2name> <a one line description of the parameter>
 *  < include all parameters in a similar way as above>
 */
```

Method with parameters and has return type

```
/**
 *  <a description of what the method does>
 *  @param <parame1name> <a one line description of the parameter>
 *  @param <parame2name> <a one line description of the parameter>
 *  < include all parameters in a similar way as above>
 * @return <a description of what is returned, including if errors are returned>
 */
```