Matthew Smith

Derek Schissler

**Clue Refactor Comments**

ISSUE: DRY code in BadConfigFormatException

Example:

**public** **void** error() {

**try** {

FileWriter fstream = **new** FileWriter("ErrorLog.txt");

BufferedWriter out = **new** BufferedWriter(fstream);

out.write("An input file is in an invalid format. Check the documents.");

out.close();

} **catch** (Exception e) {

System.*out*.println("Error: " + e.getMessage());

}

}

**public** **void** error(String name) {

**try** {

FileWriter fstream = **new** FileWriter("ErrorLog.txt");

BufferedWriter out = **new** BufferedWriter(fstream);

out.write(name + " is in an invalid format. Check the document.");

out.close();

} **catch** (Exception e) {

System.*out*.println("Error: " + e.getMessage());

}

}

Solution: Combine error function into one, use constructor call to super to create the error message, pass it to the error function, and write that to the log. Also named error something more helpful.

**public** BadConfigFormatException() {

**super**("An input file is in an invalid format. Check the documents.");

logError();

}

**public** BadConfigFormatException(String fileName) {

**super**(fileName + " is in an invalid format. Check the document.");

logError();

}

**public** **void** logError() {

**try** {

FileWriter fstream = **new** FileWriter("ErrorLog.txt");

BufferedWriter out = **new** BufferedWriter(fstream);

out.write(getMessage());

out.close();

}

**catch** (Exception e) {

System.*out*.println("Error: " + e.getMessage());

}

}

ISSUE: Needlessly complex code in Board constructor. Code was used to determine whether or not it was Rader’s test files or your own test files. Use the non-default constructor instead

Example:

**if** (raderTests) {

mapName = "ClueLayout.csv"; //used in CR tests

legendName = "ClueLegend.txt";

} **else** {

mapName = "ClueMap.csv"; //default names- change to your own

legendName = "legend.txt";

}

Solution: Delete offending code, change code in CR\* tests to use her config files.

ISSUE: Didn’t declare a type for Set, Map when declaring instance variables targets and function return types. While it works, you want to include types so we know what each variable will include, instead of having to dig through a function looking for it.

SOLUTION: Add appropriate type for each declaration of Set or Map. Change targets type in non-default constructor to HashSet<BoardCell> instead of HashSet<Integer>.

ISSUE: Code repetition in getRoomCellAt function. You have two instances of the function, one for getRoomCellAt(int cell) and one for getRoomCellAt(int row, int col). This should be simplified, see solution below.

EXAMPLE:

**public** RoomCell getRoomCellAt(**int** row, **int** col) { //gets cell if it's room, otherwise gives null

**int** index = calcIndex(row, col);

**if** (cells.get(index) **instanceof** RoomCell)

**return** (RoomCell) cells.get(index);

**else**

**return** **null**;

}

**public** BoardCell getCellAt(**int** row, **int** col) { //gets cell regardless of walkway/room definition

**int** index = calcIndex(row, col);

**return** cells.get(index); //uses index method

}

**public** RoomCell getRoomCellAt(**int** index) { //index form of getRoomCellAt

**if** (cells.get(index) **instanceof** RoomCell)

**return** (RoomCell) cells.get(index);

**else**

**return** **null**;

}

**public** BoardCell getCellAt(**int** index) {//gets cell regardless of walkway/room definition

**return** cells.get(index);

}

SOLUTION:

**public** RoomCell getRoomCellAt(**int** row, **int** col) {

**int** index = calcIndex(row, col);

**return** getRoomCellAt(index);

}

**public** RoomCell getRoomCellAt(**int** index) {

**if** (cells.get(index) **instanceof** RoomCell)

**return** (RoomCell) cells.get(index);

**else**

**return** **null**;

}

**public** BoardCell getCellAt(**int** row, **int** col) {

**int** index = calcIndex(row, col);

**return** getCellAt(index); //uses index method

}

**public** BoardCell getCellAt(**int** index) {

**return** cells.get(index);

}

ISSUE: Logic in calcTargets is needlessly complex. Instead of creating a new array ‘unvisited’, you should simple iterate through adjacent cells, and if conditions are met, the cell will be added to targets.

EXAMPLE:

**if** (adjMtx.get(index).size() == 0)

**throw** **new** RuntimeException("Invalid location");

**for** (Integer i : adjMtx.get(index))

**if** (visited[i] == **false**) unvisited.add(i);

**for** (**int** i : unvisited) {

visited[i] = **true**;

**if** ((steps == 1 || getCellAt(i).isDoorway())) targets.add(getCellAt(i));

**else** {

calcTargets(i, steps - 1);

}

}

SOLUTION:   
**if** (adjMtx.get(index).size() == 0)

**throw** **new** RuntimeException("Invalid location");

**for** (Integer i : adjMtx.get(index)) {

**if** (visited[i] == **false**) {

visited[i] = **true**;

**if** ((steps == 1 || getCellAt(i).isDoorway()))

targets.add(getCellAt(i));

**else**

calcTargets(i, steps - 1);

visited[i] = **false**;

}

}

ISSUE: getAdjList should just be a getter. Calculating adjacencies every time getAdjList is called wastes resources.

SOLUTION: Create new function, calcCellAdjacency, to perform the calculations, change getAdjList to be a getter.