# COMP 111

# Lab 5 – Winter 2018

# Scenario

“It’s only natural to be a little nervous,” you think to yourself. After all you’re going to be giving a formal presentation to the senior staff at the Enterprise Biological Solutions. Of course, your boss will be there also. You picture the pleased looks on the bosses’ faces, perhaps a standing ovation. Maybe they’ll even hoist you on their shoulders while cheering you …

“OK, I’m getting a little carried away,” you admit with a smile.

This assignment finalizes the maze result administration system prototype. It’s an enhancement and expansion of the last assignment. You review the changes:

* The array type container will be replaced with an ArrayList so that the container of MazeResult objects can grow and shrink dynamically. You learn that this is a good example of encapsulation since the public interface for many of the methods does not need to change from previous versions of the project even though you are changing the internal structure.
* The next position instance variable will no longer be required since you are now working with an ArrayList.
* All of the existing methods will be retained and the code revised to work with the ArrayList container object. That means you’ll be using ArrayList class methods such as add, get, set and remove.
* For methods that take in an object reference, or pass back an object reference, you will make "copies" of those references prior to setting instance variable values or passing back instance variable values to avoid side effects and to achieve encapsulation/data hiding.
* deleteMazeResult methods will be added to the mix, with one version taking in an array position value as a parameter and another version of the method taking in a rat RFID.
* A new sortByTime method will sort the ArrayList of MazeResult objects by maze time. It will accept one explicit parameter that will indicate the direction of the sort. A value of “Asc” will indicate to sort in ascending order (smallest time to largest), a value of ‘Des’ will indicate to sort in descending order (largest time to smallest) Using one of the sort algorithms discussed in class. (See reading guide for chapter 14, starting at page 44, for examples of the sort algorithms.)

You have some diagrams and charts in a slide show as part of your presentation. You also will demonstrate the prototype in the IDE, projecting it on a white board from your laptop. Time to practice the presentation; this is something you definitely don’t want to wing. As your mind wanders, you once again think about that promotion, the fame, the fortune …

# Assignment

1. A starter BlueJ project will be provided to you including the Competition and CompetitionTest classes.
2. You must use the provided project file and develop the lab within the BlueJ IDE to complete this assignment.
3. Do NOT change the name, return type, or parameter order/type of any of the class methods. These same attributes are used by Web-CAT to grade your submission, and you will end up losing points if they are changed. It is recommended that you do NOT alter (modify, add, or delete) any of the test objects provided for you in the setUp() method in the CompetitionTest class.
4. Each class constructor and method should have a corresponding unit test method. Besides serving to verify that your code is accomplishing what you intended, writing unit test methods helps you understand how the objects should behave (how the class methods create, access and mutate the object). That is, writing the test methods first helps guide your class coding efforts.

Follow these steps to complete this assignment:

* 1. Find and correct any syntax errors so that the classes in the project compile cleanly.
  2. Review the first JUnit test method, completing the code if necessary. Note that proper testing includes making manual calculations to verify that expected values for a method action are equivalent to actual results.
  3. Once you feel the unit test method is a valid and robust test of the method’s expected behavior, review the corresponding class constructor or method and complete the coding if necessary.
  4. Run the JUnit test for the method. If it does not pass, review and correct the code for this method.
  5. When the test passes, continue in like manner for the next and subsequent methods, iteratively completing tests and developing your class or classes. All class methods must be tested.

1. All classes must compile cleanly.
2. Check your programming style using the “Checkstyle” tool provided with BlueJ (Tools 🡪 Checkstyle).
3. Document the overall project in the README file as directed in the Action Items for the Lab.
4. Submit your completed lab to Web-CAT using BlueJ (Tools 🡪 Submit) and review the Web-CAT results for errors. Repeat the above steps as needed to resolve any errors.
5. Note that the driver class CompetitionDriver included in the project file is simply to provide a visual confirmation of correct class behavior. It is not part of the actual solution. You can modify it if that is helpful to you, but the driver class will not be reviewed by either your instructor or Web-CAT.
6. Notes regarding use of the Rat, Time, and MazeResult classes:

When you unzip the Lab 5 starter project, you will see a folder labeled “Student Distribution Lab5”. If you look inside this folder, there are two additional folders, “+lib”, which is your BlueJ starter project you will need, and “doc”, which is a folder holding the documentation for the Rat, Time, and MazeResult classes. You need to use these classes when you code the Competition class, they are included as a library jar file in the starter project. The source code for these classes is not visible to you from the BlueJ screen, yet you can access these classes as if they were other classes in the project. The API documentation for these classes is contained in an HTML files in the “doc” folder. Open the index.html file in a browser to view the API for these classes.

1. You will notice that there is code entered in the setup() method of the CompetitionTest class. This has been entered to provide data that can be used when testing the various methods. A Competition object has been created and populated with seven MazeResult objects. Each of the MazeResult objects has been populated with a Rat object and both a start and end Time object. You will notice the test cases provided to you take advantage of that populated Competition object.