# COMP 111

# Lab 4 – Winter 2018

# Scenario

Although your presentation to the boss regarding the Enterprise Biological Solutions data administration application prototype is going to be an informal, white board, brainstorming session, you plan to be well prepared. You’ll have a proposed solution in hand that you hope will be “blessed” or at least will only require some minor tweaking.

Your last assignment defined and modeled the relationship of a MazeResult with its components, instances of a Rat and a Time class. Now you will build on this project by creating a class that has an association with MazeResult of 0 to *n* MazeResult objects. This class will manage a collection of MazeResult objects. It models a race or competition in essence, so you decide to name this collection management class Competition. For this phase of the prototype, you decide to use the simplest of collection types, an array, to hold the MazeResult objects in your Competition class.

To identify a Competition you decide to use a description attribute. In addition, you will have an array of MazeResults objects that can hold up to 100 objects and a companion variable that keeps track of the next available position in the array to add a new MazeResult object. You remind yourself that for the language you are using to code the design, the first element of an array has index (subscript) of 0, and the maximum element has index of (length – 1).

After speaking with the EBF analyst, you come up with some operations that will provide the client with the ability to manage test data and extract information. These capabilities will be expanded in your final prototype.

Your ideas/design for the Competition class are as follows:

#### Attributes

* description – text description of the competition
* results – array of MazeResult objects
* nextPos – whole number indicating the next available element position (index) in the array to add an object, also serves as the populated array count

Operations (methods):

* A constructor that takes in a description
* A get and set description method
* A get method for the next position attribute (array populated element count)
* Method to add a MazeResult object to the array
* Methods to get a MazeResult object; one that takes in an array position index and one that takes in a rat ID string.
* A method with conditional logic to give a count of all rats for a certain category (*male, female, vaccinated, not vaccinated, all*) triggered by a flag passed in as a whole number (1, 2, 3, 4, 5, respectively, implemented as public constants). A similar method provides the average maze result (actual time) for each potential category.
* A toString method that gives the test description and a count of maze results in the collection as well as looping through the array and calling the toString method for each MazeResult object.

You scramble to get your notes in order before your meeting. You’ll document your plan to unit test all methods and will bring that up early on in the presentation, hoping to beat your boss to the punch this time on the importance of the Test-Driven Development methodology.

# Assignment

1. A starter BlueJ project will be provided to you. You are to complete the coding of 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.
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 domain method’s expected behavior, review the corresponding domain 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 Lab4 starter project, note that a library jar file named MazeResult class is in the +libs folder. This allows the project access to the Rat, Time, and MazeResult classes. Although you will not be able to see the source code for the classes, you will be able to use the documentation contained in the doc folder underneath +libs to review the support classes public interfaces. Open the index.html file in a browser to view the API for the classes.

1. You will notice there is code in the setup() method of the CompetitionTest class. This is data that can and should be used as appropriate when testing a number of the 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. Note some of the test cases coded for you take advantage of these already created objects.