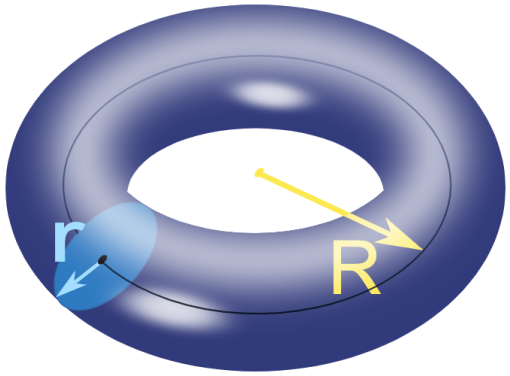
Deliverables

Your project files should be submitted to the grading system by the due date and time specified. Note that there is also an optional Skeleton Code assignment (ungraded) which will ensure that you have classes and methods named correctly and that you have the correct return types and parameter types. This ungraded assignment will also indicate level of coverage your tests have achieved. The files you submit to skeleton code assignment may be incomplete in the sense that method bodies have at least a return statement if applicable or they may be essentially completed files. To avoid a late penalty for the project, you must submit your files to the Completed Code assignment no later than 11:59 PM on the due date. Your grade will be determined, in part, by the tests that you pass or fail in your test file and by the level of coverage attained in your source file, as well as our usual correctness tests.

Files to submit to the grading system:

- RingTorus.java, RingTorusTest.java
- RingTorusList.java, RingTorusListTest.java

A **ring torus** is made by revolving a small circle with radius ***r*** along a line made by a larger circle with radius ***R*** as shown below.



Variables in formulas:

***R**: radius of large circle (or large radius)*
***r**: radius of small circle (or small radius)*
***A**: surface area*
***V**: volume*
***D**: ring diameter*

$D = 2\,(R + r)$

$A = (2\pi R)\,(2\pi r)$

$V = (2\pi R)\,(\pi r^2)$

Specifications – Use arrays in this project; ArrayLists are not allowed!

Overview: This project consists of four classes: (1) RingTorus is a class representing a RingTorus object; (2) RingTorusTest class is a JUnit test class which contains one or more test methods for each method in the RingTorus class; (3) RingTorusList is a class representing a RingTorus list object; and (4) RingTorusListTest class is a JUnit test class which contains one or more test methods for each method in the RingTorusList class. *Note that there is no requirement for a class with a main method in this project.*

You should create a new folder to hold the files for this project and add your files from Part 2 (RingTorus.java file and RingTorusTest.java). You should create a new jGRASP project for Part 3 and add RingTorus.java file and RingTorusTest.java to the project; you should see the two files in their respective categories – Source Files and Test Files. If RingTorusTest.java appears in source File category, you should right-click on the file and select "Mark As Test" from the right-click menu.

You will then be able to run the test file by clicking the JUnit run button on the Open Projects toolbar. After RingTorusList.java and RingTorusListTest.java are created as specified below, these should be added to your jGRASP project for Part 3 as well.

If you have successfully completed RingTorus.java and RingTorusTest.java in Part 2, you should go directly to RingTorusList.java on page 6.

- **RingTorus.java** (*new items for this class in Part 2 are underlined*)

  Requirements: Create a RingTorus class that stores the label, small radius, and large radius where the small radius is less than the large radius and both are positive. The RingTorus class also includes methods to set and get each of these fields, as well as methods to calculate the outside diameter, surface area, and volume of a RingTorus object, and a method to provide a String value that describes a RingTorus object. The RingTorus class includes a one static field (or class variable) to track the number of RingTorus objects that have been created, as well appropriate static methods to access and reset this field. And finally, this class provides a method that JUnit will use to test RingTorus objects for equality as well as a method required by Checkstyle. In addition, RingTorus must implement the Comparable interface for objects of type RingTorus.

  Design: The RingTorus class implements the Comparable interface for objects of type RingTorus and has fields, a constructor, and methods as outlined below (last method is new).

  (1) **Fields:** *Instance Variables* - label of type `String`, large radius of type `double`, and small radius of type `double`. Initialize the `String` to `""` and the `double` variables to 0 in their respective declarations. These instance variables should be private so that they are not directly accessible from outside of the RingTorus class, and these should be the only instance variables (fields) in the class.
  Class Variable - count of type int should be private and static, and it should be initialized to zero in the declaration.

  (2) **Constructor**: Your RingTorus class must contain a public constructor that accepts three parameters (see types of above) representing the label, large radius, and small radius. Instead of assigning the parameters directly to the fields, the respective set method for each field (described below) should be called since they are checking the validity of the parameter. For example, instead of using the statement `label = labelIn;` use the statement `setLabel(labelIn);`
  After setLabel is called be sure to call setLargeRadius before calling setSmallRadius. The constructor should increment the class variable count each time a RingTorus is constructed.

  Below are examples of how the constructor could be used to create RingTorus objects. Note that although String and numeric literals are used for the actual parameters (or arguments) in these examples, variables of the required type could have been used instead of the literals.

```
RingTorus ex1 = new RingTorus ("Small Example", 9.5, 1.25);

RingTorus ex2 = new RingTorus (" Medium Example ", 35.1, 10.4);

RingTorus ex3 = new RingTorus ("Large Example", 134.28, 32.46);
```

(3) **Methods**: Usually a class provides methods to access and modify each of its instance variables (known as get and set methods) along with any other required methods. The methods for RingTorus, which should each be public, are described below. See the formulas in the figure above and the Code and Test section below for information on constructing these methods.

- `getLabel`: Accepts no parameters and returns a `String` representing the label field.
- `setLabel`: Takes a `String` parameter and returns a `boolean`. If the `String` parameter is not `null`, then the "trimmed" `String` is set to the label field and the method returns `true`. Otherwise, the method returns `false` and the label is not set.
- `getLargeRadius`: Accepts no parameters and returns a `double` representing the large radius field.
- `setLargeRadius`: Takes a `double` parameter and returns a `boolean`. If the `double` parameter is positive and is greater than the current small radius, then the parameter is assigned to the large radius field and the method returns `true`. Otherwise, the method returns `false` and the large radius field is not set.
- `getSmallRadius`: Accepts no parameters and returns a `double` representing the small radius field.
- `setSmallRadius`: Takes a `double` parameter and returns a `boolean`. If the `double` parameter is positive and is less than the current large radius, then the parameter is assigned to the small radius field and the method returns `true`. Otherwise, the method returns `false` and the small radius field is not set.
- `diameter`: Accepts no parameters and returns a `double` representing the diameter of the RingTorus. See formula in figure on page 1.
- `surfaceArea`: Accepts no parameters and returns the `double` value for the surface area of the RingTorus. See formula in figure on page 1.
- `volume`: Accepts no parameters and returns the double value for the volume of the RingTorus. See formula in figure on page 1.
- `toString`: Returns a `String` containing the information about the RingTorus object formatted as shown below, including decimal formatting ("#,##0.0##") for the `double` values. Newline and tab escape sequences should be used to achieve the proper layout within the String but it should not begin or end with a newline. In addition to the field values (or corresponding "get" methods), the following methods should be used to compute appropriate values in the `toString` method: `diameter()`, `surfaceArea()`, and `volume()`. Each line should have no trailing spaces (e.g., there should be no spaces before a newline (\n) character). The `toString` value for `ex1`, `ex2`, and `ex3` respectively are shown below (the blank lines are not part of the `toString` values).

```
RingTorus "Small Example"
   large radius = 9.5 units
   small radius = 1.25 units
   diameter = 21.5 units
   surface area = 468.806 square units
   volume = 293.004 cubic units

RingTorus "Medium Example"
   large radius = 35.1 units
   small radius = 10.4 units
   diameter = 91.0 units
   surface area = 14,411.202 square units
   volume = 74,938.248 cubic units

RingTorus "Large Example"
   large radius = 134.28 units
   small radius = 32.46 units
   diameter = 333.48 units
   surface area = 172,075.716 square units
   volume = 2,792,788.867 cubic units
```

- `getCount`: A static method that accepts no parameters and returns an int representing the static count field.
- `resetCount`: A static method that returns nothing, accepts no parameters, and sets the static count field to zero.
- `equals`: An instance method that accepts a parameter of type Object and returns false if the Object is a not a RingTorus; otherwise, when cast to a RingTorus, if it has the same field values (ignoring case in the label field) as the RingTorus upon which the method was called, it returns true. Otherwise, it returns false. Note that this equals method with parameter type Object will be called by the JUnit Assert.assertEquals method when two RingTorus objects are checked for equality.

  Below is a version you are free to use.

```
public boolean equals(Object obj) {

   if (!(obj instanceof RingTorus )) {
      return false;
   }
   else {
      RingTorus rt = (RingTorus ) obj;
      return (label.equalsIgnoreCase(rt.getLabel())
            && (Math.abs(largeRadius - rt.getLargeRadius()) < .000001)
            && (Math.abs(smallRadius - rt.getSmallRadius()) < .000001));
   }
}
```

- `hashCode()`: Accepts no parameters and returns zero of type int. This method is required by Checkstyle if the `equals` method above is implemented.
- `compareTo`: Accepts a parameter of type RingTorus and returns an int as follows: a negative value if `this.volume()` is less than the parameter's volume; a positive value

if `this.volume()` is greater than the parameter's volume; zero if the two volumes are essentially equal. *For a hint, see the activity for this module*.

Code and Test: As you implement the methods in your RingTorus class, you should compile it and then create test methods as described below for the RingTorusTest class.

When using the **setSmallRadius** and **setLargeRadius**, the values of the current smallRadius and largeRadius must be considered to decide the order in which you invoke the associated set methods. For setSmallRadius, the "new" small radius must be positive and smaller the current large radius, and for setLargeRadius, the "new" large radius must be positive and larger the current small radius. For example, a new RingTorus object has its smallRadius and largeRadius fields initialized to zero in the field declaration, which is default initial value for doubles. However, this means in the constructor, you must invoke the setLargeRadius before the setSmallRadius. If you wanted a small radius of 2.5 and attempt to set it first, it would not be set since it will not be less than largeRadius, which is initially zero. Thus, you invoke setLargeRadius first since any positive value will be greater than the smallRadius which is initially zero. Therefore, in general you need to consider the current value the smallRadius and largeRadius to decide the order you invoke the associated set methods.

- **RingTorusTest.java**

Requirements: Create a RingTorusTest class that contains a set of *test* methods to test each of the methods in RingTorus. The goal for Part 2 is method, statement, and condition coverage.

Design: Typically, in each test method, you will need to create an instance of RingTorus, call the method you are testing, and then make an assertion about the expected result and the actual result (note that the actual result is commonly the result of invoking the method unless it has a void return type). You can think of a test method as simply formalizing or codifying what you could be doing in jGRASP interactions to make sure a method is working correctly. That is, the sequence of statements that you would enter in interactions to test a method should be entered into a single test method. You should have sufficient test methods so that each method, statement, and condition in RingTorus are covered. Collectively, these test methods are a set of test cases that can be invoked with a single click to test all the methods in your RingTorus class.

Code and Test: A good strategy would be to begin by writing test methods for those methods in RingTorus that you "know" are correct. By doing this, you will be able to concentrate on the getting the test methods correct. That is, if the test method *fails*, it is most likely due to a defect in the test method itself rather the RingTorus method being testing. As you become more familiar with the process of writing test methods, you will be better prepared to write the test methods as new methods are developed. Be sure to call the RingTorus `toString` method in one of your test methods and assert something about the return value. If you do not want to use assertEquals, which would require the return value match the expected value exactly, you could use assertTrue and check that the return value contains the expected value. For example, for RingTorus example3:

```
Assert.assertTrue(example3.toString().contains("\"Large  Example\""));
```

Also, remember that you can set a breakpoint in a JUnit test method and run the test file in Debug mode. Then, when you have an instance in the Debug tab, you can unfold it to see its values or you can open a canvas window and drag items from the Debug tab onto the canvas. You can also step-in to the method being called by the test method and then single-step through it, looking for the error.

- **RingTorusList.java** (new for Part 3) – Consider implementing this file in parallel with its test file, RingTorusListTest.java, which is described after this class.

  Requirements: Create a RingTorusList class that stores the name of the list and an array of RingTorus objects. It also includes methods that return the name of the list, number of RingTorus objects in the RingTorusList, total surface area, total volume, average surface area, and average volume for all RingTorus objects in the RingTorusList. The toString method returns summary information about the list (see below).

  Design: The RingTorusList class has three fields, a constructor, and methods as outlined below.

  (1) Fields (or instance variables): (1) a String representing the name of the list, (2) an array of RingTorus objects, and (3) an `int` representing the number of RingTorus objects in the array, which may be less than the length of the array of RingTorus objects. These instance variables should be private so that they are not directly accessible from outside of the RingTorusList class. These should be the only fields (or instance variables) in this class, and they should be initialized in the constructor described below.

  (2) Constructor: Your RingTorusList class must contain a constructor that accepts three parameters: (1) a parameter of type String representing the name of the list, (2) a parameter of type `RingTorus[]`, representing the list of RingTorus objects, and (3) a parameter of type `int` representing the number of RingTorus objects in the array. These parameters should be used to assign the fields described above (i.e., the instance variables).

  (3) Methods: The methods for RingTorusList are described below.
  - `getName`: Returns a String representing the name of the list.
  - `numberOfRingToruses`: Returns an int (the value of the third field in the RingTorusList object) representing the number of RingTorus objects in the RingTorusList.
  - `totalDiameter`: Returns a double representing the total diameters for all RingTorus objects in the list. If there are zero RingTorus objects in the list, zero should be returned.
  - `totalSurfaceArea`: Returns a double representing the total surface areas for all RingTorus objects in the list. If there are zero RingTorus objects in the list, zero should be returned.
  - `totalVolume`: Returns a double representing the total volumes for all RingTorus objects in the list. If there are zero RingTorus objects in the list, zero should be returned.

- `averageDiameter`: Returns a double representing the average diameter for all RingTorus objects in the list. If there are zero RingTorus objects in the list, zero should be returned.
- `averageSurfaceArea`: Returns a double representing the average surface area for all RingTorus objects in the list. If there are zero RingTorus objects in the list, zero should be returned.
- `averageVolume`: Returns a double representing the average volume for all RingTorus objects in the list. If there are zero RingTorus objects in the list, zero should be returned.
- `toString`: Returns a String (does not begin with \n) containing the name of the list (which can change depending on the name of the list passed as a parameter to the constructor) followed by various summary items: number of RingToruses, total surface area, total volume, average surface area, and average volume. Use "#,##0.0##" as the pattern to format the double values. Below is an example of the formatted String returned by the `toString` method, where the name of the list (name field) is `RingTorus Test List` and the array of RingTorus objects contains the three examples described above (top of page 3).

```
----- Summary for RingTorus Test List -----
Number of RingToruses: 3
Total Diameter: 445.98 units
Total Surface Area: 186,955.724 square units
Total Volume: 2,868,020.119 cubic units
Average Diameter: 148.66 units
Average Surface Area: 62,318.575 square units
Average Volume: 956,006.706 cubic units
```

- `getList`: Returns the array of RingTorus objects (the second field above).
- `addRingTorus`: Returns nothing but takes three parameters (label, large radius, and small radius), creates a new RingTorus object, and adds it to the RingTorusList object in the next available location in the RingTorus array. Be sure to increment the int field containing the number of RingTorus objects in the RingTorusList object.
- `findRingTorus`: Takes a label of a RingTorus as the String parameter and returns the corresponding RingTorus object if found in the RingTorusList object; otherwise returns null. Case should be ignored when attempting to match the label.
- `deleteRingTorus`: Takes a String as a parameter that represents the label of the RingTorus and returns the RingTorus if it is found in the RingTorusList object and deleted; otherwise returns null. Case should be ignored when attempting to match the label. When an element is deleted from an array, elements to the right of the deleted element must be shifted to the left. After shifting the items to the left, the last RingTorus element in the array should be set to null. Finally, the number of elements field must be decremented.
- `editRingTorus`: Takes three parameters (label, large radius, and small radius), uses the label to find the corresponding the RingTorus object in the list. If found, sets the large radius and small radius fields to the values passed in as parameters, and returns true. If not found, returns false.
  (*Note that the label should not be changed by this method.*)
- `findRingTorusWithLargestVolume`: Returns the RingTorus with the largest volume; if the list contains no RingTorus objects, returns null.

Code and Test: Some of the methods above require that you use a loop to go through the objects in the array. You should implement the class below in parallel with this one to facilitate testing. That is, after implementing one to the methods above, you can implement the corresponding test method in the test file described below.

- **RingTorusListTest.java** (new for Part 3) – Consider implementing this file in parallel with its source file, RingTorusList.java, which is described above this class.

  Requirements: Create a RingTorusListTest class that contains a set of *test* methods to test each of the methods in RingTorusList.

  Design: Typically, in each test method, you will need to create an instance of RingTorusList, call the method you are testing, and then make an assertion about the expected result and the actual result (note that the actual result is usually the result of invoking the method unless it has a void return type). You can think of a test method as simply formalizing or codifying what you have been doing in interactions to make sure a method is working correctly. That is, the sequence of statements that you would enter in interactions to test a method should be entered into a single test method. You should have at least one test method for each method in RingTorusList. However, if a method contains conditional statements (e.g., an *if* statement) that results in more than one distinct outcome, you need a test method for each outcome. For example, if the method returns boolean, you should have one test method where the expected return value is false and another test method that expects the return value to be true. Also, each condition in boolean expression must be exercised true and false. Collectively, these test methods are a set of test cases that can be invoked with a single click to test all the methods in your RingTorusList class.

  Code and Test: A good strategy would be to begin by writing test methods for those methods in RingTorusList that you "know" are correct. By doing this, you will be able to concentrate on the getting the test methods correct. That is, if the test method *fails*, it is most likely due to a defect in the test method itself rather the RingTorusList method being testing. As you become more familiar with the process of writing test methods, you will be better prepared to write the test methods for the new methods in RingTorusList. Be sure to call the RingTorusList toString method in one of your test cases so that the grading system will consider the toString method to be "covered" in its coverage analysis. Remember that when a test method fails, you can set a breakpoint in a JUnit test method and run the test file in Debug mode. Then, when you have an instance in the Debug tab, you can unfold it to see its values or you can open a canvas window and drag items from the Debug tab onto the canvas. You can also step-in to the method being called by the test method and then single-step through it, looking for the error.

  Finally, when comparing two arrays for equality in JUnit, be sure to use Assert.assertArrayEquals rather than Assert.assertEquals. Assert.assertArrayEquals will return true only if the two arrays are the same length and the elements are equal based on an element-by-element comparison using the equals method.

The Grading System

When you submit your files (RingTorus.java, RingTorusTest.java, RingTorusList.java, and RingTorusListTest.java), the grading system will use the results of your test methods and their level of coverage of your source files as well as the results of our reference correctness tests to determine your grade. In this project, your test files should provide method, statement, and condition coverage. Each condition in your source file must be exercised both true and false. See below for a description of how to test a boolean expression with multiple conditions. To see your code coverage in Web-CAT, on "Your Assignment Submission Results" page, in "File Details", click on your source file and look for any highlighted lines of code. Hover your mouse on the highlighted source code, to see the hint about what was not covered. Example of hints you might see include: (1) This line was never executed by your tests. (2) Not all possibilities of this decision were tested. Remember that when you have N simple conditions combined, you must test all N+1 possibilities.

Note For Testing the `equals` Method

Perhaps the most complicated method to test is the `equals` method in RingTorus. This method has three conditions in the boolean expression that are &&'d. Since Java (and most other languages) uses short-cut logic, if the first condition in an && is false, the &&'d expression is false. This means that to test the second condition, the first conditions must be true. Furthermore, to test the third conditions both the first and second conditions must be true. To have condition coverage for the equals method, you need the four test cases where the three conditions evaluate to the following, where T is true, F is false, and X is don't care (could be true or false):

FXX - returns false
TFX - returns false
TTF - returns false
TTT - returns true

Note For Testing the Methods with Two &&'d Conditions

Your setSmallRadius and setLargeRadius methods likely each have two conditions in the boolean expression that are &&'d. Since Java (and most other languages) uses short-cut logic, if the first condition in an && is false, the &&'d expression itself is false (without considering the second condition). This means that to test the second condition, the first condition must be true. To have condition coverage for the equals method, you need the three test cases where the two conditions evaluate to the following, where T is true, F is false, and X is don't care (could be true or false):

FX - returns false
TF - returns false
TT - returns false