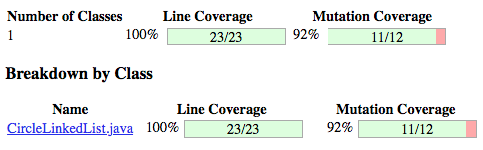
## CircularlyLinkedList:

### Running the Tests:

These tests are run in a generic JUnit test environment as well the PiTest mutation software. A viable way of running these tests would be to open the project in IntelliJ. If you do not have eclipse, please acquire it at <https://www.jetbrains.com/idea/download/> and download the community version. From this, you may open or import the project that is included with the source code. To run the tests, simply click on the testing file known as “CircleLinkedListTest.java” and open the file (within IntelliJ). You then should run the tests by either clicking the green arrow (the run button) or you may run this with code coverage (green arrow with a red and green bar). The tests should then run and the results should be shown on the left hand side of your screen. Install the PiTest plugin by clicking preferences in the File dropdown menu. Click the “Plugins” menu on the left side and then click “Browse Repositories”. Search for “Pitest” , download the plugin and restart IntelliJ. After IntelliJ has restarted, click the “run” dropdown menu at the top of the screen, and then click “edit configurations”. In the edit configurations popup menu, click the “+” to add a new configuration, and from that menu, select PIT Runner. In the new PIT Runner configuration, set the Target Class to “DataStructures.\*” and the source directory, to wherever your source files are located. Run this new configuration and a report will be generated in a corresponding reports folder in the project directory.

### **Results:**



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Method Being Tested | Criteria (Mutation Coverage) | Test Input | Narrative | Author | Result |
| getSize() | **Original:** return size;  **Mutation:** return (size == 0 ? 1: 0)  Kill Mutation | Replace return integer with boolean integer | This mutation outputs the incorrect return value of the size of the list, which is important because the size of the list is necessary for the proper functionality of the remove method. | Alex Oladele | Pass |
| append() | **Original:** if (value == null){  **Mutation:** if (value != null){  Kill Mutation | Negated conditional | This mutation is a good change, because it heavily influences how the rest of the method works. If the value != null then it’ll always throw an exception | Alex Oladele | Pass |
| append() | **Original:** size++;  **Mutation:** size--;  Kill Mutation | Replaced integer addition w/ subtraction | This mutation will adjust the size incorrectly, which can negatively impact the rest of the class so it is import to test. | Alex Oladele | Pass |
| remove() | **Original:** if (pos > size || pos < 0) {  **Mutation:** if (pos >= size || pos < 0) {  Kill Mutation | Changed conditional boundary | This mutation increases the area of input that will throw an exception, which means that a value will not be removed. This can cause incorrect states throughout the rest of the program. | Alex Oladele | Fail |
| remove() | **Original:** if (pos > size || pos < 0) {  **Mutation:** if (pos > size || pos <= 0) {  Kill Mutation | Changed conditional boundary | This mutation increases the area of input that will throw an exception, which means that a value will not be removed. This can cause incorrect states throughout the rest of the program. | Alex Oladele | Pass |
| remove() | **Original:** if (pos > size || pos < 0) {  **Mutation:** if (pos != size || pos < 0) {  Kill Mutation | Negated conditional | This mutation increases the area of input that will throw an exception, which means that a value will not be removed. This can cause incorrect states throughout the rest of the program. | Alex Oladele | Pass |
| remove() | **Original:** if (pos > size || pos < 0) {  **Mutation:** if (pos > size || pos != 0) {  Kill Mutation | Negated conditional | This mutation increases the area of input that will throw an exception, which means that a value will not be removed. This can cause incorrect states throughout the rest of the program. | Alex Oladele | Pass |
| remove() | **Original:** for (int i = 1; i <= pos; i++) {  **Mutation:** for (int i = 1; i < pos; i++) {  Kill Mutation | Changed conditional boundary | This mutation influences the value of the iterator, which is important in trying to get to the correct position to remove. This mutation can cause incorrect states throughout the program. | Alex Oladele | Pass |
| remove() | **Original:** for (int i = 1; i <= pos; i++) {  **Mutation:** for (int i = 1; i <= pos; i--) {  Kill Mutation | Changed increment from 1 to -1 | This mutation causes the iterator to infinitely be assigned an incorrect value, which means that nodes will never be deleted. This causes incorrect states in a program | Alex Oladele | Timed Out |
| remove() | **Original:** for (int i = 1; i <= pos; i++) {  **Mutation:** for (int i = 1; i != pos; i++) {  Kill Mutation | Negated conditional | This mutation changes the iterator to be infinitely assigned to an incorrect value. This is important in trying to get to the correct position to remove a node, because it will never. This mutation can cause incorrect states throughout the program. | Alex Oladele | Timed Out |
| remove() | **Original:** size--;  **Mutation:** size++;  Kill Mutation | Replaced integer subtraction with addition | This mutation will adjust the size incorrectly, which can negatively impact the rest of the class so it is import to test. | Alex Oladele | Pass |
| remove() | **Original:** return saved  **Mutation:**  new Object();  return null;  Kill Mutation | Change return of object value | This mutation will return the incorrect value of the node removed, which is the core of the remove method (popping the node from the list and returning its value) | Alex Oladele | Pass |