

<b>ArrayIntList</b>	<b>3</b>
Simple	3
First prompt 4o	3
Second prompt 4o	3
Third prompt 4o	4
First prompt 3.5	4
Second prompt 3.5	5
Third prompt 3.5	5
With context (removeAtElement())	6
First prompt 4o (no difference)	6
Second prompt 4o (no difference)	6
Third prompt 4o (no difference)	7
First prompt 3.5 (no difference)	7
Second prompt 3.5 (Explaining)	7
Third prompt 3.5 (no difference)	8
Prompt engineering	9
First prompt 4o	9
Second prompt 4o	10
Third prompt 4o	10
First prompt 3.5	10
Second prompt 3.5	11
Third prompt 3.5	11
Few shot	12
First prompt 4o [got limit]	14
Second prompt 4o [got limit]	14
Third prompt 4o [got limit]	14
First prompt 3.5	14
Second prompt 3.5	15
Third prompt 3.5	15
<b>Rational</b>	<b>16</b>
Simple	16
First prompt 4o	16
Second prompt 4o	16
Third prompt 4o	17
First prompt 3.5	17
Second prompt 3.5	17
Third prompt 3.5	18
With context (abs())	19
First prompt (understands the method) 4o	19
Second prompt (no difference) 4o	19
Third prompt (understands the method) 4o	20
First prompt (understands the method) 3.5	20
Second prompt (understands the method) 3.5	20

Third prompt (understands the method) 3.5	21
Prompt engineering	21
First prompt 4o	21
Second prompt 4o	22
Third prompt 4o	22
First prompt 3.5	23
Second prompt 3.5	23
Third prompt 3.5	23
Few shot	1
First prompt 4o	1
Second prompt 4o	1
Third prompt 4o	1
First prompt 3.5	1
Second prompt 3.5	1
Third prompt 3.5	1
<b>Keycloak</b>	<b>1</b>
Few shot	1
First prompt 4o	1
Second prompt 4o	1
Third prompt 4o	1
First prompt 3.5	1
Second prompt 3.5	1
Third prompt 3.5	1

# ArrayIntList

## Simple

Please generate a summary of the unit test given between the [CODE] and [/CODE] tags. Please put your answer between [SUMMARY] and [/SUMMARY] tags.

[CODE]

```
1.
2. @Test
3. public void test10() throws Throwable {
4.     ArrayIntList arrayIntList0 = new ArrayIntList();
5.
6.     try {
7.         arrayIntList0.add(0, 0);
8.         arrayIntList0.add(0, 1);
9.         arrayIntList0.add(0, 2);
10.        assertEquals(3, arrayIntList0.size());
11.        arrayIntList0.removeElementAt((1));
12.        fail("Expecting exception: IndexOutOfBoundsException");
13.    } catch (IndexOutOfBoundsException e) {
14.        //
15.        // Should be at least 0 and less than 0, found -1
16.        //
17.    }
18. }
```

[/CODE]

## First prompt 4o

[SUMMARY]

The unit test test10 creates an instance of ArrayIntList and adds three elements at index 0. It then checks if the size of the list is 3. After that, it attempts to remove an element at index 1 and expects an IndexOutOfBoundsException to be thrown. The test will fail if the exception is not thrown, but if it is thrown, the test will pass and the exception message will be verified to indicate the error.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Last part isn't needed*

*Naturalness: 1. Last part isn't really nice*

## Second prompt 4o

[SUMMARY]

The unit test test10 verifies the behavior of the ArrayIntList class. It performs the following actions:

1. Creates an instance of ArrayIntList.
2. Adds three elements (0, 1, and 2) at the 0th index.
3. Asserts that the size of the list is 3.

4. Attempts to remove an element at index 1, expecting an `IndexOutOfBoundsException` to be thrown.
5. Catches the expected `IndexOutOfBoundsException`, confirming that the exception message indicates the invalid index.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Third prompt 4o

[SUMMARY]

The unit test `test10` checks the functionality of the `ArrayIntList` class. It creates an instance of `ArrayIntList` and adds three elements at index 0, incrementing the list size to 3. The test then attempts to remove an element at index 1, which is expected to throw an `IndexOutOfBoundsException`. If the exception is not thrown, the test will fail with a specific message. If the exception is caught, it verifies that the error message indicates an invalid index.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## First prompt 3.5

[SUMMARY]

The unit test `test10` is testing the behavior of the `ArrayIntList` class. It first creates an instance of `ArrayIntList` and adds three elements at index 0. Then it verifies that the size of the list is indeed 3. After that, it attempts to remove an element at index 1, expecting an `IndexOutOfBoundsException` to be thrown. The catch block verifies that the correct exception is thrown with the expected error message indicating the index that caused the exception.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Second prompt 3.5

[SUMMARY]

The provided unit test is for a method called test10 in a class, where an ArrayList object is created and manipulated. Initially, three elements are added at index 0, and then an attempt is made to remove an element at index 1. The test expects an IndexOutOfBoundsException to be thrown during this removal. The test passes if the exception is caught with the correct error message indicating the out-of-bounds index.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Third prompt 3.5

[SUMMARY]

The unit test checks the functionality of adding elements to an ArrayList at specific indices and removing an element at a specified index. It verifies that an IndexOutOfBoundsException is thrown when attempting to remove an element at an invalid index. The test case expects the exception to be thrown when attempting to remove an element at index -1.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## With context (removeAtElement())

Please generate a summary of the unit test given between the [CODE] and [/CODE] tags. The method under test is between the [METHOD\_TESTED] AND [/METHOD\_TESTED] tags. Please put your answer between [SUMMARY] and [/SUMMARY] tags.

[CODE]

```
1.
2. @Test
3. public void test10() throws Throwable {
4.     ArrayIntList arrayIntList0 = new ArrayIntList();
5.
6.     try {
7.         arrayIntList0.add(0, 0);
8.         arrayIntList0.add(0, 1);
9.         arrayIntList0.add(0, 2);
10.        assertEquals(3, arrayIntList0.size());
11.        arrayIntList0.removeElementAt((1));
12.        fail("Expecting exception: IndexOutOfBoundsException");
13.    } catch (IndexOutOfBoundsException e) {
14.        //
15.        // Should be at least 0 and less than 0, found -1
16.        //
17.    }
18. }
```

[/CODE]

[METHOD\_TESTED]

```
public int removeElementAt(int index) {
    checkRange(index);
    incrModCount();
    int oldval = _data[index];
    int numtomove = _size - index - 1;
    if(numtomove > 0) {
        System.arraycopy(_data,index+1,_data,index,numtomove);
    }
    _size--;
    return oldval;
}
```

[/METHOD\_TESTED]

## First prompt 4o (no difference)

[SUMMARY]

The unit test test10 verifies the behavior of the removeElementAt method in the ArrayIntList class. The test initially adds three elements to the list and checks if the size of the list is correct. It then attempts to remove the element at index 1. The test expects an IndexOutOfBoundsException to be thrown when trying to remove an element, which suggests that the method should handle invalid index removal properly. The catch block confirms that the exception is thrown, indicating that the method's range-checking mechanism is functioning as expected.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't need the extra "which suggests that the method should handle invalid index removal properly" 2. Doesn't need catch block explanation*

*Naturalness: 1. Nothing special*

## Second prompt 4o (no difference)

[SUMMARY]

The unit test named test10 tests the removeElementAt method of the ArrayIntList class. In the test, an ArrayIntList object is created, and elements are added at index 0 three times, resulting in a list with three elements. The test then asserts that the list's size is 3. It attempts

to remove an element at index 1, which should be valid, but it incorrectly expects an `IndexOutOfBoundsException` to be thrown. The exception is caught, and a comment notes that the expected behavior is not correctly captured, as the operation should not throw an exception in this scenario.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Last part isn't needed*

*Naturalness: 1. Nothing special*

### Third prompt 4o (no difference)

[SUMMARY]

The unit test `test10` validates the behavior of the `removeElementAt` method from the `ArrayIntList` class. The test first initializes an `ArrayIntList` instance and adds three elements to it at index 0. It then asserts that the list size is 3. The test attempts to remove an element at index 1, but it expects an `IndexOutOfBoundsException` to be thrown. If the exception is not thrown, the test fails with a message indicating this expectation. The exception handling confirms that the correct exception with an appropriate message is indeed thrown.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Last part isn't needed*

*Naturalness: 1. Nothing special*

### First prompt 3.5 (no difference)

[SUMMARY]

The unit test provided checks the functionality of the `removeElementAt` method of the `ArrayIntList` class. It adds three elements to the list and then attempts to remove an element at index 1. The test expects an `IndexOutOfBoundsException` to be thrown since the index is out of bounds after removing the element. The test passes if the exception is caught, ensuring that the method handles out-of-bounds index properly.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

### Second prompt 3.5 (Explaining)

[SUMMARY]

The unit test provided checks the behavior of the `removeElementAt` method in the `ArrayIntList` class. It attempts to add elements to the list and then removes an element at a specified index. The test expects an `IndexOutOfBoundsException` to be thrown when

attempting to remove an element at an index that is out of bounds. The test verifies this behavior by catching the exception and checking its message. **The removeElementAt method removes an element at the specified index from the list, shifting subsequent elements to the left. It then decrements the size of the list and returns the removed element.**

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing 5*

*Conciseness: 1. The line "The test [...] its message" isn't needed 2. Explaining method not needed*

*Naturalness: 1. Nothing special*

### Third prompt 3.5 (no difference)

[SUMMARY]

The unit test is testing the removeElementAt method of a class, presumably ArrayList. In this particular test case (test10), the method is being called with an index that should result in an IndexOutOfBoundsException. The test checks if the exception is thrown correctly when attempting to remove an element at an invalid index. The test adds three elements to the list and then attempts to remove an element at index 1, which is expected to be out of bounds after the removal operation. The test passes if the exception is thrown as expected.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Last part isn't needed*

*Naturalness: 1. Nothing special*



# Prompt engineering

```
[INST] As a detail-oriented developer focused on enhancing the clarity of a test suite, your task is to analyze the provided Java code and generate a descriptive and concise summary of the unit test. Follow these steps:  
1. Carefully read the Java code between the [CODE] tags.  
2. Identify the primary functionality or purpose of the test.  
3. Analyze actions performed in the test.  
4. Formulate a test method summary that succinctly and concisely captures what the test case does in a bullet point format for easy, quick reading  
5. Place your complete suggested test summary between the [TESTSUMMARY] and [/TESTSUMMARY] tags, ensuring it is clear and precise without unnecessary descriptions or information.  
Remember, your focus is on clarity and precision. Use your expertise to provide a meaningful and appropriate summary.[/INST]  
  
[CODE]  
1.  
2. @Test  
3. public void test10() throws Throwable {  
4.     ArrayIntList arrayIntList0 = new ArrayIntList();  
5.  
6.     try {  
7.         arrayIntList0.add(0, 0);  
8.         arrayIntList0.add(0, 1);  
9.         arrayIntList0.add(0, 2);  
10.        assertEquals(3, arrayIntList0.size());  
11.        arrayIntList0.removeElementAt(1);  
12.        fail("Expecting exception: IndexOutOfBoundsException");  
13.    } catch (IndexOutOfBoundsException e) {  
14.        //  
15.        // Should be at least 0 and less than 0, found -1  
16.        //  
17.    }  
18. }
```

## First prompt 4o

[TESTSUMMARY]

- Initializes an empty ArrayIntList.
- Adds three elements (0, 1, 2) at index 0.
- Asserts that the size of the list is 3.
- Attempts to remove an element at index 1.
- Expects an IndexOutOfBoundsException to be thrown during removal.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Second prompt 4o

[TESTSUMMARY]

- Initializes an ArrayList object.
- Adds three elements (0, 1, 2) at index 0.
- Asserts that the list size is 3.
- Attempts to remove an element at index 1.
- Expects an IndexOutOfBoundsException due to invalid index removal.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Third prompt 4o

[TESTSUMMARY]

- Setup: Initializes an empty ArrayList.
- Actions:
  - Adds three elements (0, 1, 2) at index 0, shifting existing elements to the right.
  - Asserts that the list size is 3 after additions.
  - Attempts to remove an element at index 1.
- Expected Outcome: Catches IndexOutOfBoundsException when attempting to remove an element at index 1, validating that the exception is thrown correctly for an invalid index.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## First prompt 3.5

[TESTSUMMARY]

- Test method: test10
- Purpose: Verify the behavior of ArrayList when adding elements at specific indices and removing an element.
- Actions:
  - Adds three elements to ArrayList at index 0.

- Checks that the size of the list is 3.
- Removes an element at index 1.
- Expects an `IndexOutOfBoundsException` when attempting to remove an element at an invalid index.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

### Second prompt 3.5

- Test method "test10" verifies the behavior of adding elements to an `ArrayList` and removing an element at a specified index.
- It adds three elements to the list at index 0 and asserts that the size of the list is 3.
- Then, it attempts to remove an element at index 1, expecting an `IndexOutOfBoundsException`.
- The test passes if the exception is thrown with the expected message indicating an out-of-bounds index.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

### Third prompt 3.5

- Test method "test10" verifies the behavior of the `ArrayList` class when adding elements and removing an element at a specific index.
- Adds elements with values 0, 1, and 2 to the list.
- Asserts that the size of the list is 3.
- Removes the element at index 1.
- Expects an `IndexOutOfBoundsException` to be thrown during the removal operation.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Few shot

Your task is to generate a test summary between the [CODE] and [/CODE] tags. Your summary should be like the summaries on top of the tests between the [DEMONSTRATION] and [/DEMONSTRATION] tags in terms of size and content. Please put your test summary between [SUMMARY] and [/SUMMARY] tags.

```
Your task is to generate a test summary between the [CODE] and [/CODE] tags. Your summary should be like the summaries on top of the tests
Please put your test summary between [SUMMARY] and [/SUMMARY] tags.

[DEMONSTRATION 1]
@Test
void abstractRefreshableWAC_fallsBackToInitParam() {
    XmlWebApplicationContext ctx = new XmlWebApplicationContext();
    //ctx.setConfigLocation("programmatic.xml"); // nothing set programmatically
    ContextLoaderListener cll = new ContextLoaderListener(ctx);

    MockServletContext sc = new MockServletContext();
    sc.addInitParameter(ContextLoader.CONFIG_LOCATION_PARAM, "from-init-param.xml");

    assertThatExceptionOfType(Throwable.class).isThrownBy(() ->
        cll.contextInitialized(new ServletContextEvent(sc)))
        .withMessageEndingWith("ServletContext resource [/from-init-param.xml]");
}

[SUMMARY]
/**
 * If setConfigLocation has not been called explicitly against the application context,
 * then fall back to the ContextLoaderListener init-param if present.
 */
[/SUMMARY]

[/DEMONSTRATION 1]

[DEMONSTRATION 2]
@Test
public void testSemgrexFilter() throws IOException {
    String expected="{ \"sentences\": [ true, false ]}".replaceAll(" ", "");

    String query = "The dog ate a fish.  He went home.";
    byte[] message = query.getBytes("utf-8");
    Properties props = new Properties();
    props.setProperty("annotators", "tokenize,ssplit,pos,parse");
    String queryParams = String.format("pattern=%s&properties=%s&filter=true",
        URLEncoder.encode("{}=verb >obj {}=obj", "utf-8"),
        URLEncoder.encode(PropertiesUtils.propsAsJsonString(props), "utf-8"));
    URL serverURL = new URL("http", "localhost", port, "/semgrex?" + queryParams);
    String response = slurpURL(serverURL, message);
}
```

```

    Assert.assertEquals(expected, response.replaceAll(" ", "").replaceAll("\n", ""));
}

    [SUMMARY]
    /**
     * The first sentence should match this query, but the second one should not.
     */
    [SUMMARY]
[/DEMONSTRATION 2]

[DEMONSTRATION 3]
@Test
public void longCycle() {
    LongCycleComponent longCycleComponent = DaggerLongCycle_LongCycleComponent.create();
    assertNotNull(longCycleComponent.class1());
}

    [SUMMARY]
    /**
     * Tests a cycle long enough that the real factory is created in a separate initialize method from
     * the delegate factory.
     */
    [SUMMARY]
[/DEMONSTRATION 3]

[CODE]
1.
2. @Test
3. public void test10() throws Throwable {
4.     ArrayIntList arrayIntList0 = new ArrayIntList();
5.
6.     try {
7.         arrayIntList0.add(0, 0);
8.         arrayIntList0.add(0, 1);
9.         arrayIntList0.add(0, 2);
10.        assertEquals(3, arrayIntList0.size());
11.        arrayIntList0.removeElementAt((1));

12.        fail("Expecting exception: IndexOutOfBoundsException");
13.    } catch (IndexOutOfBoundsException e) {
14.        //
15.        // Should be at least 0 and less than 0, found -1
16.        //
17.    }
18. }
[/CODE]

```

First prompt 4o [got limit]

[SUMMARY]

/\*\*

\*Tests adding and removing elements in an ArrayList, ensuring proper exception handling when attempting to access an invalid index.

\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

Second prompt 4o [got limit]

[SUMMARY]

/\*\*

\*Tests adding elements to an ArrayList and verifies that removing an element at an invalid index throws an IndexOutOfBoundsException.

\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

Third prompt 4o [got limit]

[SUMMARY]

/\*\*

\* Tests the addition of elements to an ArrayList and verifies that an IndexOutOfBoundsException

\* is thrown when attempting to remove an element at an invalid index.

\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

First prompt 3.5

[SUMMARY]

/\*\*

\* This test verifies the behavior of adding elements to an ArrayList  
\* and removing an element at a specific index. It checks if the size of  
\* the list is updated correctly after each addition and ensures that  
\* removing an element at an invalid index throws an IndexOutOfBoundsException.  
\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing 3. Also doesn't check after each addition if size is good, only once*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Second prompt 3.5

[SUMMARY]

/\*\*

\* Tests adding elements to an ArrayList and then removing an element at an invalid index,  
\* expecting an IndexOutOfBoundsException to be thrown.  
\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Third prompt 3.5

/\*\*

\* This test validates the behavior of adding elements to an ArrayList at specific indices  
\* and removing elements at specified indices. It expects an IndexOutOfBoundsException to  
be thrown  
\* when attempting to remove an element at an index outside the valid range.  
\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

# Rational

## Simple

Please generate a summary of the unit test given between the [CODE] and [/CODE] tags. Please put your answer between [SUMMARY] and [/SUMMARY] tags.

[CODE]

```
1. @Test
2. public void test3() throws Throwable {
3.     Rational rational0 = new Rational(1L, 3215L);
4.     Rational rational1 = rational0.abs();
5.     assertEquals(1L, rational0.numerator);
6.     assertEquals(3215L, rational0.denominator);
7.     assertEquals(3.11041E-4F, rational1.floatValue(), 0.01F);
8. }
```

[/CODE]

## First prompt 4o

[SUMMARY]

The unit test named test3 verifies the behavior of the Rational class when creating a rational number and **obtaining its absolute value**. Specifically, it:

- Creates a Rational object rational0 with a numerator of 1 and a denominator of 3215.
- Calls the abs method on rational0, storing the result in rational1.
- Asserts that the numerator of rational0 is 1.
- Asserts that the denominator of rational0 is 3215.
- Asserts that the float value of rational1 is approximately 3.11041E-4 within a tolerance of 0.01.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Has multiple lines the same 2. Has repetitive numbers*

*Naturalness: 1. Nothing special*

## Second prompt 4o

[SUMMARY]

The unit test test3 verifies the behavior of the Rational class, particularly the abs() method. It creates a Rational object rational0 with a numerator of 1 and a denominator of 3215. It then calls the abs() method on rational0 to get rational1. The test asserts that the numerator and denominator of rational0 remain 1 and 3215, respectively. Finally, it checks that the float value of rational1 is approximately 3.11041E-4 within a tolerance of 0.01.

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Has repetitive numbers*

*Naturalness: 1. Nothing special*



[/SUMMARY]

### Third prompt 40

[SUMMARY]

The unit test test3 verifies the behavior of the abs() method in the Rational class. The test creates a Rational object rational0 with a numerator of 1L and a denominator of 3215L. It then calls the abs() method on this object, creating rational1. The test checks that the numerator and denominator of rational0 remain unchanged. Additionally, it verifies that the float value of rational1 is approximately 3.11041E-4 within a tolerance of 0.01F.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

### First prompt 3.5

[SUMMARY]

The given unit test checks the functionality of the abs() method in the Rational class. It creates a Rational object with numerator 1 and denominator 3215, then calls the abs() method on it. The test verifies that the original Rational object remains unchanged and that the **absolute value is computed correctly** by checking the numerator, denominator, and the float value of the result.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

### Second prompt 3.5

[SUMMARY]

The unit test provided tests a method named abs() of the Rational class. It creates a Rational object with numerator 1 and denominator 3215. Then it invokes the abs() method on this object. The test asserts that the numerator and denominator of the original object remain unchanged, and it checks whether the **absolute value of the rational number** approximately equals 3.11041E-4 with a tolerance of 0.01.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

### Third prompt 3.5

[SUMMARY]

The unit test provided verifies the functionality of the `abs()` method in the `Rational` class. It creates a `Rational` object with numerator 1 and denominator 3215, then invokes the `abs()` method on it. The test checks if the numerator and denominator of the original object remain unchanged. Additionally, it verifies if the absolute value of the `Rational` object approximately equals  $3.11041\text{E-}4$  with a delta of 0.01.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## With context (abs())

```
Please generate a summary of the unit test given between the [CODE] and [/CODE] tags. The method under test is between the [METHOD_TESTED] AND [/METHOD_TESTED] tags. Please put your answer between [SUMMARY] and [/SUMMARY] tags.

[CODE]
1. @Test
2. public void test3() throws Throwable {
3.     Rational rational0 = new Rational(1L, 3215L);
4.     Rational rational1 = rational0.abs();
5.     assertEquals(1L, rational0.numerator);
6.     assertEquals(3215L, rational0.denominator);
7.     assertEquals(3.11041E-4F, rational1.floatValue(), 0.01F);
8. }
[/CODE]

[METHOD_TESTED]
    public Rational abs() {
        return new Rational((numerator < 0L) ? -numerator : numerator, (denominator < 0L) ? -denominator : denominator).reduce();
    }
[/METHOD_TESTED]
```

## First prompt (understands the method) 4o

[SUMMARY]

The unit test test3 verifies the behavior of the abs method in the Rational class. **The method is supposed to return the absolute value of a Rational number.** In the test, a Rational object rational0 is created with a numerator of 1 and a denominator of 3215. The abs method is called on this object, resulting in a new Rational object rational1. The test then asserts that the original rational0 object remains unchanged with a numerator of 1 and a denominator of 3215. Finally, it checks that the floatValue of the new Rational object rational1 is approximately 0.000311041, allowing a margin of error of 0.01.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't need to write out how much approximately it is*

*Naturalness: 1. Nothing special*

## Second prompt (no difference) 4o

[SUMMARY]

The unit test named test3 verifies the functionality of the abs method of the Rational class. The test creates a Rational object with a numerator of 1 and a denominator of 3215. It then calls the abs method on this object and asserts that **the original object's numerator and denominator remain unchanged**. Additionally, it checks that the floatValue of the resulting Rational object is approximately 0.000311041 within a tolerance of 0.01.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't need to write it out*

*Naturalness: 1. Nothing special*

### Third prompt (understands the method) 4o

[SUMMARY]

The unit test named test3 tests the abs method of the Rational class. **The abs method returns a new Rational object with the absolute values of the numerator and denominator of the original object.** In the test, a Rational object is created with a numerator of 1 and a denominator of 3215. The abs method is called on this object, and the test asserts that the original object's numerator and denominator remain unchanged. It also checks that the floatValue of the resulting Rational object is approximately 3.11041E-4 with a tolerance of 0.01.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. I think doesn't need to write what the abs is in a new sentence*

*Naturalness: 1. Nothing special*

### First prompt (understands the method) 3.5

[SUMMARY]

The unit test provided tests the abs() method of the Rational class. Within the test method, a Rational object is initialized with numerator 1 and denominator 3215. Then, the abs() method is called on this object. The test verifies that after calling abs(), the original object remains unchanged, with its numerator and denominator intact. Additionally, it checks whether the absolute value of the Rational object's numerator is correctly calculated, with a tolerance of 0.01. Overall, the test ensures the correctness of the abs() method in terms of returning the **absolute value of a rational number.**

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

### Second prompt (understands the method) 3.5

[SUMMARY]

The unit test provided aims to verify the functionality of the abs() method within the Rational class. The test creates a Rational object initialized with numerator 1 and denominator 3215. It then calls the abs() method on this object. The test asserts that the numerator and denominator of the original object remain unchanged. Additionally, it asserts that the float value of the returned Rational object is approximately equal to 0.000311041. This test ensures that the abs() **method correctly returns the absolute value of the rational number.**

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't need to write it out*

*Naturalness: 1. Nothing special*

### Third prompt (understands the method) 3.5

[SUMMARY]

The provided unit test verifies the functionality of the `abs()` method in the `Rational` class. The test instantiates a `Rational` object with a numerator of 1 and a denominator of 3215. It then calls the `abs()` method on this object and asserts that the resulting numerator and denominator of the original object remain unchanged. Additionally, it checks whether the `floatValue()` of the resulting object is approximately equal to `3.11041E-4` within a delta of 0.01. This test ensures that the `abs()` method **correctly returns the absolute value of a rational number**.

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Prompt engineering

```
[INST] As a detail-oriented developer focused on enhancing the clarity of a test suite, your task is to analyze the provided Java code and generate a descriptive and concise summary of the unit test. Follow these steps:
  1. Carefully read the Java code between the [CODE] tags.
  2. Identify the primary functionality or purpose of the test.
  3. Analyze actions performed in the test.
  4. Formulate a test method summary that succinctly and concisely captures what the test case does in a bullet point format for easy, quick reading
  5. Place your complete suggested test summary between the [TESTSUMMARY] and [/TESTSUMMARY] tags, ensuring it is clear and precise without unnecessary descriptions or information.
  Remember, your focus is on clarity and precision. Use your expertise to provide a meaningful and appropriate summary.[/INST]

[CODE]
1. @Test
2. public void test3() throws Throwable {
3.     Rational rational0 = new Rational(1L, 3215L);
4.     Rational rational1 = rational0.abs();
5.     assertEquals(1L, rational0.numerator);
6.     assertEquals(3215L, rational0.denominator);
7.     assertEquals(3.11041E-4F, rational1.floatValue(), 0.01F);
8. }
[/CODE]
```

### First prompt 4o

[TESTSUMMARY]

- Purpose: Verify the absolute value computation of a Rational number.
- Actions:
  - Create a `Rational` object with a numerator of 1 and a denominator of 3215.
  - **Calculate the absolute value of the Rational object.**
  - Assert that the original `Rational` object's numerator and denominator remain unchanged.

- Assert that the float value of the absolute Rational object is approximately 3.11041E-4 within a tolerance of 0.01.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Second prompt 4o

[TESTSUMMARY]

- Initializes a Rational object with a numerator of 1 and a denominator of 3215.
- **Computes the absolute value of the Rational object.**
- Asserts that the numerator and denominator of the original Rational object remain unchanged.
- Asserts that the float value of the absolute Rational object is approximately 3.11041E-4 within a tolerance of 0.01.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Third prompt 4o

[TESTSUMMARY]

- Test Method Name: test3
- Purpose: Verify the behavior of the abs method in the Rational class.
- Actions:
  - Create a Rational object with numerator 1L and denominator 3215L.
  - Call the abs method on the Rational object.
  - Assert that the original Rational object's numerator and denominator remain unchanged.
  - Assert that the absolute value (float representation) of the Rational object is approximately 3.11041E-4 with a tolerance of 0.01.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't need the name*

*Naturalness: 1. Nothing special*

## First prompt 3.5

[TESTSUMMARY]

- Test method summary:
  - Test the abs() method of the Rational class.
  - Create a Rational object with numerator 1L and denominator 3215L.
  - Call the abs() method on the Rational object.
  - Verify that the original Rational object remains unchanged.
  - Verify that the absolute value of the Rational object is approximately 0.000311041.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Multiple lines the same*

*Naturalness: 1. Nothing special*

## Second prompt 3.5

[TESTSUMMARY]

- Test method name: test3()
- Purpose: Validate the behavior of the abs() method in the Rational class.
- Actions:
  - Create a Rational object with numerator 1L and denominator 3215L.
  - Call the abs() method on the Rational object.
  - Assert that the original Rational object's numerator and denominator remain unchanged.
  - Assert that the float value of the absolute Rational object is approximately 3.11041E-4 with a delta of 0.01F.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't need the name*

*Naturalness: 1. Nothing special*

## Third prompt 3.5

[TESTSUMMARY]

- Test method: "test3()"
- Purpose: Verify the functionality of the "abs()" method in the Rational class.
- Actions:
  - Creates a Rational object with numerator 1 and denominator 3215.

- Calls the "abs()" method on the Rational object.
- Asserts that the original Rational object remains unchanged.
- Asserts that the absolute value of the Rational object is approximately 3.11041E-4F.

[/TESTSUMMARY]

*Context: 1. Has what it is testing 2. Has how it is testing*

*Conciseness: 1. Doesn't need the name*

*Naturalness: 1. Nothing special*



## Few shot

Your task is to generate a test summary between the [CODE] and [/CODE] tags. Your summary should be like the summaries on top of the tests between the [DEMONSTRATION] and [/DEMONSTRATION] tags in terms of size and content. Please put your test summary between [SUMMARY] and [/SUMMARY] tags.

[same examples as few shot of arrayintlist]

```
[CODE]
1. @Test
2. public void test3() throws Throwable {
3.     Rational rational0 = new Rational(1L, 3215L);
4.     Rational rational1 = rational0.abs();
5.     assertEquals(1L, rational0.numerator);
6.     assertEquals(3215L, rational0.denominator);
7.     assertEquals(3.11041E-4F, rational1.floatValue(), 0.01F);
8. }
[/CODE]
```

## First prompt 4o

[SUMMARY]

/\*\*

\* Tests the absolute value computation of a Rational number and verifies the numerator, denominator, and float value.

\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

## Second prompt 4o

[SUMMARY]

/\*\*

\* Tests the absolute value function of the Rational class to ensure it correctly handles positive values and returns the expected float representation.

\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

### Third prompt 4o

[SUMMARY]

/\*\*

\* Verifies the absolute value method for a Rational number with positive numerator and denominator, ensuring the original Rational object remains unchanged.

\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

### First prompt 3.5

[SUMMARY]

/\*\*

\* This test validates the functionality of the 'abs' method in the Rational class.

\* It ensures that the method correctly computes the absolute value of a Rational number,

\* maintaining the original numerator and denominator.

\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

### Second prompt 3.5

[SUMMARY]

/\*\*

\* This test validates the abs() method of the Rational class.

\* It checks whether the absolute value of a Rational number is calculated correctly,

\* including the numerator, denominator, and the float value.

\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

### Third prompt 3.5

[SUMMARY]

/\*\*

\* This test verifies the abs() method of the Rational class.

\* It checks if the method correctly returns the absolute value of the Rational number.

\*/

[/SUMMARY]

*Context: 1. Has what it is testing 2. Has not how it is testing*

*Conciseness: 1. Doesn't have unnecessary information*

*Naturalness: 1. Nothing special*

# Keycloak

## Few shot

Your task is to generate a test summary between the [CODE] and [/CODE] tags. Your summary should be like the summaries on top of the tests between the [DEMONSTRATION] and [/DEMONSTRATION] tags in terms of size and content. Please put your test summary between [SUMMARY] and [/SUMMARY] tags.

[same examples as few shot of arrayintlist]

```
@Test(timeout = 4000)
public void testEncodedPath() throws Throwable {
    KeycloakUriBuilder uri = KeycloakUriBuilder.fromPath("x");
    HashMap<String, Integer> map = new HashMap<String, Integer>();
    URI result = uri.buildFromEncodedMap(map);
    assertEquals("x", result.getRawPath());
}
```

## First prompt 4o

[SUMMARY]

/\*\*

\* Tests that the URI builder correctly constructs a URI from an encoded map with a simple path.

\*/

[/SUMMARY]

## Second prompt 4o

[SUMMARY]

/\*\*

\* Tests that a URI built from an empty encoded map retains the specified raw path.

\*/

[/SUMMARY]

## Third prompt 4o

[SUMMARY]

/\*\*

\* Verifies that the URI builder correctly constructs a URI from an encoded path with an empty parameter map.

\*/

[/SUMMARY]

### First prompt 3.5

[SUMMARY]

/\*\*

\* Verifies that the Keycloak URI builder constructs the path correctly when building from an encoded map.

\*/

[/SUMMARY]

### Second prompt 3.5

[SUMMARY]

/\*\*

\* Verifies that building a URI from an empty map with a given path preserves the path when encoded.

\*/

[/SUMMARY]

### Third prompt 3.5

[SUMMARY]

/\*\*

\* Tests that KeycloakUriBuilder correctly builds a URI from an encoded map,  
\* ensuring that the raw path remains unchanged.

\*/

[/SUMMARY]

# ArrayIntList2

## Few shot

Your task is to generate a test summary between the [CODE] and [/CODE] tags. Your summary should be like the summaries on top of the tests between the [DEMONSTRATION] and [/DEMONSTRATION] tags in terms of size and content. Please put your test summary between [SUMMARY] and [/SUMMARY] tags.

[same examples as few shot of arrayintlist]

```
1  @Test
2  public void test8() throws Throwable {
3      ArrayIntList arrayIntList0 = new ArrayIntList();
4      arrayIntList0.ensureCapacity(9);
5      assertEquals(0, arrayIntList0.size());
6  }
```

## First prompt 4o

[SUMMARY]

/\*\*

\* Ensures that the ArrayIntList can expand its capacity and still report the correct size.

\*/

[/SUMMARY]

## Second prompt 4o

[SUMMARY]

/\*\*

\* Ensures that the capacity of an ArrayIntList can be increased to a specified value without affecting its size.

\*/

[/SUMMARY]

### Third prompt 4o

[SUMMARY]

/\*\*

\* Ensures that calling ensureCapacity on an empty ArrayList does not change its size.

\*/

[/SUMMARY]

### First prompt 3.5

[SUMMARY]

/\*\*

\* Verifies that calling ensureCapacity with a value larger than the current capacity increases the capacity accordingly.

\*/

[/SUMMARY]

### Second prompt 3.5

[SUMMARY]

/\*\*

\* Tests that the size of an ArrayList is zero after ensuring capacity for 9 elements.

\*/

[/SUMMARY]

### Third prompt 3.5

[SUMMARY]

/\*\*

\* Verifies that an ArrayList properly initializes with a specified capacity,

\* and its size remains zero initially.

\*/

[/SUMMARY]