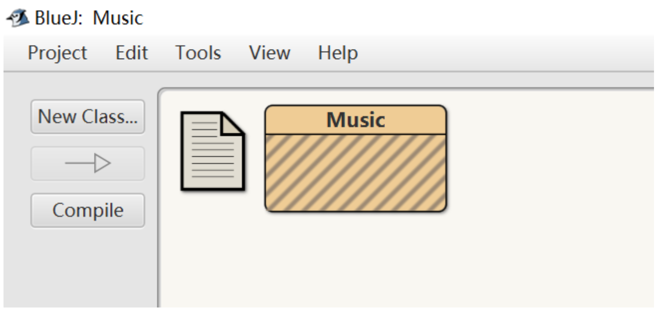
BlueJ Project

Yufei LIU & Qilian GU

The idea of this tutorial is to help you better understand how to play with java classes and use the methods and tests to verify the implementation. In this tutorial, we will explain to you step by step how it works, here we take an example of a class of music, with only 2 attributes, the title of the music and the edition of the music, then we will create a method to get the age of the music, etc, to discover more, please follow our tutorial. Don't worry if you are new to Java, we will use images at each step to illustrate what we've done. Let's embark on an exciting journey to craft a brand new Music class that will serve as the foundation for our adventures!

Step 1: create a java class named music.



What are we waiting for one piece of music? Of course not a hamburger or a kebab, added 2 attributes then.

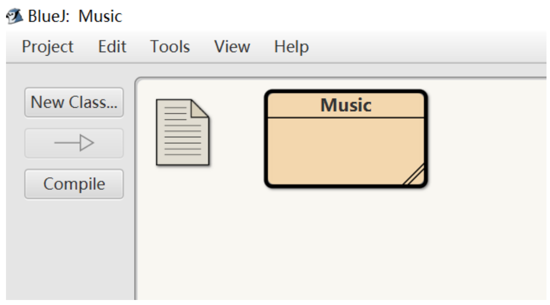
```
public class Music
{
    // instance variables - replac
    private String nom;
    private int edition;
```

Let's now explore the constructor() method, a fundamental building block in object-oriented programming that enables the proper initialization of our class instances.

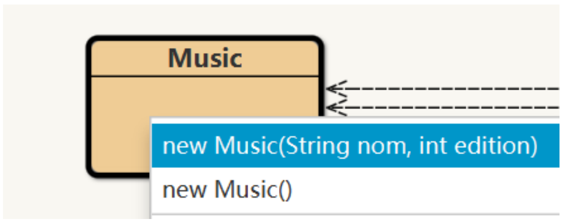
```
/**
 * Constructor for objects of class Music
 */
public Music()
{

}
```

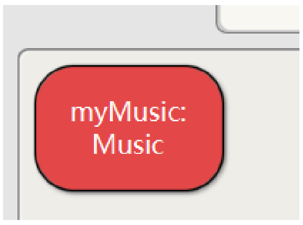
```
    public Music(String nom, int edition)
{
    this.nom = nom;
    this.edition = edition;

}
```

The class was compiled and the gray lines disappeared meaning that the compilation was successful. If not ? Check your code !

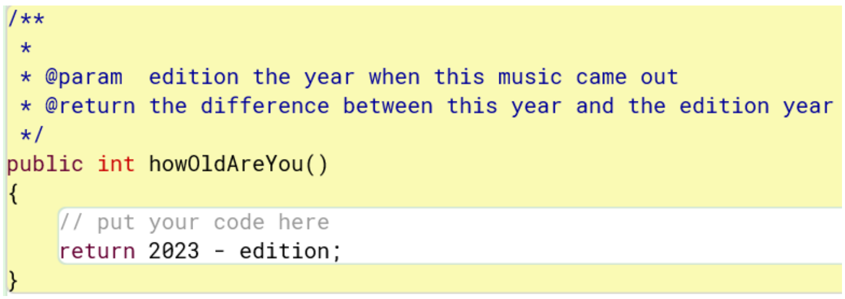


With a simple right click, an instance of the class Music was created, look at this beautiful red square, genius !



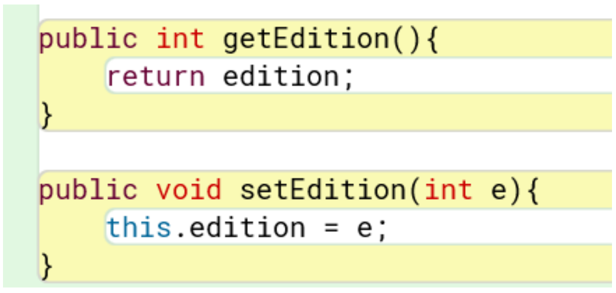


And, behold, a method to unravel the music's age is coming!
It is a simple method implemented to the Music class, a very easy calculation.



```
/**
 *
 * @param  edition the year when this music came out
 * @return the difference between this year and the edition year
 */
public int howOldAreYou()
{
    // put your code here
    return 2023 - edition;
}
```

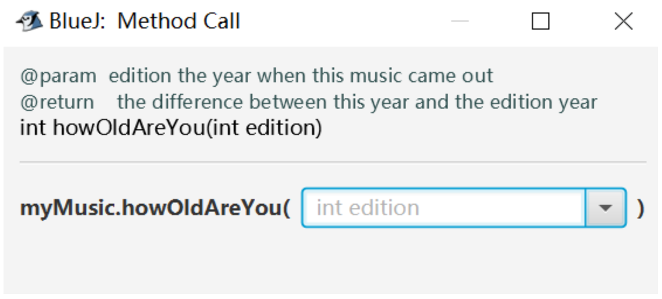
Think further now, what do we need for our next steps? We may need to set or get the edition of the music, so we added a getter and setter of the edition, i.e the year of the song when it came out. Which could certainly make our life easier for the next steps.



```
public int getEdition(){
    return edition;
}

public void setEdition(int e){
    this.edition = e;
}
```

Ok till now, a basic class and a simple method are created, what do we need to make sure that all works well? Right, a test class.

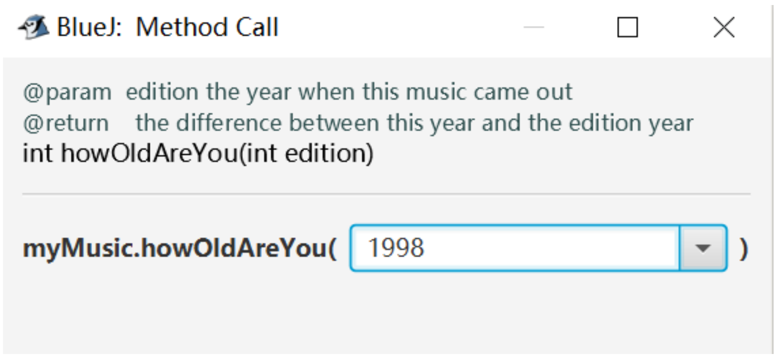
When you run the function, an interactive bar shows up, allowing you to give an edition of your music. I typed 1998 because I was born this year (you can check any age you want 😀 extraordinarily robust!)



BlueJ: Method Call

@param edition the year when this music came out
@return the difference between this year and the edition year
int howOldAreYou(int edition)

myMusic.howOldAreYou(int edition)



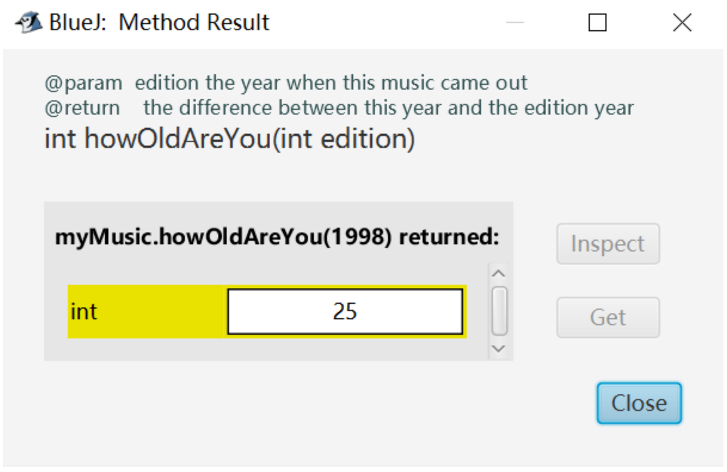
BlueJ: Method Call

@param edition the year when this music came out
@return the difference between this year and the edition year
int howOldAreYou(int edition)

myMusic.howOldAreYou(1998)

You can see your parametre appears in parentheses of the function, pretty cool isn't it? Let's just do a calculation in our head, so if the song came out in 1998, what is the age of the song, knowing that we are in 2023.

Bingo! It is 25, not difficult, no?


BlueJ: Method Result

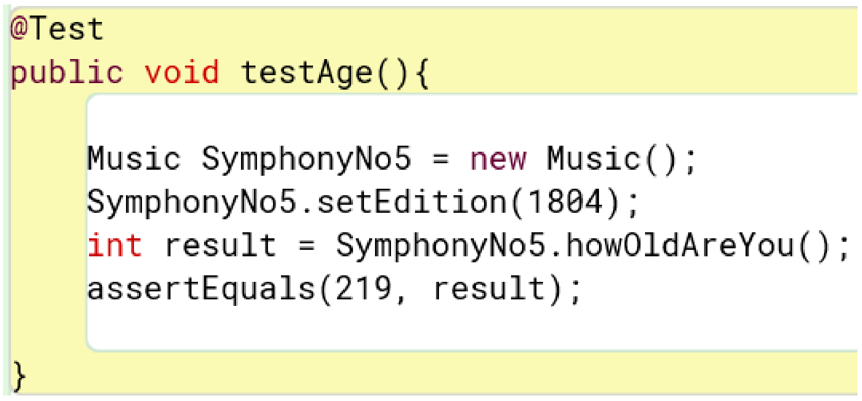
@param edition the year when this music came out
@return the difference between this year and the edition year
int howOldAreYou(int edition)

myMusic.howOldAreYou(1998) returned:

```
int 25
```

Is this enough? Of course not! Let's discover unit tests. Unit testing offers advantages such as isolation, automation, faster feedback, easier debugging, and improved code quality, making it more effective than just running a function as a standalone test.

So then, I will show you how to test our age function with a unit test, don't blink your eyes! Beethoven's Symphony No. 5 was released in 1804, so our goal is to test if the result of our function is equal to the number we calculate, which is 219 in this case.

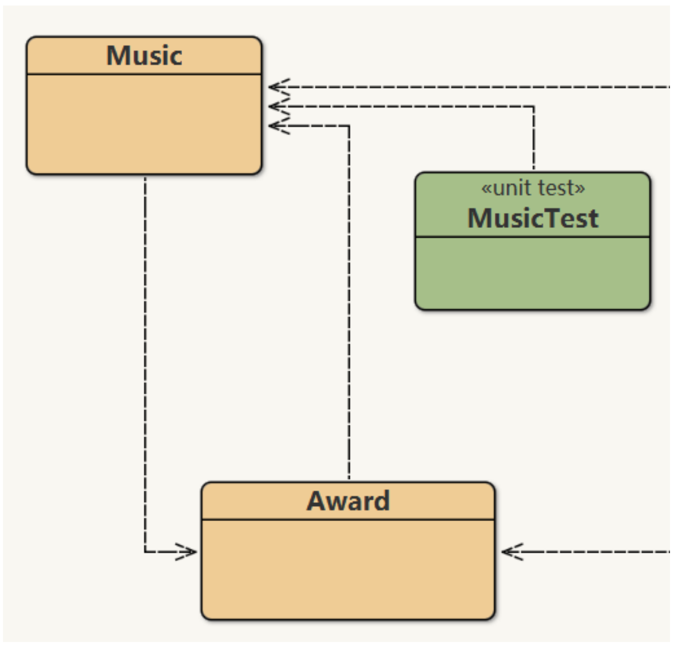


```
@Test
public void testAge(){

    Music SymphonyNo5 = new Music();
    SymphonyNo5.setEdition(1804);
    int result = SymphonyNo5.howOldAreYou();
    assertEquals(219, result);

}
```

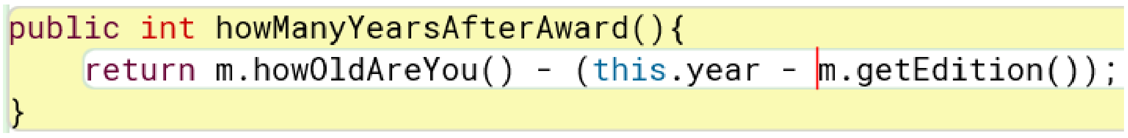
Just music? So boring!

Add the second class **Award** to my principal class Music with a relation 0..1 and 0..1, Here I define a WorstSongAward which means every music has zero or one award and every award belongs to zero or one music. Although I don't wish you to get this award 😂.

Here is how the classes and MusicTest linked.

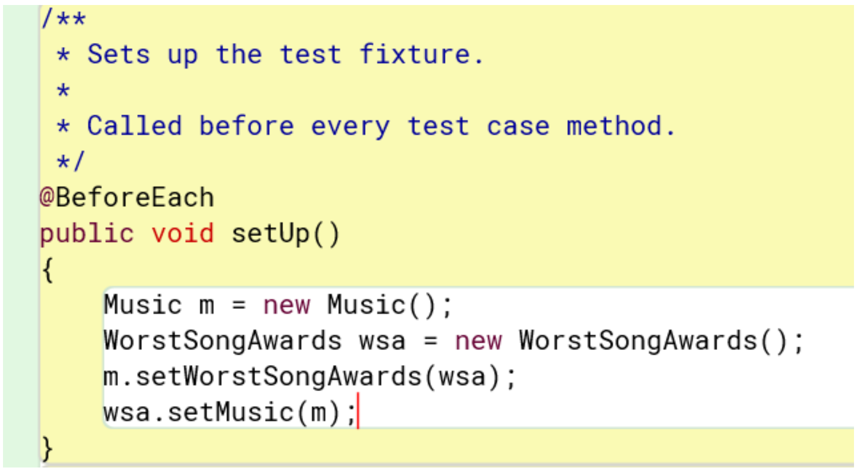


We could then add a second function in our new class which interacts with the first function. Pay attention to “the spaces” we should follow java coding conventions.



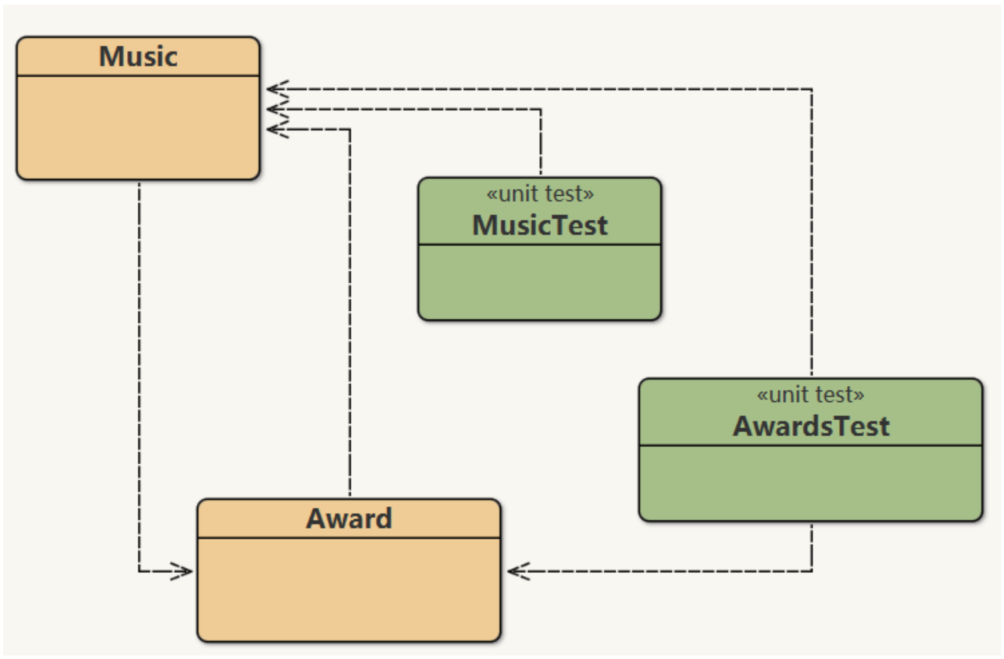
```
public int howManyYearsAfterAward(){
    return m.howOldAreYou() - (this.year - m.getEdition());
}
```

To batch the test cases, instantiating an object in every test is not convenient. A setup before the Unit test is very necessary.



```
/**
 * Sets up the test fixture.
 *
 * Called before every test case method.
 */
@BeforeEach
public void setUp()
{
    Music m = new Music();
    WorstSongAwards wsa = new WorstSongAwards();
    m.setWorstSongAwards(wsa);
    wsa.setMusic(m);
}
```

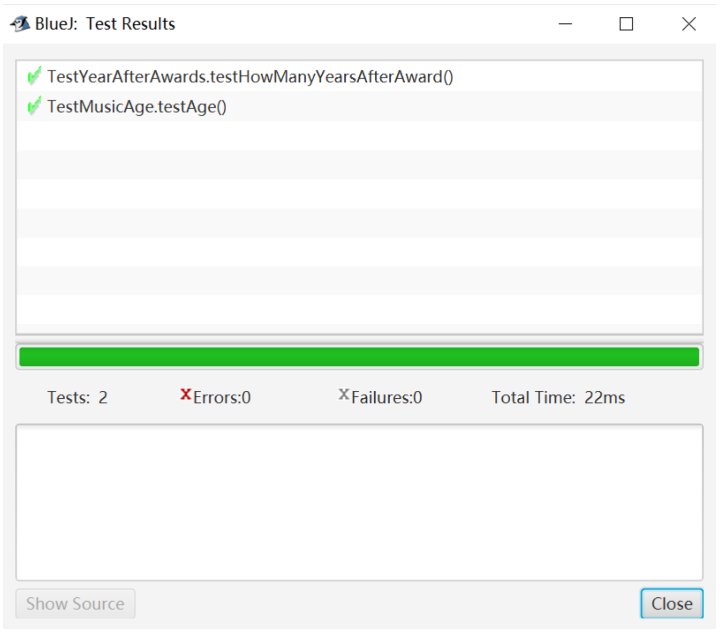
New member to our family and how they link to each other :



And the code for the second test case, we are in 2023, and the song won the worst song prize in 2013, so it has been 10 years since the song held this prize.

```
@Test
public void testHowManyYearsAfterAward(){
    int result = worstSongAwards.howManyYearsAfterAward();
    assertEquals(10, result);

}
```

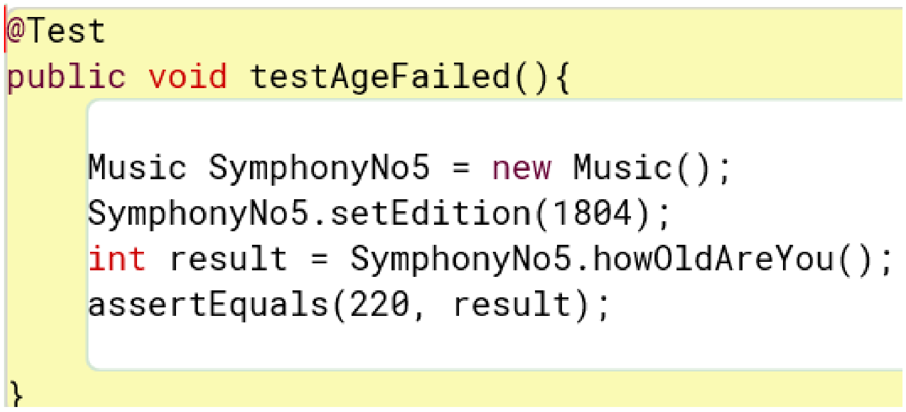
The 2 tests succeed, excellent 👍



```
✓ TestYearAfterAwards.testHowManyYearsAfterAward()
✓ TestMusicAge.testAge()

Tests:  2        ✗Errors:0        ✗Failures:0        Total Time:  22ms
```

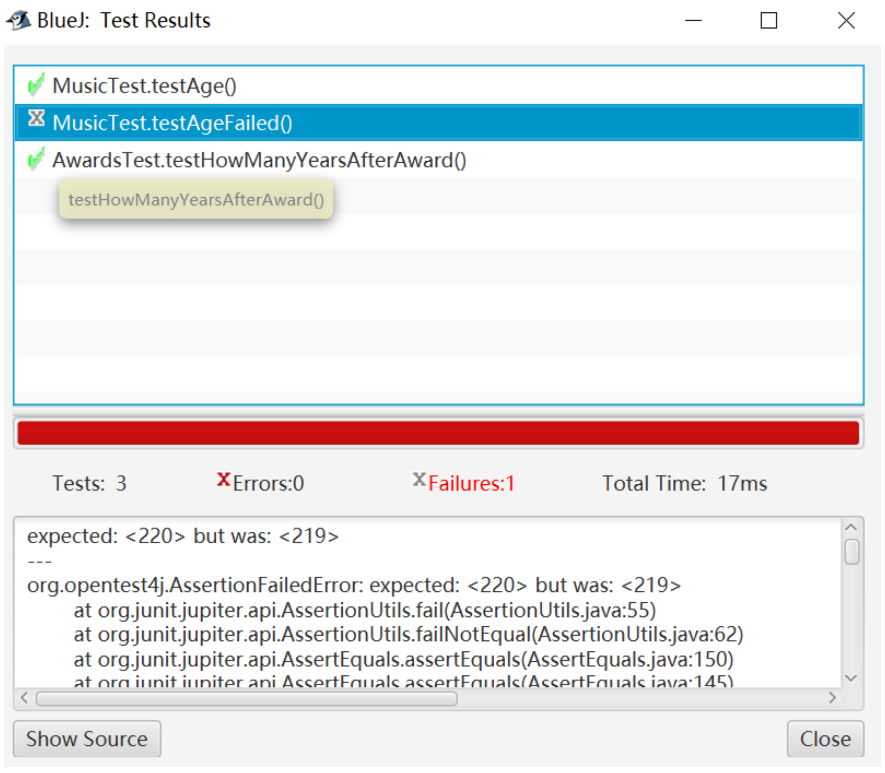
To go further, we would like to test more to get more credibility of our codes.

So let’s make a test for failure cases : Sometimes it’s good to see red in your test (to avoid being always green. it means the test works!) The age of this song which came out should be 219 years, we asserted 220 for the test, which will show a red test line result if our method works well.



```
@Test
public void testAgeFailed(){

    Music SymphonyNo5 = new Music();
    SymphonyNo5.setEdition(1804);
    int result = SymphonyNo5.howOldAreYou();
    assertEquals(220, result);

}
```

Red lines matter, that’s exactly what we expected.



BlueJ: Test Results

MusicTest.testAge()
MusicTest.testAgeFailed()
AwardsTest.testHowManyYearsAfterAward()
testHowManyYearsAfterAward()

Tests: 3   Errors:0   Failures:1   Total Time: 17ms

```
expected: <220> but was: <219>
---
org.opentest4j.AssertionFailedError: expected: <220> but was: <219>
	at org.junit.jupiter.api.AssertionUtils.fail(AssertionUtils.java:55)
	at org.junit.jupiter.api.AssertionUtils.failNotEqual(AssertionUtils.java:62)
	at org.junit.jupiter.api.AssertEquals.assertEquals(AssertEquals.java:150)
	at org.junit.jupiter.api.AssertEquals.assertEquals(AssertEquals.java:145)
```

All works well till now, let’s get more tests for getter, constructor and setter :

```java
@Test
public void testDefaultConstructor() {
    Music defaultMusic = new Music();
    assertNotNull(defaultMusic, "The default constructor should create a non-null Music object.");
}

@Test
public void testConstructorWithParameters() {
    Music customMusic = new Music("Custom Song", 2005);
    assertEquals("Custom Song", customMusic.getNom(), "The constructor should set the correct 'nom' attribute.");
    assertEquals(2005, customMusic.getEdition(), "The constructor should set the correct 'edition' attribute.");
}

@Test
public void testGetEdition() {
    int edition = music.getEdition();
    assertEquals(2021, edition, "The getEdition method should return the correct edition year.");
}

@Test
public void testSetEdition() {
    music.setEdition(2022);
    int updatedEdition = music.getEdition();
    assertEquals(2022, updatedEdition, "The setEdition method should update the edition year.");
}
```

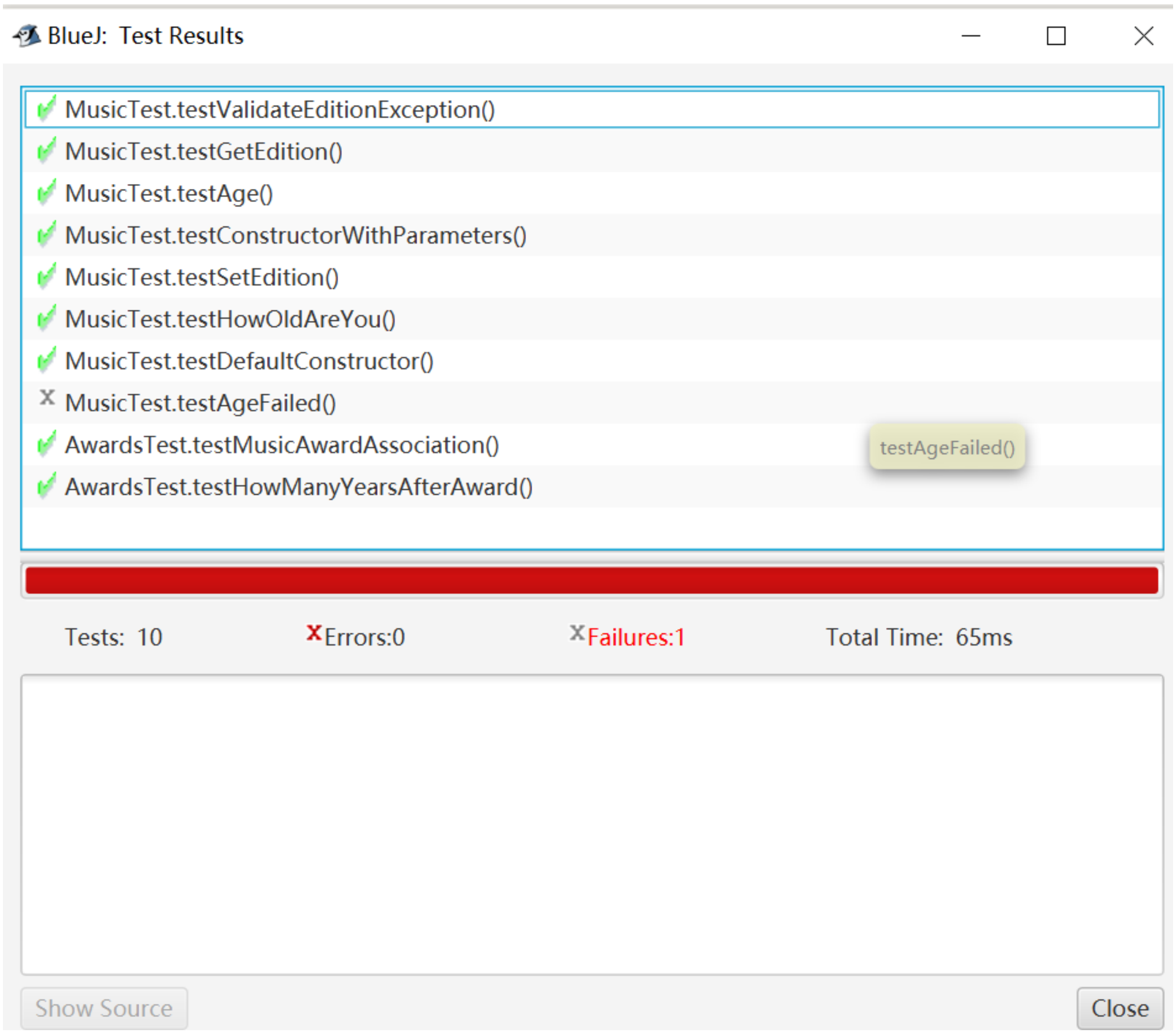
All the test classes of getter, setter and constructors work well, now we are going to test the association between 2 classes, the class of music and the class of award. Surely, we’ll use the getter and setter we tested for this association test.

```java
@Test
public void testMusicAwardAssociation() {
    Music music = new Music("Test Song", 2010);
    Award award = new Award();

    music.setWorstSongAwards(award);
    award.setMusic(music);

    assertEquals(music, award.getMusic(), "The Award object should have the correct Music object associated.");
    assertEquals(award, music.getWorstSongAwards(), "The Music object should have the correct Award object associated.");
}
```

And finally, don’t forget to make a test of exception :

```java
@Test
public void testValidateEditionException() {
    Music music = new Music("Test Song", 2010);

    assertThrows(IllegalArgumentException.class, () -> {
        music.validateEdition(-1);
    }, "The validateEdition method should throw an IllegalArgumentException for invalid edition years.");
}
```

Congratulations, you are arriving at the end of this tutorial, we hope you’ve learned something new about the implementation of java classes and tests. This is just a start of your long journey of learning Java, hope you find this tutorial helpful. If you have any questions about this tutorial or anything you do not agree with, we are happy to discuss with you. You could contact us by email javamusic@dauphine.eu,so see you soon!



BlueJ: Test Results

```
✓ MusicTest.testValidateEditionException()
✓ MusicTest.testGetEdition()
✓ MusicTest.testAge()
✓ MusicTest.testConstructorWithParameters()
✓ MusicTest.testSetEdition()
✓ MusicTest.testHowOldAreYou()
✓ MusicTest.testDefaultConstructor()
X MusicTest.testAgeFailed()
✓ AwardsTest.testMusicAwardAssociation()
✓ AwardsTest.testHowManyYearsAfterAward()

Tests:  10        XErrors:0        XFailures:1        Total Time:  65ms
```