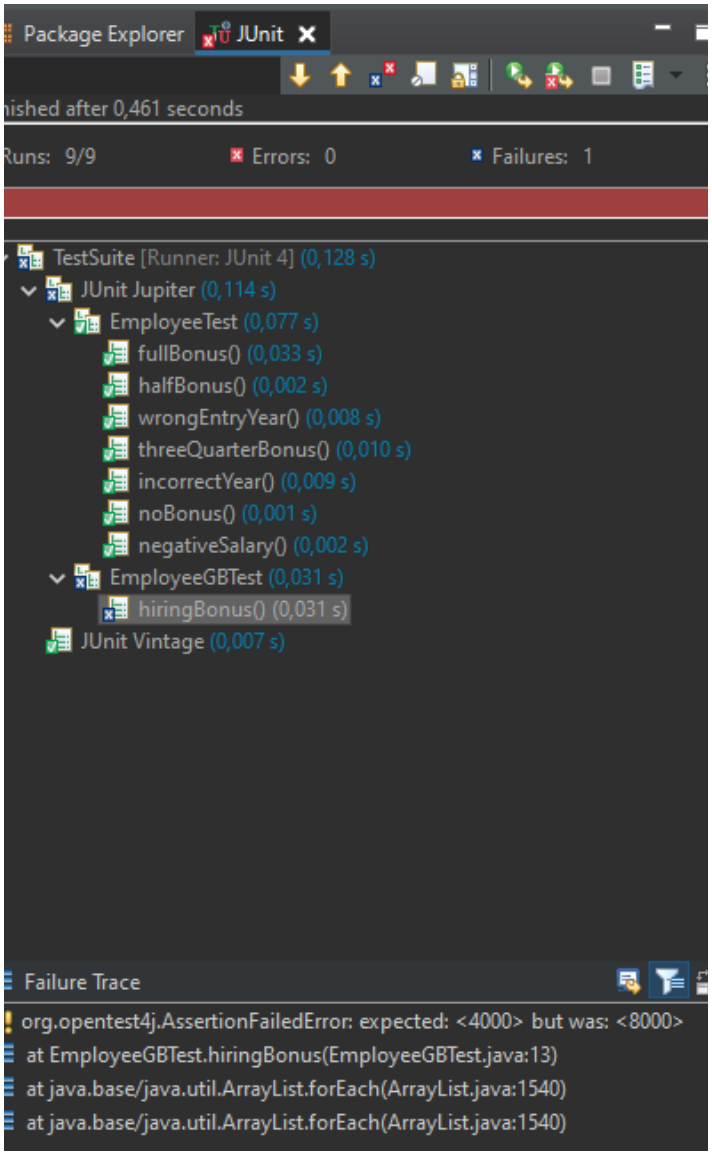
1 Step 1: Create Black Box-Test Cases

Define the equivalence classes for a black-box test of the constructor **`Employee(int, int)`** and the method **`bonus(int):int`** of an employee; select the equivalence classes, select representatives and define the associated test cases with expected result. Document your test cases in the following table. Actually this is the most time-consuming part of the exercise.

No.	Equivalence class (as a test comment)	Parameter for construction an `Employee`	Parameters for `bonus`	Expected result of `Employee` or `bonus`
1	No Bonus	`yoh: 2020` `sal: 4000`	`yoc: 2020`	`0 eur`
2	50% Bonus	yoh: 2016 sal: 4000	`yoc: 2020`	`2000 eur`
3	75% Bonus	yoh: 2013 sal: 4000	`yoc: 2020`	`3000 eur`
4	100% Bonus	yoh: 2010 sal: 4000	`yoc: 2020`	`4000 eur`
5	Wrong Entry Year	yoh: 1989 sal: 4000		`Runtime Exception: "Wrong entry year"`
6	Negative Salary	yoh: 2011 sal: -4000		`Runtime Exception "Negative Salary"`
7	Incorrect Year	yoh: 2020 sal: 4000	`yoc: 2018`	`Runtime Exception: "Wrong calculation year"`

JUNIT RESULTS



COVERAGE RESULTS

erage ✕ Problems Javadoc Declaration

21 5:05:56 μ.μ.)

	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Test	100,0 %	75	0	75
	100,0 %	75	0	75
package)	100,0 %	75	0	75
oyee.java	100,0 %	75	0	75
nployee	100,0 %	75	0	75
version()	100,0 %	2	0	2
Employee(int, int)	100,0 %	24	0	24
bonus(int)	100,0 %	49	0	49

Black Box Test Code

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;


class EmployeeTest {

	@BeforeAll
	public static void info() {
		System.out.println("Employee Version: " +
Employee.version());
	}

	@Test
	void noBonus() {
		Employee e = new Employee(2020, 4000);
		int actual = e.bonus(2020);
		int expected = 0;
		assertEquals(expected, actual);
	}

	@Test
	void halfBonus() {
		Employee e = new Employee(2016, 4000);
		int actual = e.bonus(2020);
		int expected = 2000;
		assertEquals(expected, actual);
	}

	@Test
	void threeQuarterBonus() {
		Employee e = new Employee(2013, 4000);
		int actual = e.bonus(2020);
		int expected = 3000;
		assertEquals(expected, actual);
	}

	@Test
	void fullBonus() {
		Employee e = new Employee(2010, 4000);
		int actual = e.bonus(2020);
		int expected = 4000;
		assertEquals(expected, actual);
	}

	@Test
	void wrongEntryYear() {
		int n = 1989;
```

```
		assertThrows(RuntimeException.class, () -> new Employee(n,
4000),"Wrong entry year.");
	}

	@Test
	void negativeSalary() {
		int n = -4000;
		assertThrows(RuntimeException.class, () -> new
Employee(2011, n), "Negative salary.");
	}

	@Test
	void incorrectYear() {
		Employee e = new Employee(2020, 4000);
		assertThrows(RuntimeException.class, () ->
e.bonus(2018),"Wrong calculation year");
	}

}
```

Glass Box Test Code

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class EmployeeGBTest {

	@Test
	void hiringBonus() {
		Employee e = new Employee(1993, 4000);
		int actual = e.bonus(2020);
		int expected = 4000;
		assertEquals(expected, actual);
	}

}
```

Test Suite Code

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

class EmployeeGBTest {

	@BeforeAll
	public static void info() {
		System.out.println("Employee Version: " +
Employee.version());
	}

	@Test
	void hiringBonus() {
		Employee e = new Employee(1993, 4000);
		int actual = e.bonus(2020);
		int expected = 8000;
		assertEquals(expected, actual);
	}

}
```