

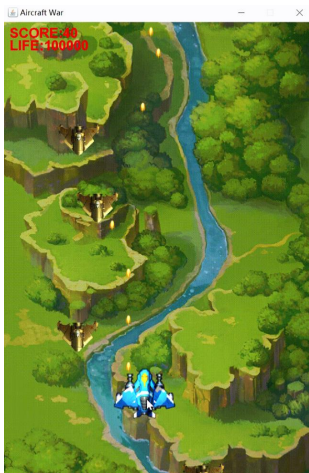


实验三：JUnit单元测试

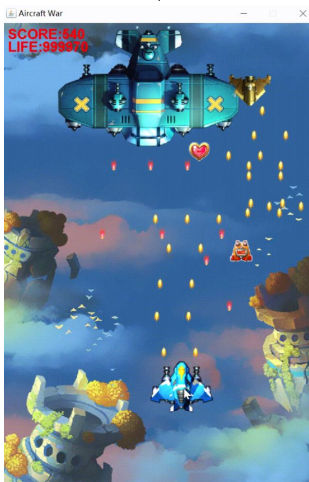
实验与创新实践教育中心 · 计算机与数据技术实验教学部

本学期实验总体安排

初始版本



最终版本



游戏主界面
英雄机移动
英雄机子弹直射
碰撞检测
统计得分和生命值

重构代码，采用**单例模式**
创建英雄机
重构代码，采用**工厂模式**
创建敌机和道具

重构代码，采用**策略模式**
实现不同弹道发射
采用**数据访问对象模式**
实现得分排行榜

采用**观察者模式**
实现炸弹道具生效
采用**模板模式**
实现三种游戏难度

初始版本

01

绘制UML类图
创建精英敌机并直射子弹
精英敌机随机掉落三种道具
加血道具生效

02

03

添加**JUnit单元测试**
创建**Boss**和**超级精英**敌机

04

05

使用**Swing**添加游戏难度选择和排行榜界面
使用**多线程**实现音效的开启/关闭、及火力道具

06

本学期实验总体安排

实验项目	一	二	三	四	五	六
学时数	2	2	2	2	4	4
实验内容	飞机大战 功能分析	单例模式 工厂模式	Junit 单元测试	策略模式 数据访问对 象模式	Swing 多线程	观察者模式 模板模式
分数	4	6	4	6	6	14 (6+8)
提交内容	UML类图、 代码	UML类图、 代码	测试报告、 代码	UML类图、 代码	代码	项目代码、实 验报告、展示 视频

实验课程共16个学时，6个实验项目，总成绩为40分。

CONTENTS

目录

01 实验目的

02 实验任务

03 实验步骤

04 作业提交

小调查



请选择：你不愿意做代码重构的原因

- ☐ A . 纯粹因为懒
- ☐ B . 担心重构引发Bug
- ☐ C . 我的程序很完美

实验目的

- 了解单元测试的定义及其重要性；
- 掌握JUnit5的常见用法。



实验任务

1. 用JUnit5对StrassenMatrixMultiplication类的所有方法进行单元测试，并使用Jacoco统计代码覆盖率，**要求行 (lines) 覆盖率达到100%**。
2. 对**英雄机类**进行单元测试，要求至少选择3个方法（**包含其父类方法**）作为测试对象，设计测试用例并完成单元测试（不要求代码覆盖率）。
3. 重构代码，完成本次迭代开发的目标：

① 添加**超级精英敌机**，实现**散射**弹道 **(需求变更)**



② 添加**Boss敌机**，实现**环射**弹道



JUnit与单元测试

➤ 单元测试

- 是指对软件中的**最小可测试单元**进行检查和验证;
- 自动化测试, 一般由程序员自己编写;
- 提升软件质量, 增加重构自信。

➤ JUnit

- 一个Java 语言的单元测试框架;
- 促进了测试驱动开发的发展;
- 大部分的Java IDE都集成了JUnit作为单元测试工具;
- 官方文档: [JUnit 5 User Guide](#)



实验步骤

假如，有一个实现简易计算器功能的类Calculator，有加、减、乘、除功能。如何用JUnit5对它进行单元测试？



测它→

```
public class Calculator {  
    public int add(int x, int y) { //加法  
        return x + y;  
    }  
  
    public int sub(int x, int y) { //减法  
        return x - y;  
    }  
  
    public int mul(int x, int y) { //乘法  
        return x * y;  
    }  
  
    public int div(int x, int y) { //除法  
        return x / y;  
    }  
  
    public int div2(int x, int y) { //除法 做了异常判断  
        try {  
            int z = x / y;  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
        return x / y;  
    }  
  
    public void unCompleted(int x, int y) { //未完成的模块:例如平方、开方等等  
        //TODO  
    }  
}
```

实验步骤：如何用JUnit5进行单元测试

① 创建测试类文件夹

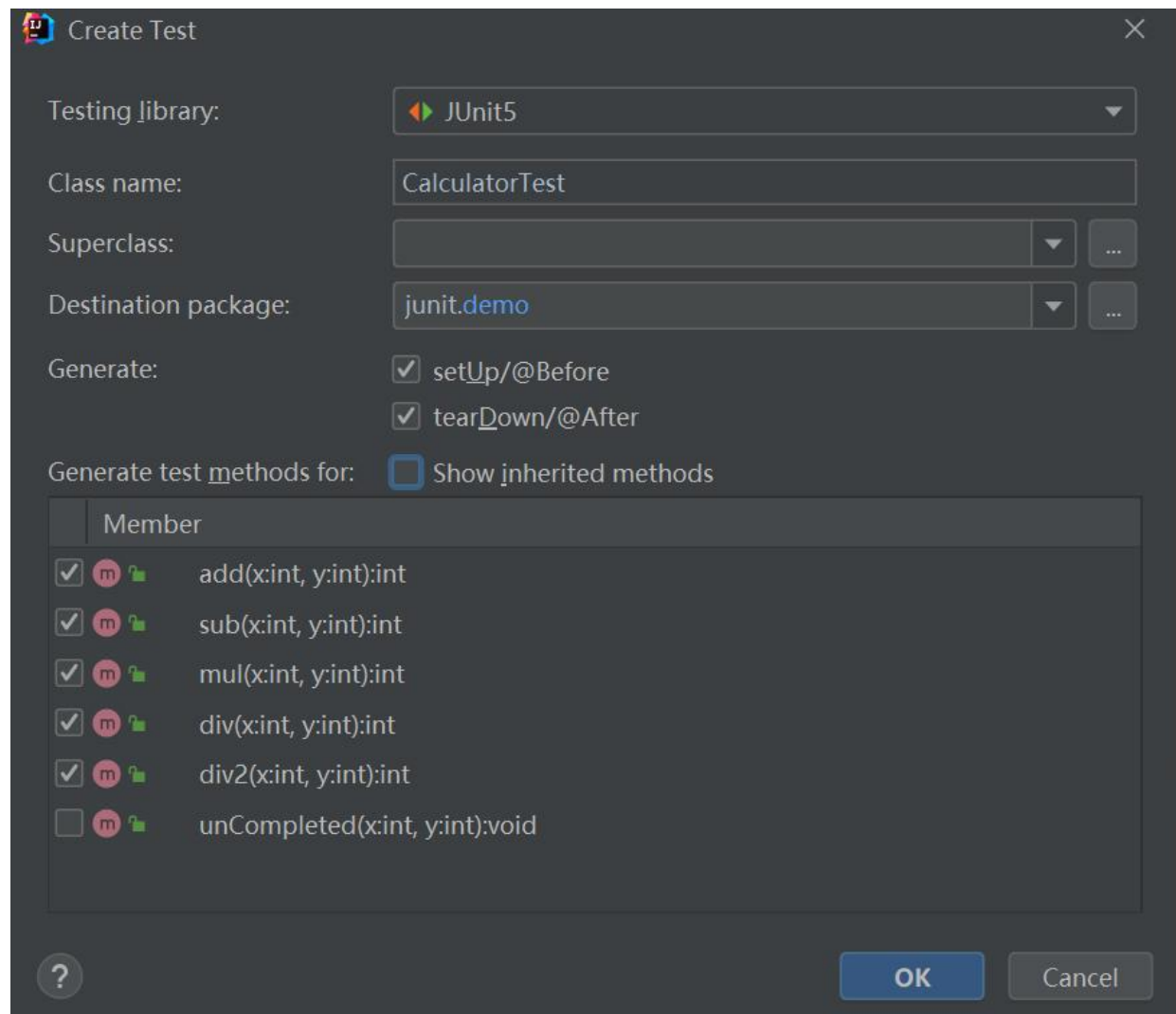
新建test文件夹，右键选择 Mark Directory as → **Test Sources Root**。

② 创建JUnit单元测试类

在待测试的类中，按下快捷键ctrl + shift + T，选择**Create New Test**。

③ 勾选需要测试的方法

在Member中勾选Calculator 中需要进行单元测试的方法。



实验步骤：如何用JUnit5进行单元测试

④ 确认单元测试类生成

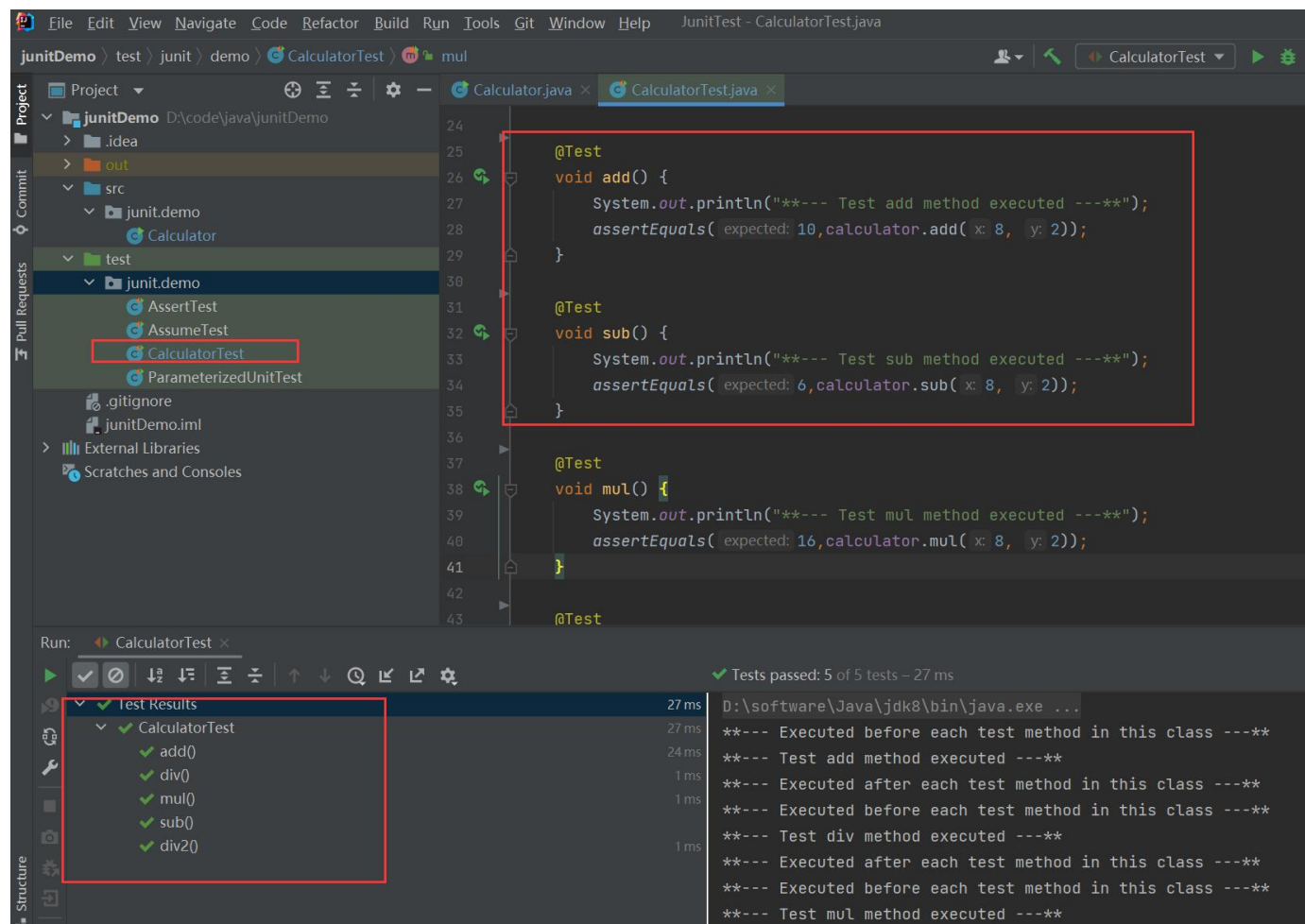
查看test目录下生成的单元测试类
CalculatorTest。

⑤ 编写单元测试代码

设计测试用例，编写单元测试代码，
修改详见指导书4.2节。

⑥ 运行单元测试代码

右键CalculatorTest类，选择 **Run CalculatorTest**，即可运行。



实验步骤：JUnit5 的常见用法

(1) JUnit5注解 (Annotations)

Annotation	Description
@Test	Denotes a test method
@DisplayName	Declares a custom display name for the test class or test method
@BeforeEach	Denotes that the annotated method should be executed before each test method
@AfterEach	Denotes that the annotated method should be executed after each test method
@BeforeAll	Denotes that the annotated method should be executed before all test methods
@AfterAll	Denotes that the annotated method should be executed after all test methods
@Disable	Used to disable a test class or test method
@Nested	Denotes that the annotated class is a nested, non-static test class
@Tag	Declare tags for filtering tests
@ExtendWith	Register custom extensions

实验步骤：JUnit5 的常见用法

```
class CalculatorTest {  
    private Calculator calculator;  
  
    ...  
  
    @BeforeEach  
    void setUp() {  
        calculator = new Calculator();  
    }  
  
    @AfterEach  
    void tearDown() {  
        calculator = null;  
    }  
  
    @DisplayName("Test add method")  
    @Test  
    void add() {  
        assertEquals(10, calculator.add(8, 2));  
    }  
  
    @Test  
    @Disabled("implementation pending")  
    void div2() {}  
  
    ...  
}
```

The screenshot shows an IDE with the following components:

- Project Explorer:** Shows the project structure with 'junitDemo' containing 'src' (Calculator) and 'test' (CalculatorTest).
- Editor:** Displays the code for 'CalculatorTest.java', showing methods 'mul()' and 'div()' with annotations like '@DisplayName', '@Test', and '@Disabled'.
- Run:** Shows the execution of 'CalculatorTest'.
- Test Results:** A table showing the results of the tests:

Test Results	Duration
CalculatorTest	20 ms
Test add method	16 ms
Test div method	1 ms
Test mul method	2 ms
Test sub method	1 ms
div2()	
- Output:** A log of the test execution, showing the sequence of events: 'Executed once before all test methods in this class', 'Executed before each test method in this class', 'Test add method executed', 'Executed after each test method in this class', 'Test div method executed', 'Test mul method executed', 'Test sub method executed', and 'Executed once after all test methods in this class'. The output also shows the 'implementation pending' message for the 'div2()' method.

实验步骤：JUnit5 的常见用法

(2) JUnit5断言 (Assertions)

必须使用断言将每个测试方法的条件评估为true，以便测试可以继续执行。

Assertion	Description
<code>assertEquals(expected, actual)</code>	Fails when expected does not equal actual
<code>assertFalse(expression)</code>	Fails when expression is not false
<code>assertNull(actual)</code>	Fails when actual is not null
<code>assertNotNull(actual)</code>	Fails when actual is null
<code>assertAll()</code>	Group many assertions and every assertion is executed even if one or more of them fails
<code>assertTrue(expression)</code>	Fails if expression is not true
<code>assertThrows()</code>	Class to be tested is expected to throw an exception

实验步骤: JUnit5 的常见用法

```
public class AssertTest {  
    @Test  
    void testAssertEqual() {  
        assertEquals("ABC", "ABC");  
        assertEquals(2 + 2, 4);  
    }  
    @Test  
    void testAssertFalse() {  
        assertFalse("FirstName".length() == 2);  
        assertFalse(10 < 20, "assertion message:  
            test assert false");  
    }  
    @Test  
    void testAssertNull() {  
        String str1 = null;  
        String str2 = "abc";  
        assertNull(str1);  
        assertNotNull(str2);  
    }  
    ...  
}
```

The screenshot displays the IDE interface with the 'Test Results' window open. It shows the following details:

- Test Results:** A list of tests with their durations. 'testAssertFalse()' is marked as failed (3 ms).
- Error Message:** `org.opentest4j.AssertionFailedError: assertion message: test assert false ==>`
- Expected vs Actual:** `Expected :false` and `Actual :true`.
- Stack Trace:** The failure occurred at `AssertTest.java:19` within the `testAssertFalse` method.
- Detailed Comparison:** `expected: <xyz> but was: <zzz>`, `Comparison Failure:`, `Expected :xyz`, `Actual :zzz`.

实验步骤：JUnit5 的常见用法

(3) JUnit5假设 (Assumptions)

仅在满足指定条件时执行测试，否则测试将中止。

Assumptions	Description
assumeTrue	Execute the body of lamda when the positive condition hold else test will be skipped
assumeFalse	Execute the body of lamda when the negative condition hold else test will be skipped
assumingThat	Portion of the test method will execute if an assumption holds true and everything after the lambda will execute irrespective of the assumption in assumingThat() holds

实验步骤: JUnit5 的常见用法

```
public class AssumeTest {  
    @Test  
    void testAssumeFalse() {  
        assumeFalse(false);  
        System.out.println("This will be  
                           implemented.");  
        assertEquals("Hello", "Hello2");  
    }  
  
    @Test  
    @DisplayName("executes only on Saturday")  
    public void testAssumeTrueSaturday() {  
        LocalDateTime dt = LocalDateTime.now();  
        assumeTrue(dt.getDayOfWeek().getValue()  
                  == 6);  
        System.out.println("further code will  
                           execute only if above assumption holds  
                           true");  
    }  
    ...  
}
```

```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help JUnit5 - AssumeTest.java  
junitDemo > test > junit > demo > AssumeTest > testAssumeTrueSaturday  
Project  
  Project  
  > junitDemo D:\code\java\junitDemo  
  > .idea  
  > out  
  > src  
  > junit.demo  
  > Calculator  
  > test  
  > junit.demo  
  > AssertTest  
  > AssumeTest  
  > CalculatorTest  
  > ParameterizedUnitTest  
  .gitignore  
  junitDemo.iml  
  External Libraries  
  Scratches and Consoles  
Run: AssumeTest  
Test Results  
  AssumeTest  
    testAssumeTrue() 34 ms  
    testAssumeFalse() 18 ms  
    test executes only on Saturday 3 ms  
Tests failed: 1, ignored: 2 of 3 tests - 34 ms  
This will be implemented.  
org.opentest4j.AssertionFailedError:  
Expected :Hello  
Actual :Hello2  
<Click to see difference>
```

实验步骤：JUnit5 的常见用法

(4) JUnit5测试异常 (Test Exception)

在某些情况下，期望方法在特定条件下引发异常。如果给定方法未引发指定的异常，则`assertThrows`将使测试失败。

```
public static <T extends Throwable> T assertThrows(Class<T>  
expectedType, Executable executable)
```

它断言所提供的executable的执行将引发expectedType的异常并返回该异常。

实验步骤：JUnit5 的常见用法

```
class CalculatorTest {  
    private Calculator calculator;  
    ...  
    @Test  
    @DisplayName("Test div2 method  
                with expected exception")  
    void div2() {  
        Exception exception = assertThrows(  
            ArithmeticException.class,  
            () -> calculator.div2(2, 0));  
        assertEquals("/ by zero",  
            exception.getMessage());  
        assertTrue(exception.getMessage()  
            .contains("zero"));  
    }  
}
```

The screenshot displays an IDE environment with the following components:

- Project Explorer:** Shows the project structure with folders like 'src' and 'test', and files like 'CalculatorTest.java'.
- Source Editor:** Displays the code for 'CalculatorTest.java'. Key annotations include:
 - `@Test` and `@DisplayName("Test div2 method with expected exception")` for the `div2()` method.
 - `assertThrows(ArithmeticException.class, () -> calculator.div2(2, 0))` to expect an exception.
 - `assertEquals("/ by zero", exception.getMessage())` to verify the exception message.
 - `assertTrue(exception.getMessage().contains("zero"))` to verify the message contains "zero".
 - `@AfterAll` for a static `afterAll()` method.
- Test Results:** Shows a list of test results. The test 'Test div2 method with expected exception' is marked as passed (green checkmark).
- Console:** Displays the output of the test execution, including the assertion of the `ArithmeticException` and the final exit code 0.

实验步骤：JUnit5 的常见用法

(5) JUnit5参数测试 (Parameterized Tests)

@ParameterizedTest 作为参数化测试的必要注解，替代了 @Test 注解。任何一个参数化测试方法都需要标记上该注解。

📌 基本数据源测试：@ValueSource

@ValueSource 是 JUnit 5 提供的最简单的数据参数源，支持 Java 的八大基本类型、字符串和Class，使用时赋值给注解上对应类型属性，以数组方式传递。

📌 CSV 数据源测试：@CsvSource

通过 @CsvSource 可以注入指定 CSV 格式 (comma-separated-values) 的一组数据，用每个逗号分隔的值来匹配一个测试方法对应的参数。

实验步骤: JUnit5 的常见用法

```
public class ParameterizedUnitTest {
```

```
    @ParameterizedTest
```

```
    @DisplayName("Test value source1")
```

```
    @ValueSource(ints = {2, 4, 8})
```

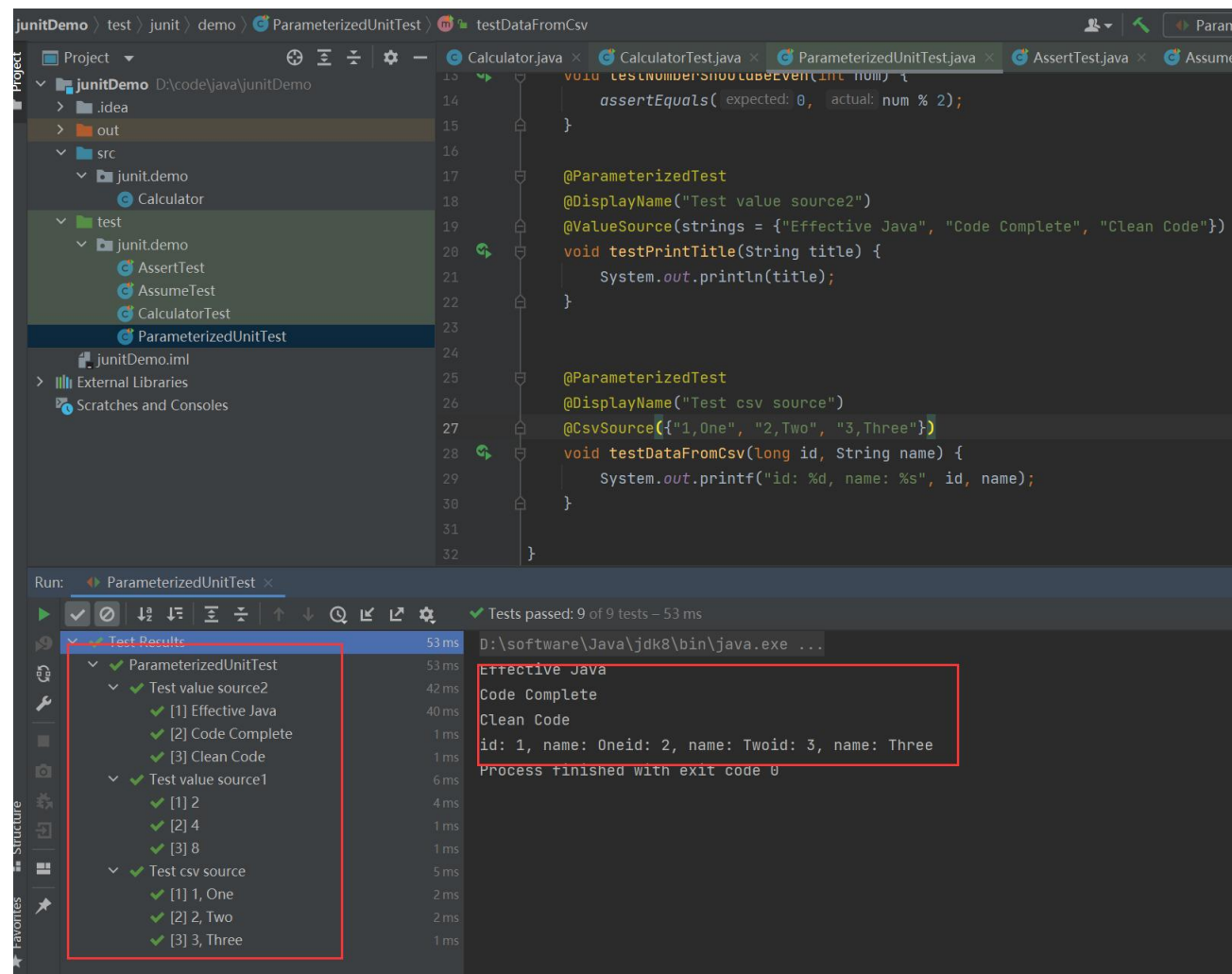
```
    void testNumberShouldBeEven(int num) {  
        assertEquals(0, num % 2);  
    }  
    ...
```

```
    @ParameterizedTest
```

```
    @DisplayName("Test csv source")
```

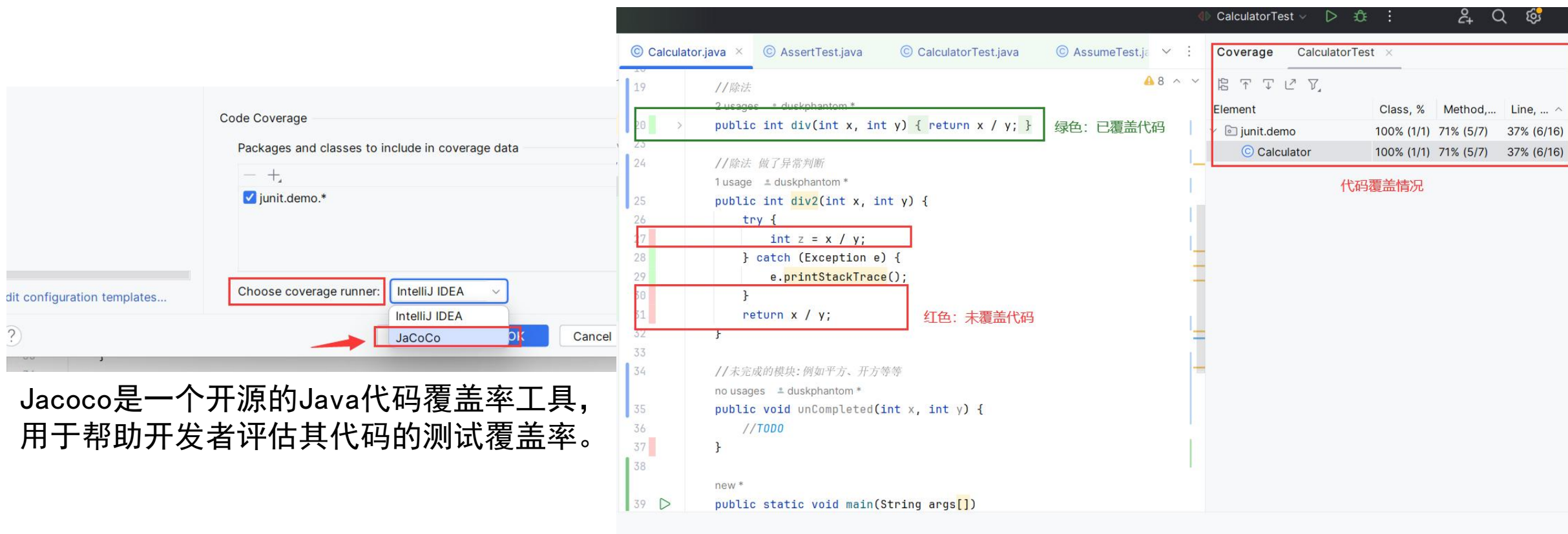
```
    @CsvSource({"1,One", "2,Two", "3,Three"})
```

```
    void testDataFromCsv(long id, String name) {  
        System.out.printf("id: %d, name: %s", id,  
            name);  
    }  
}
```



实验步骤：代码覆盖率统计

代码覆盖率 (Code Coverage) 是指单元测试运行时覆盖到的代码量，通常以行数、分支或方法为单位来度量。



The screenshot illustrates the process of setting up and viewing code coverage in IntelliJ IDEA. On the left, the 'Code Coverage' configuration window is shown, where 'junit.demo.*' is selected under 'Packages and classes to include in coverage data'. The 'Choose coverage runner' dropdown is set to 'JaCoCo'. On the right, the IDE editor displays the 'CalculatorTest.java' file. The code is color-coded: green lines indicate covered code (e.g., line 20: `public int div(int x, int y) { return x / y; }`), and red lines indicate uncovered code (e.g., lines 27-31: `try { int z = x / y; } catch (Exception e) { e.printStackTrace(); } return x / y;`). A red box highlights the uncovered code section with the label '红色：未覆盖代码'. On the far right, the 'Coverage' tool window shows a table summarizing the coverage for the 'CalculatorTest' class.

Element	Class, %	Method, ...	Line, ...
junit.demo	100% (1/1)	71% (5/7)	37% (6/16)
Calculator	100% (1/1)	71% (5/7)	37% (6/16)

代码覆盖情况

Jacoco是一个开源的Java代码覆盖率工具，用于帮助开发者评估其代码的测试覆盖率。

本次迭代开发的目标

✓ 采用**工厂模式**添加**超级精英**敌机和**Boss**敌机;

	超级精英敌机	Boss敌机
出现	每隔一定周期 随机 产生	分数达到设定 阈值 ，可多次出现
移动	向屏幕下方左右移动	悬浮于界面上方左右移动
火力	散射 弹道 同时发射3颗子弹，呈扇形	环射 弹道 同时发射20颗子弹，呈环形
坠毁	随机掉落<=1个道具	随机掉落<=3个道具
		

作业提交

- 提交内容

- ① 项目代码（包含单元测试代码，压缩成zip包）；
- ② 单元测试报告，包括测试用例描述及JUnit结果截图。

- 截止时间

实验课后一周内提交至HITsz Grader 作业提交平台，具体截止日期参考平台发布。登录网址：：<http://grader.tery.top:8000/#/login>

注意：上传后请自行下载确认是否提交成功。



同学们，请开始实验吧！

THANK YOU