

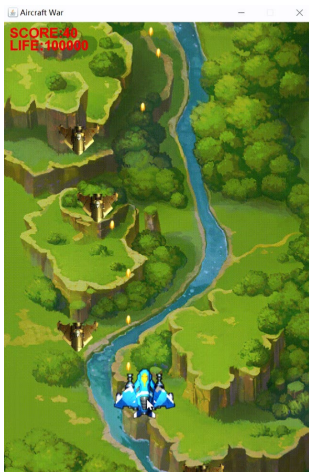


# 实验三：JUnit单元测试

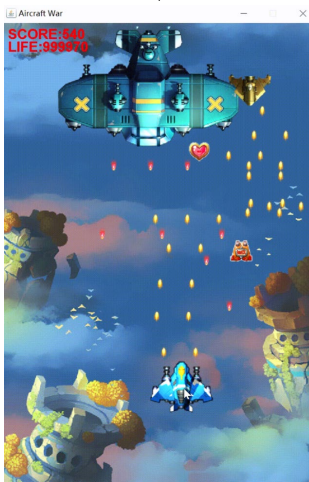
实验与创新实践教育中心 • 计算机与数据技术实验教学部

# 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**。

# 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。



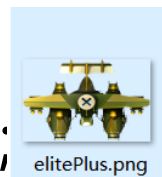
# 实验任务

## 1. 用JUnit5进行单元测试：

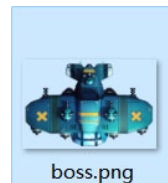
- ① 完成对StrassenMatrixMultiplication类所有方法的单元测试，并使用Jacoco统计代码覆盖率，要求行（lines）覆盖率达到100%；
- ② 为飞机大战游戏设计测试用例，选择英雄机类的3个方法作为单元测试的对象。

## 2. 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；



- 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，有一个实现矩阵乘法的类  
StrassenMatrixMultiplication，如  
何用JUnit5对它进行单元测试？



```
public class StrassenMatrixMultiplication {  
  
    // Function to multiply matrices  
    8 usages  
    public int[][] multiply(int[][] A, int[][] B) {...}  
  
    // Function to subtract two matrices  
    7 usages  
    public int[][] sub(int[][] A, int[][] B) {...}  
  
    // Function to add two matrices  
    13 usages  
    public int[][] add(int[][] A, int[][] B) {...}  
  
    // Function to split parent matrix into child matrices  
    12 usages  
    public void split(int[][] P, int[][] C, int iB, int jB) {...}  
  
    // Function to join child matrices into (to) parent matrix  
    8 usages  
    public void join(int[][] C, int[][] P, int iB, int jB) {...}  
}
```

# 实验步骤：如何用JUnit5进行单元测试

## ① 创建测试类文件夹

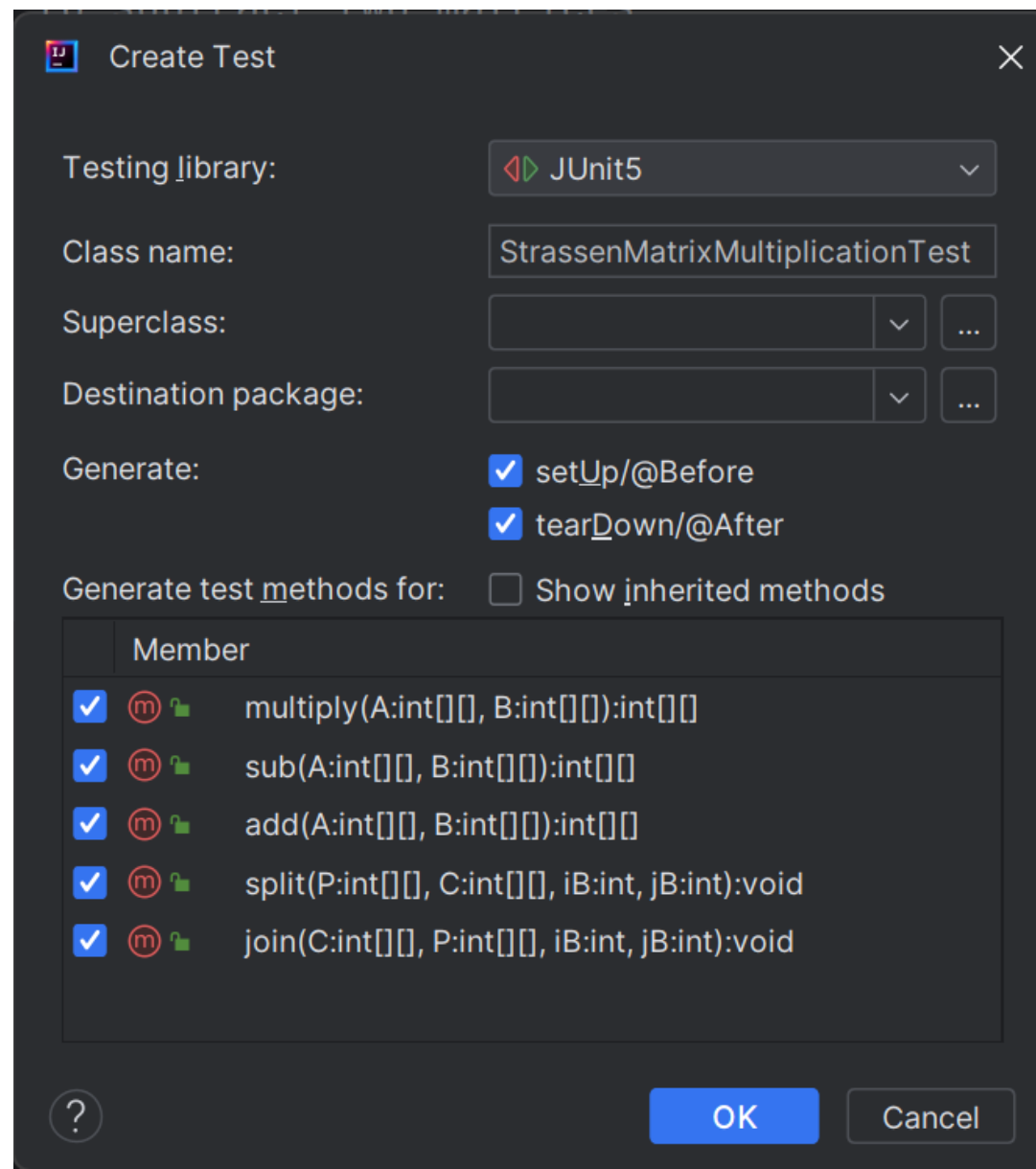
新建test文件夹，右键选择 Mark Directory as → **Test Sources Root**。

## 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

**StrassenMatrixMultiplicationTest。**

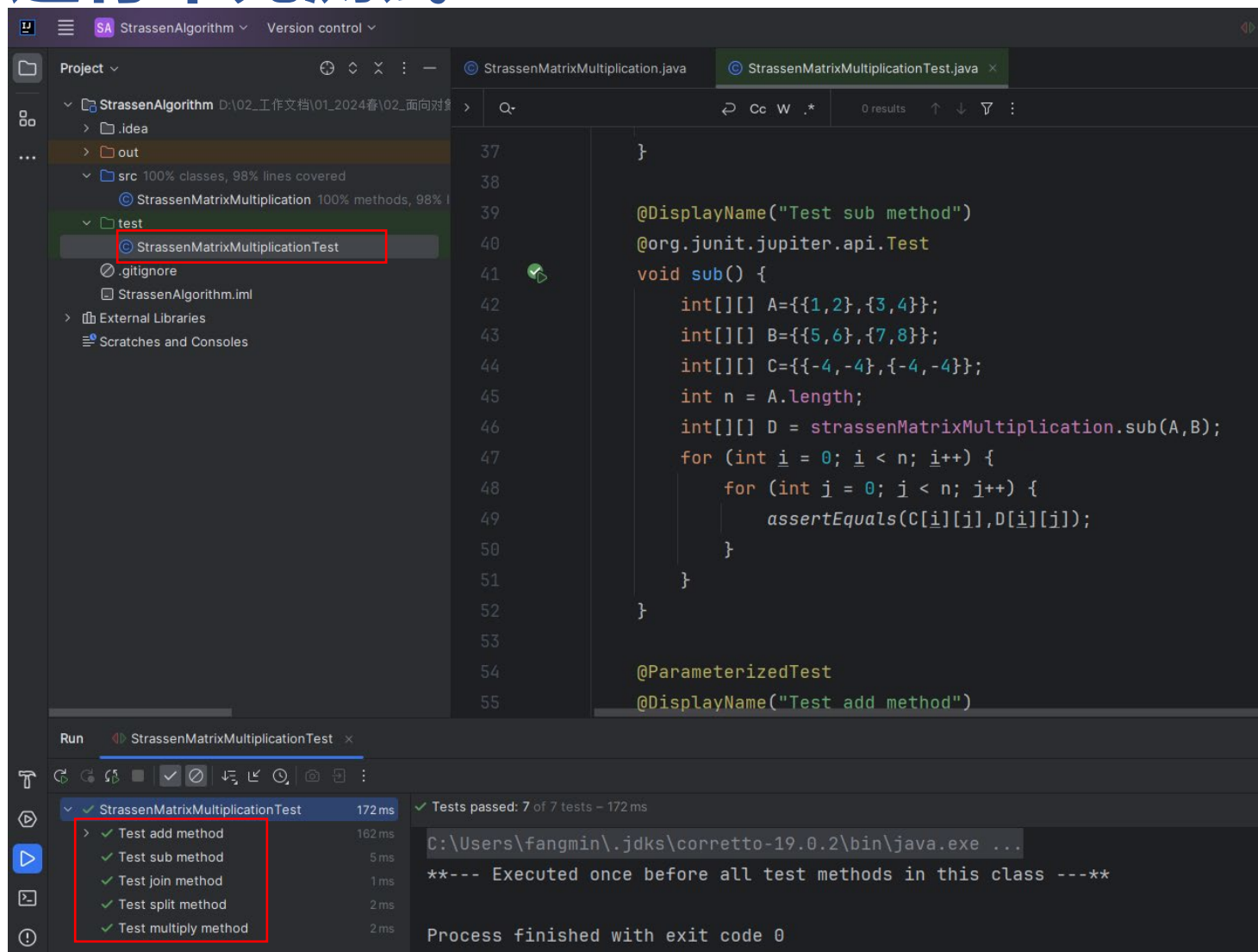
## ⑤ 编写单元测试代码

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StrassenMatrixMultiplicationTest类，选择 **Run**

**StrassenMatrixMultiplicationTest**，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

The screenshot displays an IDE interface with the following components:

- Project Explorer:** Shows the project structure with folders like `.idea`, `out`, `src`, and `test`. The `test` folder contains `StrassenMatrixMultiplicationTest`.
- Code Editor:** Displays the `StrassenMatrixMultiplicationTest.java` file. The code includes:
  - Class declaration: `class StrassenMatrixMultiplicationTest {`
  - Usage count: `13 usages`
  - Private field: `private StrassenMatrixMultiplication strassenMatrixMultiplication;`
  - Annotations and methods:
    - `@BeforeAll` annotation pointing to `static void beforeAll()` (indicated by a red arrow).
    - `@BeforeEach` annotation pointing to `void setUp()` (indicated by a red arrow).
    - `@AfterEach` annotation pointing to `void tearDown()` (indicated by a red arrow).
    - `@DisplayName("Test multiply method")`
    - `@org.junit.jupiter.api.Test`
- Run Console:** Shows the execution of `StrassenMatrixMultiplicationTest`. The output includes:
  - Execution time: `107 ms`
  - Test results: `Tests passed: 4, ignored: 1 of 5 tests`
  - Test methods and their durations:
    - `Test add method`: 91 ms
    - `Test sub method`: 5 ms
    - `Test join method`: (skipped)
    - `Test split method`: 5 ms
    - `Test multiply method`: 6 ms
  - Output text: `**--- Executed once before all test methods in this class ---**`
  - Warning: `Do not test the test method temporarily`



# 实验步骤：JUnit5 的常见用法

The screenshot displays an IDE with a project named 'StrassenAlgorithm'. The 'test' directory contains the file 'StrassenMatrixMultiplicationTest'. The code in this file includes annotations like `@DisplayName` and `@org.junit.jupiter.api.Test`, which are highlighted with red arrows. The `sub()` method is defined, performing matrix multiplication and using `assertEquals` for assertions. Below the code editor, the 'Run' tab shows the test results for 'StrassenMatrixMultiplicationTest', indicating that all 7 tests passed. A red arrow points to the 'Test sub method' entry in the results list. The bottom status bar shows the command used to run the tests: `C:\Users\fangmin\jdk\corretto-19.0.2\bin\java.exe ...`.

```
37     }
38
39     @DisplayName("Test sub method")
40     @org.junit.jupiter.api.Test
41     void sub() {
42         int[][] A={{1,2},{3,4}};
43         int[][] B={{5,6},{7,8}};
44         int[][] C={{-4,-4},{-4,-4}};
45         int n = A.length;
46         int[][] D = strassenMatrixMultiplication.sub(A,B);
47         for (int i = 0; i < n; i++) {
48             for (int j = 0; j < n; j++) {
49                 assertEquals(C[i][j],D[i][j]);
50             }
51         }
52     }
53
54     @ParameterizedTest
55     @DisplayName("Test add method")
```

Run StrassenMatrixMultiplicationTest

✓ StrassenMatrixMultiplicationTest 172 ms

- ✓ Test add method 162 ms
- ✓ Test sub method 5 ms
- ✓ Test join method 1 ms
- ✓ Test split method 2 ms
- ✓ Test multiply method 2 ms

✓ Tests passed: 7 of 7 tests - 172 ms

C:\Users\fangmin\jdk\corretto-19.0.2\bin\java.exe ...

\*\*\* Executed once before all test methods in this class \*\*\*

Process finished with exit code 0

# 实验步骤：JUnit5 的常见用法

The screenshot displays the IntelliJ IDEA IDE with a project named 'StrassenAlgorithm'. The left sidebar shows the project structure, including a 'test' directory containing 'StrassenMatrixMultiplicationTest'. The main editor shows the source code for 'StrassenMatrixMultiplicationTest.java'. A red arrow points to the '@Disabled("Do not test the sub method temporarily.")' annotation on line 40. The code includes a 'multiply()' method and a 'sub()' method. The 'sub()' method contains a nested loop that calculates the product of two matrices and asserts the result against a pre-calculated value. The bottom panel shows the 'Run' output, indicating that 6 tests passed and 1 test was ignored. A red arrow points to the 'Test sub method' entry in the test results list, which is marked as ignored. The output text shows the command used to run the tests and the message 'Do not test the sub method temporarily.'.

```
28 void multiply() {...}
38
39 @DisplayName("Test sub method")
40 @Disabled("Do not test the sub method temporarily.")
41 @org.junit.jupiter.api.Test
42 void sub() {
43     int[][] A={{1,2},{3,4}};
44     int[][] B={{5,6},{7,8}};
45     int[][] C={{-4,-4},{-4,-4}};
46     int n = A.length;
47     int[][] D = strassenMatrixMultiplication.sub(A,B);
48     for (int i = 0; i < n; i++) {
49         for (int j = 0; j < n; j++) {
50             assertEquals(C[i][j],D[i][j]);
51         }
52     }
53 }
```

Run StrassenMatrixMultiplicationTest

Tests passed: 6, ignored: 1 of 7 tests - 189 ms

C:\Users\fangmin\jdk\corretto-19.0.2\bin\java.exe ...

\*\*\* Executed once before all test methods in this class \*\*\*

Do not test the sub method temporarily.

Process finished with exit code 0

# 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

```
@DisplayName("Test sub method")
@org.junit.jupiter.api.Test
void sub() {
    int[][] A={{1,2},{3,4}};
    int[][] B={{5,6},{7,8}};
    int[][] C={{-4,-4},{-4,-4}};
    int n = A.length;
    int[][] D = strassenMatrixMultiplication.sub(A,B);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
             assertEquals(C[i][j],D[i][j]);
        }
    }
}
```

# 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

7 usages

```
void CompareMatrix(int[][] C1, int[][] C2)
{
    int n = C1.length;
    → assertTrue(assumption: n>0);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            assertEquals(C1[i][j], C2[i][j]);
        }
    }
}
```

# 实验步骤：JUnit5 的常见用法

## 假设 VS 断言

7 usages

```
void CompareMatrix(int[][] C1, int[][] C2)
{
    int n = C1.length;
    assumeTrue(assumption: n>0);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            assertEquals(C1[i][j], C2[i][j]);
        }
    }
}
```

### 1.假设 (Assumptions) :

- ① 假设是用于在测试执行过程中进行前提条件的验证, 并根据验证结果来决定是否继续执行测试的机制。
- ② 如果假设条件不满足, 假设会导致测试被标记为“忽略” (Skipped) 。
- ③ 假设通常用于验证测试环境的一些前提条件, 如特定的系统属性、环境变量、配置等, 以确保测试的可靠性和有效性。

### 2.断言 (Assertions) :

- ① 断言是用于在测试中验证实际结果与预期结果是否一致的机制。
- ② 如果断言失败, 测试将被标记为“失败” (Failed) , 并显示具体的失败信息, 帮助开发者定位问题。
- ③ 断言通常用于验证代码的行为是否符合预期, 包括返回值、异常抛出、状态变化等方面。

## 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

## 被测方法：

```
public int div2(int x, int y)
{ //除法 做了异常判断

    try {

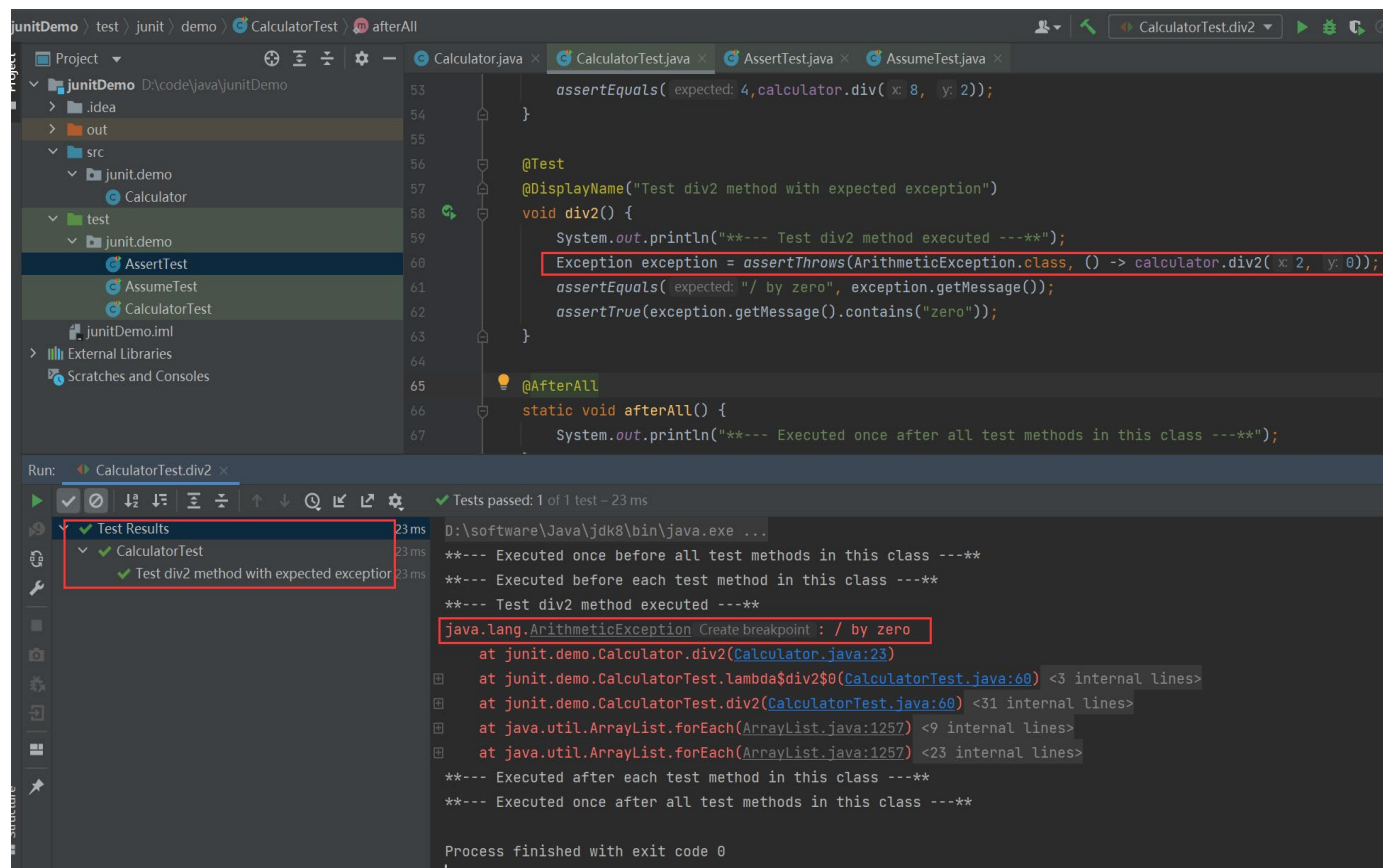
        int z = x / y;

    } catch (Exception e) {

        e.printStackTrace();

    }

    return x / y;
}
```



# 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

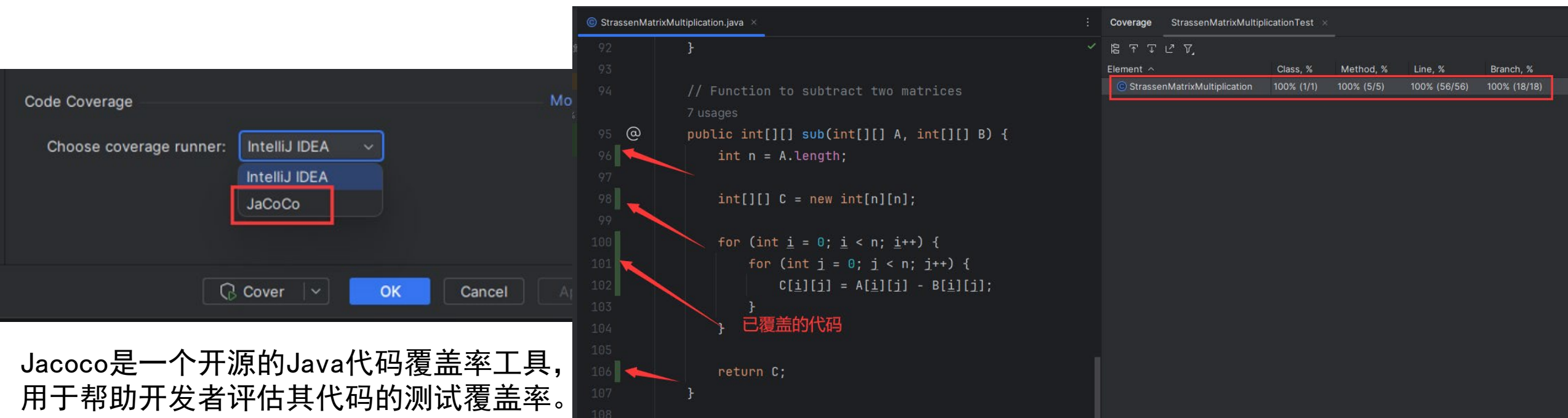
```
@ParameterizedTest
    @DisplayName("Test add method")
@CsvSource({"1,2,3,4,5,6,7,8,6,8,10,12",
            "2,4,3,5,4,6,3,2,6,10,6,7",
            "11,2,6,4,9,16,7,3,20,18,13,7"})
    void add(int A00,int A01, int A10, int A11, int B00,int B01, int B10, int B11,
            int C00,int C01, int C10, int C11) {
        int[][] A={{A00,A01},{A10,A11}};
        int[][] B={{B00,B01},{B10,B11}};
        int[][] C={{C00,C01},{C10,C11}};
        int n = A.length;
        int[][] D = strassenMatrixMultiplication.add(A,B);
        CompareMatrix(C,D);
    }
```

✓ StrassenMatrixMultiplicationTest	172 ms
✓ Test add method	162 ms
✓ [1] 1, 2, 3, 4, 5, 6, 7, 8, 6, 8, 10, 11	151 ms
✓ [2] 2, 4, 3, 5, 4, 6, 3, 2, 6, 10, 6, 7	5 ms
✓ [3] 11, 2, 6, 4, 9, 16, 7, 3, 20, 18, 13, 7	6 ms



## 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 displays the IntelliJ IDEA interface with the Code Coverage tool integrated. On the left, the 'Code Coverage' dialog box is open, showing 'IntelliJ IDEA' as the selected coverage runner. The 'JaCoCo' option is highlighted with a red box. Below the dialog, a text box explains that Jacoco is an open-source Java code coverage tool used for evaluating test coverage.

The main editor shows the source code of `StrassenMatrixMultiplication.java`. The code includes a function to subtract two matrices. Red arrows point to specific lines of code (95, 96, 98, 100, 101, 102, 106) that are highlighted in green, indicating they are covered by tests. A red label '已覆盖的代码' (Covered Code) points to these lines.

On the right, the 'Coverage' tab is active, displaying a table of coverage statistics for the `StrassenMatrixMultiplicationTest` class. The table shows 100% coverage for the class, method, lines, and branches.

Element	Class, %	Method, %	Line, %	Branch, %
StrassenMatrixMultiplication	100% (1/1)	100% (5/5)	100% (56/56)	100% (18/18)

# 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 <b>随机</b> 产生	分数达到设定 <b>阈值</b> ，可多次出现
移动	向屏幕下方 <b>左右</b> 移动	<b>悬浮</b> 于界面上方 <b>左右</b> 移动
火力	<b>散射</b> 弹道 同时发射3颗子弹，呈扇形	<b>环射</b> 弹道 同时发射20颗子弹，呈环形
坠毁	随机掉落 $\leq 1$ 个道具	随机掉落 $\leq 3$ 个道具
		

# 作业提交

- 提交内容

把下面内容压缩成zip包：

- ① 飞机大战项目代码（包含单元测试代码）；
- ② 单元测试报告（使用下发的模板）；
- ③ 矩阵乘法项目代码（包含单元测试代码）。

- 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。

# 作业提交

1. 测试用例（一个方法一个测试用例）				
用例编号	1			
待测试类及方法	HeroAircraft.****			
测试类及方法				
前提条件（如有）				
用例描述	测试步骤	期望结果	实际输出	测试结果

用例编号：唯一标识测试用例的序号，一般是数字或模块名首字母大写+数字序号。

待测试类及方法：该用例所测试的类名和方法名

测试类及方法：相应的测试代码的类名和方法名

前提条件（如有）：执行该测试用例的前提条件，比如碰撞检测，需已创建英雄机和敌机（或道具）。

用例描述：用一句话简单总结该测试用例的用意和目的。

测试步骤：详细完整的操作过程描述。

期望结果：正常情况下的响应结果。

实际结果：程序通过测试

测试结果：通过或失败。

实际结果：程序通过测试  
测试结果：通过或失败。

用例编号	1				
待测试类及方法	HeroAircraft.crash				
测试类及方法	HeroAircraftTest.crash				
前提条件（如有）	创建指定坐标的道具或敌机				
用例描述	测试步骤	期望结果	实际输出	测试结果	
测试两个飞行物体的碰撞检测是否正常。	1. 英雄机初始化，指定坐标（x,y）（具体值） 2. 创建火力道具，指定坐标（x,y）（具体值，可以相撞） 3. 调用英雄机的 crash 方法，传入火力道具 4. 判断是否检测到碰撞	Crash 返回 true	True	Pass	
用例编号	2				
待测试类及方法	HeroAircraftTest.crash				
测试类及方法	HeroAircraftTest.crash				
前提条件（如有）	创建指定坐标的道具或敌机				
用例描述	测试步骤	期望结果	实际输出	测试结果	
测试两个飞行物体的碰撞检测是否正常。	1. 英雄机初始化，指定坐标（x,y）（具体值） 2. 创建火力道具，指定坐标（x,y）（具体值，可以相撞） 3. 调用英雄机的 crash 方法，传入火力道具 4. 判断是否检测到碰撞	Crash 返回 true	True	Pass	

# 作业提交

## 2. JUnit 单元测试结果

请截图 JUnit 测试类（包含多个方法）的运行结果。

```
Project
├── StrassenAlgorithm
│   ├── .idea
│   ├── out
│   ├── src
│   │   ├── StrassenMatrixMultiplication
│   │   └── test
│   │       ├── StrassenMatrixMultiplicationTest
│   │       ├── .gitignore
│   │       └── StrassenAlgorithm.iml
│   └── External Libraries
│       └── Scratches and Consoles
└── StrassenMatrixMultiplicationTest.java
└── StrassenMatrixMultiplicationTest.java

@DisplayName("Test sub method")
@org.junit.jupiter.api.Test
void sub() {
    int[][] A={{1,2},{3,4}};
    int[][] B={{5,6},{7,8}};
    int[][] C={{-4,-4},{-4,-4}};
    int n = A.length;
    int[][] D = strassenMatrixMultiplication.sub(A,B);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            assertEquals(C[i][j],D[i][j]);
        }
    }
}

@ParameterizedTest
@DisplayName("Test add method")

Run StrassenMatrixMultiplicationTest
├── StrassenMatrixMultiplicationTest 172 ms
│   ├── Test add method 162 ms
│   ├── Test sub method 5 ms
│   ├── Test join method 1 ms
│   ├── Test split method 2 ms
│   └── Test multiply method 2 ms
└── Tests passed: 7 of 7 tests - 172 ms
C:\Users\fangmin\.jdk\corretto-19.0.2\bin\java.exe ...
**--- Executed once before all test methods in this class ---**
Process finished with exit code 0
```

参考图



# 同学们，请开始实验吧！

## THANK YOU