



哈爾濱工業大學(深圳)  
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

規格嚴格



功夫到家

# 面向对象的软件构造导论

## 实验三：JUnit单元测试

2023春

哈尔滨工业大学（深圳）

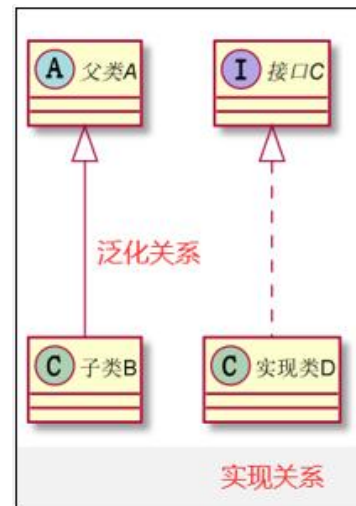


# 实验回顾

## 实验二中存在的几个问题：

### ① UML类图的规范性

- 接口、抽象类、具体类的图标
- 各种关系的图标
- 可见性、静态成员变量和方法



### ② 职责分配的合理性

- **请思考：** 道具掉落、道具生效的职责应该分给哪些类？



# 实验回顾

## 实验二中存在的几个问题：

### ③ 创建细节的封装

- **请思考：**是否真的对使用者隐藏了英雄机、敌机和道具的创建细节？
- 下面2段来自Game的代码，完成的都是初始化英雄机的工作，哪个更好一些？

```
// 初始化英雄机
heroAircraft = HeroAircraft.getHeroAircraft(
    locationX: WINDOW_WIDTH / 2,
    locationY: WINDOW_HEIGHT - ImageManager.HERO_IMAGE.getHeight() ,
    speedX: 0, speedY: 0, hp: 1000);
```

```
// 初始化英雄机
heroAircraft = HeroAircraft.getHeroAircraft();
```

# 实验回顾

## 实验二中存在的几个问题：

### ④ 多态的使用

- 请思考：什么是多态？什么是里氏代换原则？
- 实现“开-闭”原则的关键步骤就是抽象化。

```
public static void main(String[] args) {  
    ShapeFactory shapeFactory;  
    Shape shape;  
  
    //获取 Circle 的对象，并调用它的 draw 方法  
    shapeFactory = new CircleFactory();  
    shape = shapeFactory.createShape();  
    shape.draw();  
  
    //获取 Rectangle 的对象，并调用它的 draw 方法  
    shapeFactory = new RectangleFactory();  
    shape = shapeFactory.createShape();  
    shape.draw();  
  
    //获取 Square 的对象，并调用它的 draw 方法  
    shapeFactory = new SquareFactory();  
    shape = shapeFactory.createShape();  
    shape.draw();  
}
```

定义变量时使用基类

赋值时使用子类的实例化对象

```
MobEnemyFactory mobEnemyFactory  
EliteEnemyFactory eliteEnemyFactory
```



```
List<MobEnemy> mobEnemies;  
List<EliteEnemy> eliteEnemies;
```



```
List<BombSupply> bombSupplies;  
List<FireSupply> fireSupplies;  
List<HpSupply> hpSupplies;
```





# 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 (2+2)	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	模板模式 观察者模式
分数	4	6	4	6	6	14
提交内容	UML类图、 代码	UML类图、 代码	单元测试 代码 测试报告	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;
- 理解编码规范的重要性, 熟悉阿里编码规约插件的使用方法。





# 实验任务

---

1. 为飞机大战系统设计测试用例，并用JUnit5 对代码进行单元测试；
2. 重构代码，添加Boss敌机，实现散射弹道。



# 实验步骤

---

## ➤ 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

## 1 用JUnit5进行单元测试

假如，有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  
        //TODO  
    }  
}
```



# 实验步骤

## 1 用JUnit5进行单元测试

### ① 创建测试类文件夹

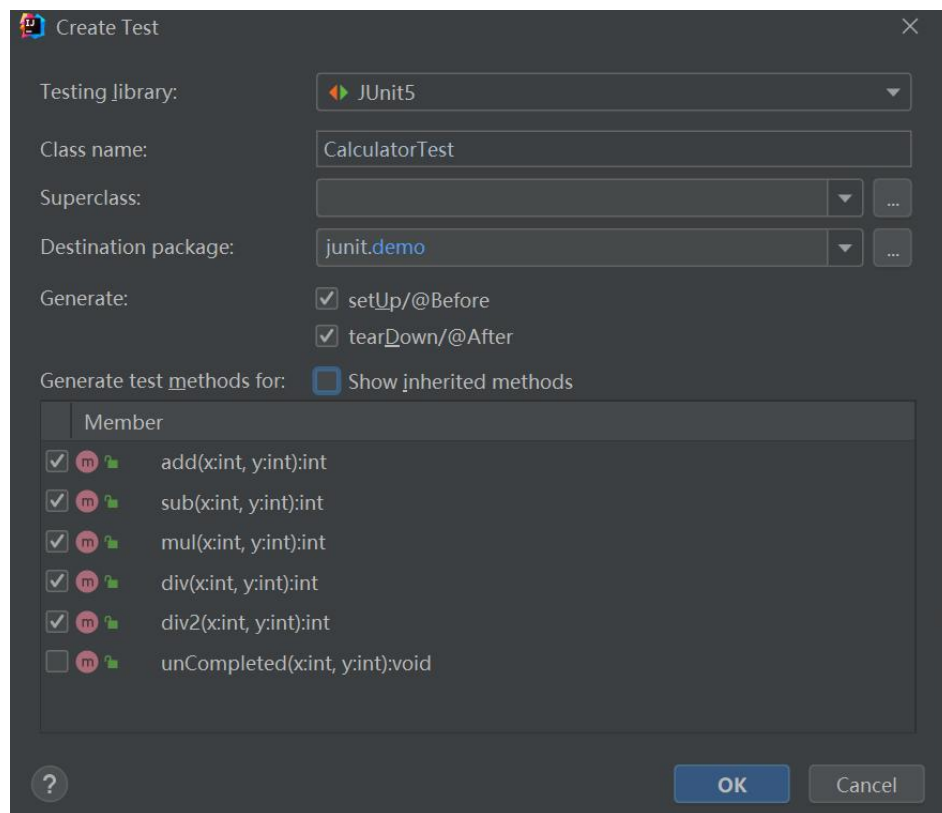
新建test文件夹，右键选择  
Mark Directory as → **Test**  
**Sources Root**。

### 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

## 1 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**，即可运行该单元测试类。

The screenshot displays an IDE with the `CalculatorTest.java` file open. The file contains three test methods: `add()`, `sub()`, and `mul()`. Each method is annotated with `@Test` and uses `assertEquals` to verify the results of the corresponding calculator methods. The `add()` method expects 10, `sub()` expects 6, and `mul()` expects 16. The IDE's output window at the bottom shows the execution results, indicating that all 5 tests passed successfully within 27 ms. The output also includes timing information for each test method and the overall execution time.

```
JUnit5 - CalculatorTest.java
junitDemo > test > junit > demo > CalculatorTest > mul

Project
  junitDemo
    .idea
    out
    src
      junit.demo
        Calculator
    test
      junit.demo
        AssertTest
        AssumeTest
        CalculatorTest
        ParameterizedUnitTest
    .gitignore
    junitDemo.iml
  External Libraries
  Scratches and Consoles

Run: CalculatorTest x
Test Results
  CalculatorTest
    add()
    div()
    mul()
    sub()
    div2()

Tests passed: 5 of 5 tests - 27 ms
D:\software\Java\jdk8\bin\java.exe ...
**** Executed before each test method in this class ****
**** Test add method executed ****
**** Executed after each test method in this class ****
**** Executed before each test method in this class ****
**** Test div method executed ****
**** Executed after each test method in this class ****
**** Executed before each test method in this class ****
**** Test mul method executed ****
```



## 2

## 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

## 2

## 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 folders like `src`, `test`, and `CalculatorTest`.
- Source Editor:** Displays the `CalculatorTest.java` file with the following code:

```
42 @DisplayName("Test mul method")  
43 @Test  
44 void mul() {  
45     System.out.println("*** Test mul method executed ***");  
46     assertEquals( expected: 16, calculator.mul( x: 8, y: 2));  
47 }  
48  
49 @DisplayName("Test div method")  
50 @Test  
51 void div() {  
52     System.out.println("*** Test div method executed ***");  
53     assertEquals( expected: 4, calculator.div( x: 8, y: 2));  
54 }  
55  
56 @Test  
57 @Disabled("implementation pending")
```
- Run Console:** Shows the output of the test run:

```
Tests passed: 4, ignored: 1 of 5 tests - 20 ms  
D:\software\Java\jdk8\bin\java.exe ...  
*** 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 ***  
*** Executed before each test method in this class ***  
*** Test div method executed ***  
*** Executed after each test method in this class ***  
*** Executed before each test method in this class ***  
*** Test mul method executed ***  
*** Executed after each test method in this class ***  
*** Executed before each test method in this class ***  
*** Test sub method executed ***  
*** Executed after each test method in this class ***  
  
implementation pending  
*** Executed once after all test methods in this class ***
```



## 2 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

## 2

## JUnit5 的常见用法

```
public class AssertTest {
```

```
    @Test
```

```
    void testAssertEqual() {  
        assertEquals("ABC", "ABC");  
        assertEquals(2 + 2, 4);  
    }
```

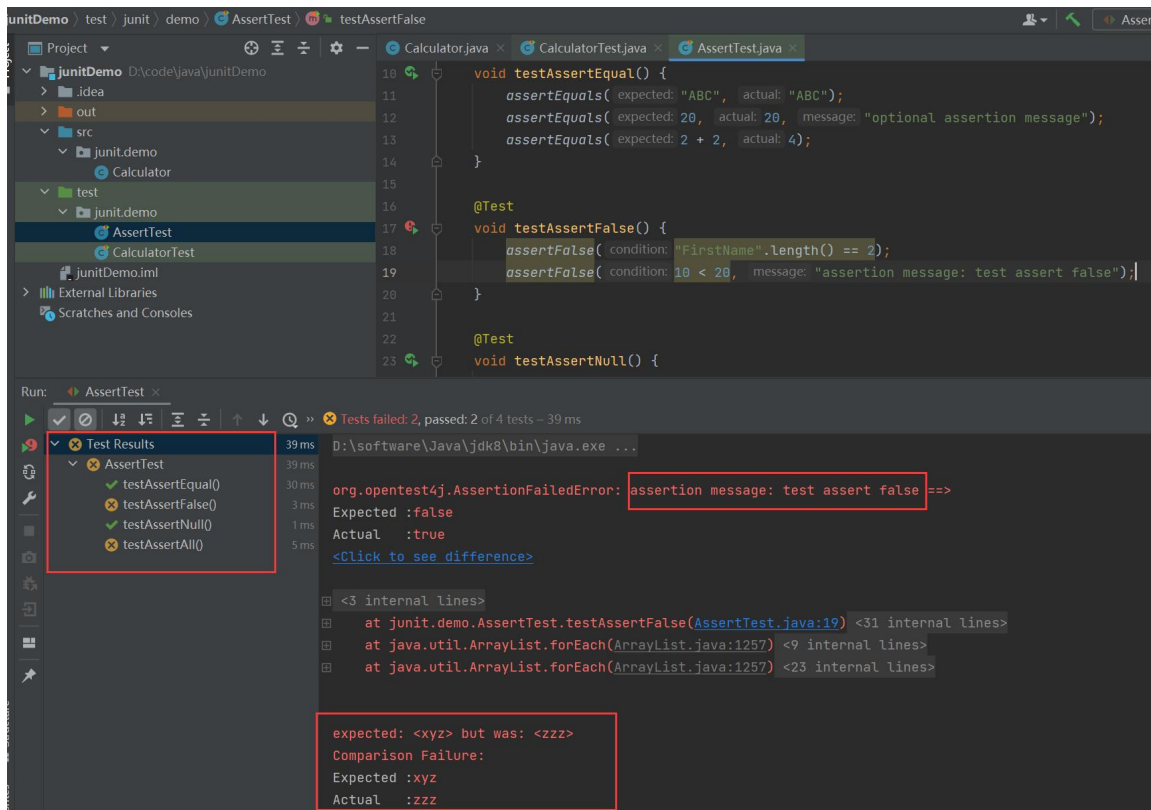
```
    @Test
```

```
    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);  
    }
```

```
...  
}
```





## 2

## 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

## 2

## JUnit5 的常见用法

```
public class AssumeTest {
```

```
    @Test
```

```
    void testAssumeFalse() {  
        assumeFalse(false);  
        System.out.println("This will be  
                           implemented.");  
        assertEquals("Hello", "Hello2");  
    }
```

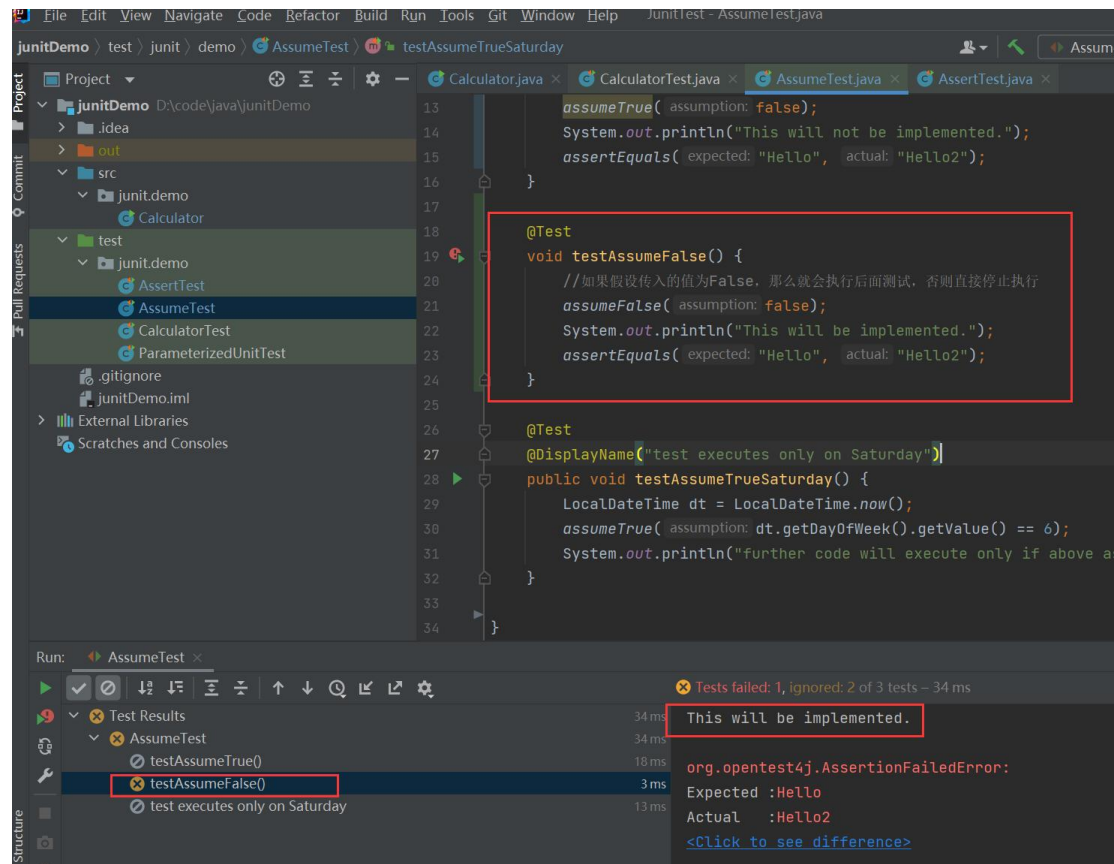
```
    @Test
```

```
    @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");  
    }
```

```
    ...
```

```
}
```





## 2 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

## 2

## 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 with the following components:

- Project Explorer:** Shows the project structure with folders like 'src' and 'test'. The 'test' folder contains 'CalculatorTest'.
- Code Editor:** Shows the source code for 'CalculatorTest.java'. It includes an '@Test' method 'div2()' that uses 'assertThrows' to expect an 'ArithmeticException' and 'assertEquals' to verify the message. An '@AfterAll' method is also present.
- Run Console:** Shows the output of the test execution. It indicates that the test passed. The output includes the exception message: 'java.lang.ArithmeticException: / by zero'.



## 2 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。

## 2

## 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);  
    }  
}
```

The screenshot displays an IDE with a project named 'junitDemo'. The left sidebar shows the project structure with folders for 'src' and 'test'. The 'test' folder contains 'ParameterizedUnitTest'. The main editor shows the source code of 'ParameterizedUnitTest.java', which includes two test methods: 'testNumberShouldBeEven' and 'testDataFromCsv'. The 'testDataFromCsv' method is annotated with '@CsvSource' and uses 'System.out.printf' to print the test data. The bottom panel shows the 'Run' output, indicating that 9 tests passed in 53 ms. A red box highlights the test results for 'Test csv source', showing three successful test cases: '[1] 1, One', '[2] 2, Two', and '[3] 3, Three'. The output also shows the printed values for each test case: 'id: 1, name: One', 'id: 2, name: Two', and 'id: 3, name: Three'.

```
Run: ParameterizedUnitTest x  
Tests passed: 9 of 9 tests - 53 ms  
Test Results  
ParameterizedUnitTest 53ms  
  Test value source2 53ms  
    [1] Effective Java 42ms  
    [2] Code Complete 40ms  
    [3] Clean Code 1ms  
  Test value source1 6ms  
    [1] 2 4ms  
    [2] 4 1ms  
    [3] 8 1ms  
  Test csv source 5ms  
    [1] 1, One 2ms  
    [2] 2, Two 2ms  
    [3] 3, Three 1ms  
Effective Java  
Code Complete  
Clean Code  
id: 1, name: Oneid: 2, name: Twoid: 3, name: Three  
Process finished with exit code 0
```



# 实验步骤

## 3 用JUnit5对飞机大战代码进行单元测试

为飞机大战系统设计测试用例，选择英雄机、敌机、子弹和道具类的方法（包含其父类方法）作为单元测试的对象，用JUnit5进行单元测试。

实验要求：至少选择3个类，每个类至少测试2个方法。







# 实验步骤

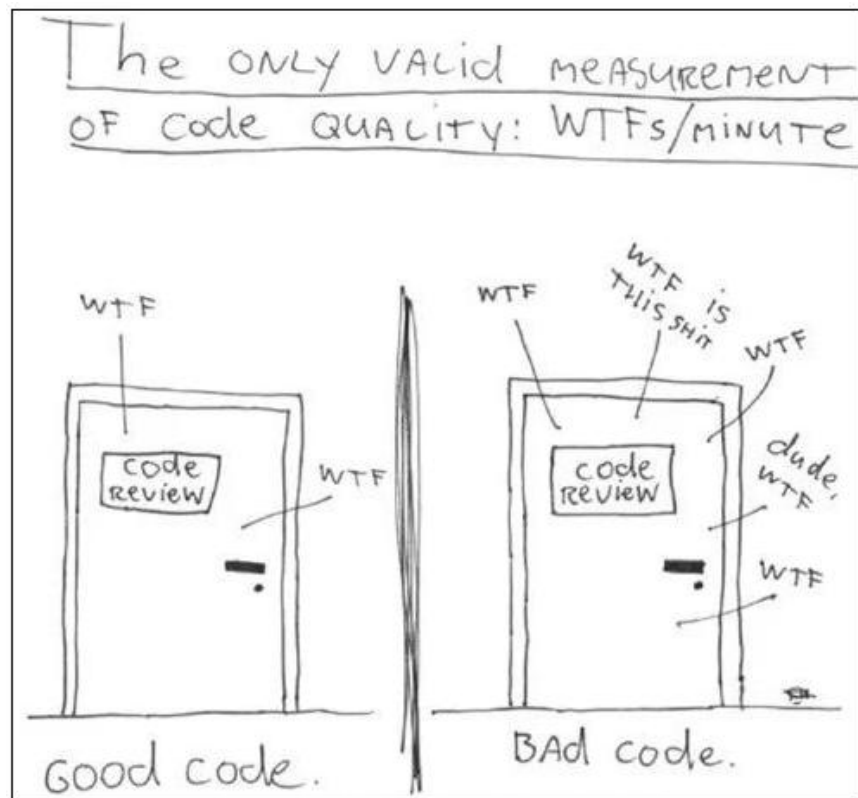
## ➤ 编码规范

- 是程序编码所要遵循的规则，要注意代码的正确性、稳定性、可读性；
- 好的编码规范是提高我们[代码质量](#)的最有效的工具之一。

## ➤ 阿里编码规约插件

(Alibaba Java Coding Guidelines)

- 是对《阿里巴巴 Java 开发规约》的一个延伸；
- 以一个 IDE 的插件存在，可自动对手册中的 Java 不规范问题进行提示。

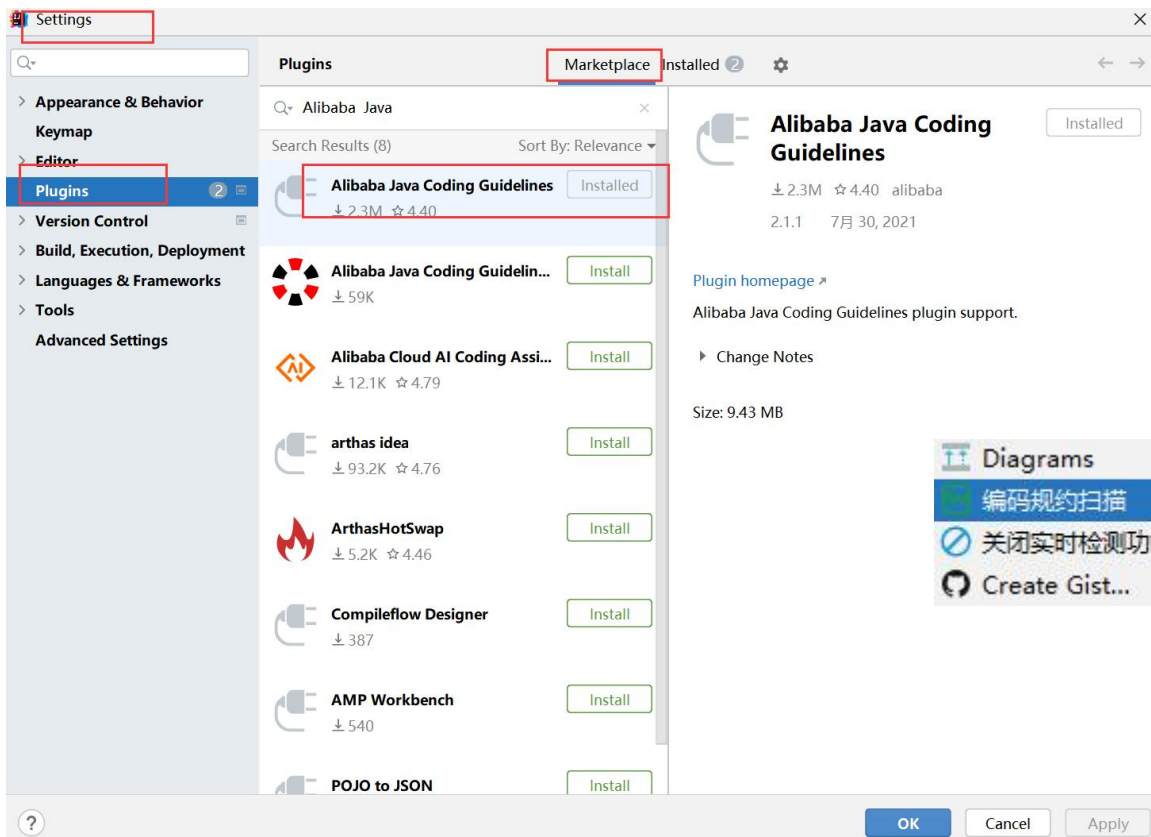




# 实验步骤

## 4 安装阿里编码规约插件

- 通过插件管理器搜索 **Alibaba** 并集成安装后，重启IDEA即可。



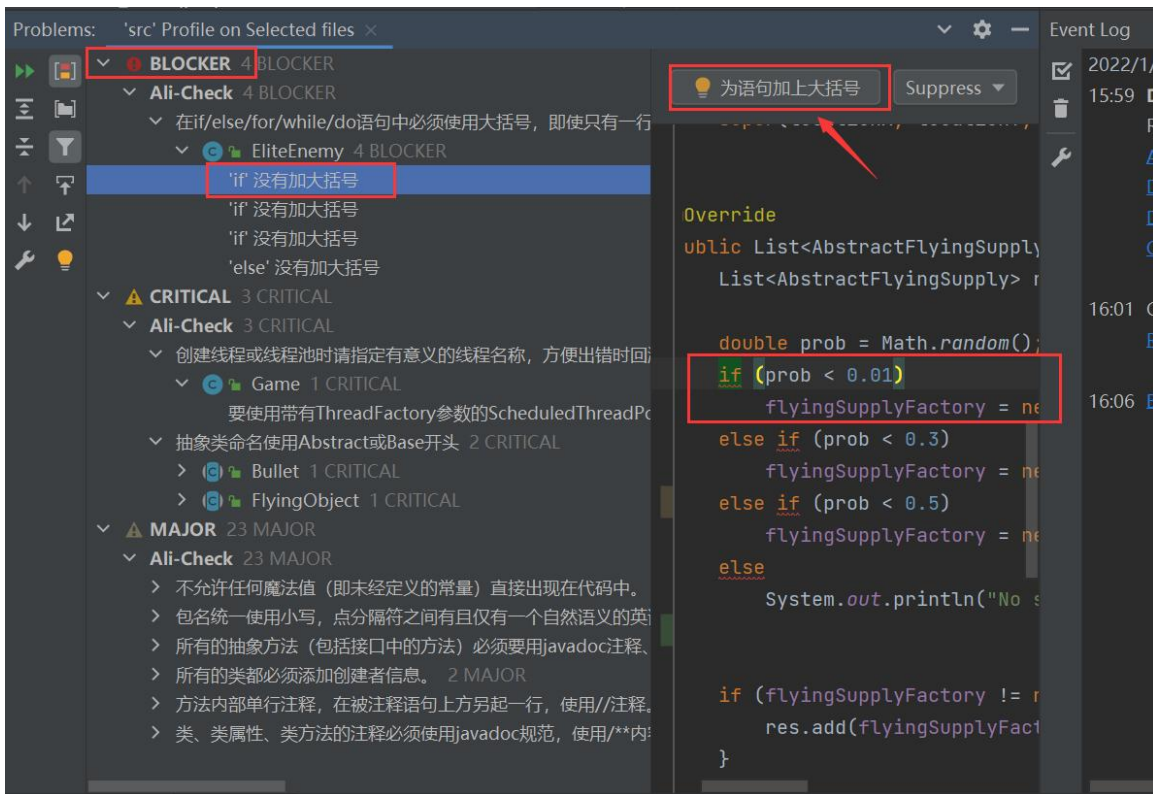


# 实验步骤

## 5 使用阿里编码规约插件

扫描代码后，按 **Blocker**（崩溃） / **Critical**（严重） / **Major**（重要）三个等级显示问题，双击可以定位至代码处，右侧窗口还有针对代码的批量修复功能。

- 建议处理掉全部 Blocker/Critical级别的问题
- 修改前请备份代码





# 实验步骤

## 6

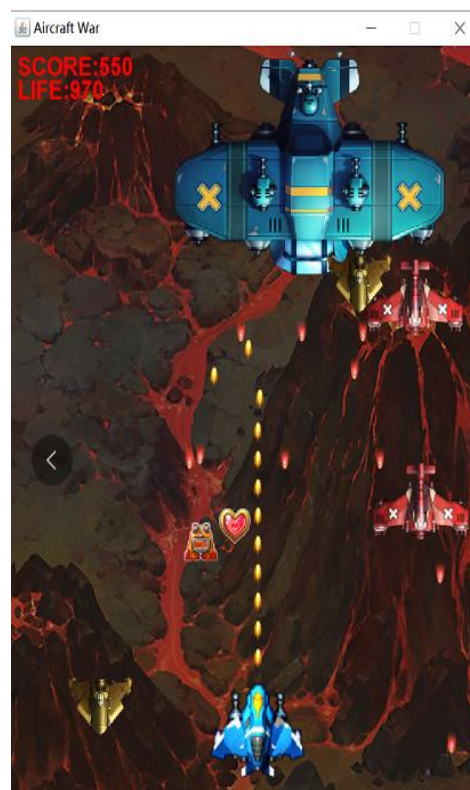
## 重构代码，添加Boss敌机，实现散射弹道

本次实验的目标：使用工厂模式添加Boss敌机，并实现散射弹道。

- 出现：分数达到设定**阈值**，可多次出现
- 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**）
- 坠毁：随机掉落**3个**道具



只有敲代码才能  
感受到温暖





# 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>



# 作业提交

## 实验三报告

采用黑盒测试和白盒测试的常用方法，为飞机大战系统设计测试用例。在系统中选择英雄机、敌机、子弹和道具类的方法（包含其父类方法）作为单元测试的对象，为每个测试对象编写单元测试代码。要求至少选择3个类，每个类至少测试2个方法，并截图 JUnit 单元测试的结果。

### 1. 测试用例（一个方法一个测试用例）

用例编号					
待测试类及方法					
测试类及方法					
前提条件（如有）					
用例描述	用例编号	1			
<p>用例编号：唯一标识测试用例</p> <p>待测试类及方法：该用例所测试类及方法；相应的测试前提条件（如有）：执行该用例（或道具）。</p> <p>用例描述：用一句话简单总结测试步骤；详细完整的操作</p> <p>期望结果：正常情况下的响</p> <p>实际结果：程序通过测试步</p> <p>测试结果：通过或失败</p> <p>2. JUnit 单元测试结果</p> <p>请截图 JUnit 每个测试类（</p>	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



---

**同学们  
请开始实验吧！**