

# 杀戮尖塔MOD入门教程

- 杀戮尖塔MOD入门教程
  - 简介
  - 准备工作
    - 游戏本体安装
    - 下载必要的mod支持库
    - 搭建mod开发环境
      - IDE的安装
      - maven环境配置
  - 开始构建项目
    - 构建步骤
    - 测试构建效果
    - 作业
  - Mod开发语法基础
    - 创建新mod
    - 控制台输出
    - 函数基础
      - 变量
      - 函数
      - 类
      - 作用域
    - 作业2
  - Mod开发初步
    - 监听函数
    - 引用
    - 继承

## 简介

杀戮尖塔的MOD都需要modthespire,stslib和basemod这三大支持库来支持，杀戮尖塔mod的本质是在游戏本体中通过第三方API [\[1\]](#)来添加自己的代码

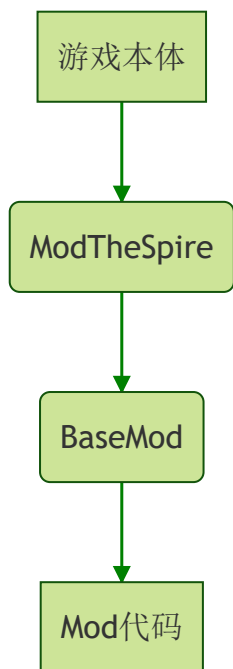


图1 杀戮尖塔mod结构

杀戮尖塔本体是由java语言构建的，构建方式属于典型的面向对象式的。具体而言就是将众多方法分类封装，然后在程序需要的时候调用。对此，mod代码也可以仿照杀戮尖塔的本体结构来搭建。

▼ desktop-1.0.jar.src E:\modifys\desktop-1.0.jar.src

- > .idea
- > achievements
- > audio
- > beyondScene
- > bottomScene
- > cards
- > cardui
- > changelog
- > cityScene
- ▼ com
  - > .idea
  - > badlogic
  - > codedisaster
  - > esotericsoftware
  - > fasterxml
  - > gikk
  - > google
  - > jcraft
  - > megacrit
  - > sun
- > darwin
- > de
- > endingScene
- > font
- > images
- > io
- > javazoom
- > linux
- > localization
- > META-INF
- > net
- > oldCards
- > orbs
- > org
- > powers
- > shaders
- > title
- > vfx
- > win-x64
- > win-x86

## 准备工作

## 游戏本体安装

（略）

## 下载必要的mod支持库

1. 在杀戮尖塔的steam页面中找到创意工坊（有条件的同学可以科学上网，也可使用第三方steam加速器）
2. 搜索stslib，basemod，以及modthespire。或是在右侧菜单勾选api寻找这三项。
3. 点击订阅即可。

以上也是想要测试以及游玩其他人提供的mod的先决条件。

## 搭建mod开发环境

### IDE的安装

杀戮尖塔由java语言编写，为了搭建mod这种大型java工程，我们可以使用IDE（集成开发环境）。目前主流的java IDE<sup>[2]</sup>有Eclipse和Intelij idea。本教程后面将以IJ作为示例进行讲解。

这两种IDE可以在官网下载，其中Eclipse和Intelij的社区版都是免费的，可以直接使用。点击上文中的超链接即可在官网进行下载并安装。

### maven环境配置

maven是一款程序框架搭建平台，使用它就可以搭建类似于杀戮尖塔这样结构的java程序。下面讲解下windows环境的配置方法。其他操作系统的配置方法可参考[此处](#)。

首先通过超链接下载最新版本maven，然后在系统中设置环境变量如下：

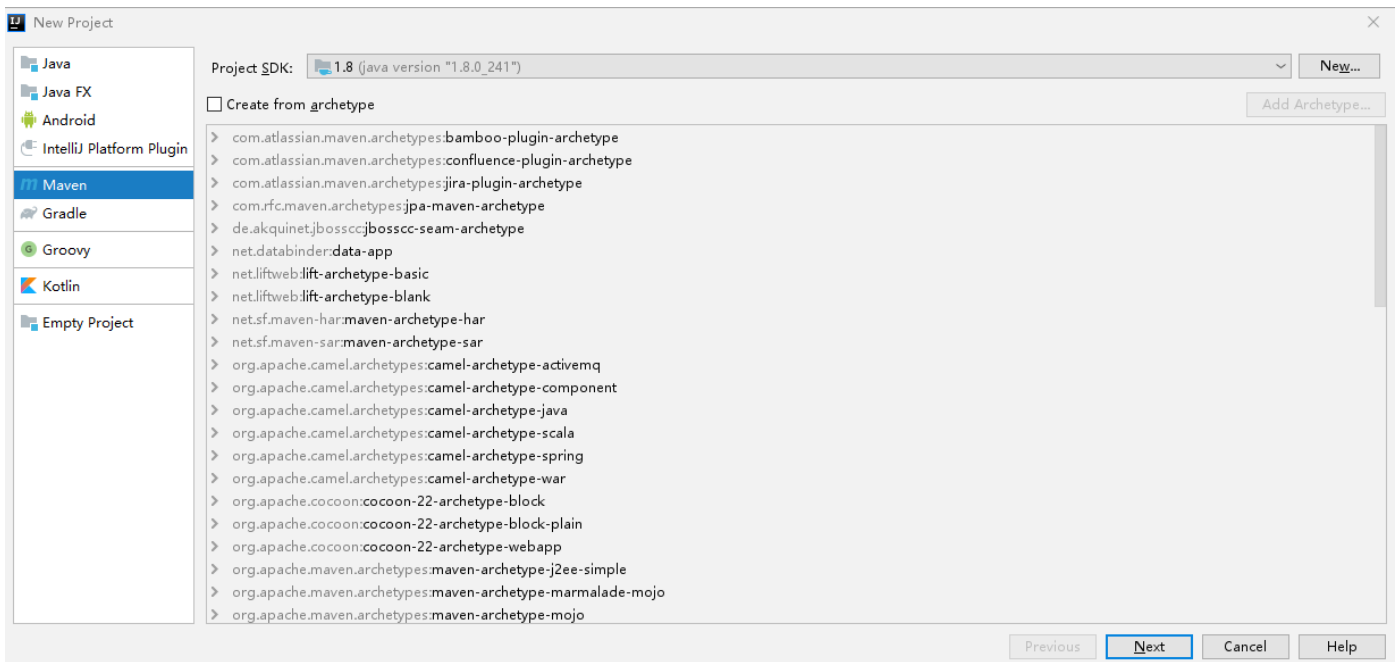
1. 找到 计算机-属性-高级系统设置，点击环境变量。
2. 新建系统变量 **MAVEN\_HOME**，设置变量值：E:\Maven\apache-maven-3.3.9（若已存在则可跳过此步）
3. 寻找系统变量 **Path**，选择编辑，之后选择新建，写入 ;%MAVEN\_HOME%\bin

## 开始构建项目

### 构建步骤

至此我们完成了开发环境的配置，可以进行mod的开发工作了。下面以IJ为例来介绍如何构建一个项目：

1. 打开ij，新建一个maven项目，不勾选选择框。随后填写项目名和文件地址。（本项目取名为tutorial作为示例）



2. 进去之后如下编辑pom.xml:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>org.example</groupId>
8     <artifactId>tutorial</artifactId>
9     <version>0.1</version>
10    <packaging>jar</packaging>
11
12    <name>tutorialMod</name>
13    <description>tutorial</description>
14
15    <properties>
16        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17        <steam.path>E:\SteamLibrary\steamapps</steam.path>
18    </properties>
19
20    <dependencies>
21        <dependency>
22            <groupId>bin</groupId>
23            <artifactId>BaseMod</artifactId>
24            <scope>system</scope>
25            <version>1.0</version>
26            <systemPath>${steam.path}/workshop/content/646570/1605833019/BaseMod.jar</systemPath>
27        </dependency>
28        <dependency>
29            <groupId>bin</groupId>
30            <artifactId>SlayTheSpire</artifactId>
31            <scope>system</scope>
32            <version>1.0</version>
33            <systemPath>${steam.path}/common/SlayTheSpire/desktop-1.0.jar</systemPath>
34        </dependency>
35        <dependency>
36            <groupId>bin</groupId>
37            <artifactId>ModTheSpire</artifactId>
38            <scope>system</scope>
39            <version>1.0</version>
40            <systemPath>${steam.path}/workshop/content/646570/1605060445/ModTheSpire.jar</systemPath>
41        </dependency>
42    </dependencies>
43
44    <build>
45        <finalName>tutorialMod</finalName>
46        <plugins>
47            <plugin>
48                <groupId>org.apache.maven.plugins</groupId>
49                <artifactId>maven-compiler-plugin</artifactId>
50                <version>3.7.0</version>
51                <configuration>

```

```

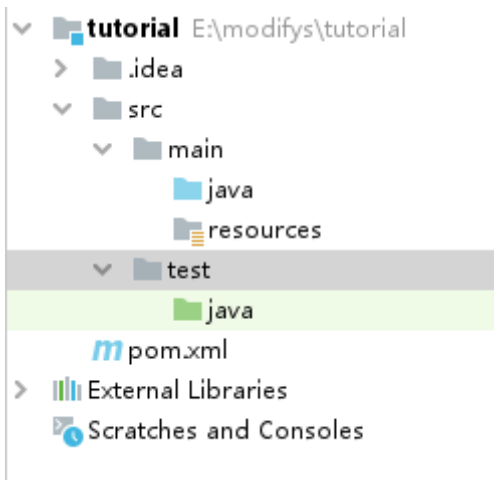
51         <source>1.8</source>
52         <target>1.8</target>
53     </configuration>
54 </plugin>
55 <plugin>
56     <groupId>org.apache.maven.plugins</groupId>
57     <artifactId>maven-antrun-plugin</artifactId>
58     <version>1.8</version>
59     <executions>
60         <execution>
61             <phase>package</phase>
62             <configuration>
63                 <target>
64                     <copy file="target/tutorialMod.jar" tofile="${steam.path}/
65                 </target>
66             </configuration>
67             <goals>
68                 <goal>run</goal>
69             </goals>
70         </execution>
71     </executions>
72 </plugin>
73 </plugins>
74 </build>
75 </project>
76

```

在输入完代码后右下角会显示maven配置发生更改，此时选择import Changes即可。这段代码是maven的配置信息，其中包含了以下关键点：

- 这段代码的意思向指定目标文件夹输出编译好的jar文件，jar文件是代码包，也是所有mod的使用格式。在游玩时mod加载器会加载代码包中的文件，或者使用代码包中的代码覆盖游戏代码。version指的是版本控制中的版本号。
- properties项中规定了该文件的编码方式为UTF-8，这种编码方式允许我们使用中文定义变量，也保证了输出代码的准确性。steam/path中是steam游戏所在的文件夹，可根据steamlibrary文件夹的位置修改。
- dependencies加载了三个我们所需要的代码包，也就是上文所介绍的游戏主文件 desktop-1.0.jar，和两个支持包 BaseMod.jar ModTheSpire.jar
- build部分是关于jar的编译输出的，这里提示了maven的运作规律，会先生成测试文件夹，然后再将测试文件夹生成的文件拷贝到目标文件夹。

3. 此时可以看出文件的组织结构如下：



其中：

- .idea文件夹是IDE的辅助文件，与mod本身无关。
- src是程序的主体部分，分成main和test两部分，其中test是maven自动生成的测试文件夹，maven会自动处理好相关细节。最重要的是main文件夹，java是我们存放代码的地方，而resource文件夹用于存放角色或卡面等图片或json格式的文字素材。

这时我们可以在resource目录下写入 `ModTheSpire.json` <sup>[3]</sup>，该文件向ModTheSpire标识了该Mod的加载信息，格式如下：

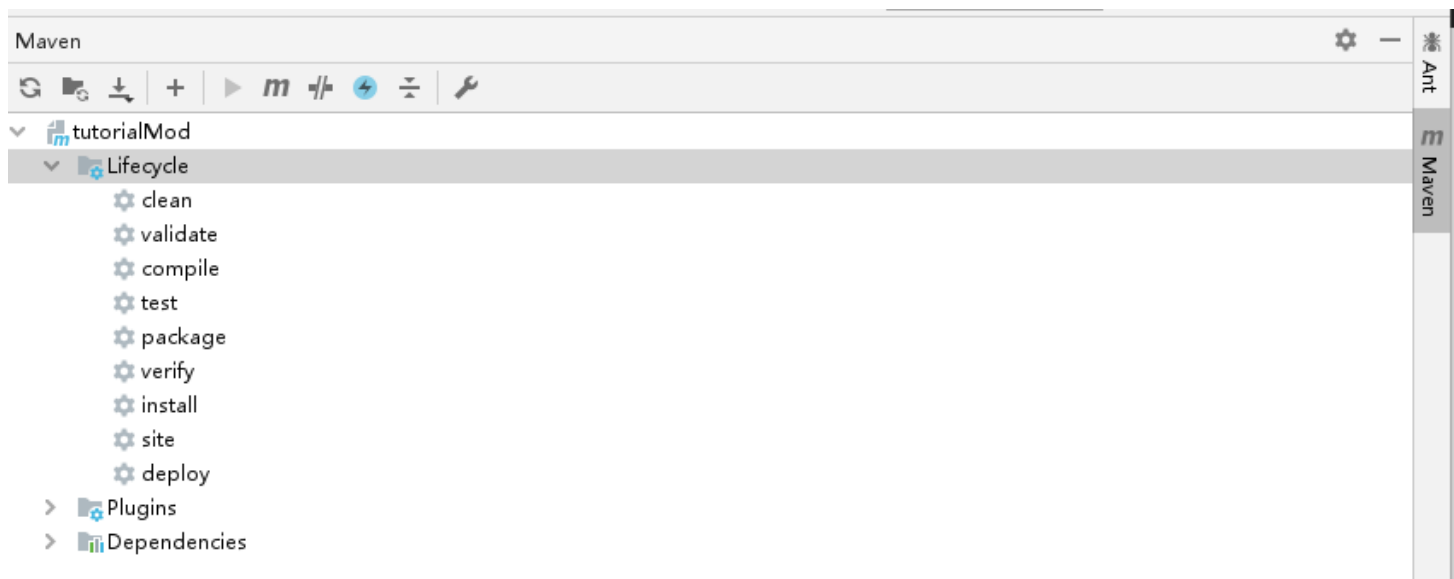
```
1  {
2    "modid": "tutoiral",
3    "name": "tutoiralMod",
4    "author_list": ["A","B"],
5    "description": "新手教程mod",
6    "version": "0.0.1",
7    "sts_version": "03-29-2018",
8    "mts_version": "2.6.0",
9    "dependencies": ["basemod"]
10 }
```

Json文件中用花括号{}代表代码块，方括号[]代表数组，要注意他们的配对。这样我们就完成了mod初期的搭建工作。

## 测试构建效果

构建框架的目的是为了输出mod文件，格式为jar。为此我们要进行maven的package操作来测试搭建效果：



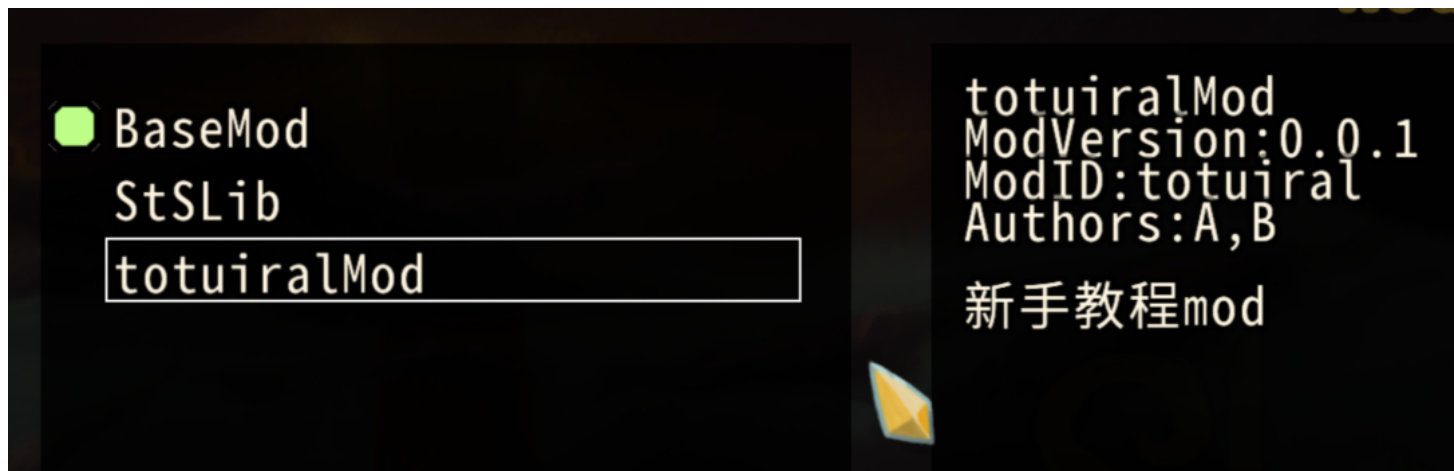


点击右边的maven侧边，展开Lifecycle，这个指的是程序在输出执行的步骤。我们在这些步骤中双击package即可像杀戮尖塔的mod文件夹输出jar格式的mod。也可以点击上面的m按钮，在其中输入 `mvn package`。

如果输出成功后，在steam中用withmod模式打开杀戮尖塔，即可在ModTheSpire画面中看到你刚才输出的文件中的标识信息。



在界面中我们勾选Basemod和ModTheSpire，以及刚刚输出的mod（totuiralMod），然后选择运行就可以进入游戏。



游戏中进入mod菜单，我们可以看到我们写下的标识信息，但是因为没有任何代码，这个mod暂时还没有任何功能。

## 作业1

1. 仿照上文的方式，输出一个jar格式的mod文件并在游戏中运行。给mod取一个名字，并在作者（Author）一栏写上你的ID。
2. 研读加载mod时ModTheSpire的log框中的信息，尝试揣测他们是什么意思。

## Mod语法基础

### 创建新mod

Mod是通过支持库来对游戏代码进行添加，删改工作。所以第一步我们就是要让支持库知道我们创建了Mod，首先在java里面建立一个package命名为该mod的小写名称（本例为tutorial），package有点类似于文件夹。我们在package里面添加第一个Class文件，将其命名为Mod的名称（比如在此案例中为Tutorial，或者也可命名为TutorialHook）。class是一个类文件，而类是java的一个代码单元。我们命名类的时候采用：首字母大写单词，之间大小写交错的形式。

在这个类中写入如下代码：

```
1  import com.evacipated.cardcrawl.modthespire.lib.SpireInitializer;
2  @SpireInitializer
3  public class Tutorial {
4      public Tutorial() {
5      }
6      public static void initialize() {
7          new Tutorial();
8      }
9  }
```

其中import指的是这里用到了别人写的类，在这里指的就是ModTheSpire里的代码。这几行代码非常简单，就是告诉ModTheSpire说，创建了一个新mod，叫做Tutorial。在这里面 @SpireInitializer 表示以下代码是初始化内容。

有了这些代码我们就成功创建了一个新的mod，现在我们可以写点简单的功能。

### 控制台输出

我们之前观察过，在加载mod的时候和游戏过程中ModTheSpire会产生log信息，现在我们可以命令控制台主动产生log。在这里面我们在mod初始化完成后在控制台输出经典的"hello world!"，我们只要在Initialize()的代码块里面加入这么一行命令即可：

```
System.out.print("hello world!");
```

每个语句用半角分号表示结束。该命令不仅能输出文本（字符串），也可以输出数字甚至表达式：

```
System.out.print("hello world! 5*6+30=");
System.out.print(5*6+30);
System.out.print("end of line!\n2+2="+4);//此处+代表文本之间相连，\n在此是一个字符，专门表示该行已结
```

将其输出为jar后我们可以观察加载该mod时产生的log。事实上在游戏中我们也可以插入这样的命令输出log，用于调试。我们可以看到控制台显示了这组输出。其中2ms表示2毫秒，显示了加载该mod所需时间

```
hello world! 5*6+30=60end of line!
2+2=4    - 2ms
Done.
```

## 函数基础

除了输出字符，mod最重要的还是跟游戏中的各种行为进行互动。而这些依赖于java的函数，由于java是由各种各样的类构成的，所以函数都是成员函数（或称方法）。

## 变量

函数可以用于处理输入和输出（也可以都不处理），为了表述输入这种行为，我们引入了变量去衡量输入的值。变量可以随着数值的不同而改变自己的值。在杀戮尖塔经常用的是这几种变量：整形，浮点型，布尔型，字符串型。

- 整形用于表示整数，根据正负分为有符号和无符号，也可根据占空间大小来区分，一般来讲支持的位数越高越占空间。杀戮尖塔mod开发中最常见的整数类型是整型，标识符为int，它占四个字节，可以表示从 $-2^{31}$ 到 $2^{31} - 1$ 之间的整数。
- 浮点型表示小数，顾名思义它通过在二进制数位之间浮动小数点来表示对应的小数，在这我们不展开研究浮点数的表述方法。最常用的浮点数类型是float，占用4个字节，另一种类型double则要占用8个字节，精度也更高。我们会在一个数字后面加上f表示他为float,否则为double。
- 布尔型只存储两个值：true和false，表示真与假。用于做逻辑判断，对应计算机表示为0和1。但是java不允许将数字转化为布尔值，布尔值类型的标识符为boolean。
- 字符串是一串文本，是由很多字符组成的。他的使用与其他变量并不一致，较为特殊。使用String标识符

Java中创建变量由以下几个部分组成：

```
1 | public int number;
2 | private float dot=2.4f;
3 | public static boolean judge=true;
4 | public final String HELLO="Hello world!";
```

1. 属性： public 表示它可以被类外的代码访问， private 则表示只能在这个类里面被访问。 static 表示在函数结束后变量仍然被保留， final 指变量的值不能被修改，表示一个常量。更

多属性可以查阅相关java资料。

2. 标识符：用于表示变量的类型。
3. 名称：变量的名称一般使用大小写混合，以小写开头，避免使用单字母变量。如果带有**final**属性则表示为常量，名字全部用大写并使用下划线分隔单词。
4. 初始化：变量使用=赋值，在创建变量时赋值被称为初始化。未初始化的变量不能直接访问，需要赋值后访问。常量必须初始化。

## 函数

函数具有输入端和输出端，表示方法如下：

```
1 | private int print(String string,int num){
2 |     System.out.print(string);
3 |     return num;
4 | }
```

1. 与变量类似，函数也有属性，例如**public**和**private**。
2. 函数用变量标识符表示具有该类型的输出值，可以被赋给某个变量或被另一个函数使用。
3. 函数括号内表示接受的变量类型，被称为形式参数。真正在调用过程中被传入的变量被称为实际参数，两者的命名没有任何关系。
4. 函数命名规则与变量类似，使用小写来命名
5. 函数使用**return** 语句来返回一个值，同时跳出函数体。

值得一提的是函数可以没有返回值，也可以没有输入值。我们可以这样声明这种函数：

```
1 | private void hello(){
2 |     System.out.print("Hello world!");
3 |     return;
4 | }
```

此处使用了**void**标识符来表示没有返回值，**return**语句单纯表示跳出函数，可以省略等函数所有语句正常执行完自动跳出。

## 类

java的基本代码单元就是类，在杀戮尖塔里面类有着非常清晰地实际意义。你使用的卡都是这种卡中的其中一张（对象），生成的一个电球也属于电球这个类型的一员。所以函数和变量都是放在类里面去讨论的，例如电球就具有放电攻击敌人，激发等函数，他的伤害作为变量又能随时受到集中值的影响。

类具有对象，而作为类的对象又能在其他类中使用。在一场战斗中，战场上就具有卡类的对象，玩家类的对象以及怪物类的对象同时出现。所以对象具有很重要的地位，申明一个对象的代码如下：

```
A a=new A();
```

其中**A**表示一个名为**A**的类，而这个**a**则是这个对象的代号（引用）。**new** 代表**A**创建了新对象，这里面出现了一个特殊的函数。表示创建一个新对象所需要的操作，这种函数叫做构造函数<sup>[4]</sup>。比如创建一个新卡牌时要写入他的攻击力和费用，才能结束新卡牌的创建。

由此我们可以看出，一个类是由对象所拥有的变量<sup>[5]</sup>（例如卡的攻击力，玩家的生命值）和所拥有的函数（又称方法）构成的。

我们再回过头来审视我们创建的第一个类，这个类就有我们所说的部分功能

```
1  import com.evacipated.cardcrawl.modthespire.lib.SpireInitializer;
2  @SpireInitializer
3  public class Totutorial {
4      public Totutorial() {
5      }
6      public static void initialize() {
7          new Totutorial();
8      }
9  }
```

首先这里面具有一个构造函数，我们发现构造函数没有类型标识符，因为它必然不能被用于直接调用而返回一个参数。在 **Initialize()** 方法里面我们就通过构造函数创建了该**mod**的对象传入**ModTheSpire**，从而让**ModTheSpire**识别我们的**Mod**。

## 作用域

之前我们多次提到了**public**和**private**这两个重要属性，为什么会有这种区别？**public**代表其具有全局属性，包括类。**IDE**为了结构的整齐性，强制要求公有类必须以单独文件的形式存在，这样就可以通过其他类中调用该类来实现类的全局化。调用公有类使用 **import**：

```
import com.evacipated.cardcrawl.modthespire.lib.SpireInitializer;
```

其中最后一个单词代表类名，前面的单词代表该类所在的包名，这种格式十分类似于文件与文件夹。其他类只有通过这种方式才能访问到该公有类中的成员和函数。而其他类也只可访问这个类中的公有成员和函数，不能访问私有的。

作用域中还有一个属性非常重要，一般的成员都归对象所有，比如说手牌上有很多张打击，如果一张打击受到干瘪的手影响，这时其他打击的费用仍然是不变的。说明虽然都为打击类的对象，但是他们的成员能量是互不干涉的。但是如果具有 **static** 静态属性的话这个成员或者函数就归为类所有，所有该类对象都能共享这个静态成员和方法。我们也就不需要再用对象去访问他了，可以直接用类本身去访问。

假设打击<sup>[6]</sup>这个类有两个成员，一个是打击卡的攻击力，一个是总共生成了多少张卡，以下是他们的访问方式：

```
strike a=new strike();
int dam=a.damage;//dam变量得到的是某张打击“a”的伤害值
int summary=strike.sum //summary得到的是打击这个类总共生成了多少张打击卡
```

## 作业2

以下提供了一个监听(hook)函数，根据给出的代码和提示实现以下功能：(改编自basemodWiki)

```
1  package totutorial;
2
3  import basemod.interfaces.*;
4  import com.evacipated.cardcrawl.modthespire.lib.SpireInitializer;
5  import com.megacrit.cardcrawl.cards.DamageInfo;
6  import com.megacrit.cardcrawl.rooms.AbstractRoom;
7
8  import basemod.BaseMod;
9
10 @SpireInitializer
11 public class Totutorial implements OnPlayerDamagedSubscriber,PostBattleSubscriber {
12
13
14     public Totutorial() {
15         BaseMod.subscribe(this);
16     }
17
18     public static void initialize() {
19         new Totutorial();
20     }
21
22     @Override
23     public int receiveOnPlayerDamaged(int damage, DamageInfo damageInfo) {
24         //当玩家受到伤害，该函数接受玩家受到的伤害和伤害种类
25         return damage;
26     }
27
28     @Override
29     public void receivePostBattle(AbstractRoom r) {
30         //该函数中的内容将在战斗结束后执行一次
31         System.out.println();
32     }
33
34 }
```

请在类中和函数体内补充代码，使得log能够输出每次战斗玩家受到的伤害总和。

# Mod开发初步

通过上面的讲述，我们对类和函数有了初步的认识。现在我们将这些知识运用在实际的Mod开发，并来补充学习类与函数其他特性。

## 监听函数

我们将函数的知识用于对游戏的互动来看，我们可以通过ModeTheSpire来让mod接受游戏中的行为，并用监听函数来对游戏内的行为做出反应。

现在我们扩充最开始的类文件：

```
1  package tutorial;
2
3  import basemod.BaseMod;
4  import basemod.interfaces.PostDrawSubscriber;
5  import com.evacipated.cardcrawl.modthespire.lib.SpireInitializer;
6  import com.megacrit.cardcrawl.cards.AbstractCard;
7
8  @SpireInitializer
9  public class Totutorial implements PostDrawSubscriber {
10     public Totutorial() {
11         BaseMod.subscribe(this);
12     }
13     public static void initialize() {
14         new Totutorial();
15         System.out.print("hello world! 5*6+30=");
16         System.out.print(5*6+30);
17         System.out.print("end of line!\n2+2="+4); //此处+代表文本之间相连，\n在此是一个字符，专
18     }
19
20     @Override
21     public void receivePostDraw(AbstractCard abstractCard) {
22         System.out.println(abstractCard.name); //与print的区别在于println会自动换行
23     }
24 }
```

这个类的意思是通过监听函数来得知玩家在游戏中抽到了了什么卡，并将结果输出在log上。我们将形参传入的对象做处理，得到了该形参的名字并输出。

```
17:31:27.896 INFO basemod.BaseMod> publishPostDraw  
防御  
17:31:27.987 INFO basemod.BaseMod> publishPostDraw  
发泄  
17:31:28.104 INFO basemod.BaseMod> publishPostDraw  
暴怒  
17:31:28.221 INFO basemod.BaseMod> publishPostDraw  
打击  
17:31:28.337 INFO basemod.BaseMod> publishPostDraw  
打击  
..
```

这段代码中涉及到了很多类的特性，接下来将逐步讲解。

## 引用

我们使用该程序再次做实验，这次我们先在初始化的时候交换两个int变量，之后再对抽取到的卡牌进行操作，观察结果：



```

1 package tutorial;
2
3 import basemod.BaseMod;
4 import basemod.interfaces.PostDrawSubscriber;
5 import com.evacipated.cardcrawl.modthespire.lib.SpireInitializer;
6 import com.megacrit.cardcrawl.cards.AbstractCard;
7
8 @SpireInitializer
9 public class Tutorial implements PostDrawSubscriber {
10     public Tutorial() {
11         BaseMod.subscribe(this);
12     }
13     public static void initialize() {
14         new Tutorial();
15         int a=1, b=100
16         System.out.println(a+"    "+b);
17         swap(a,b);//调用函数swap交换a, b数值
18         System.out.println(a+"    "+b);//这一行会输出什么?
19     }
20
21     @Override
22     public void receivePostDraw(AbstractCard abstractCard) {
23         System.out.println(abstractCard.name);
24         abstractCard.name="NULL";
25     }
26
27     public static void swap(int a,int b){//swap函数采用静态方法表示该方法不需要经过对象去调用
28         int temp=a;
29         a=b;
30         b=temp;
31         System.out.println(a+"    "+b);//输出交换后数值
32     }
33 }

```

交换输出结果:

```

- tutorialMod
- example_mod.Tutorial
1    100
100    1
1    100
- 3ms
Done.

```



很容易就能发现，`swap`函数没有改变真实的数值，但是`receivePostDraw`函数却在函数结束后真实改变了卡牌的名字。这一点说明对象的传递机制和变量并不一样。

回忆对象的声明和变量的声明，这两个有着很大的差别：

```
A a=new A();
public int number=0;
```

我们在对象的声明中特意加入了`new`关键字以创建真正的对象，而`a`不是真正的对象，只是对象的引用[7]。这样就产生了在函数传递过程中的不同情形。

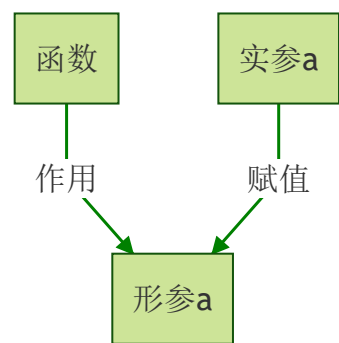


图2 变量的传递过程，函数没有修改实参a

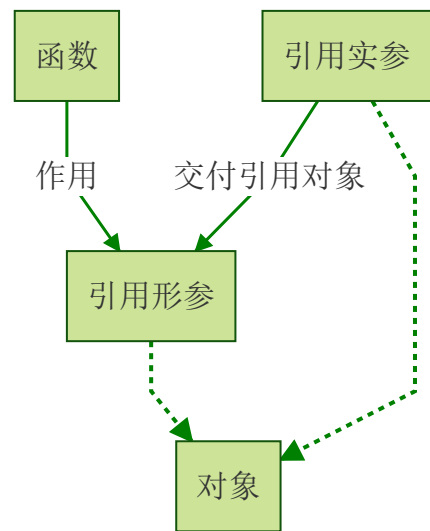


图3 引用的传递过程，函数参与修改了对象[8]

由此可见传递对象的引用可以让函数对原值进行修改，而非复制。**Java**对此设计了封装基本变量的类，将变量转化为对象处理。

例如：

```
Integer i = new Integer(0); //对应int
Boolean b = new Boolean(False); //对应boolean
```

这种封装处理在之后的泛型操作中也会体现到。

## 继承

机器人具有多种能量球，但是这些能量球都有共同的行为：有共同的活动区域，能被激发，每回合会执行某种操作，能够响应弹幕齐射或者编译冲击。我们为了归纳这些种类不同但是本质上都是能量球的类型，我们引入了继承的概念。

首先我们创建一个总体的能量球类，然后再让各种各样的能量球继承这个能量球类。这样我们既可以在父类中兼顾到他们的共性，也能实现他们具有的不同的功能。这是面向对象设计的核心内容，也是杀戮尖塔**mod**制作的核心概念。

现在我们以一张卡牌的源代码为例来详细讲述继承的有关概念。

1. 应用程序接口，提供了代码抽象功能。↩
2. 集成开发环境，除了文中介绍的关于**java**的**IDE**。还常用**visual studio**或者**xcode**用于开发。↩
3. 右键**resource**目录，选择**New-file**，将新建文件改名为**ModTheSpire.json**并编辑即可。↩
4. 构造函数往往需要传入参数，这里假设构造函数不需要传入参数。↩
5. 也被称为成员，代表一个对象的组成成员。从内存上看，一个对象所占的内存确实是由所有成员所占的内存拼凑而来。也可被称为字段（**Field**），这里不探讨字段与成员的差别。↩
6. 游戏中的打击类远比这个复杂。↩
7. 你甚至能把他理解为替身。↩
8. ~~替身受到伤害主人也会受到伤害~~↩