Shoot射击游戏第一天：

1. 创建6个对象类，并创建World类并测试

Shoot射击游戏第二天：

1. 给6个对象类添加构造方法，并测试

Shoot射击游戏第三天：

1. 设计小敌机数组、大敌机数组、小蜜蜂数组、子弹数组，并测试

Shoot射击游戏第四天：

1. 设计FlyingObject超类，6个对象类继承超类
2. 在超类中设计2个构造方法，6个对象类分别调用
3. 画窗口

Shoot射击游戏第五天：

1. 将小敌机数组、大敌机数组、小蜜蜂数组组合为FlyingObject数组，并测试
2. 在6个派生类中重写step方法

Shoot射击游戏第六天：

1. 给类中成员添加访问控制修饰符
2. 设计Images工具类给对象准备图片

Shoot射击游戏第七天：

1. 设计窗口的宽和高为常量、适当地方做修改
2. 画对象：

1) 想画对象需要获取对象的图片，每个对象都得获取图片，意味着获取图片行为为共有行为，

所以将获取图片行为设计在超类中，每个对象获取图片的方式都是不一样的，所以设计为抽象方法

----在FlyingObject中设计抽象方法getImage()来获取对象的图片

2) 想获取图片需要考虑对象的状态，因为在不同状态下获取的图片是不一样的，因为每个对象都有状态，

所以将状态设计在超类中，状态一般都设计为常量，同时再设计变量state表示当前状态

----在 FlyingObject中设计LIFE、DEAD、REMOVE常量，state变量

状态设计好了，那么对象在获取图片时需要去判断对象的状态，每个对象都需要判断状态，为共有行为，

所以将判断状态行为设计在超类中，每个对象判断状态的方式都是一样的，所以设计为普通方法

----在FlyingObject中设计isLife()、isDead()、isRemove()来判断对象的状态

3) ----重写超类的getImage()获取图片:

3.1) 天空Sky，直接返回sky图片即可

3.2) 子弹Bullet:

3.2.1) 若活着的，直接返回bullet图片即可

3.2.1) 若死了的，直接删除

3.3) 英雄机Hero:

3.3.1) 若活着的，直接返回两张英雄机图片的来回切换

3.4) 小敌机Airplane:

3.4.1) 若活着的，直接返回第一张图片即可

3.4.2) 若死了的，返回从第二张图到第五张图的切换，5后删除

3.5) 大敌机BigAirplane:

3.5.1) 若活着的，直接返回第一张图片即可

3.5.2) 若死了的，返回从第二张图到第五张图的切换，5后删除

3.6) 小蜜蜂Bee:

3.6.1) 若活着的，直接返回第一张图片即可

3.6.2) 若死了的，返回从第二张图到第五张图的切换，5后删除

4) 图片有了就可以开画了，在World类中重写paint()画方法----此内容不需要掌握

画对象-----需要三个数据(图片，x坐标、y坐标)

<=

；

“”

abc-------------bbc

方法名写错

airs-------------ars

调方法不写括号

文件没有保存

晚上7点晚课------------10点

白天

若不考虑爆破，对象状态分两种:

1. 活着的LIFE
2. 删除的REMOVE

若考虑爆破，对象状态分三种:

1. 活着的LIFE
2. 死了的DEAD(还没有死彻底，还得爆破)
3. 删除的REMOVE

回顾：

1. package和import:
2. 访问控制修饰符:

1) public: 公开的，任何类

2) private: 私有的，本类

3) protected: 受保护的，本类、派生类、同包类

4) 默认的: 什么也不写，本类、同包类

1. static: 静态的

1) 静态变量：static，类，方法区、一份、类名点访问

所有对象共享的数据(图片、音频、视频等)

2) 静态方法：static，类，方法区、一份、类名点访问，没有this，不能直接访问实例成员

方法操作与对象无关

3) 静态块：static，类，类被加载时自动执行，只执行一次

加载/初始化静态资源(图片、音频、视频等)

正课：

1. final: 最终的、不可改变的，单独应用率低

1) 修饰变量：变量不能被改变

2) 修饰方法：方法不能被重写

3) 修饰类：类不能被继承

1. static final常量: 应用率比较高

1) 必须声明同时初始化

2) 通过类名来访问，不能被改变

3) 建议：常量名所有字母都大写，多个单词用下划线分隔

4) 编译器在编译时会将常量直接替换为具体的值，效率高

5) 何时用：数据永远不变，并且经常使用

1. 抽象方法:

1) 由abstract修饰

2) 只有方法的定义，没有具体的实现(连大括号都没有)

1. 抽象类:

1) 由abstract修饰

2) 包含抽象方法的类必须是抽象类

不包含抽象方法的类也可以声明为抽象类------我乐意

3) 抽象类不能被实例化(new对象)

4) 抽象类是需要被继承的，派生类:

4.1) 重写所有抽象方法---------变不完整为完整

4.2) 也声明为抽象类-----------一般不这么用

5) 抽象类的意义:

5.1) 封装派生类所共有的属性和行为-------代码复用(超类的意义)

5.2) 为所有派生类提供统一的类型---------向上造型(超类的意义)

5.3) 可以包含抽象方法，为所有派生类提供统一的入口(造型之后通过超类引用能点出来方法)

派生类的具体实现不同，但入口是一致的

1) 问: 抽象方法step()有什么意义?

答: 当向上造型的时候，通过超类的引用可以点出step()，没有它就点不出step()---能点出来什么看引用的类型

但是调用的不是整个抽象方法，真正调用的还是重写之后的------------重写方法被调用，看对象的类型

2) 问: 把抽象方法就做成普通方法不是也可以吗?

答: 可以，但工作中大家都会把它做成抽象的

普通方法被重写也可以，不重写也可以------达不到限制作用的

抽象方法是必须被重写的------------------达到限制的作用

设计规则:

1. 将派生类所共有的属性和行为，抽到超类中--------抽共性
2. 所有派生类的行为都一样，设计为普通方法

所有派生类的行为都不一样，设计为抽象方法

下周讲

abstract class Aoo{

成员变量

普通方法

}

abstract class FlyingObject{

width,height,x,y

abstract void step();

}

class Airplane extends FlyingObject{

void step(){ //重写抽象方法------变不完整为完整

y向下

}

}

class Bee extends FlyingObject{

void step(){

x左右动，y向下

}

}

A项目-------调用B项目

new FlyingObject(); //编译错误

FlyingObject o; //正确

**new** FlyingObject(); //创建FlyingObject对象----编译错误

**new** FlyingObject[5]; //创建FlyingObject数组对象

//只要是个数据类型就能创建数组对象

step()

泛化----就是继承

窗口的宽和高