Shoot射击游戏第一天：

理论:

1. 什么是类？什么是对象
2. 如何创建类？如何创建对象？如何访问成员？

案例:

1. 创建6个对象类，并创建World类并测试

Shoot射击游戏第二天：

理论：

1. 方法的重载
2. 构造方法
3. this

案例：

1. 给6个对象类添加构造方法，并测试

Shoot射击游戏第三天：

理论：

1. null和NullPointerException
2. 引用类型之间画等号
3. 引用类型数组

案例：

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：

理论：

1. 继承
2. super

案例：

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：

理论：

1. 向上造型
2. 方法的重写

案例：

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：

理论：

1. package和import
2. 访问控制修饰符
3. static

案例：

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：

理论：

1. final
2. static final常量
3. 抽象方法
4. 抽象类

案例：

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

Shoot射击游戏第八天：

理论：

1. 成员内部类
2. 匿名内部类

案例：

1. 敌人入场:

1) 创建敌人为窗口所特有的行为，所以在World中创建nextOne()生成敌人

2) 敌人入场为定时发生的，所以在run()中调用enterAction()实现敌人入场---enterAction()下调用repaint()重画

在enterAction中:

每400毫秒，获取敌人obj，enemies扩容，并将obj装到最后一个元素位置上

1. 子弹入场:

1) 创建子弹为英雄机所特有的行为，所以在Hero中创建shoot()生成子弹

2) 子弹入场为定时发生的，所以在run()中调用shootAction()实现子弹入场

在shootAction中:

每300毫秒，获取子弹对象bs，bullets数组扩容，数组的追加

Shoot射击游戏第九天：

案例：

1. 飞行物移动: ----先要画出天空的第二张图片

1) 飞行物移动为派生类所共有的行为，所以在超类中设计抽象step()方法实现飞行物移动，派生类重写step()

2) 飞行物移动为定时发生的，所以在run()中调用stepAction()实现飞行物的移动

在stepAction()中:

天空动，遍历敌人数组敌人动，遍历子弹数组子弹动

1. 英雄机随着鼠标移动:

1) 英雄机随着鼠标动为英雄机的行为，所以在Hero中设计moveTo()实现英雄机随着鼠标动

2) 英雄机随着鼠标移动为事件触发的，所以在侦听器中重写mouseMoved()鼠标移动事件

在mouseMoved()中:

获取鼠标的x坐标和y坐标，调用moveTo()实现英雄机移动

1. 删除越界的敌人和子弹:

1) 在FlyingObject中设计outOfBounds()检测敌人是否越界，在Bullet中重写outOfBounds()检测子弹是否越界

2) 删除越界敌人和子弹为定时发生的，所以在run中调用outOfBoundsAction()删除越界敌人和子弹

在outOfBoundsAction()中:

声明不越界数组，遍历enemies/bullets数组，

判断若不越界: 将不越界的敌人/子弹装到不越界数组中

最后将不越界数组复制到enemies/bullets中

Shoot射击游戏第十天：

理论：

1. 接口
2. 多态-1

案例：

1. 设计Enemy得分接口，Airplane和BigAirplane实现Enemy得分接口

设计Award奖励接口，Bee实现Award奖励接口

Shoot射击游戏第十一天：

理论：

1. 多态-2

案例：

1. 子弹与敌人的碰撞:

1) 在超类FlyingObject中设计hit()检测碰撞，goDead()飞行物去死

在英雄机Hero中设计addLife()增命、addFire()增火力

2) 子弹与敌人的碰撞为定时发生的，所以在run()中调用bulletBangAction()实现子弹与敌人的碰撞

在bulletBangAction()中:

遍历子弹得子弹，遍历敌人得敌人，判断若都活着并且撞上了:

敌人去死、子弹去死

判断若为Enemy得分，则玩家增分

若为Award奖励，则英雄机得奖励

1. 画分和画命:

1) 在英雄机Hero中，设计getLife()获取英雄机命数

2) 在World类的paint()方法中，画分和画命

1. 英雄机与敌人的碰撞:

1) 借用FlyingObject类的hit()碰撞检测、goDead()去死

在Hero中设计subtractLife()减命、clearFire()清空火力值

2) 英雄机与敌人的碰撞为定时发生的，所以在run()中调用heroBangAction()实现英雄机与敌人的碰撞

在heroBangAction中:

遍历敌人得敌人，判断若都活着并且撞上了:

敌人去死、英雄机减命、英雄机清空火力值

Shoot射击游戏第十二天：

理论：

1. 内存管理
2. 面向对象三大特征

案例：

1. 检测游戏结束:

1) 借用Hero的getLife()获取命数

2) 检测游戏结束为定时发生的，所以在run()中调用checkGameOverAction()检测游戏结束

在checkGameOverAction()中:

判断当英雄机的命数<=0时，表示游戏结束，……(将当前状态修改为GAME\_OVER状态)

1. 画状态:

1) 在World中设计START、RUNNING、PAUSE、GAME\_OVER状态常量，state变量表示当前状态

在Images中设计start、pause、gameover状态图片

在World的paint方法中：实现在不同状态下画不同的对象

2) 将run中的那一堆action设计为仅在运行状态下执行

将英雄机移动也设计为仅在运行状态下执行

3) 重写mouseClicked()鼠标点击事件: 启动变运行，游戏结束先清理现场再变启动

重写mouseExited()鼠标移出事件: 运行变暂停

重写mouseEntered()鼠标移入事件: 暂停变运行

从头开始做项目，应该如何入手:

1. 首先要了解业务需求，而后先设计数据结构，再写算法
2. 设计数据结构: 类+成员变量

1) 设计超类FlyingObject、接口Enemy、Award

2) 设计派生类继承超类、实现接口

3) 设计图片工具类Images

4) 设计World窗口类，画窗口. ------建议先把接口内容重做两次、尝试着自己抽下那两个接口

1. 设计算法: 方法 -------今天6:30开始上职业素质课，7:30开始上晚课

1) 画对象

2) 敌人入场、子弹入场

3) 飞行物移动、英雄机随着鼠标移动

4) 删除越界敌人和子弹

class FlyingObject{

LIFE,DEAD,REMOVE,state

width,height,x,y

两个构造方法

}

interface Enemy{

getScore();

}

interface Award{

FIRE,LIFE

getAwardType()

}

class Airplane extends FlyingObject implements Enemy{

speed

构造

重写getScore()

}

剩余5个派生类省略…

class Images{

准备图片

}

class World{

WIDTH,HEIGHT

sky,hero,enemies,bullets

把窗口画出来

把对象画出来

敌人入场

子弹入场

飞行物移动

英雄机随着鼠标动

删除越界敌人和子弹

子弹和敌人碰撞

英雄机和敌人碰撞

检测游戏结束

画状态

}

回顾：

\1. 多态:

1) 意义:

2) 向上造型/自动类型转换:

3) 强转类型转换，成功的条件只有如下两种:

3.1) 引用所指向的对象，就是该类型

3.2) 引用所指向的对象，实现了该类或继承了该接口

4) 强转若不符合如上条件，则发生ClassCastException类型转换异常

建议:强转之前先通过instanceof判断引用的对象是否是该类型

正课：

1. 内存管理: 由JVM来管理

1) 堆:

1.1) 存储new出来的对象(包括实例变量)

1.2) 垃圾: 没有任何引用所指向的对象

垃圾回收器(GC)不定时到堆中清扫垃圾，回收过程是透明(看不到)的，不一定一发现垃圾就立刻回收

通过调用System.gc()可以建议JVM尽快调度GC来回收垃圾

1.3) 内存泄漏: 不再使用的对象还没有被及时的回收，严重的泄漏会导致系统的崩溃

----建议: 不再使用的对象要及时将引用设置为null

1.4) 实例变量的生命周期:

创建(new)对象时存储在堆中，对象被回收时被一并回收

2) 栈:

2.1) 存储正在调用的方法中的所有局部变量(包括方法的参数)

2.2) 调用方法时，会在栈中为该方法分配一块对应的栈帧，栈帧中存储局部变量和方法的参数

方法调用结束时，栈帧被自动清除，局部变量与方法的参数也一并被清除

2.3) 局部变量的生命周期:

调用方法时存在栈中，方法调用结束时与栈帧一并被清除

3) 方法区:

3.1) 存储.class字节码文件(包括静态变量、方法)

3.2) 方法只有一份，通过this来区分具体的对象

1. 面向对象三大特征:-------------常见的面试题(问: 你对面向对象的理解)

1) 封装:

1.1) 类: 封装的是对象的属性和行为

1.2) 方法: 封装一段特定的业务逻辑功能

1.3) 访问控制修饰符: 封装的是具体的访问权限

2) 继承:

2.1) 作用: 代码复用

2.2) 超类/父类: 所有派生类所共有的属性和行为

接口: 部分派生类所共有的行为

派生类/子类: 派生类所特有的属性和行为

2.3) 单一继承、多接口实现，传递性

3) 多态:

3.1) 对象的多态( 所有对象都是多态的 )

行为的多态( 所有抽象方法都是多态的 )

3.2) 向上造型、强转类型转换、instanceof

3.3) 多态的表现形式: 方法的重写

现在的程序是:

只要不越界，就要了

真正的程序应该是:

要: 不越界的，并且，LIFE或DEAD的

==

不越界的，并且，非REMOVE的

现在程序效果是-----死了之后只是修改了状态，但是没有从数组中真正的删除

最后敌人飞出窗口还是会被删掉----但是数据会有延迟

敌人默认为LIFE的

一旦被子弹打中，状态改为DEAD

先爆破，爆破完之后，状态改为REMOVE

3条命-----死2次-----只剩1条命-----负1

hero.getLife()<=0

Aoo o = new Aoo();

Object o = new Aoo();

Boo o = new Boo();

Aoo o = new Boo();

Object o = new Boo();

Airplane----FlyingObject-----Enemy-----Object

Object: 所有类的鼻祖

//下课休息15分钟--------11:30上课

Aoo o = new Aoo();---------------a=0

o.show(5);-----------------------b=5

class Aoo{

int a;

void show( int b ){

int c;

System.out.println( a ); //0

System.out.println( b ); //5

System.out.println( c ); //编译错误

}

}

变量在用之前必须声明并初始化

class Aoo{

int a;

void show(int b){

int c = 8;

}

}

假设没有---删除越界敌人的功能:

----意味着所有敌人都在enemies数组中，也就是说enemies中有引用指向敌人对象

----就算敌人飞出窗口，那enemies中的引用也依然在指向那个对象

----虽然飞出窗口的对象以后永远也不用了，但GC不会回收它，因为enemies中还有引用呢

----结论: 运行时间一长，就特别卡

有了删除越界功能:

----意味着enemies里面的都是没有出界的----相当于出界的都删掉了

//------------------10:27上课

垃圾回收器(GC)

项目写串了------7、8个

getScore()

gotScore()

API