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

Shoot射击游戏第八天：

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：

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：

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：

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

回顾:

1. 接口:

引用数据类型，interface定义，只能包含常量和抽象方法，不能被实例化

需要被实现/继承，实现类/派生类必须重写接口中的使用抽象方法。

一个类可以实现多个接口，用逗号分隔，接口可以继承接口

1. 多态:

1) 意义:

1.1) 同一个对象被造型为不同的类型时，有不同的功能

----对象的多态: 我，你，水…

1.2) 同一类型的引用指向不同的对象时，有不同的实现

----行为的多态: cut(), run(), step(), getImage()

2) 向上造型:

2.1) 超类型的引用指向派生类的对象

2.2) 能造型成为的数据类型有: 超类+所实现的接口

2.3) 能点出来什么，看引用的类型

设计规则:

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

所有派生类行为不一样，设计为抽象方法

1. 将部分派生类所共有的行为，抽到接口中

是对类继承的单根性的扩展------------------实现多继承

正课:

1. 多态:

1) 意义:

1.1) 同一个对象被造型为不同的类型时，有不同的功能

----对象的多态: 我，你，水…

1.2) 同一类型的引用指向不同的对象时，有不同的实现

----行为的多态: cut(), run(), step(), getImage()

2) 向上造型/自动类型转换:

2.1) 超类型的引用指向派生类的对象

2.2) 能造型成为的数据类型有: 超类+所实现的接口

2.3) 能点出来什么，看引用的类型

3) 强制类型转换，能成功的条件只有如下两种情况:

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，实现了该接口或继承了该类

4) 强转时若不符合如上两种情况，则发生ClassCastException类型转换异常

建议: 在强转之前先通过instanceof来判断引用指向的对象是否是该类型

int score = 0; //得分

/\*\* 子弹与敌人的碰撞 \*/

**public** **void** bulletBangAction(){ //每10毫秒走一次

for(int i=0;i<bullets.length;i++){ //遍历所有子弹

Bullet b = bullets[i]; //获取每一个子弹

for(int j=0;j<enemies.length;j++){ //遍历所有敌人

FlyingObject f = enemies[j]; //获取每一个敌人

if(f.isLife() && b.isLife() && f.hit(b)){ //撞上了

f.goDead();

b.goDead();

Enemy e = (Enemy)f; //小敌机和大敌机好使，小蜜蜂发生类型转换异常

Award a = (Award)f; //小蜜蜂好使，小敌机和大敌机发生类型转换异常

//何时需要强转?-----当向上造型后，想访问的那个东西在超类中没有的时候需要强转

//------isLife()、hit()、goDead()这三个方法在超类中有，所以不需要强转

//------getScore()这个方法在超类中没有，所以想访问必须强转

//------------------------复用性好、扩展性好、维护性好

//Airplane(1)

//BigAirplane(3)

//BigYellowBee(6)

if(f instanceof Enemy){ //-------代表所有能得分的

Enemy e = (Enemy)f;

score += e.getScore();

}

//Bee()

//BigYellowBee()

if(f instanceof Award){ //-------代表所有能得奖励的

Award a = (Award)f;

int type = a.getAwardType();

switch(type){

case 0:

hero.addFire();

break;

case 1:

hero.addLife();

break;

}

}

//---------------------------复用性差、扩展性差、维护性差

if(f instanceof Airplane){ //只能代表小敌机

Airplane a = (Airplane)f;

score += a.getScore();

}

if(f instanceof BigAirplane){ //只能代表大敌机

BigAirplane ba = (BigAirplane)f;

score += ba.getScore();

}

if(f instanceof Bee){ //只能代表小蜜蜂

Bee be = (Bee)f;

int type = be.getAwardType();

switch(type){

case 0:

hero.addFire();

break;

case 1:

hero.addLife();

break;

}

}

if(f instanceof BigYellowBee){ //只能代表大黄蜂

BigYellowBee byb = (BigYellowBee)f;

score += byb.getScore();

int type = byb.getAwardType();

switch(type){

case 0:

hero.addFire();

break;

case 1:

hero.addLife();

break;

}

}

}

}

}

}

活着的，死了的，删除的

若撞上了：

1. 子弹去死、敌人去死
2. 若打掉的是小敌机，玩家得1分

大敌机，玩家得3分

小蜜蜂，英雄机得奖励(得1条命，40活力值)

LIFE状态变为DEAD状态----子弹getImage()中，DEAD时直接变为REMOVE了

敌人getImage()中，DEAD时先返回爆破图，再变为REMOVE

子弹与敌人的碰撞:------检测碰撞

英雄机与敌人的碰撞:----检测碰撞

class FlyingObject{ //敌人

//敌人撞子弹/英雄机

public boolean hit( FlyingObject other ){

this: 敌人

other: 子弹/英雄机

}

}

?处设计为什么类型

既能接收子弹对象

也能接收英雄机对象

能想到给子弹和英雄机设计接口--------非常好

能想到设计为超类FlyingObject---------非常好

class Bullet{ //子弹. --------碰撞检测放在子弹中不好是因为，后期做英雄机与敌人的碰撞时，此方法就用不上了

//子弹撞敌人

public boolean hit( FlyingObject other ){

this: 子弹

other: 敌人

}

}

现在只讲了多态的语法------具体的强转的应用下午案例中讲

造型之后能点出来的东西少了，如果还想点很多的话，可以再把它强转回来

超类大，派生类小

大 小

FlyingObject o = new Airplane();

向上造型之后----能点出来的东西是多了还是少了?-------或少

因为派生类的东西多，超类的东西少

interface Inter1{

}

interface Inter2{

}

interface Inter3 extends Inter1,Inter2{ //正确

}

项目功能: