面向对象第9天：

潜艇游戏第一天：

1. 创建6个类，创建World类并测试

潜艇游戏第二天：

1. 给6个类添加构造方法，并测试

潜艇游戏第三天：

1. 创建侦察潜艇数组、鱼雷潜艇数组、水雷潜艇数组，水雷数组，炸弹数组，并测试
2. 设计SeaObject超类，6个类继承超类
3. 在SeaObject中设计两个构造方法，6个派生类分别调用

潜艇游戏第四天：

1. 将侦察潜艇数组、鱼雷潜艇数组、水雷潜艇数组统一组合为SeaObject超类数组，并测试
2. 在6个类中重写move()移动，并测试
3. 画窗口:------共3步

潜艇游戏第五天：

1. 给类中成员添加访问控制修饰符
2. 设计Images图片类

潜艇游戏第六天：

1. 设计窗口的宽和高为常量，适当地方做修改
2. 画对象：----------能够按照我的笔记步骤写出来，就OK了

```
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，所以设计在SeaObject超类中，
  每个对象获取图片的代码都是不一样的，所以设计为抽象方法
  ----在SeaObject中设计为抽象方法getImage()获取对象的图片
2)在派生类中重写getImage()获取对象图片
  ----在6个类中重写getImage()返回不同的图片
3)因为只有活着的对象才需要画到窗口中，所以需要设计对象的状态(活着还是死了)，
  每个对象都有状态，意味着状态为共有属性，所以设计在SeaObject超类中，
  状态一般都设计为常量，同时再设计state变量表示当前状态
  ----在SeaObject中设计LIVE、DEAD常量，state变量表示当前状态
  在后期的业务中经常需要判断对象的状态，每个对象都得判断，
  意味着判断状态的行为为共有行为，所以设计在SeaObject超类中，
  每个对象判断状态的代码都是一样的，所以设计为普通方法
  ----在SeaObject中设计isLive()、isDead()判断对象的状态
4)数据(状态、图片、x坐标、y坐标)都有了就可以开画了，每个对象都得画，
  意味着画对象行为为共有行为，所以设计在SeaObject超类中，
  每个对象画的代码都是一样的，所以设计为普通方法
  ----在SeaObject中设计paintImage()画对象
5)画对象的行为做好了，在窗口world类中调用即可
```

```
5.1)准备对象
5.2)重写paint()方法-----在paint()中调用paintImage()画对象即可
```

潜艇游戏第七天：-----能够按照我的步骤写出来就可以

1. 潜艇入场：
   - 潜艇是由窗口产生的，所以在窗口World类中设计nextSubmarine()生成潜艇对象
   - 潜艇入场为定时发生的，所以在run()中调用submarineEnterAction()实现潜艇入场

     在submarineEnterAction()中：

     每400毫秒，获取潜艇对象obj，submarines扩容，将obj添加到最后一个元素上

     > 注意：在run()中调用submarineEnterAction()之后，一定要调用repaint()来重画
2. 水雷入场：---------前半段
   - 水雷是由水雷潜艇发射出来的，所以在MineSubmarine中设计shootMine()生成水雷对象
   - 水雷入场为定时发生的，所以在run()中调用mineEnterAction()实现水雷入场

     在mineEnterAction()中：

     每1000毫秒，.....----暂时搁置
3. 海洋对象移动：
   - 海洋对象移动为共有行为，所以在SeaObject中设计抽象方法move()实现移动，派生类中重写
   - 海洋对象移动为定时发生的，所以在run()中调用moveAction()实现海洋对象移动

     在moveAction()中：

     遍历所有潜艇--潜艇动，遍历所有水雷--水雷动，遍历所有炸弹--炸弹动

潜艇游戏第八天：-----能够按照我的步骤写出来就可以

1. 炸弹入场：
   - 炸弹是由战舰发射出来的，所以在Battleship中设计shootBomb()生成炸弹对象
   - 炸弹入场为事件触发的，所以在侦听器中重写keyReleased()按键抬起事件，在抬起事件中：
     - 判断若抬起的是空格键，则：

       获取炸弹对象obj，bombs扩容，将obj添加到bombs的最后一个元素上
2. 战舰移动：
   - 战舰移动为战舰的行为，所以在Battleship中设计moveLeft()左移、moveRight()右移
   - 战舰移动为事件触发的，所以在侦听器的重写keyReleased()按键抬起事件中：
     - 判断若抬起的是左箭头，则战舰左移
     - 判断若抬起的是右箭头，则战舰右移
3. 删除越界的海洋对象：
   - 在SeaObject中设计isOutOfBounds()检测潜艇是否越界，在Bomb和Mine中重写isOutOfBounds()检测炸弹和水雷是否越界
   - 删除越界海洋对象为定时发生的，所以在run()中调用outOfBoundsAction()删除越界海洋对象

     在outOfBoundsAction()中：

     遍历所有潜艇/水雷/炸弹数组，判断若越界了：

将越界元素替换为数组的最后一个元素，缩容

4. 设计EnemyScore得分接口，侦察潜艇和鱼雷潜艇实现得分接口

   设计EnemyLife得命接口，水雷潜艇实现得命接口

潜艇游戏第九天：-----能够按照我的步骤写出来就可以

1. 水雷入场：
   - 水雷是由水雷潜艇发射出来的，所以在MineSubmarine中设计shootMine()生成水雷对象
   - 水雷入场为定时发生的，所以在run()中调用mineEnterAction()实现水雷入场

     在mineEnterAction()中：

     每1000毫秒，遍历所有潜艇，判断若为水雷潜艇，则：

     将潜艇转换为水雷潜艇类型，获取水雷对象obj，mines扩容，将obj装到最后一个元素上
2. 炸弹与潜艇的碰撞：
   - 在SeaObject中设计isHit()检测碰撞、goDead()去死，在Battleship中设计addLife()增命
   - 炸弹与潜艇的碰撞为定时发生的，所以在run()中设计bombBangAction()实现炸弹与潜艇的碰撞

     在bombBangAction()中：

     遍历所有炸弹获取炸弹，遍历所有潜艇获取潜艇，判断若都活着并且还撞上了：

     炸弹去死、潜艇去死

     判断若是分，则强转为得分接口，玩家得分

     判断若是命，则强转为得命接口，获取命数，战舰得命
3. 画分和画命：
   - 在Battleship中设计getLife()获取命数
   - 在World类的paint()中：画分和画命------不要求掌握

回顾：

1. 接口：

精华笔记：

1. 多态：多种形态
   - 同一个对象被造型为不同的类型时，有不同的功能-------所有对象都是多态的(明天总结详细讲)
     - 对象的多态：水、我、你.....

     同一类型的引用在指向不同的对象时，有不同的实现----所有抽象方法都是多态的
     - 行为的多态：cut()、getImage()、move().....
   - 向上造型/自动类型转换：
     - 超类型的引用指向派生类的对象------前面是超类型，后面是派生类型
     - 能点出来什么，看引用的类型
     - 能造型成为的数据类型：超类+所实现的接口
   - 强制类型转换，成功的条件只有两种：

    - 引用所指向的对象，就是该类型
    - 引用所指向的对象，实现了该接口或继承了该类
- 强转时若不满足如上条件，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

  > 说明：instanceof会返回true或false的结果
  >
  > 如果满足强转成功的条件则返回true，否则返回false
  >
  > 何时需要强转：向上造型后，若想访问的东西在超类中没有，则需要强转

笔记：

1. 多态：多种形态
    - 同一个对象被造型为不同的类型时，有不同的功能------所有对象都是多态的(明天总结详细讲)
        - 对象的多态：水、我、你.....

      同一类型的引用在指向不同的对象时，有不同的实现----所有抽象方法都是多态的

        - 行为的多态：cut()、getImage()、move().....
    - 向上造型/自动类型转换：
        - 超类型的引用指向派生类的对象------前面是超类型，后面是派生类型
        - 能点出来什么，看引用的类型
        - 能造型成为的数据类型：超类+所实现的接口
    - 强制类型转换，成功的条件只有两种：
        - 引用所指向的对象，就是该类型
        - 引用所指向的对象，实现了该接口或继承了该类
    - 强转时若不满足如上条件，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

      > 说明：instanceof会返回true或false的结果
      >
      > 如果满足强转成功的条件则返回true，否则返回false
      >
      > 何时需要强转：向上造型后，若想访问的东西在超类中没有，则需要强转

```
public class MultiTypeDemo {
    public static void main(String[] args) {
        //成功的条件1:引用所指向的对象，就是该类型
        //成功的条件2:引用所指向的对象，实现了该接口或继承了该类
        Aoo o = new Boo(); //向上造型
        Boo o1 = (Boo)o; //引用o所指向的对象，就是Boo类型-----------符合条件1
        Inter o2 = (Inter)o; //引用o所指向的对象，实现了Inter接口---符合条件2
        //Coo o3 = (Coo)o; //运行时会发生ClassCastException类型转换异常
        if(o instanceof Coo){ //false
            Coo o4 = (Coo)o;
        }else{
            System.out.println("o不是Coo类型");
        }

    }
}
interface Inter{ }
```

```
class Aoo{ }
class Boo extends Aoo implements Inter{ }
class Coo extends Aoo{ }
```

补充:

1. 体会接口的好处:

```
//复用性好、扩展性好、维护性好------------------高质量代码
//被撞的是ObserveSubmarine-----调用ObserveSubmarine的getScore()-----10分
//被撞的是TorpedoSubmarine-----调用TorpedoSubmarine的getScore()-----40分
//被撞的是NuclearSubmarine-----调用NuclearSubmarine的getScore()-----100分
if(s instanceof EnemyScore){ //------适用于所有实现EnemyScore接口的
    EnemyScore es = (EnemyScore)s;
    score += es.getScore();
}
//被撞的是MineSubmarine--------调用MineSubmarine的getLife()---------1
//被撞的是NuclearSubmarine-----调用NuclearSubmarine的getLife()------3
if(s instanceof EnemyLife){ //-------适用于所有实现EnemyLife接口的
    EnemyLife el = (EnemyLife)s;
    int num = el.getLife();
    ship.addLife(num);
}

//复用性差、扩展性差、维护性差------------------垃圾代码
if(s instanceof ObserveSubmarine){ //---------只能适用于ObserveSubmarine的
    ObserveSubmarine os = (ObserveSubmarine)s;
    score += os.getScore();
}
if(s instanceof TorpedoSubmarine){ //---------只能适用于TorpedoSubmarine的
    TorpedoSubmarine ts = (TorpedoSubmarine)s;
    score += ts.getScore();
}
if(s instanceof MineSubmarine){    //---------只能适用于MineSubmarine的
    MineSubmarine ms = (MineSubmarine)s;
    int num = ms.getLife();
    ship.addLife(num);
}
if(s instanceof NuclearSubmarine){ //---------只能适用于NuclearSubmarine
    NuclearSubmarine ns = (NuclearSubmarine)s;
    score += ns.getScore();
    int num = ns.getLife();
    ship.addLife(num);
}
```

2. 明日单词:

```
1)subtract:减
2)gameover:结束
3)running:运行
```