面向对象第10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：-----能够按照我的步骤写出来就可以

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

潜艇游戏第十天：-----能够按照我的步骤写出来就可以

1. 水雷与战舰的碰撞：
   - 在Battleship中设计subtractLife()减命
   - 水雷与战舰的碰撞为定时发生的，所以在run()中调用mineBangAction()来实现水雷与战舰的碰撞

     在mineBangAction()中：

     遍历所有水雷得水雷，判断若都活着并且还和战舰撞上了：

     水雷去死、战舰减命
2. 检测游戏结束：
   - 借用Battleship中的getLife()获取命
   - 检测游戏结束为定时发生的，所以在run()中调用checkGameOverAction()实现检测游戏结束

     在checkGameOverAction()中：

     判断若战舰的命数<=0，表示游戏结束，则.....
3. 画状态：

- 在World类中设计RUNNING、PAUSE、GAME_OVER状态常量，state变量表示当前状态

  在checkGameOverAction()中，设计若游戏结束，则将state修改为GAME_OVER游戏结束状态

  在paint()中，设计若当前状态为游戏结束状态时，画游戏结束图
- 将run中的那一堆代码设计为，仅在运行状态时执行

  按按空格、左移、右移代码设计为，仅在运行状态时执行

  设计按下P键后，运行状态时修改为暂停状态，暂停状态时修改为运行状态

回顾：

1. 多态：
   - 对象多态(所有对象都是多态的)、行为多态
   - 向上造型：超类+所实现的接口
   - 如果想访问的东西在超类中没有，那就需要强制类型转换，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
   - 强转时若不符合如上条件，则发生ClassCastException类型转换异常，

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

精华笔记：

1. 内存管理：由JVM管理
   - 堆：
     - 存储new出来的对象(包括实例变量、数组的元素)
     - 垃圾：没有任何引用所指向的对象

       垃圾回收器(GC)不定时到内存堆中清扫垃圾，回收过程是透明的(看不到的)，

       不一定一发现垃圾就立刻回收，通过调用System.gc()可以建议虚拟机尽快调度GC来回收
     - 实例变量的生命周期：

       创建(new)对象时存储在堆中，对象被回收时一并被回收
     - 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：不再使用的对象应及时将引用设置为null
   - 栈：
     - 存储正在调用的方法中的局部变量(包括方法的参数)
     - 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(包括方法的参数)，

       方法调用结束时，栈帧被自动清除，局部变量一并被清除。
     - 局部变量的生命周期：

       调用方法时存储在栈中，方法结束时与栈帧一并被清除
   - 方法区：

        - 存储.class字节码文件(包括静态变量、所有方法)
        - 方法只有一份，通过this来区分具体的调用对象
2. 面向对象三大特征总结：
    - 封装：
        - 类：封装的是对象的属性和行为
        - 方法：封装的是具体的业务逻辑功能实现
        - 访问控制修饰符：封装的是具体的访问权限
    - 继承：
        - 作用：代码复用
        - 超类：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属性和行为

          派生类：派生类所特有的属性和行为
        - 单一继承、多接口实现，具有传递性
    - 多态：
        - 所有对象都是多态的，通过向上造型来体现

          所有抽象方法都是多态的，通过方法的重写来体现
        - 向上造型、强制类型转换、instanceof判断
3. String字符串类型：
    - java.lang.String类使用final修饰，不能被继承
    - String的底层封装的是一个字符数组
    - String在内存中采用Unicode编码格式，每个字符占用2个字节的空间
    - 字符串对象一旦创建，对象内容永远无法改变，但字符串引用可以重新赋值(指向新的对象)
        - 不变对象
4. 字符串常量池：
    - java对String字符串有一个优化措施：字符串常量池(堆中)
    - java推荐我们使用字面量/直接量(直接"")的方式来创建对象，并且会缓存所有以字面量形式创建的字符串对象到常量池中，当使用相同字面量再创建对象时将会复用常量池中的对象，以减少内存开销

笔记：

1. 内存管理：由JVM管理
    - 堆：
        - 存储new出来的对象(包括实例变量、数组的元素)
        - 垃圾：没有任何引用所指向的对象

          垃圾回收器(GC)不定时到内存堆中清扫垃圾，回收过程是透明的(看不到的)，

          不一定一发现垃圾就立刻回收，通过调用System.gc()可以建议虚拟机尽快调度GC来回收
        - 实例变量的生命周期：

            创建(new)对象时存储在堆中，对象被回收时一并被回收
        - 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：不再使用的对象应及时将引用设置为null

- 栈：
  - 存储正在调用的方法中的局部变量(包括方法的参数)
  - 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(包括方法的参数)，

    方法调用结束时，栈帧被自动清除，局部变量一并被清除。
  - 局部变量的生命周期：

    调用方法时存储在栈中，方法结束时与栈帧一并被清除
- 方法区：
  - 存储.class字节码文件(包括静态变量、所有方法)
  - 方法只有一份，通过this来区分具体的调用对象

2. 面向对象三大特征总结：
   - 封装：
     - 类：封装的是对象的属性和行为
     - 方法：封装的是具体的业务逻辑功能实现
     - 访问控制修饰符：封装的是具体的访问权限
   - 继承：
     - 作用：代码复用
     - 超类：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属性和行为

       派生类：派生类所特有的属性和行为
     - 单一继承、多接口实现，具有传递性
   - 多态：
     - 所有对象都是多态的，通过向上造型来体现

       所有抽象方法都是多态的，通过方法的重写来体现
     - 向上造型、强制类型转换、instanceof判断

3. String字符串类型：
   - java.lang.String类使用final修饰，不能被继承
   - String的底层封装的是一个字符数组
   - String在内存中采用Unicode编码格式，每个字符占用2个字节的空间
   - 字符串对象一旦创建，对象内容永远无法改变，但字符串引用可以重新赋值(指向新的对象)
     - 不变对象

4. 字符串常量池：
   - java对String字符串有一个优化措施：字符串常量池(堆中)
   - java推荐我们使用字面量/直接量(直接"")的方式来创建对象，并且会缓存所有以字面量形式创建的字符串对象到常量池中，当使用相同字面量再创建对象时将会复用常量池中的对象，以减少内存开销

```
/*
    使用字面量来创建字符串对象时，JVM会检查常量池中是否有该对象：
    1)若没有，则会创建该字符串对象，并存入常量池中
    2)若有，则直接将常量池中的对象(引用)返回---并不会创建新的字符串对象
```

```
*/
String s1 = "123abc"; //常量池还没有，因此创建该字符串对象，并存入常量池中
String s2 = "123abc"; //常量池中已经有了，直接复用对象
String s3 = "123abc"; //常量池中已经有了，直接复用对象
//引用类型==，比较的是地址是否相同-----这是规定
System.out.println(s1==s2); //true
System.out.println(s1==s3); //true
System.out.println(s2==s3); //true

s1 = s1 + "!"; //创建新的字符串对象(123abc!)并将地址赋值给s1
System.out.println(s1==s2); //false
```

补充:

1. 明日单词:

```
1)last:最后的
2)trim:剪去、截掉
3)start:开始
4)end:结束
5)uppercase:大写字母
6)lowercase:小写字母
7)value:值
8)builder:建造
9)append:追加
10)replace:替换
11)delete:删除
12)insert:插入
```