面向对象第十天：

潜艇游戏第十天：

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

      水雷去死、战舰减命
2. 检测游戏结束：
    - 借用Battleship的getLife()获取命数
    - 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：

      判断若战舰的命数<=0，表示游戏结束了
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，若游戏结束，则将当前状态修改为GAME_OVER游戏结束状态
    - 在paint()中设计：若为游戏结束状态则画游戏结束图
    - 设计run()中那一堆代码，为仅在运行状态下执行
    - 设计键盘抬起时，按下空格键、左移键、右移键，也为仅在运行状态下执行
    - 在键盘抬起时，设计若按的是P键，则运行状态变暂停状态，暂停状态变运行状态

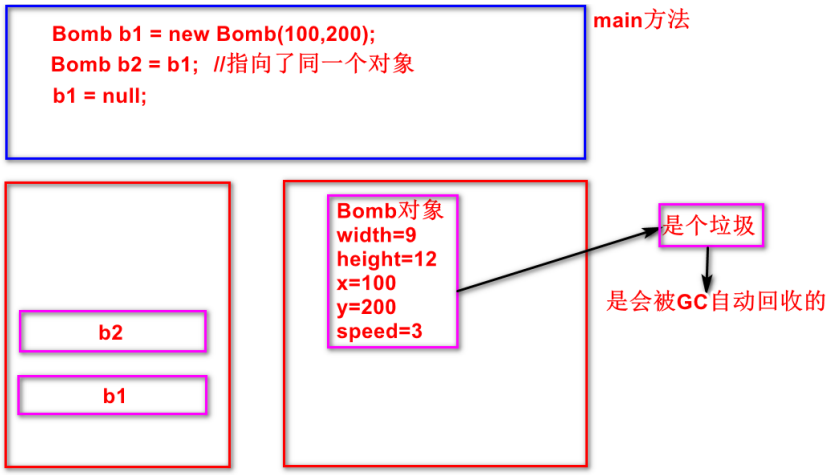
笔记：

1. 内存管理：由JVM来管理
    - 堆：
        - 存储的是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：Garbage Collector)不定时到堆中清扫垃圾，回收过程是看不到的，并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：

          在创建对象时存储在堆中，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回收，严重的内存泄漏会导致系统崩溃

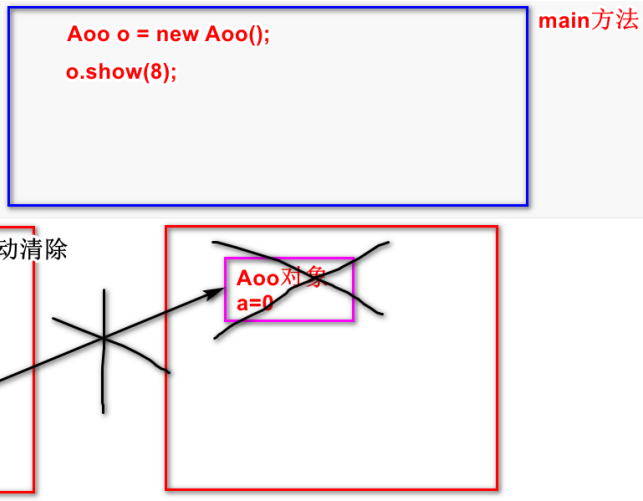
          建议：不再使用的对象应及时将引用设置为null



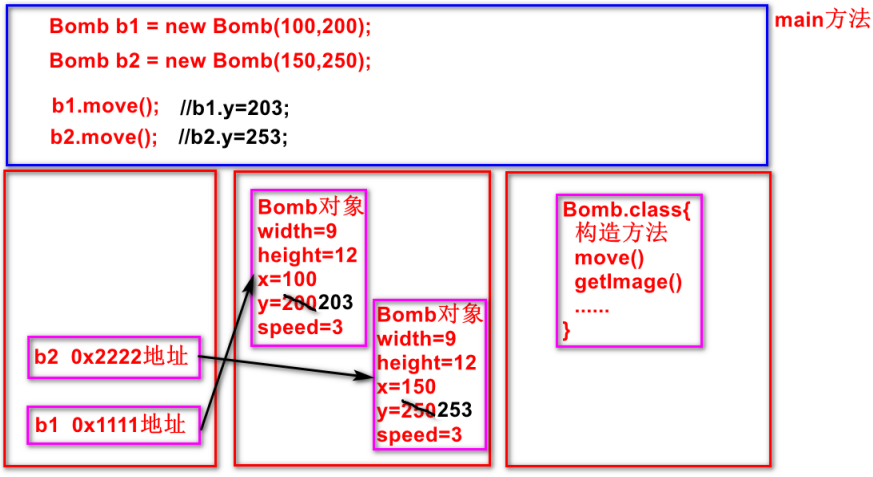
- 栈：
  - 存储正在调用的方法中的局部变量(包括方法的参数)
  - 调用方法时会在栈中为该方法分配一块对应的栈帧，栈帧中存储局部变量(包括方法的参数)，方法调用结束时，栈帧被自动清除，局部变量一并被清除
  - 局部变量的生命周期：

    调用方法时存储在栈中，方法调用结束时与栈帧一并被清除

```
class Aoo{
    int a;
    void show(int b){
        int c = 5;
    }
}
```



- 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：（非常重要，一定要记住！）
   - 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：给封装的东西加具体的访问权限
    - 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的（Object类是所有类的超类）

          所有抽象方法都是多态的：通过方法的重写来体现的
        - 向上造型、强制类型转换（保证能点出来）、instanceof（强转前先判断）

3. String：字符串类型
    - java.lang.String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，每个字符占用两个字节的空间
    - 字符串一旦创建，new的对象的内容永远无法改变，但字符串引用可以重新赋值(指向新的对象的地址)

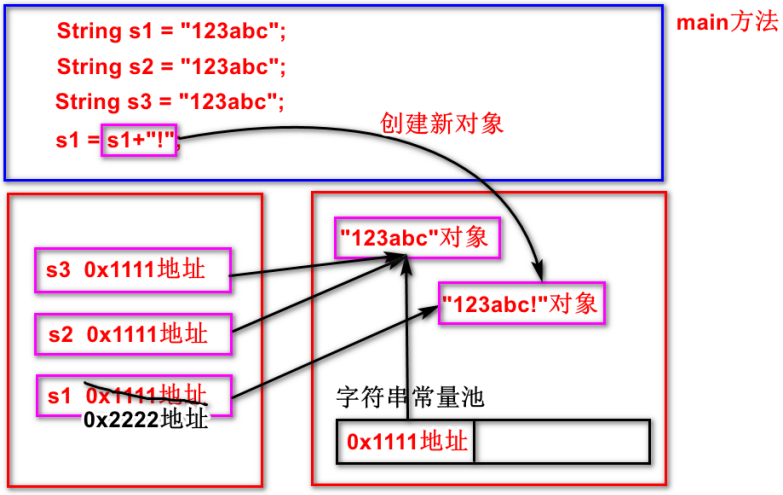
      （不变对象）

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(在堆中)
    - java推荐我们使用字面量/直接量的方式来创建对象，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。

      > 注意：只有使用字面量方式创建的对象，才会存储在字符串常量池中

```
public class StringDemo {
    public static void main(String[] args) {
            /*
              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（不会创建新的字符串对象）
             */
        String s1 = "123abc"; //常量池中还没有，因此创建该字符串对象，并存入常量池
        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
        //引用类型去做比较相等运算==，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，(123abc!)不会在常量池中创建
        System.out.println(s1 == s2); //false
    }
}
```




```
public class StringDemo {
    public static void main(String[] args) {
        String s1 = "123abc"; //堆中创建一个123abc对象，常量池中存储这个对象的引用
        //编译器在编译时，若发现是两个字面量连接，则直接运算好并将结果保存起来，
        //如下代码相当于 String s2 = "123abc";
        String s2 = "123" + "abc"; //复用常量池中的123abc对象
        System.out.println(s1 == s2); //true

        //如下代码会在堆中创建新的123abc对象，而不会重用常量池中的对象
        String s3 = "123";
        String s4 = s3 + "abc";
        System.out.println(s1 == s4); //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
```