面向对象第5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

笔记：

1. package：声明包
    - 作用：避免类的命名冲突
    - 同包中的类不能同名，但不同包中的类可以同名
    - 类的全称：包名.类名，常常有层次结构
    - 建议：包名所有字母都小写

   import：导入类、引入类
    - 同包中的类可以直接访问，但不同包中的类不能直接访问，若想访问：
        - 先import导入类再访问类（建议）
        - 类的全称（太繁琐，不建议）

   > 注意：
   >
   > 1. 顺序问题：package--->import--->class
   > 2. import 包名.*; 表示导入了包中的所有类，但不建议，建议用哪个类导哪个类，因为.*会影响性能

2. 访问控制修饰符：（保护数据的安全）
    - public：公开的，任何类
    - protected：受保护的，本类、同包类、派生类
    - 默认的：什么也不写，本类、同包类

- private：私有的，本类

> 注意：
>
> 1. java不建议默认权限
> 2. 类的访问权限只能是public或默认，类中成员的访问权限以上4种都可以
> 3. 访问权限由大到小依次为：public--->protected--->默认的--->private

```
public class Card { //银行卡
    private String cardId;  //卡号
    private String cardPwd; //密码
    private double balance; //余额

    public boolean payMoney(double money) { //支付金额
        if (balance >= money) {
            balance -= money;
            return true;
        } else {
            return false;
        }
    }

    public boolean checkPwd(String pwd) { //检测密码（营业员可以调用）
        if (pwd == cardPwd) {
            return true;
        } else {
            return false;
        }
    }
}
```

```
//访问控制修饰符的演示

package ooday05;

public class Aoo {
    public int a;    //任何类
    protected int b; //本类、同包类、派生类
    int c;           //本类、同包类
    private int d;   //本类

    void show() {
        a = 1;
        b = 2;
        c = 3;
        d = 4;
    }
}

//演示private
class Boo { //同包类
    void show() {
        Aoo o = new Aoo();
        o.a = 1;
        o.b = 2;
        o.c = 3;
        //o.d = 4; //编译错误
```

```
    }
}
```

```
//访问控制修饰符的演示-接上

package ooday05_vis;

import ooday05.Aoo;

//演示public
public class Coo { //不同包的类
    void show() {
        Aoo o = new Aoo();
        o.a = 1;
        //o.b = 2; //编译错误
        //o.c = 3; //编译错误
        //o.d = 4; //编译错误
    }
}

//演示protected
class Doo extends Aoo { //跨包继承
    void show() {
        a = 1;
        b = 2;
        //c = 3; //编译错误
        //d = 4; //编译错误
    }
}
```

3. final：最终的、不可改变的（单独应用的几率低）
    - 修饰变量：变量不能被改变

```
//演示final修饰变量
class Eoo {
    final int a = 5;

    void show() {
        //a = 55; //编译错误，final的变量不能被改变
    }
}
```

    - 修饰方法：方法不能被重写

```
//演示final修饰方法
class Foo {
    final void show() {}
    void test() {}
}

class Goo extends Foo {
    //void show(){} //编译错误，final的方法不能被重写
    void test() {}
}
```

- 修饰类：类不能被继承

```
//演示final修饰类
final class Hoo {}

//class Ioo extends Hoo{} //编译错误，final的类不能被继承
class Joo {}

final class Koo extends Joo {} //正确
```

4. static：静态的
   - 静态变量：
     - 由static修饰
     - 属于类，存储在方法区中，只有一份
     - 常常通过类名点来访问（如下：Loo.b）
     - 何时用：所有对象所共享的数据(图片、音频、视频等)

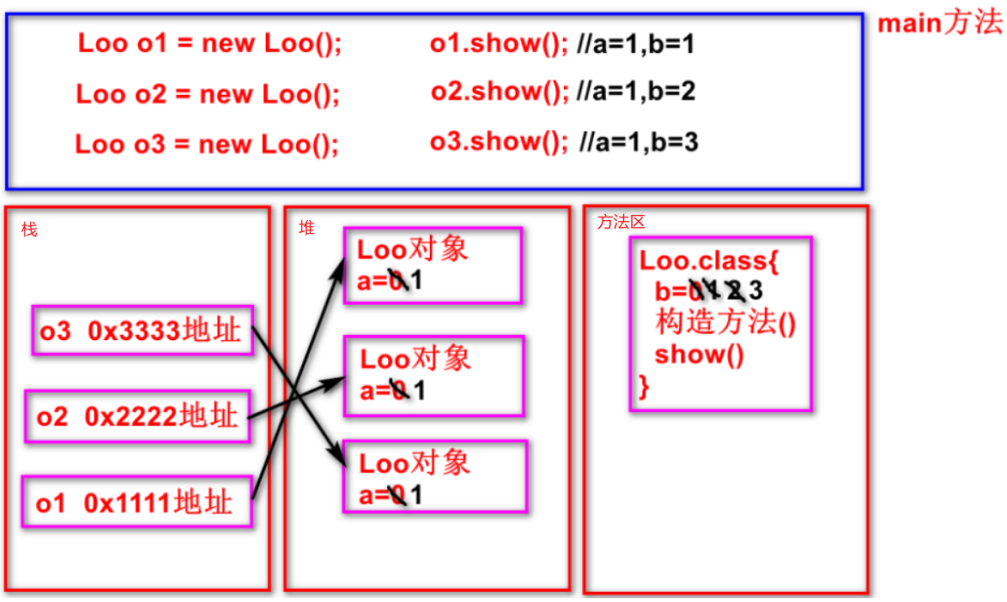
```
class Loo {
    int a;
    static int b;

    Loo() {
        a++;
        b++;
    }

    void show() {
        System.out.println("a=" + a + ",b=" + b);
    }
}

public class StaticDemo {
    public static void main(String[] args) {
        Loo o1 = new Loo();
        o1.show();
        Loo o2 = new Loo();
        o2.show();
        Loo o3 = new Loo();
        o3.show();
        System.out.println(Loo.b); //常常通过"类名.变量名"来访问
    }
}
```

- 内存图（上述代码）：



- 静态方法：
  - 由static修饰
  - 属于类，存储在方法区中，只有一份
  - 常常通过类名点来访问（如下：Moo.test()）
  - 静态方法中没有隐式this传递，所以不能直接访问实例成员
  - 何时用：当方法的操作与对象无关时

```
//演示静态方法
class Moo {
    int a; //实例变量(对象来访问)
    static int b; //静态变量(类名点来访问)

    void show() { //有隐式this
        System.out.println(this.a);
        System.out.println(Moo.b);
    }

    static void test() { //没有隐式this
        //静态方法中没有隐式this传递，没有this就意味着没有对象，而实例变量a是必须通过对象点来访问的，所以如下语句发生编译错误
        //System.out.println(a); //编译错误
        System.out.println(Moo.b);
    }
}

//演示静态方法何时用
class Noo {
    int a; //实例变量（属于对象，描述对象的属性）

    //show()中需要访问对象的属性a，说明此方法与对象有关，不能设计为静态方法
    void show() {
        System.out.println(a);
    }

    //plus()中没有访问对象的属性或行为，说明此方法与对象无关，可以设计为静态方法
    static int plus(int num1, int num2) {
```

```
        int num = num1 + num2;
        return num;
    }
}

public class StaticDemo {
    public static void main(String[] args) {
        Moo.test(); //常常通过类名点来访问
    }
}
```

- 静态块：
  - 由static修饰
  - 属于类，在类被加载期间自动执行，因为一个类只被加载一次，所以静态块也只执行一次
  - 何时用：加载/初始化静态资源(图片、音频、视频等)

```
//演示静态块
class Poo {
    static {
        System.out.println("静态块");
    }

    Poo() {
        System.out.println("构造方法");
    }
}

public class StaticDemo {
    public static void main(String[] args) {
        Poo o4 = new Poo();
        Poo o5 = new Poo();
        Poo o6 = new Poo();
    }
}
```

补充：

1. 数据(成员变量)私有化(private)，行为(方法)大部分公开化(public)
2. 成员变量分两种：实例变量和静态变量
   - 实例变量：没有static修饰，属于对象的，存储在堆中，有几个对象就有几份，通过引用打点来访问
   - 静态变量：由static修饰，属于类的，存储在方法区中，只有一份，通过类名点来访问
3. 内存管理：由JVM管理的
   - 堆：new出来的对象(包括实例变量、数组的元素)
   - 栈：局部变量(包括方法的参数)
   - 方法区：.class字节码文件(包括静态变量、所有方法)
4. 一般情况下，凡是静态的成员，都是公开的
5. 明日单词：

```
1)graphics:图像/画笔
2)PI:圆周率
3)count:数量
4)abstract:抽象的
5)live:活着的
6)dead:死了的
7)state:状态
8)is:是
9)paint:画
```

6. 自己补充：不同包的同名类，调用规则：import的类优先