面向对象第五天：

回顾：

1. 多态：
    - 向上造型：
        - 超类型的引用指向派生类的对象
        - 能点出来什么，看引用的类型
        - 能造型成为的数据类型有：超类+所实现的接口
    - 向下转型/强制类型转换，成功的条件只有如下两种：
        - 引用所指向的对象，就是该类型
        - 引用所指向的对象，继承了该类或实现了该接口
    - 强转时若不符合如上条件，则发生ClassCastException类型转换异常

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

      > 若想访问派生类所特有的属性/行为时，需要强制类型转换/向下转型
2. 成员内部类：应用率低
    - 类中套类，内部类对外不具备可见性，内部类对象一般在外部类中创建
    - 内部类中可以直接访问外部类的成员----外部类名.this指代创建它的外部类对象
    - 何时用：类A只让类B用，在类A中还需要访问类B的东西，可以将类A设计为成员内部类
3. 匿名内部类：应用率高----------大大简化代码
    - 若想创建一个派生类的对象，并且对象只被创建一次，可以设计为匿名内部类
4. package和import：

精华笔记：

1. 访问控制修饰符：----------保护数据的安全(隐藏数据、暴露行为)，实现封装
    - public：公开的，任何类
    - private：私有的，本类
    - protected：受保护的，本类、派生类、同包类
    - 默认的：什么也不写，本类、同包类----------java不建议

      > 注意：
      >
      > 1. 访问权限由低到高依次为：private<默认的<protected<public
      > 2. 类的访问权限只能是public或默认的，类中成员的访问权限如上4种都可以。
2. final：最终的、不能改变的----------单独应用几率低
    - 修饰变量：变量不能被改变
    - 修饰方法：方法不能被重写
    - 修饰类：类不能被继承
3. 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
        - 何时用：对象所共享的数据
    - 静态块：
        - 由static修饰
        - 属于类，在类被加载期间自动执行，一个类只被加载一次，所以静态块也只执行一次
        - 何时用：初始化/加载静态资源/静态变量
    - 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
        - 静态方法中没有隐式this传递，所以静态方法中不能直接访问实例成员(实例变量/实例方法)
        - 何时用：方法的操作与对象无关(不需要访问对象的属性/行为)
4. static final常量：应用率高
    - 必须声明同时初始化
    - 常常通过类名点来访问，不能被改变
    - 建议：常量名所有字母都大写，多个单词之间用_分隔
    - 编译器在编译时，会将常量直接替换为具体的数，效率高
    - 何时用：在程序运行过程中数据永远不变，并且经常使用
5. 枚举：
    - 是一种引用数据类型
    - 特点：枚举类型的对象数目是固定的，常常用于定义一组常量

      > 例如：季节、星期、月份、订单状态、性别.....
    - 所有枚举都继承自Enum类，其中提供了一组方法供我们使用
    - 枚举的构造方法都是私有的

笔记：

1. 访问控制修饰符：----------保护数据的安全(隐藏数据、暴露行为)，实现封装
    - public：公开的，任何类
    - private：私有的，本类
    - protected：受保护的，本类、派生类、同包类
    - 默认的：什么也不写，本类、同包类----------java不建议

      > 注意：
      >
      > 1. 访问权限由低到高依次为：private<默认的<protected<public
      > 2. 类的访问权限只能是public或默认的，类中成员的访问权限如上4种都可以。

```
package ooday05;
public class Aoo {
    public int a;    //任何类
    protected int b; //本类、派生类、同包类
    int c;           //本类、同包类
```

```
    private int d;   //本类

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

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

package ooday05_vis;
import ooday05.Aoo;
public class Coo { //--------------------演示同包的
    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
    }
}

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

2. final：最终的、不能改变的----------单独应用几率低
    - 修饰变量：变量不能被改变

```
//演示final修饰变量
class Eoo{
    final int num = 5;
    void show(){
        //num = 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

```
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

```
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，不能当老爸，但能当儿子
```

3. 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
     - 何时用：对象所共享的数据

```
public class StaticVar {
    int a; //实例变量
    static int b; //静态变量
    StaticVar(){
        a++;
        b++;
    }
    void show(){
        System.out.println("a="+a+", b="+b);
    }
}

public class StaticDemo {
    public static void main(String[] args) {
        StaticVar o1 = new StaticVar();
        o1.show();
        StaticVar o2 = new StaticVar();
        o2.show();
        StaticVar o3 = new StaticVar();
        o3.show();
        System.out.println(StaticVar.b); //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，一个类只被加载一次，所以静态块也只执行一次

- 何时用：初始化/加载静态资源/静态变量

```
public class StaticBlock {
    static{
        System.out.println("静态块");
    }
    StaticBlock(){
        System.out.println("构造方法");
    }
}

public class StaticDemo {
    public static void main(String[] args) {
        StaticBlock o4 = new StaticBlock(); //加载类时自动执行静态块
        StaticBlock o5 = new StaticBlock();
        StaticBlock o6 = new StaticBlock();
    }
}
```

- 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
  - 静态方法中没有隐式this传递，所以静态方法中不能直接访问实例成员(实例变量/实例方法)
  - 何时用：方法的操作与对象无关(不需要访问对象的属性/行为)

```
public class StaticMethod {
    int a; //实例变量(对象来访问)--------------属于对象的
    static int b; //静态变量(类名来访问)-------属于类的

    //方法的操作与对象无关(不需要访问对象的属性/行为)

    //在say()中需要访问对象的属性a，所以认为say的操作与对象有关，不适合设计为静态方法
    void say(){
        System.out.println(a);
    }
    //在plus()中不需要访问对象的属性/行为，所以认为plus的操作与对象无关，可以设计为静态方法
    static int plus(int num1,int num2){
        int num = num1+num2;
        return num;
    }

    void show(){ //有隐式this
        System.out.println(this.a);
        System.out.println(StaticMethod.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必须通过对象来访问
        //所以如下语句发生编译错误
        //System.out.println(a); //编译错误，静态方法中不能直接访问实例成员
        System.out.println(StaticMethod.b);
    }
}

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

4. static final常量：应用率高
    - 必须声明同时初始化
    - 常常通过类名点来访问，不能被改变
    - 建议：常量名所有字母都大写，多个单词之间用_分隔
    - 编译器在编译时，会将常量直接替换为具体的数，效率高
    - 何时用：在程序运行过程中数据永远不变，并且经常使用

```
public class StaticFinalDemo {
    public static void main(String[] args) {
        System.out.println(Loo.PI); //常常通过类名点来访问
        //Loo.PI = 3.1415926; //编译错误，常量不能被改变

        //1)加载Loo.class到方法区中
        //2)静态变量num一并存储到方法区中
        //3)到方法区中获取num的值并输出
        System.out.println(Loo.num);

        //编译器在编译时会将常量直接替换为具体的数，效率高
        //相当于System.out.println(5);
        System.out.println(Loo.COUNT);
    }
}

class Loo{
    public static int num = 5; //静态变量
    public static final int COUNT = 5; //常量(静态常量)

    public static final double PI = 3.14159;
    //public static final int NUM; //编译错误，常量必须声明同时初始化
}
```

5. 枚举：
    - 是一种引用数据类型
    - 特点：枚举类型的对象数目是固定的，常常用于定义一组常量
    - 所有枚举都继承自Enum类，其中提供了一组方法供我们使用
    - 枚举的构造方法都是私有的

```
//简单版:
package ooday05;
/**
 * 季节枚举
 */
public enum Seasons {
    SPRING,SUMMER,AUTUMN,WINTER //表示Seasons的固定的4个对象，都是常量
}

package ooday05;
/**
 * 枚举的测试类
 */
public class EnumTest {
    public static void main(String[] args) {
        Seasons[] seasons = Seasons.values(); //获取所有枚举对象
        for(int i=0;i<seasons.length;i++){
            System.out.println(seasons[i]);
        }
        /*
        Seasons s = Seasons.WINTER; //获取WINTER对象
        switch(s){
            case SPRING:
                System.out.println("春天到了...");
                break;
            case SUMMER:
                System.out.println("夏天到了...");
                break;
            case AUTUMN:
                System.out.println("秋天到了...");
                break;
            case WINTER:
                System.out.println("冬天到了...");
                break;
        }
         */
    }
}
```

```
//复杂版:
package ooday05_vis;
/**
 * 季节枚举
 */
public enum Seasons {
    SPRING("春天","暖和"),
    SUMMER("夏天","热"),
    AUTUMN("秋天","凉爽"),
    WINTER("冬天","冷");
    private String seasonName;
    private String seasonDesc;
    Seasons(String seasonName, String seasonDesc) {
        this.seasonName = seasonName;
```

```
        this.seasonDesc = seasonDesc;
    }

    public String getSeasonName() {
        return seasonName;
    }
    public void setSeasonName(String seasonName) {
        this.seasonName = seasonName;
    }
    public String getSeasonDesc() {
        return seasonDesc;
    }
    public void setSeasonDesc(String seasonDesc) {
        this.seasonDesc = seasonDesc;
    }
}

package ooday05_vis;

/**
 * 枚举的演示
 */
public class EnumTest {
    public static void main(String[] args) {
        Seasons s = Seasons.WINTER;
        System.out.println(s.getSeasonName()+","+s.getSeasonDesc());

        Seasons[] seasons = Seasons.values();
        for(int i=0;i<seasons.length;i++){
            System.out.println(seasons[i]);
            System.out.println(seasons[i].getSeasonName());
            System.out.println(seasons[i].getSeasonDesc());
        }
    }
}
```

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. getter/setter：行业标准
3. 成员变量分两种：
    - 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/对象打点来访问
    - 静态变量：有static修饰，属于类的，存储在方法区中，只有一份，通过类名打点来访问
4. 工具类：

```
1)Math:数学工具类，里面封装了很多数学相关的静态方法/工具
2)Arrays:数组工具类，里面封装了很多数组相关的静态方法/工具
```

5. 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