Java程序设计基础

面向对象基本特征

面向对象基本特征主要内容

- 8.1 抽象和封装（重点）
- 8.2 继承（重点）
- 8.3 多态（重点、难点）

教学目标

- 能理解抽象的概念
- 能正确使用抽象的思想设计程序
- 能理解继承的概念
- 能正确描述无法使用继承的两种情况
- 能描述 final 关键字的作用
- 能正确区分 super 与 this 关键字的作用
- 能理解多态的概念
- 能正确使用多态简化程序
- 能理解面向基类的编程思想
- 能理解向上转型和向下转型的概念
- 能正确区分向上转型和向下转型

什么是抽象

- 有些资料把抽象和封装、继承、多态一起并称为面向对象的四大特征（但主流的说法，仍然是三大特征）。
- 面向对象设计首先要做的就是抽象，也就是根据用户的业务需求抽象出类，并关注这些类的属性和方法，将现实世界中的对象抽象成程序设计中的类。
- 程序员开发出来的软件是需要满足用户需求的，所以程序员做分析和设计的依据是用户需求，这通常就是软件开发前期形成的“需求规格说明书”。面向对象设计时，首先要阅读用户需求，找出需求中名词部分用来确定类和拥有静态特征的属性，找出动词部分确定动态行为的方法。

“租车系统”需求抽象分析

- 分析一个“租车系统”的部分需求。

在控制台输出“请选择要租车的类型：（1 代表轿车，2 代表卡车）”，等待用户输入。

如果用户选择的是轿车，则在控制台输出“请选择轿车品牌：（1 代表红旗，2 代表长城）”，等待用户输入。

如果用户选择的是卡车，则在控制台输出“请选择卡车吨位：（1 代表 5 吨，2 代表 10 吨）”，等待用户输入。

在控制台输出“请给所租的车起名：”，等待用户输入车名。

所租的车油量默认值为 20 升，车辆损耗度为 0（表示刚保养完的车，无损耗）。

具有显示所租车辆信息功能，显示的信息包括车名、品牌/吨位、油量和车损度。

“租车系统”需求抽象分析

- 首先我们进行抽象，发现需求里的类并定义类的属性和方法：

发现名词。通过阅读需求，发现需求中有类型、轿车、卡车、品牌、红旗、长城等名词。

确定类和属性。通过分析，车名、油量、车损度、品牌这些名词依附于轿车这个名词，车名、油量、车损度、吨位依附于卡车这个名词，所以可以将轿车、卡车抽象成类，依附于这些类的名词抽象成属性。

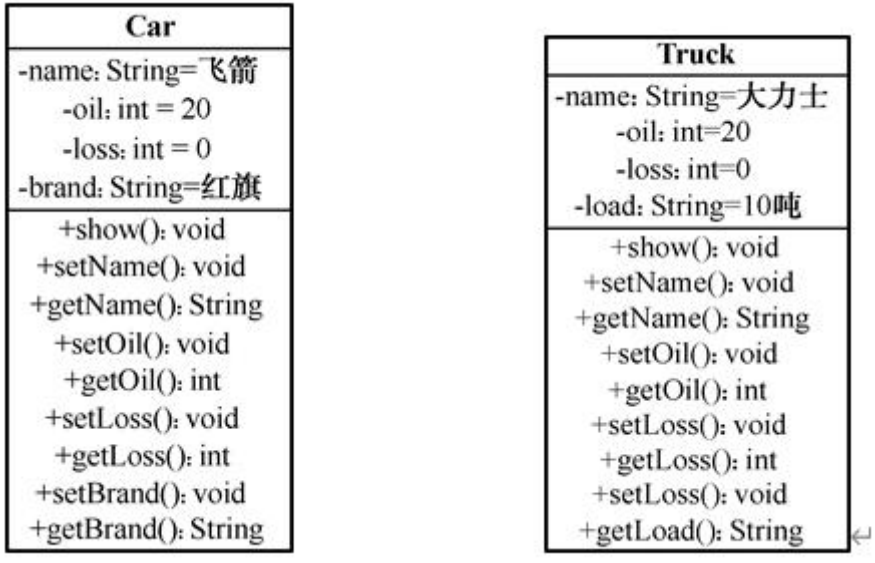
确定方法。通过分析需求的动词，发现显示车辆信息是轿车和卡车的行为，所以可以将这个行为抽象成类的方法。同样地，不是所有依附于类名词的动词都需要抽象成类的方法，只有需要参与业务处理的动词才能确定成方法。

“租车系统”需求抽象分析

- 据对轿车和卡车的抽象，可以得到如图所示的结果

封装“租车系统”的轿车类和卡车类

- 把所有的属性都设置为私有属性，每个私有属性都提供 getter 和 setter 公有的方法，封装后的类图如下图所示



封装“租车系统”的轿车类和卡车类

- 上述封装过于简单，没有考虑需求，接下来进一步阅读需求，可以发现以下几点。

租车时可以指定车的类型和品牌（或吨位），之后不允许修改。

油量和车损度租车时取默认值，只有通过车的加油和行驶的行为改变其油量和车损度值，不允许修改。

- 根据需求，应对轿车类和卡车类做如下修改。

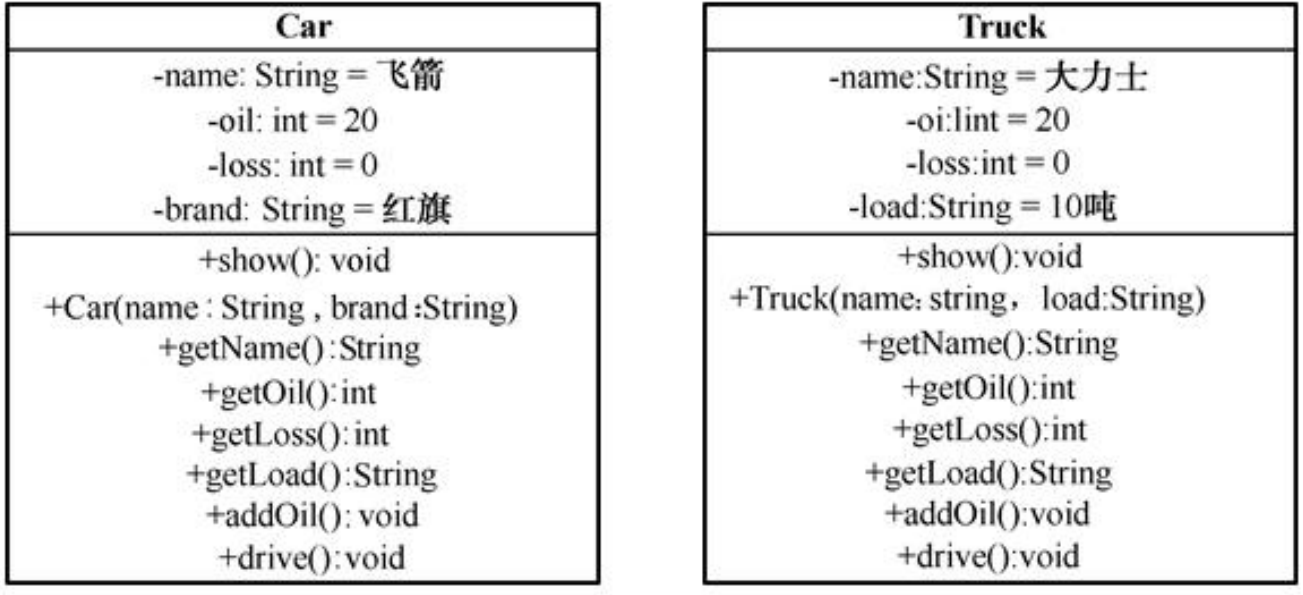
由于要求了车的属性值不允许修改，因此去掉所有的 setter 方法，但保留所有的 getter 方法。

提供 addOil()、drive()这两个公有的方法，实现车的加油和行驶的行为。

至少需要提供一个构造方法，实现对类型和品牌（或吨位）的初始化。

封装“租车系统”的轿车类和卡车类

- 调整后的类图如图所示



抽象和封装的具体体现

- 抽象，实际上是一个分析的过程，是根据需求的表述归纳实体的类型、属性和行为，其产出物是类图。类图勾勒了实体应该具备哪些属性和行为，但未涉及细节。
- 封装，实则就是将抽象得到的模型转变为具体实现。它的要点是，尽可能对外隐藏细节，Java 中的手段就是使用 private，所以在前面案例中所有的属性都是 private。
- 抽象，是归纳提炼；封装则是在实现中依据业务需求尽量隐藏细节。

为什么要使用继承

- 用继承可以大大减少冗余代码，提高代码的复用性。
- 我们之间添加的 Car.java 和 Truck.java 中的代码，可以发现现有代码存在着以下的不足

Car 类和 Truck 类中的代码大量重复：例如二者都存在 name、oil、loss 属性，相应的 getter 方法，以及 addOil( )、drive( )等方法。

对于整体结构相似的 Car 类和 Truck 类，如果要修改其中的一个类的方法，另外一个类的方法要不要修改呢？例如，如果要将 Car 类中的 addOil()方法改为 fuelUp()方法，那么 Truck 类中是否也需要做相应的修改呢？显然，目前这种做法给后期的代码维护带来了麻烦。

为什么要使用继承

- 上述的问题可以使用继承解决。
- 继承可以使得子类沿用父类的成员（属性和方法）。当多个子类具有相同的成员时，就可以考虑将这些成员提取出来，放到父类中，然后再用子类去继承这个父类，也就是将一些相同的成员提取到了更高的层次中。
- 继承的关键字是 extends，语法形式如下。

```
class A extends B{
    类定义部分
}
```

以上表示 A 类继承 B 类，B 类称为父类、超类或基类；A 类称为子类、衍生类或导出类。

无法继承的两种特殊情况

- 以下是两种特殊的情况不能使用继承

  子类无法继承父类的构造方法

  子类不能继承父类中不符合访问权限的成员

方法重写

- 有的时候，父类继承而来的方法不能满足子类的需要，此时就可以在子类中对父类的同名方法进行覆盖，这就是重写。

final 关键字

- 重写在语法上需要满足如下条件。

重写方法与被重写方法同名，参数列表也必须相同。

重写方法的返回值类型必须和被重写方法的返回值类型相同或是其子类。

重写方法不能缩小被重写方法的访问权限。

同时调用父类子类

- 以下是借助于super()同时调用父类及子类的示例代码

```
public class Car extends Vehicle {
    private String brand = "红旗";
    public Car(String name, String brand) {
        super(name);            //使用super关键字，调用父类的构造方法
        this.brand = brand;
    }
    ...
}
```

父类中没有无参构造器的两种常见处理方式

- 在子类构造方法第一行显式的通过 super([参数列表])，调用父类的某一个有参构造方法

```
public class Sup {
    public Sup( String arg){          //父类中没有无参构造方法，只有有参构造方法
    }
}
class Sub extends Sup {
    public Sub(){
      super("argValue") ;            //显式的通过super([参数列表])，调用父类的有参构造方法
      // ..
    }
}
```

父类中没有无参构造器的两种常见处理方式

- 先通过 this([参数列表]) 调用本类中的其他构造方法，再在其他构造方法中通过 super([参数列表]) 显式的调用父类的构造方法

```
public class Sup {
    public Sup( String arg){
    }
}
class Sub extends Sup {
    public Sub(){
        this(1) ;                    //先调用本类的其他构造方法
    }
    public Sub(int a){
        super("argValue") ;          //再调用父类的构造方法
    }
}
```

调用父类中被重写的方法

- 但如何调用父类中被重写的那个方法呢？使用 super 关键字，即可以使用 super 明确调用父类中的方法

```
public class Sup {
    public void info(){ }
}
class Sub extends Sup{
    public void info(){ }
    public void show(){
        info();             //或this.info()。调用的是子类中的方法
        super.info();       //使用super调用父类中的方法
    }
}
```

调用父类中被重写的方法

- 在使用 super 明确调用父类中的方法时，一种常见的做法是，通过 super 对父类中已有的方法进行补充

```
class Sub extends Sup{
    public void info(){
        super.info();                          //调用父类的info()
        System.out.println("this is sub class info");     //进行一些额外的代码补充
    }
    ...
}
```

继承相关问题的总结

- 对于“在子类继承了父类后，子类继承父类的方法”这一问题总结如下：

子类可以直接沿用父类的方法；

子类可以重写父类的方法；

子类可以在父类提供方法的基础上，额外新增一些功能。

super 与 this 关键字的区别与联系

- super与this的区别如下：

this 代表当前对象本身，而 super 代表父类对象。

使用 this 可以调用当前对象的属性或方法，使用 super 可以调用父类对象的属性或方法。

使用 this 可以调用当前类中的其他构造方法，使用 super 可以调用父类中的构造方法。

super 与 this 关键字的区别与联系

- super与this的联系如下：

super 和 this 都指向一个对象，因此 super 和 this 的本质都是引用。二者都可以调用类或对象的属性、方法。

在使用 super()或 this()调用构造方法时，都必须写在构造方法的第一行。因此，在一个构造方法内部，不可能同时使用 super()和 this()来调用其他的构造方法。

什么是多态

- 在汉语中经常存在着“一词多义”的情景。例如“打”这个字是什么意思呢？“打篮球”、“打水”、“打架”中“打”的含义各不相同，因此要想确定某个具体的“打”字的含义，就必须将“打”放入具体的语境，通过上下文来断定。
- 在程序开发中也存在着类似于这种“一词多义”的特征——多态。

向上转型

- 见向上转型，实际就是父类的引用指向子类对象，如下

    父类 引用 = new 子类() ;

示例

    Vehicle 是父类，Car 是子类

        Vehicle v = new Car() ;

动态绑定与静态绑定

- 动态绑定是指在编译期间方法并不会和“引用”的类型绑定在一起，而是在程序运行的过程中，JVM 需要根据具体的实例对象才能确定此时要调用的是哪个方法。
- 重写方法遵循的就是动态绑定。
- 例如，父类 Sup 和子类 Sub 都定义了 info()方法，此时执行以下代码。

```
Sup x= new Sub() ;
x.info() ;
```

由于 Sub 类重写了 Sup 中的 info()方法，在编译期间 info()方法不会和任何一个具体的类绑定起来，而是在运行期间，会列举出 Sub 类中 info()的方法和从父类 Sup 继承过来的 info()方法，然后根据当前的实例对象是 Sub 对象，选择调用 Sub 类中的 info()方法。

动态绑定与静态绑定

- 有动态绑定，自然就有静态绑定。静态绑定是指程序编译期的绑定。以下的类信息，使用的就是静态绑定机制：

  final、static 或 private 修饰的方法，以及重载方法和构造方法。

  成员变量。

面向基类的编程思想

- 在程序设计时，一种推荐的编程思想是“面向基类”编程。也就是建议将面向的“对象”抽象为更高层次的基类。
- 例如，Car 和 Truct 是子类，Vehicle 是父类。那么不建议通过 drive(Car car) 或 drive(Truck truck) 等方法，限制方法接收的参数是某一个具体的对象类型，而建议将这些对象抽象成一个共同的基类，如 drive(Vehicle vehicle)，这样一来就可以利用多态的特性方便的对程序进行扩展，不必每增加一个具体的子类就得新增一个具体的方法。例如 drive(Vehicle vehicle) 就可以接收任何 Vehicle 以及子类对象，因此可以大大减少代码的冗余度。

向上转型的局限性

- 向上转型虽然可以减少代码量，增加程序的可扩展性，但同时也有自身的问题。例如，在 callShow(Vehicle v) 方法中，只能调用 Vehicle 类的方法，不能调用 Vehicle 类子类特有的方法，这就是向上转型的局限性。

- 这个问题的解决办法就是向下转型。顾名思义，向下转型就是将一个父类转换成一个子类的动作。但要注意的是，向下转型不是自动进行的，需要人为的进行强制类型转换。

保证强制转换的准确性

- 程序员编程的过程中，在进行对象的强制类型转换时，如何保证转换的正确性呢？可以使用 Java 提供的 instanceof 运算符（不是方法）来进行预判断。instanceof 运算符的语法形式如下。

  对象 instanceof 类

- 该运算符判断一个对象是否属于一个类，返回值为 true 或 false。

笔试题

- 下列关于继承的哪项叙述是正确的（　　）？

A 在java中类允许多继承

B 在java中一个类只能实现一个接口

C 在java中一个类不能同时继承一个类和实现一个接口

D java的单一继承使代码更可靠

笔试题

- 关于this与super的区别下面那项描述是错误的（　　）？

A　this和super都可以调用类中的属性、方法、构造方法

B　super表示本类实例化对象，而this表示父类实例化对象

C　使用“this.属性”或者“this.方法()”时都会先从本类查找方法，如果本类没有定义，则通过父类查找

D　子类可以利用“super.方法()”调用父类方法

笔试题

- 涉及类MyClass的方法签名是：public void find（MyClass a），那么该方法不能接收的实际参数类型有（　　）？

  A MyClass类型

  B MyClass的子类

  C Object类型

  D MyClass子类的子类

笔试题

- 以下程序的运行结果是（　　）？

```
class A{
    static{
        System.out.print("1");
    }
    public A(){
        System.out.print("2");
    }
}
class B extends A{
    static {
        System.out.print("a");
    }
    public B(){
        System.out.print("b");
    }
}
public class Hello{
    public static void main(String[] ars){
        A ab=new B();
        ab=new B();
    }
}
```