# 面向对象编程(上)

天津卓讯科技有限公司

# 第3章 面向对象编程(上)

- 3.1 面向对象概述
- 3.2 类和对象的关系
- 3.3 封装类
- 3.4 方法重载(Overload)
- 3.5 this关键字
- 3.6 包

# 3.1.1面向过程和面向对象

- 面向过程：考虑问题时，以一个具体的流程为单位，考虑它的实现办法。关心的是功能的实现。
  - 强调的是功能行为
- 面向对象：考虑问题时，以具体的事物为单位，考虑它的属性(特征)及动作(行为)。
  - 将功能封装进对象，强调的是具备了功能的对象。
- 面向对象是基于面向过程的。
- 示例
  - 用洗衣机洗衣服
  - 窗体操作

# 3.1.2 面向对象的特点

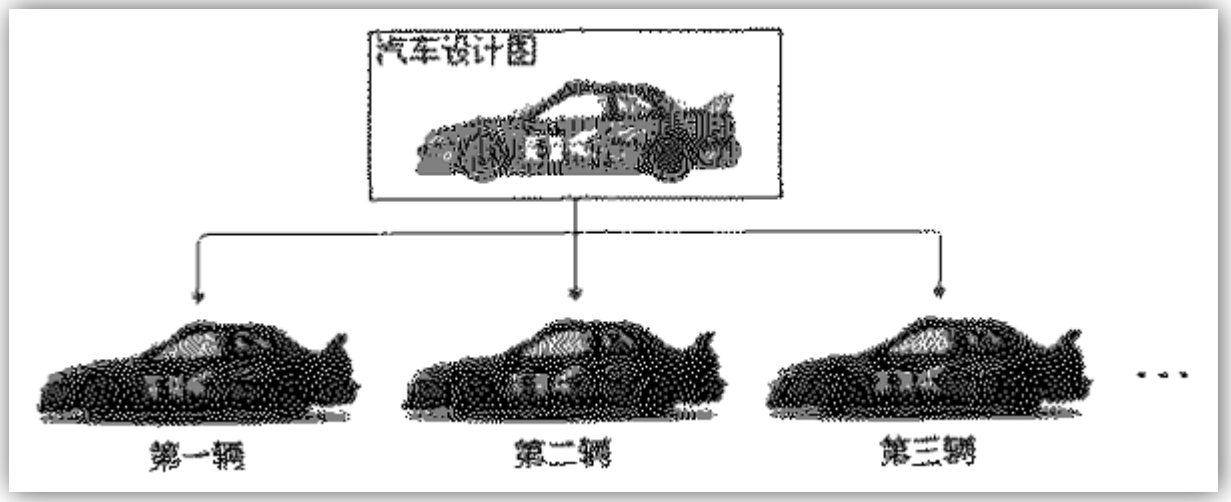
- 是一种符合人们思考习惯的思想
- 可以将复杂的事情简单化
- 将程序员从执行者转换成了指挥者
- 完成需求时：
  - 先要去找具有所需的功能的对象来用。
  - 如果该对象不存在，那么创建一个具有所需功能的对象。
  - 这样简化开发并提高复用。

# 3.1.3 面向对象开发，设计，特征

- 开发的过程：其实就是不断的创建对象，使用对象，指挥对象做事情。
- 设计的过程：其实就是在管理和维护对象之间的关系。
- 面向对象的特征：
- 抽象
  - 封装(encapsulation)
  - 继承(inheritance)
  - 多态(polymorphism)

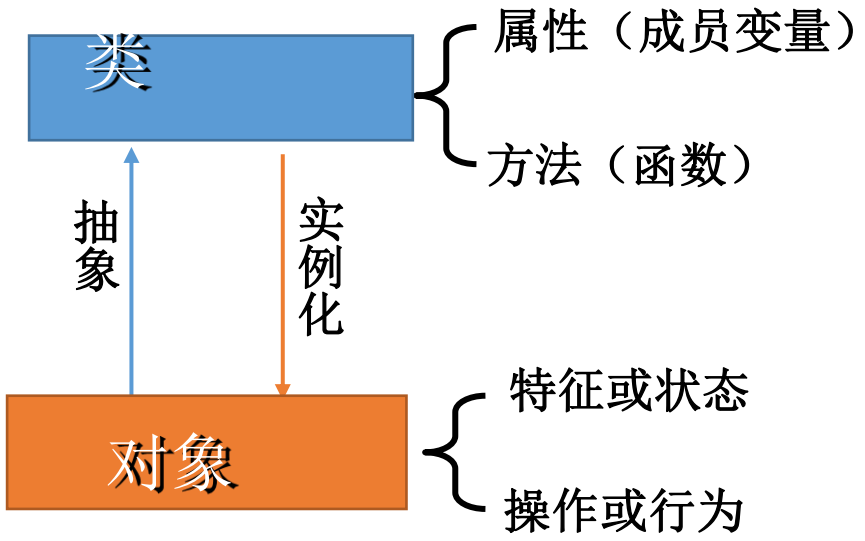
# 3.2 类和对象

- 类(class)和对象(object)是面向对象思想的核心概念
  - 类是对一类事物的描述，是抽象的、概念上的定义；
  - 对象(实体)是实际存在的该类事物的每个个体，因而也称实例(instance)。



# 3.2 类和对象

- 对象是Java程序的核心，在Java程序中“万事万物皆对象”
- 类描述了对象的属性和对象的行为，类是对象的模板，图纸。
- 对象是类的一个实例，是一个实实在在的个体 。
- JDK提供了很多类供编程人员使用，编程人员也可以定义自己想要的类。

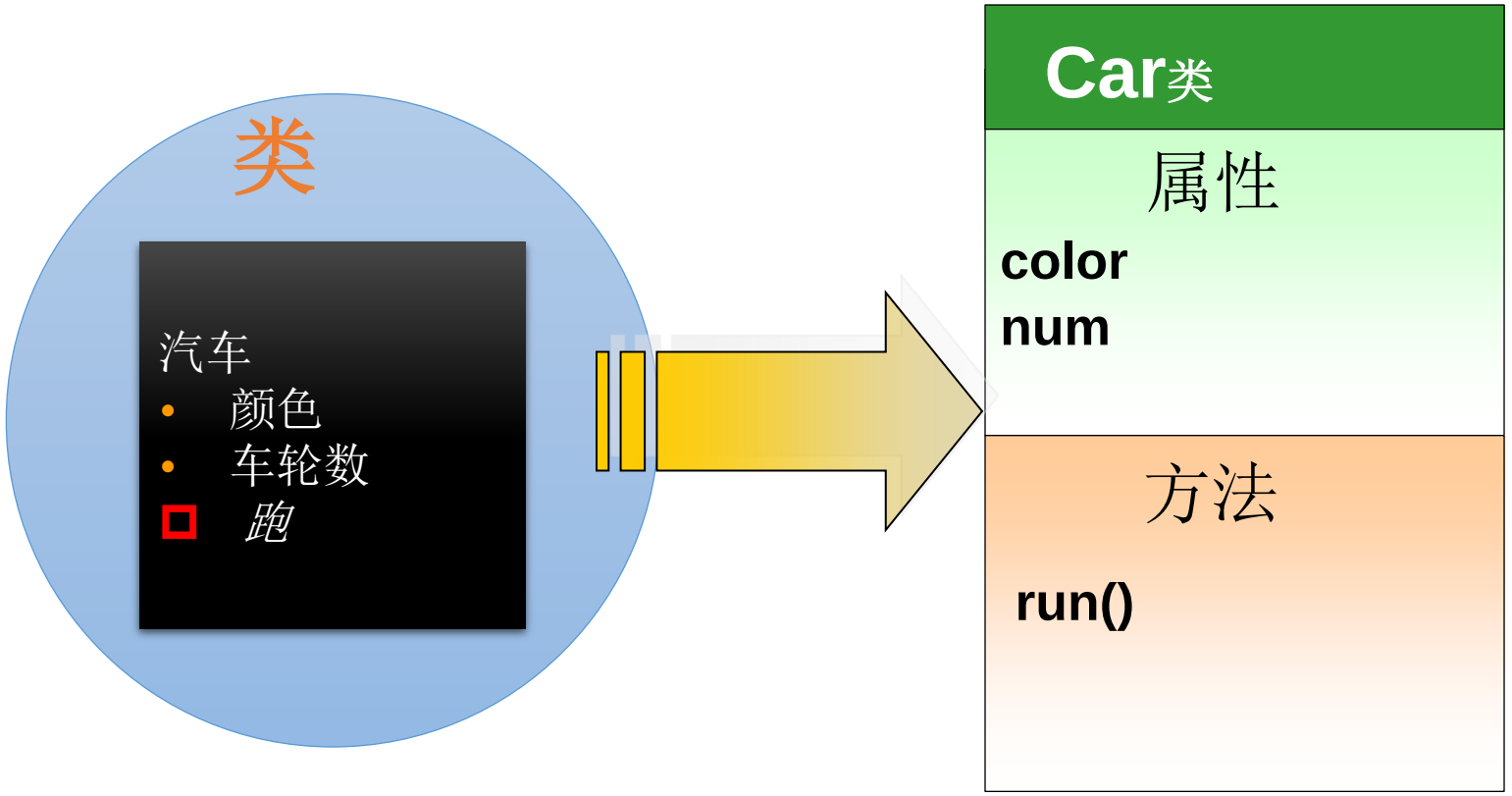


# 3.2.1 Java类的定义

- 生活中描述事物无非就是描述事物的属性和行为。
  - 如：人有身高，体重等属性，有说话，打球等行为。
- Java中用类class来描述事物也是如此
  - 属性：对应类中的成员变量。
  - 行为：对应类中的成员方法。
- 定义类其实在定义类中的成员(成员变量和成员方法)。
- 语法：
  - [修饰符] class 类名 {
  - //这里放属性和方法的声明
  - }

# 示例

- 封装一个汽车类



# 3.2.2创建对象，使用对象

```
class Car {//对Car这类事物进行描述
        String color = "red";
        int num = 4;
        void show(){
                System.out.println("color="+color+"..num="+num);
        }
}
class CarTest {
        public static void main(String[] args) {
                Car c = new Car();  //创建对象
                c.color = “black”;  //操作对象的属性
                c.show();  //使用对象的功能。
        }
}
```

# 细节：类中成员变量的定义

- 定义成员变量的语法：
  - [访问修饰符] 数据类型 成员变量名 [= 初始值];
- 成员变量的类型可以使用Java语言中的任何一种数据类型(包括基本类型和引用类型)。
- 在定义成员变量时可以对其进行初始化。如果不对其初始化，Java会使用默认的值对其进行初始化。
- 成员变量的作用域是整个类体。

| 成员变量类型 | 默认值 |
| --- | --- |
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |
| char | '\u0000' |
| float | 0.0F |
| double | 0.0D |
| boolean | false |
| 所有的引用类型 | null |

# 成员变量和局部变量

- 成员变量：
  - 成员变量定义在类中，在整个类中都可以被访问。
  - 成员变量随着对象的建立而建立，存在于对象所在的堆内存中。
  - 成员变量有默认初始化值。
- 局部变量：
  - 局部变量只定义在局部范围内，如：方法内，语句内等。
  - 局部变量存在于栈内存中。
  - 作用的范围结束，变量空间会自动释放。
  - 局部变量没有默认初始化值。

# 细节：类中方法的定义

- 定义方法的语法：
  - [修饰符]　返回值类型　方法名 (参数类型 参数名1,...) {
  -　　// 这里放方法的具体实现语句
  - }

# 细节：构造方法

- 构造方法的作用在于构造并初始化对象。
  - 利用new关键词调用类的构造方法(类的一种特殊方法)就可创建该类的一个对象。
- 语法要求
  - 构造方法的名字和类名相同，并且没有返回值，也不要加void。
- 两种构造方法
  - 参数化构造方法
  - 默认(隐式)构造方法

# 参数化构造

```
public class Car{
    private String color;        //颜色
    private int num ;            //车轮数

    public Car(String c, int n){
        color = c;
        num = n;
    }


     public void run(){
        …
    }

    public static void main(String [] args){
        Car c = new Car(“黑色”,4);
        c.run();
    }
}
```

参数化构造方法：
访问修饰符 类名(参数类型 参数名,…){
…
}

用new调用该类的构造方法来创建一个对象
注意：传递的值和构造方法的参数在个数、次序和类型上要匹配

普通方法用：
对象变量名.方法 来调用

# 默认构造方法

- 默认构造方法就是指不带参数的构造方法。
- Java的类都要求有构造方法，如果没有定义构造方法，Java编译器会为我们提供一个默认的构造方法。

```
public class Car {
    private String color;        //颜色
    private int num ;            //车轮数

    public void run(){
        …
    }

    public static void main(String [] args){
        Car c = new Car();
        c.run();
    }
}
```

用new调用该类的默认构造方法来创建一个对象

# 一个类多个构造方法

- 如果类中有一个自己编写的构造方法时，编译器就不会为我们再提供那个默认构造方法。此时又希望还可以用默认构造方法来创建类的实例时，那就必须在类中明确添加这个默认构造方法。

```
public class Car{
    String color;        //颜色
    int num ;            //车轮数
    public Car(String c, int n){
        color = c;
        num = n;
    }
    public void run(){
        …
    }
    public static void main(String [] args){
        Car c = new Car();
        c.run();
    }
}
```

编译报错

# 3.3 封装(Encapsulation)

- 封装：就是隐藏对象的属性和实现细节，仅对外提供公共访问方式。
  - private(私有的)、public(公开的)
- 好处：将变化隔离。便于使用。提高重用性。提高安全性。
- 封装原则：
  - 将不需要对外提供的内容都隐藏起来。
  - 把属性都隐藏，提供公共方法对其访问。
- 抽象：将客观存在的事物特征用Java语言描述出来。抽象只关注一个主题中与当前目标有关的方面，而忽略与当前目标无关的那些方面。

# 小结

- 类描述了对象的特征和对象的行为，类是对象的模板，图纸。
- 对象是类的一个实例，是一个实实在在的个体 。
- 封装一个类，就是根据具体的应用从同一类型对象中抽象出相关的属性(成员变量)和方法(函数)。
- 构造方法的作用在于构造并初始化对象。
  - 两种构造方法：参数化构造方法和默认构造方法。
- 方法的调用
  - 构造方法：用*new*调用。
  - 普通方法：用*对象变量名.方法()* 调用。

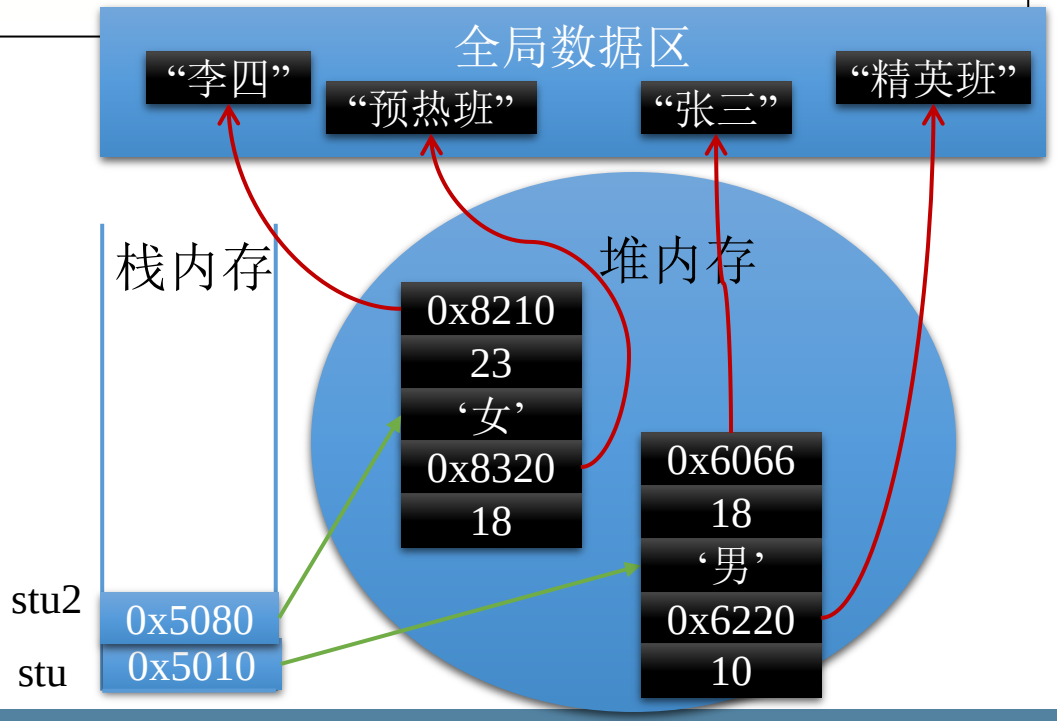
# 练习

- 封装一个学生(Student)类：
  - 学生应该有姓名(name)、年龄(age)、性别(gender)，班级号(classNum)，座位号(sno)。
  - 提供一个参数化构造化方法以便能很方便的创建一个学生对象。
  - 提供一个方法(displayInfo())用来输出显示这个学生的姓名、年龄、性别、所在的班级和他的座位号。
  - 写一个main方法创建两个学生对象，分别调用displayInfo()方法显示各自的信息。

# 内存分析

```
public class Student{
    ……
    public static void main(String[] args){
        Student stu = new Student(“张三”, 18, ‘男’, “精英班”, 10);
        stu.displayInfo();
        Student stu2 = new Student(“李四”, 23, ‘女’, “预热班”, 18);
        stu2.displayInfo();
    }
}
```

**stu变量是堆内存中创建的“对象的引用“，但常说成是一个对象。**



# 3.4 方法重载(Overload)

- 方法重载指的是同一个类中可以定义有相同名字，但参数列表不同的多个方法。调用时，会根据不同的参数选择对应的方法。
  - 参数列表是指参数的类型，个数或顺序
- 类中定义的普通方法、构造方法都可以重载
- 重载的特点：
  - 与返回值类型无关，只看参数列表。
- 作用：
  - 方便于使用者，优化了程序设计。

# 示例

```
public class Person {
    private String name;        //姓名
    private boolean sex;        //性别
    private int age;            //年龄

    public Person(String n, boolean s, int a){
        name = n;
        sex = s;
        age = a;
    }
    public Person(){}

    public void speak(String word){  //说话
        System.out.println(name + "说：" + word);
    }
    public void speak(){
        System.out.println("无语…");
    }

    public static void main(String [] args){
        Person person = new Person();
        person.speak("你好");
        person.speak();
    }
}
```

# 3.5 this关键词

- 每个类的每个非静态方法(没有被static修饰)都会隐含一个this引用名称，它指向调用这个方法的对象
- 当在方法中使用本类的属性时，都会隐含地使用this名称，当然也可以明确指定。
- this可以看作是一个变量，它的值就是当前对象的引用

```
//Persion类的构造方法
public Person(String n, boolean s, int a){
    this.name = n;
    this.sex = s;
    this.age = a;
}
```

# this用法一

- 当类中某个非静态方法的参数名跟类的某个成员变量名相同时，为了避免参数的作用范围覆盖了成员变量的作用范围，必须显式地使用this关键字来指定成员变量

```
public class Employee {
    private String name;        //姓名
    private int age;            //年龄
    private double salary;      //薪水

    public Employee(String name, int age, double salary){    //构造方法
        this.name = name;
        this.age = age;
        this.salary = salary;
    }
}
```

# this用法二

- 如果某个构造方法的第一条语句具有形式this(...)，那么这个构造方法将调用本类中的其他构造方法。

```
public class Employee {
    private String name;        //姓名
    private int age;            //年龄
    private double salary;      //薪水

    public Employee(String name, int age, double salary){    //构造方法1
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public Employee(){    //构造方法2
        this(“无名”, 18, 800.0);    //调用到了构造方法1
    }
}
```

# 3.6 包

- 为了便于管理大型软件系统中数目众多的类，解决类命名冲突的问题，Java引入了包。
- 用package来声明包，package语句必须是java源文件中的第一条语句。(若无这条语句，则放置在无名包下)
- 在package语句中，用"."来指明包（目录）的层次。包对应着文件系统的目录层次结构。
  - 如：package com.qiujy; →编译后对应的类文件位于com\qiujy目录下。

# 创建包

```
package com.qiujy;

public class Employee {
   private String name;        //姓名
    private int age;           //年龄
    private double salary;     //薪水

    public Employee(String name, int age, double salary){
        this.name = name;
        this.age = age;
        this.salary = salary;
    }
    ……
}
```

声明包

# 编译和运行带包的类

- 在当前目录下生成带包结构的字节码
  - cmd>javac –d . Employee.java
- 在指定目录下生成带包结构的字节码
  - cmd>javac –d D:\share Employee.java
- 运行带包的类：
  - cmd>java 包名.类名
    - 包名.类名 叫做类的全限定名

# 导入包

```
import com.qiujy.Employee;

public class PackageDemo {
    public static void main(String args[]){
        Employee employee = new Employee( );
        …
    }
}
```

导入包中的这个类

- 导入某个包中的所有类使用：包名.*
  - 如：import  com.qiujy.*;
- 同一包中的类之间直接引用，无需import语句
- 无名包中的类，无法在其它带包的类中使用。
- 建议：自定义类都要放置在包中。

# JDK中主要的包介绍

- java.lang - 包含一些Java语言的核心类，如：Object、String、Math、Integer、System和Thread，提供常用功能。
  - 此包非常常用，所以在任何类中不用导入就可能直接使用。
- java.util - 包含一些实用工具类，如定义系统特性、日期时间、日历、集合类等。
- java.io - 包含能提供多种输入输出的流类。
- java.net - 包含执行网络相关的操作的类。
- java.sql - java操作数据库的一些API。
- java.text - 包含一些用来处理文本、日期、数字和消息的类和接口
- java.awt - 包含了构成抽象窗口工具集的多个类，这些类被用来构建和管理应用程序的图形用户界面(GUI)。
- javax.swing - 包含了构成"轻量级"窗口的组件。

# 总结

- OOP思想
- class vs object
- 封装类
  - 定义成员变量
  - 定义方法
- 构造方法 & new
- 方法重载(Overload)
- this
- package & import