Chapter4 面向对象的程序设计

4.1类和对象

4.1.1 类和对象的概念

Java编程语言中的**抽象数据类型概念**被认为是**class**，类给对象的特殊类型提供定义。它规定对象内部的**数据**，创建该对象的**特性**，

以及对象在其自己的数据上运行的**功能**。因此**类就是一块模板**。

- Java类由**状态**（或属性）和**行为**两部分组成。
- Java程序中，用**变量**来描述类的**状态**，用**方法**来实现类的**行为**。方法定义了可以在对象上进行的操作。

eg.4.1 学生类

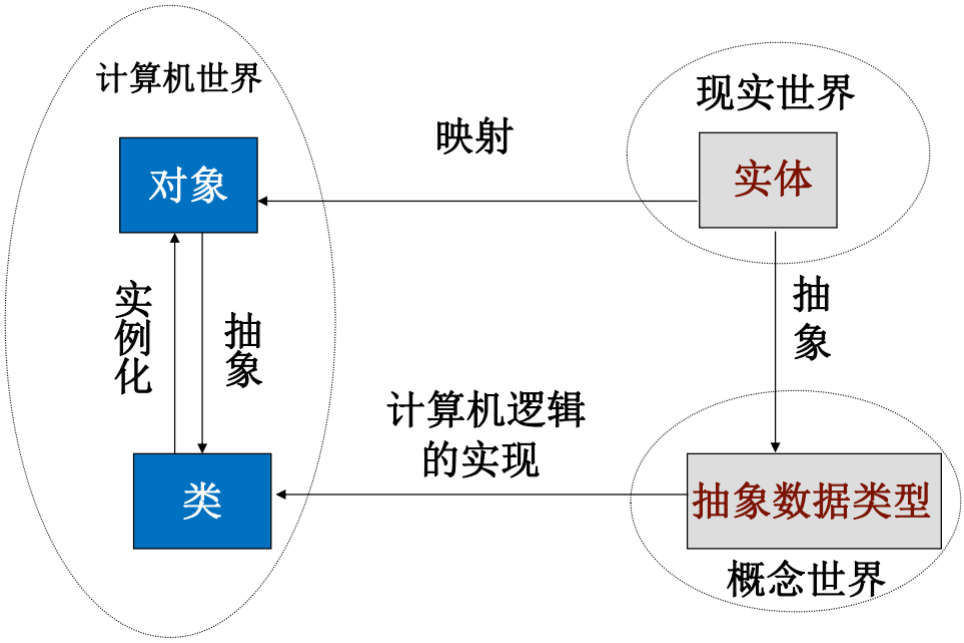
```
public class Student{
    int id;
    String name;
    double marks;
    boolean pass;
    //定义（属性）变量
    public void display(){
        System.out.println("id = "+id);
        System.out.println("name = "+name);
        System.out.println("marks = "+marks);
        System.out.println("pass = "+pass);
    }
    public static void main(String[] args){
        Student s = new Student();
        s.display();
    }
}
```

eg.4.2 雇员类

```
class Employee{
    String name;
    int age;
    float salary;

    void print(){
        System.out.println(name+"age: "+age+"salary: "+salary);
    }
    void upSalary(){
        salary+=inc;
    }
}
```

- 对象是类的唯一性实列



- Java中的类结构

  基类：所有Java类别都派生自java.lang.Object；

  Java的类可组织在包（package）中；

  Java提供的标准类库分布在一系列的包中，如java.lang，java.util，java.net等；

  编程的过程就是继承基类或标准类而创建、定义特殊子类的过程.

4.1.2 类和对象的申明

1. 类的申明

类的声明格式如下： [类的修饰符] class 类名 [extends 父类名] [implements 接口名]

说明：

1. 类的修饰符包括**public**,**abstract**和**final**,也可以缺省（无修饰符);

修饰符	说明
public	允许其他类（没有限制）访问本类，一个.java文件中有且仅有一个public类
abstract	没有实例的抽象概念类，是它所有子类公共属性和公共方法的集合
final	不能被继承，不能有子类的类；易于安全性的实现，或用于创造固定功能的类
缺省	可以被当前包中其他的类访问

2. **class**是关键字，用来标识声明类的语句；
3. 类名必须是合法的Java标识符，用来标识所声明的类；
4. **extends**是关键字，表示所声明的类继承了指定的父类**[extends 父类名]**；
5. **implement**接口名表表示所声明的类继承了指定的接口**[implements 接口名]**.

eg.4.3 类的申明

```
public abstract class shape{……}
class rectangle extends shape{……}
final class roundrect extends rectangle{……}
```

eg.4.4

```
class Man {
    String name ;
    int age;
    float weight;
    Man(String name,int age,float weight){}
    void walk() {…}
    void speek() {…}
    void sleep() {…}
}
```

2. 对象的申明

对象（实例：instance）：以类为模板创建的具体实例（实际个体）

对象的申明语法如下： 类名 欲创建的对象名 = new 类名(参数表)；

```
String s = new String("Hello!") ;
```

实例化一个对象:现实生活中某个具体的人，即对象"Jack"

这些变量(name, age, weight)被称为类Man的**成员。**

其中Man(String n,int a,float w){ ..... } 是类的**构造方法。**

4.1.3 变量

1. 成员变量(实例变量)

声明**成员变量(member)**的语法 **[修饰符] 数据类型 变量名**;

引用：对象名.变量名;

修饰符:

a.访问控制符

修饰符	说明
public	可被所有其他类引用
private	仅可被该类自身引用和修改，不能被其他任何类（包括该类的子类）引用，它提供了最高级别的保护
protected	该类自身、子类、同一包中的其他类
无	本包中的类

b.静态变量static

类成员变量，对所有的实例一致，引用时前缀可使用类名或对象名

c.常量final

数值不变常量，定义同时应对其进行初始化

eg.4.5 成员变量的申明

```
class Shape{
    private protected int x;
    private protected int y;
    static final float MaxArea=100.0f; //final 类型 常量名 = 值
    ……
}
```

eg.4.6

```
class Employee{
    String name;
    int age;
    float salary;
}
public static void main(Strings[] args){
    Employee e = new Employee( ) ;
    e.name="张立";
    e.age=21;
    e.salary = 528.37;
}
```

- 成员变量的生存周期和对象存在的时间相同.

2. 静态变量(类成员变量)

为类的各个实例所共享的变量，位于类的内存区域中。

- 无此限定符的变量是实例变量

```
class ex{
    int i ;
    static int j ;
    static final int k=10 ; //final常量可同时定义为static
}
```

- 实现各个实例之间的通讯

  对于该类的任何一个变量访问该静态变量时，取得的都是相同的值
- 跟踪创建的实例数

  ```
  public class Count{
      private int serial;
      private static int counter = 0;
      public Count(){
          counter++;
          serial = counter;
      }
  }
  ```

- 静态变量类似于某些语言中的全局变量
- 非private的静态变量无需创建实例，就可以从类的外部进行访问

```
class StaticVar{
    static int x = 100;
}
public class test{
    public void m(){
        int m = StaticVar.x;
    }
}
```

eg.4.7

```
class PhoneCard200 {
    static final String connectNumber = “200”;
    static double additoryFee;
    long cardNumber;
    int password;
    boolean connected;
    double balance;
    …
}
public class AA {
    public static void main(String args[]){
        PhoneCard200 my200_1 = new PhoneCard200();
        PhoneCard200 my200_2 = new PhoneCard200();
        PhoneCard200.additoryFee = 0.1;
        System.out.println("第一张200卡的接入号码:" + my200_1.connectNumber);
        System.out.println("第二张200卡的附加费:"+ my200_2.additoryFee);
        System.out.println("200卡类的附加费:" + PhoneCard200.additoryFee);
        System.out.println("200卡接入号码:" + PhoneCard200.connectNumber);
    }
}
```

eg.4.8

```
public class MyDate {
    int day;
    int month;
    int year;
    public static void main(String args[]){
    MyDate today;
    today = new MyDate();
    today.day=25;
    today.month=9;
    today.year=2007;
    System.out.println(today.year+"年"+today.month+"月"+today.day+"日");
    }
    public void tomorrow() {
        // code to increment day
        // like day++;
    }
}
```

3. 类和变量的关系

- 包含关系：当对象B是对象A的属性时，我们称对象A包含对象B；
- 关联关系：当对象A的引用是对象B的属性时，我们称对象A和对象B之间是关联关系；
- 类之间的继承关系：B类继承了A类，就是继承了A类的属性和方法， **A类称之为父类，B类称之为子类。**

4.2类的方法

4.2.1 方法的申明

- 方法的申明语法

```
<modifiers> <return_type> <name> ([<argument_list>])
[throws <exception>]
{
    <block>
}
public void addDays(int days){
}
```

方法是类的动态属性,标志了类所具有的功能和操作;

方法是定义对类内成员变量(数据)的操作.

- 说明:
  1. 方法名后的小括号是方法的标志。
  2. 形式参数是方法从调用它的环境输入的数据。
  3. 返回值是方法在操作完成后返还给调用它的环境的数据。
  4. 要指定方法返回值类型。如没有返回值，类型要写 void。
  5. 方法名相同,但参数不同,是不同的方法。
  6. 与类名相同的所有方法都是类的构造方法。
- 返回类型:void,return
- 参数列表:参数类型可以为类类型
- throws:通告本方法中会产生的例外（异常）类型，提醒调用者 要对相应的可能异常（例外）进行处理。当执行此方法并发生了异常时，程序会转入调用者编制的异常处理程序段。

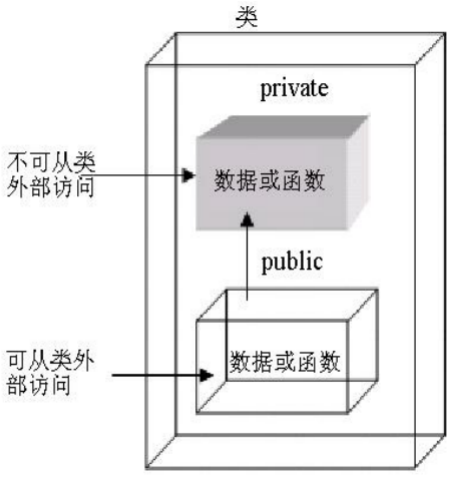
- 修饰符（Modifier）
  - 访问控制符:public,private,protected,private protected
  - 静态方法: static(类方法),
    - 使用类名或对象名作为前缀，建议使用类名；
    - 在方法体中只能使用static变量和static方法.
  - 抽象方法:abstract
    - 只有方法头，没有具体的方法体和操作实现的方法，要在子类中通过重载实现（override）来实现.
  - 最终方法:final
    - 不能被重载(override)的方法.

- 访问控制符

- 访问控制符是一组起到限定类、域或方法是否可以被程序里的其他部分访问和调用的修饰符。
- 类访问控制符
  - 公共类： public 类名
  - 一般类(缺省)
- 一个类被声明为**公共类**，**表明它可以被所有的其他类所访问和引用。**程序的其他部分可以创建这个类的对象、访问这个类的内部可见成员变量和调用它的可见方法
- **一般类只能被同一包中的类访问和引用。**
- 定义在同一个程序中的所有类属于一个包。

- 域和方法的访问限定符
  - 一个类作为整体对程序的其他部分可见，并**不能代表类内的所有成员变量和方法也同时对程序的其他部分可见**，前者只是后者的必要条件
  - 成员变量和方法的可见性

访问限定符	说明
public	公共变量和公共方法,可被任何类使用.
protected	可被同包中的其他类使用,可被子类(包括不同包中的子类)继承
无访问限定符	可被同包中的其他类使用
private	只限在本类内使用

  - 为使对象具有良好的封装性,一般将类的实例变量设置为私有.
  - 为使其他类或对象能访问私有变量,本类必须提供读写问私有变量的公共方法(getter,setter).
  - 

- 高级访问控制
  - 成员变量和方法有四种访问级别:public,protected,default(package),private;
  - 类有两种访问级别:protected,default;
  - 修饰符的作用范围

Modifier	Same class	Same Package	Subclass	Universe
public	√	√	√	√
protected	√	√	√	
default	√	√		
private	√			

eg.4.9

```
public class Student{
    private String name;
    private String sex;
    private int age;

    public Student(String name,String sex,int age){
        this.name=name;
        this.sex=sex;
        this.age=age;
    }

    public String getName(){
        return name;
    }

    public String getSex(){
        return sex;
    }

    public int getAge(){
        return age;
    }
}
```

- 方法的参数
  - *<argument_list>*允许将参数值传递到方法中。列举的元素由逗号分开，而每一个元素包含一个类型和一个标识符。方法定义中的参数并没有实际值，仅仅是为了描述处理过程而引入的，因此称为**形式参数**（简称**形参**）。使用方法时给出参数的实际值，这些实际值称为**实际参数**（简称**实参**）
  - 方法的参数要有名有型，**参数的作用域在本方法中**，在方法体中可以像方法体自己定义的变量一样使用
  - 参数的值传递:对象的内容可以改变,但对象的引用不会改变.
  - 方法的参数可以与类的成员变量同名,这时,参在方法体重将隐藏同名的成员变量.

```
class Circle {
    int x,y, radius;
    setCircle(int x, int y, int radius){
        …
    }
}
```

eg.4.10 参数传递

```
public class PassTest{
    float m_float;
    void change1(int pi){
        pi = 100;//未引用
    }
    void change2(String ps){
        ps=new String(“Right”);//未引用
    }
    void change3(PassTest po){
        po.m_float=100.0f;//引用
    }
    public static void main(String[] args){
        PassTest pt = new PassTest();
        int i = 22;
        pt.change1(i);
        System.out.println("i value is" + i);//i value is 22
        String s = new String("Hello");
        pt.change2(s);
        System.out.println("s value is" + s);//s value is Hello
        pt.m_float = 22.0F;
        pt.change3(pt);
        System.out.println("Current pt.m_float is" + pt.m_float);
        //Current pt.m_float is 100.0
    }
}
```