第 2 章 类与对象（二）

清华大学　　郑　莉

类的访问权限控制

<2.2.6>

类的访问权限控制

- 类在不同范围是否可以被访问：

类型	无修饰（默认）	public
同一包中的类	是	是
不同包中的类	否	是

类的成员访问权限控制

- 公有(public)
  - 可以被其他任何方法访问(前提是对类成员所属的类有访问权限)
- 保护(protected)
  - 只可被同一类及其子类的方法访问
- 私有(private)
  - 只可被同一类的方法访问
- 默认(default)
  - 仅允许同一个包内的访问；又被称为“包（package)访问权限”

类成员在不同范围是否可以被访问

类型	private	无修饰	protected	public
同一类	是	是	是	是
同一包中的子类	否	是	是	是
同一包中的非子类	否	是	是	是
不同包中的子类	否	否	是	是
不同包中的非子类	否	否	否	是

例：改进的圆类

将实例变量设置为private

```
public class Circle {
        static double PI = 3.14159265;
        private int radius;
        public double circumference() {
            return 2 * PI * radius;
        }
 }
```

例：改进的圆类

- 再编译CircumferenceTester.java

```
public class CircumferenceTester {
    public static void main(String args[]) {
        Circle c1 = new Circle();
        c1.radius = 50;
        Circle c2 = new Circle();
        c2.radius = 10;
        double circum1 = c1.circumference();
        double circum2 = c2.circumference();
        System.out.println("Circle 1 has circumference " + circum1);
        System.out.println("Circle 2 has circumference " + circum2);
    }
}
```

例：改进的圆类

- 编译时会提示出错

  在编译语句“c1.radius = 50;”及“c2.radius =10;”时会提示存在语法错误

  “radius has private access in Circle”

- 说错误原因
  - Circle类中变量radius被声明为private，在其它类中不能直接访问radius；

- 如果要允许其它类访问radius的值，需要在Circle类中声明相应的公有方法。
- 通常有两类典型的方法用于访问属性值，get方法及set方法

get方法

- 功能是取得属性变量的值
- get方法名以“get”开头，后面是实例变量的名字
- 例如：

```
public int getRadius(){
  return radius;
}
```

set方法

- 功能是修改属性变量的值
- set方法名以“set”开头，后面是实例变量的名字
- 例如：

```
public void setRadius(int r){
  radius = r;
}
```

- 如果方法的局部变量（包括形参）与类的非静态成员同名怎么办？

this关键字

- 如果方法内的局部变量（包括形参）名与实例变量名相同，则方法体内访问实例变量时需要this关键字。
- 例如：

```
public void setRadius(int radius){
  this.radius = radius;
}
```

- 在这一节我们学习了类和类成员的访问控制。

对象初始化

<2.3.1>

- 对象初始化
  - 系统在生成对象时，会为对象分配内存空间，并自动调用构造方法对实例变量进行初始化
- 对象回收
  - 对象不再使用时，系统会调用垃圾回收程序将其占用的内存回收

构造方法

- 用来初始化对象
- 每个类都需要有构造方法

构造方法

- 方法名与类名相同；
- 不定义返回类型；
- 通常被声明为公有的(public)；
- 可以有任意多个参数；
- 主要作用是完成对象的初始化工作；
- 不能在程序中显式的调用；
- 在生成一个对象时，会自动调用该类的构造方法为新对象初始化；
- 若未显式声明构造方法，编译器隐含生成默认的构造方法。

默认构造方法

- 没有参数（内部类除外），方法体为空；
- 使用默认的构造方法初始化对象时，如果在类声明中没有给实例变量赋初值，则对象的属性值为零或空；

- 下面我们来看一个默认构造方法的例子

例：一个银行帐户类及测试代码

```
public class BankAccount{
    String    ownerName;
    int       accountNumber;
    float     balance;
}
public class BankTester{
     public static void main(String args[]){
       BankAccount myAccount = new BankAccount();
       System.out.println("ownerName=" + myAccount.ownerName);
       System.out.println("accountNumber=" + myAccount.accountNumber);
            System.out.println("balance=" + myAccount.balance);
        }
}
```

运行结果
ownerName=null
accountNumber=0
balance=0.0

- 如何自定义构造方法，使新对象按照程序员设计的方式被初始化？——自定义构造方法。

自定义构造方法与方法重载

- 在生成对象时给构造方法传送初始值，为对象进行初始化。
- 构造方法可以被重载
  - 一个类中有两个及以上同名的方法，但参数表不同，这种情况就被称为方法重载。在方法调用时，可以通过参数列表的不同来辨别应调用哪一个方法。
- 只要显式声明了构造方法，编译器就不再生成默认的构造方法。
- 也可以显式声明无参数的造方法，方法体中可以定义默认初始化方式。

- 在下面的例子中，我们为银行账号类声明构造方法

例：为银行帐户类增加构造方法

- 为BankAccount声明一个有三个参数的构造方法

```
public BankAccount(String initName, int initAccountNumber, float initBalance) {
        ownerName = initName;
        accountNumber = initAccountNumber;
        balance = initBalance;
    }
```

- 假设一个新帐号的初始余额可以为0，则可增加一个带有两个参数的构造方法

```
public BankAccount(String initName, int initAccountNumber) {
        ownerName = initName;
        accountNumber = initAccountNumber;
        balance = 0.0f;
    }
```

- 无参数的构造方法——自定义默认的初始化方式

```
public BankAccount() {
        ownerName = "";
        accountNumber = 999999;
        balance = 0.0f;
    }
```

- 初始化逻辑相同，只是初始值不同的多个重载构造方法，需要冗余地写多个方法体吗？

声明构造方法时使用this关键字

- 可以使用this关键字在一个构造方法中调用另外的构造方法；
- 代码更简洁，维护起来也更容易；
- 通常用参数个数比较少的构造方法调用参数个数最多的构造方法。

例：使用this的重载构造方法

```
public BankAccount() {
        this("", 999999, 0.0f);
}
public BankAccount(String initName, int initAccountNumber) {
          this(initName, initAccountNumber, 0.0f);
}
public BankAccount(String initName, int initAccountNumber,
 float initBalance) {
        ownerName = initName;
        accountNumber = initAccountNumber;
        balance = initBalance;
 }
```

- 在类中定义变量时，加上final修饰，那便是说，这个变量一旦被初始化便不可改变；

final变量的初始化

- 实例变量和类变量都可被声明为final;
- final实例变量可以在类中定义时给出初始值，或者在每个构造方法结束之前完成初始化;
- final类变量必须在声明的同时初始化。

- 这一节我们学习了在构造对象时如何进行初始化。

内存回收

2.3.2

- 当一个对象在程序中不再被使用时，就成为一个无用对象
- 将在必要时被自动回收

对象的自动回收

- 无用对象
  - 离开了作用域的对象；
  - 无引用指向对象。
- Java运行时系统通过垃圾收集器周期性地释放无用对象所使用的内存。
- Java运行时系统会在对对象进行自动垃圾回收前，自动调用对象的finalize()方法。

垃圾收集器

- 自动扫描对象的动态内存区，对不再使用的对象做上标记以进行垃圾回收
- 作为一个后台线程运行，通常在系统空闲时异步地执行。

- 在对象被回收之前的最后时刻，会自动调用名为finalize的方法。
- 可以在这个方法中释放资源。

finalize()方法

- 在类java.lang.Object中声明，因此Java中的每一个类都有该方法：

  protected void finalize() throws throwable
- 用于释放资源。
- 类可以覆盖（重写）finalize()方法。
- finalize()方法有可能在任何时机以任何次序执行。

- 在这一节我们初步了解了内存回收技术。

枚举类

<2.4>

- 对象的可取值为可列举的特定的值时，可以使用枚举类型

声明枚举类

[**public**] enum 枚举类型名称 [**implements** 接口名称列表]
{
    枚举值;
    变量成员声明及初始化;
    方法声明及方法体;
}

- 下面看一个简单的枚举类型例子

例：简单的枚举类型

```
enum Score {
        EXCELLENT,
        QUALIFIED,
        FAILED;
  };
public class ScoreTester {
        public static void main(String[] args) {
                giveScore(Score.EXCELLENT);
        }
```

例：简单的枚举类型（续）

```
public static void giveScore(Score s){
        switch(s){
            case EXCELLENT:
                System.out.println("Excellent");
                break;
            case QUALIFIED:
                System.out.println("Qualified");
                break;
            case FAILED:
                System.out.println("Failed");
                break;
        }
    }
}
```

- 枚举类中也可以声明构造方法和其他用于操作枚举对象的方法

枚举类的特点

- 枚举定义实际上是定义了一个类；
- 所有枚举类型都隐含继承（扩展）自java.lang.Enum，因此枚举类型不能再继承其他任何类；
- 枚举类型的类体中可以包括方法和变量；
- 枚举类型的构造方法必须是包内私有或者私有的。定义在枚举开头的常量会被自动创建，不能显式地调用枚举类的构造方法。

枚举类型的默认方法

- 静态的values()方法用于获得枚举类型的枚举值的数组；
- toString方法返回枚举值的字符串描述；
- valueOf方法将以字符串形式表示的枚举值转化为枚举类型的对象；
- ordinal方法获得对象在枚举类型中的位置索引。

例：声明了变量和方法的枚举类

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7);

    private final double mass;   // in kilograms
    private final double radius; // in meters
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
```

说明：

- 本例取自于《The Java Tutorials 》（ http://docs.oracle.com/javase/tutorial/java/TOC.html ）
- Planet是一个代表太阳系中行星的枚举类型，其中定义了 mass 和 radius常量（ final ）成员。

例：声明了变量和方法的枚举类（续）

```
private double mass() { return mass; }
private double radius() { return radius; }

// universal gravitational constant  (m3 kg-1 s-2)
public static final double G = 6.67300E-11;

double surfaceGravity() {
    return G * mass / (radius * radius);
}
double surfaceWeight(double otherMass) {
    return otherMass * surfaceGravity();
}
```

例：声明了变量和方法的枚举类（续）

```
    public static void main(String[] args) {
      if (args.length != 1) {
        System.err.println("Usage: java Planet <earth_weight>");
          System.exit(-1);
      }
      double earthWeight = Double.parseDouble(args[0]);
      double mass = earthWeight/EARTH.surfaceGravity();
      for (Planet p : Planet.values())
        System.out.printf("Your weight on %s is %f%n",
                          p, p.surfaceWeight(mass));
    }
}
```

例：声明了变量和方法的枚举类（续）

从命令行运行程序：

```
java Planet 175
Your weight on MERCURY is 66.107583
Your weight on VENUS is 158.374842
Your weight on EARTH is 175.000000
Your weight on MARS is 66.279007
Your weight on JUPITER is 442.847567
Your weight on SATURN is 186.552719
Your weight on URANUS is 158.397260
Your weight on NEPTUNE is 199.207413
```

- 这一节我们学习了Java的枚举类

应用举例

<2.5>

- 在这一节以银行账户类为例，演示和复习一下本章的语法。首先看一下我们前面例题中设计的初步的银行账户类。

初步的BankAccount类—BankAccount.java

包括状态、构造方法、get方法及set方法

```
public class BankAccount{
    private String ownerName;
    private int accountNumber;
    private float balance;
    public BankAccount() {
       this("", 0, 0);
    }
    public BankAccount(String initName, int initAccNum, float initBal) {
      ownerName = initName;
      accountNumber = initAccNum;
      balance = initBal;
    }
```

BankAccount.java(续)

```
    public String getOwnerName()  { return ownerName; }
     public int getAccountNumber() { return accountNumber; }
     public float getBalance()  { return balance; }
     public void setOwnerName(String newName) {
         ownerName = newName;
     }
     public void setAccountNumber(int newNum) {
         accountNumber = newNum;
     }
     public void setBalance(float newBalance) {
         balance = newBalance;
    }
}
```

测试类——AccountTester.java

```
public class AccountTester {
    public static void main(String args[]) {
        BankAccount    anAccount;
        anAccount = new BankAccount("ZhangLi", 100023,0);
        anAccount.setBalance(anAccount.getBalance() + 100);
        System.out.println("Here is the account: " + anAccount);
        System.out.println("Account name: "+anAccount.getOwnerName());
        System.out.println("Account number: anAccount.getAccountNumber());
        System.out.println("Balance: $" + anAccount.getBalance());
    }
}
```

测试结果：

Here is the account: BankAccount@372a1a

Account name: ZhangLi

Account number: 100023

Balance: $100.0

- 接下来我们对银行帐户类BankAccount进行一系列修改和测试

对银行帐户类进行修改

- 覆盖toString()方法；
- 声明存取款方法；
- 使用DecimalFormat类；
- 声明类方法生成特殊的实例；
- 声明类变量。

覆盖toString()方法

- toString()方法
  - 将对象的内容转换为字符串
    - 下面的两行代码等价

```
System.out.println(anAccount);
System.out.println(anAccount.toString());
```

- 如果需要特殊的转换功能，则需要自己覆盖toString()方法：
  - 必须被声明为public；
  - 返回类型为String；
  - 方法的名称必须为toString，且没有参数；
  - 在方法体中不要使用输出方法System.out.println()。

- 下面我们来看一下在改进的银行账户类中如何覆盖toString()方法

为BankAccount覆盖toString()方法

- 覆盖的toString()方法

```
public String toString() {
   return("Account #" + accountNumber + " with balance $" + balance);
}
```

- 重新编译BankAccount类，并运行测试类BankAccountTester，结果如下：

```
Here is the account: Account #100023 with balance $100.0
Account name: ZhangLi
Account number: 100023
Balance: $100.0
```

- 一般来说，余额因发生存取款而变动，不应人为设置余额。接下来，我们为银行账户类添加存取款方法

存取款方法

- 给BankAccount类增加存款及取款方法

```
//存款
    public float deposit(float anAmount) {
        balance += anAmount;
        return(balance);
        }
// 取款
    public float withdraw(float anAmount) {
        balance -= anAmount;
        return(anAmount);
    }
```

测试存取款——修改AccountTester.java

```
public class AccountTester {
   public static void main(String args[]) {
        BankAccount    anAccount;
        anAccount = new BankAccount("ZhangLi", 100023,0);
        anAccount.setBalance(anAccount.getBalance() + 100);
        System.out.println(anAccount);
        System.out.println();
        anAccount = new BankAccount("WangFang", 100024,0);
        System.out.println(anAccount);
        anAccount.deposit(225.67f);
        anAccount.deposit(300.00f);
        System.out.println(anAccount);
        anAccount.withdraw(400.17f);
        System.out.println(anAccount);
    }
}
```

测试结果

```
Account #100023 with balance $100.0

Account #100024 with balance $0.0
Account #100024 with balance $525.67
Account #100024 with balance $125.49997
```

- 如果需要对输出的数据进行格式化，可以使用java.text包中的DecimalFormat类

DecimalFormat类

- DecimalFormat类在java.text包中。
- 在toString()方法中使用DecimalFormat类的实例方法format对数据进行格式化。

修改后的toString()方法

```
public String toString() {
    return("Account #" + accountNumber + " with balance " +
     new java.text.DecimalFormat("$0.00").format(balance));
}
```

重新编译BankAccount类，再运行BankTester类，运行结果如下：

```
Account #100023 with balance $100.00

Account #100024 with balance $0.00
Account #100024 with balance $525.67
Account #100024 with balance $125.50
```

- 如果需要生成一些特殊的样例账户，可以声明类方法，生成特殊的实例。

使用类方法生成特殊的实例

```
public static BankAccount example1() {
        BankAccount ba = new BankAccount();
        ba.setOwnerName("LiHong");
          ba.setAccountNumber(554000);
        ba.deposit(1000);
           return ba;
}
public static BankAccount example2() {
      BankAccount ba = new BankAccount();
      ba.setOwnerName("ZhaoWei");
      ba.setAccountNumber(554001);
      ba.deposit(1000);
      ba.deposit(2000);
      return ba;
 }
```

```
public static BankAccount
emptyAccountExample()
{
    BankAccount ba = new BankAccount();
    ba.setOwnerName("HeLi");
    ba.setAccountNumber(554002);
    return ba;
}
```

- 通常账号是按照某种规则自动生成的，不应该随意设置账号。因此我们需要修改构造方法自动设置账号，本例中将以对象的序号作为账号。取消设置账号的方法。

修改账号生成、余额变动方式

- 修改构造方法，取消帐号参数；
- 不允许直接修改账号，取消setAccountNumber方法；
- 增加类变量LAST_ACCOUNT_NUMBER，初始值为0，当生成一个新的BankAccount对象时，其帐号（accountNumber）自动设置为为LAST_ACCOUNT_NUMBER的值累加1；
- 取消setBalance方法，仅通过存取款在操作改变余额。

- 下面我们来看一下修改后的完整程序

完整的BankAccount2.java

```
public class BankAccount2 {
    private static int LAST_ACCOUNT_NUMBER = 0;
    private int accountNumber;
    private String ownerName;
    private float balance;
    public BankAccount2() {  this("", 0);  }
    public BankAccount2(String initName) { this(initName, 0); }
    public BankAccount2(String initName, float initBal) {
      ownerName = initName;
      accountNumber = ++LAST_ACCOUNT_NUMBER;
      balance = initBal;
    }
```

完整的BankAccount2.java（续）

```
public static BankAccount2 example1() {
  BankAccount2 ba = new BankAccount2();
  ba.setOwnerName("LiHong");
  ba.deposit(1000);
  return ba;
}
public static BankAccount2 example2() {
  BankAccount2 ba = new BankAccount2();
  ba.setOwnerName("ZhaoWei");
  ba.deposit(1000);
  ba.deposit(2000);
  return ba;
}
public static BankAccount2 emptyAccountExample() {
  BankAccount2 ba = new BankAccount2();
  ba.setOwnerName("HeLi");
  return ba;
}
```

完整的BankAccount2.java（续）

```
public int getAccountNumber() {
  return accountNumber;
}
public String getOwnerName() {
  return ownerName;
}
public float getBalance() {
  return balance;
}
public void setOwnerName(String aName) {
  ownerName = aName;
}
```

完整的BankAccount2.java（续）

```
    public String toString() {
      return("Account #"+
               new java.text.DecimalFormat("000000").format(accountNumber) +
              " with balance " +
              new java.text.DecimalFormat("$0.00").format(balance));
    }
    public float deposit(float anAmount) {
      balance += anAmount;
      return balance;
    }
    public float withdraw(float anAmount) {
      if (anAmount <= balance)
      balance -= anAmount;
      return anAmount;
    }
}
```

测试程序AccountTester2.java

```
public class AccountTester2 {
    public static void main(String args[]) {
        BankAccount2 bobsAccount, marysAccount, biffsAccount;
        bobsAccount = BankAccount2.example1();
        marysAccount = BankAccount2.example1();
        biffsAccount = BankAccount2.example2();
        marysAccount.setOwnerName("Mary");
        marysAccount.deposit(250);
        System.out.println(bobsAccount);
        System.out.println(marysAccount);
        System.out.println(biffsAccount);
    }
}
```

测试结果

```
Account #000001 with balance $1000.00
Account #000002 with balance $1250.00
Account #000003 with balance $3000.00
```

- 在这一节中我们通过一个例子，体验和复习了本章介绍的语法。

第2章小结

本章主要内容

- 面向对象程序设计的基本概念和思想
- Java语言类与对象的基本概念和语法
  - 类的声明
  - 类成员的访问
  - 对象的构造
  - 初始化和回收