实用Java教程

基于BlueJ的对象优先方法（第3版）

[英] David J. Barnes　Michael Kölling 著

第九章 继承深入

Java语言程序设计
课程教案

本章将涉及到的主要概念

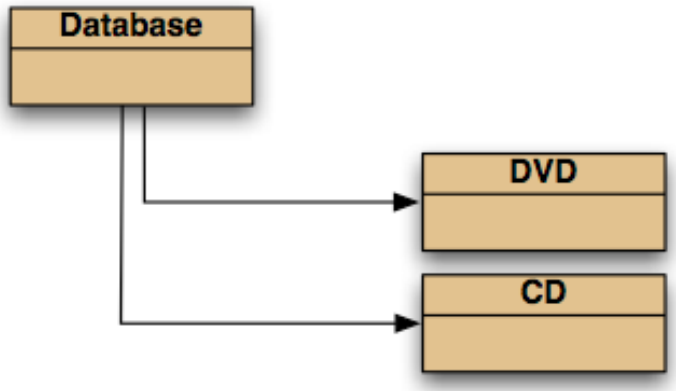
- 静态类型和动态类型 (static and dynamic type)
- 覆写 (overriding)
- 动态方法查找 (dynamic method lookup)
- 方法多态性 (method polymorphic)
- 受保护的访问 (protected access)

本章的目标

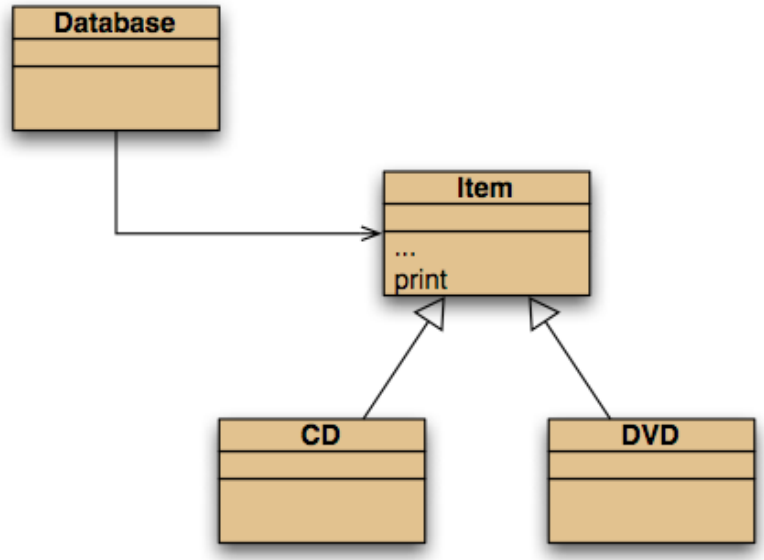
深入了解Java继承机制中的更多细节来提高程序设计质量

对 DoME 案例的进一步分析

Database of Multimedia Entertainment: 一个存储CD和DVD信息的应用程序



version 1



version 2

DoME 中的打印方法

DoME 第二个版本中的打印方法没有显示出每个条目的所有数据

版本一的输出:

```
CD: A Swingin' Affair (64 mins)*
    Frank Sinatra
    tracks: 16
    my favourite Sinatra album

DVD: O Brother, Where Art Thou? (106 mins)
     Joel & Ethan Coen
     The Coen brothers' best movie!
```

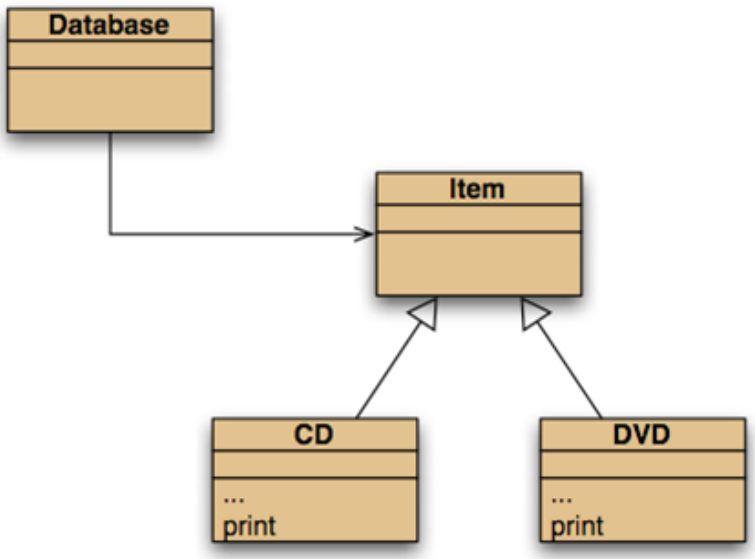
版本二的输出:

```
title: A Swingin' Affair (64 mins)*
       my favourite Sinatra album

title: O Brother, Where Art Thou? (106 mins)
       The Coen brothers' best movie!
```

DoME 中的打印方法

- print方法是在父类Item中实现的，而不是在CD类或者DVD类中实现的
  - CD类或DVD类只是继承了Item类中的print方法
- 继承是单向的
  - 在Item类的方法中，只有在Item中定义的字段才是有效的
  - 如果在print方法中试图访问CD类或DVD类的字段，则会导致程序错误

如何解决这个问题



- 把print方法移到子类中去
  - 在Item类中删除print方法
  - 在CD类和DVD类中分别定义各自的print方法，访问各自的字段
- 但是，编译以后系统会报错
  - CD类和DVD类中的方法不能访问其父类中定义的字段
  - Database类找不到Item类中的print方法

动态类型与静态类型

Java中的类型检查机制导致了上述第二个问题。参考下例:

c1的类型是什么:

```
Car c1 = new Car();
```

v1的类型是什么:

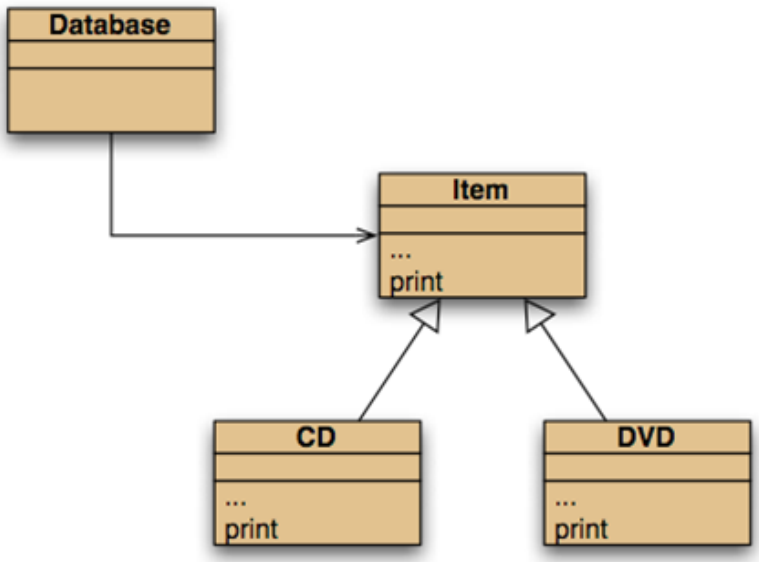
```
Vehicle v1 = new Car();
```

静态类型 变量的静态类型是其在源代码的变量声明语句中声明的类型。

动态类型 变量的动态类型是运行时其当前保存的对象的类型。

- 注意: Java编译器在做类型检查时是使用静态类型。

如何解决这个问题



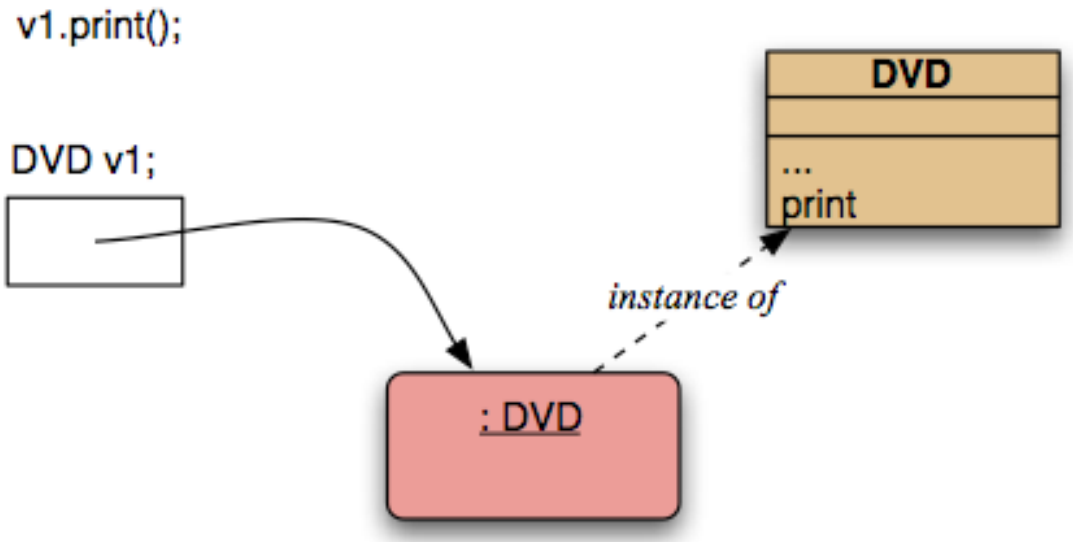
- 在Item类、CD类和DVD类中都定义print方法
  - 能够满足类型检查的规则
  - 三个类中的print方法分别打印各自类中定义的字段
  - 在Java中，这种解决方案被称为“覆写”或“重定义”

- 覆写（overriding）
  - 子类可以覆写父类中定义的方法
  - 子类中定义的方法与父类中的方法具有完全相同的签名，但是方法体不同
  - 每个方法都能够访问其所在类中所定义的字段
  - 父类中的方法能够满足静态类型检查时的要求
  - 子类中的方法在子类对象对该方法调用时具有优先权

动态查找方法

在没有继承或多态的应用场景下，方法调用的步骤相对简单

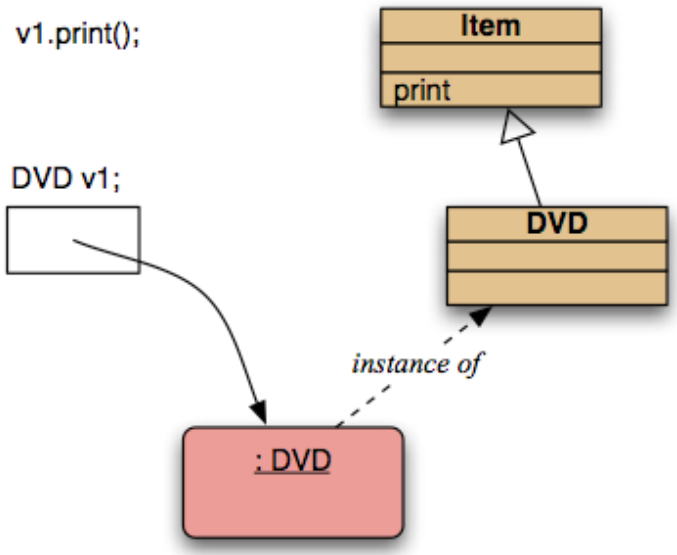
- 访问变量
- 根据引用找到变量中存储的对象
- 根据引用找到对象所属的类
- 在对象所属的类中找到相应方法的实现并执行它



动态查找方法

在使用继承（没有覆写）的应用场景下，方法调用的步骤相对复杂

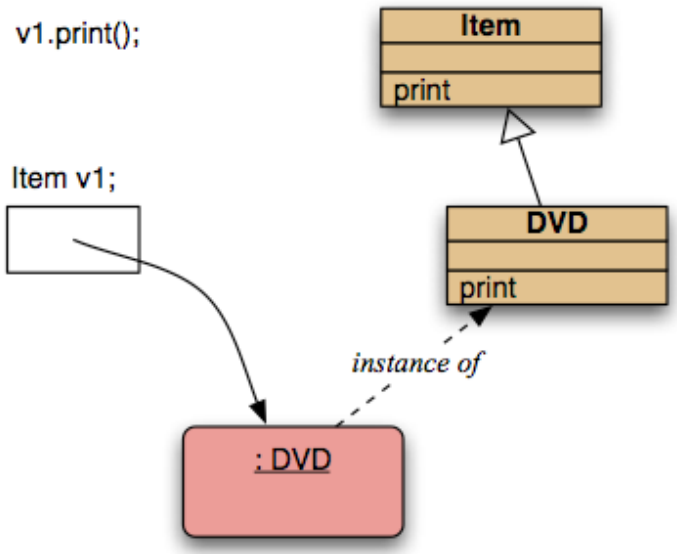
- 访问变量
- 根据引用找到变量中存储的对象
- 根据引用找到对象所属的类
- 在对象所属的类中没有找到相应方法
- 在对象所属的类的父类中查找相应方法的实现并执行它（如果父类中还找不到，就沿类继承层次向上查找，直到Object类）



动态查找方法

在使用继承，并同时具有覆写的应用场景下，方法调用的步骤与普通继承的规则是一样的，但需注意：

- 查找哪个类中的方法并执行它，是由动态类型而不是静态类型决定的
- 子类中的覆写方法比父类方法的优先级高
- 一个方法被覆写后，只有最后的版本（在继承层次中最低的）会被执行



方法中的父类调用

- 由于方法的动态查找机制，父类中被覆写的方法对子类对象来说是被隐藏的
- 但是可以通过super.methodname(...)的形式在子类方法中调用父类中被覆写的方法

```
public class CD extends Item
{
    ...
    public void print()
    {
        super.print();
        System.out.println("    " + artist);
        System.out.println("    tracks: " + numberOfTracks);
    }
    ...
}
```

两种父类调用的区别

- 与在构造器中的父类调用不同，方法中的父类调用需要显式地标明方法名。
  - super.方法名(参数列表)
- 与在构造器中的父类调用不同，方法中的父类调用可以出现在方法体中的任何地方，不一定要在第一行。
- 与在构造器中的父类调用不同，方法中的父类调用不会自动生成，也不要求一定有父类调用。

方法多态性

- 前述内容所讨论的内容就是多态性的一种形式，也叫做动态方法分派（dynamic method dispatch），或方法多态性（method polymorphism）。
- Java中的每个对象变量都是（潜在）的多态变量。
  - 一个多态变量可以存储不同类型的对象，
- Java的方法调用也是多态的。
  - 相同的方法调用语句在不同的时候可能会调用不同的方法，取决于变量中对象的动态类型。

方法多态性的例子：Object的toString方法

- Object类中的方法会被所有的类所继承
- Object类中的方法可能被其子类所覆写
  - Object类中的toString方法就常常被其子类覆写
  - public String toString()
  - 该方法的作用是给出对象的字符串表达形式

在 DoME 中覆写 toString 方法

Item.java

```
public class Item
{
    ...
    public String toString()
    {
        String line1 = title + " (" + playingTime + " mins)");
        if(gotIt) {
            return line1 + "*\n" + "    " + comment + "\n");
        } else {
            return line1 + "\n" + "    " + comment + "\n");
        }
    }

    public void print()

        System.out.println(toString());

    ...
}
```

在 DoME 中覆写 toString 方法

CD.java

```
public class CD extends Item
{
    ...
    public String toString()
    {
        return super.toString() +  "    " +
               artist + "\n    tracks: " + numberOfTracks + "\n";
    }

    public void print()
    {
        System.out.println(toString());
    }
    ...
}
```

在 DoME 中覆写 toString 方法

- 在覆写了toString方法的情况下，可以不再需要在Item、CD和DVD类中分别定义print方法
- 如果传给System.out.println和System.out.print方法的参数如果不是一个字符串对象，系统会自动调用该对象的toString方法

```
CD cd1 = new CD();
cd1.print();

CD cd2 = new CD();
System.out.println(cd2.toString);

CD cd3 = new CD();
System.out.println(cd3);
```

方法多态性的例子：对象相等性比较

- 程序经常需要比较两个对象是否“相等”？
  - 引用相等（reference equality），通过“==”运算符比较
  - 内容相等（content equality），通过“equals”方法比较
- 有些程序可能需要定义自己的“相等”比较规则
  - 先进行引用相等性比较
  - 如果引用不相等，再进行自定义的内容相等性比较
  - 可以通过覆写equals方法实现自定义的相等性比较

方法多态性的例子：对象相等性比较

- 如何比较两个对象是否表示同一个“学生”？

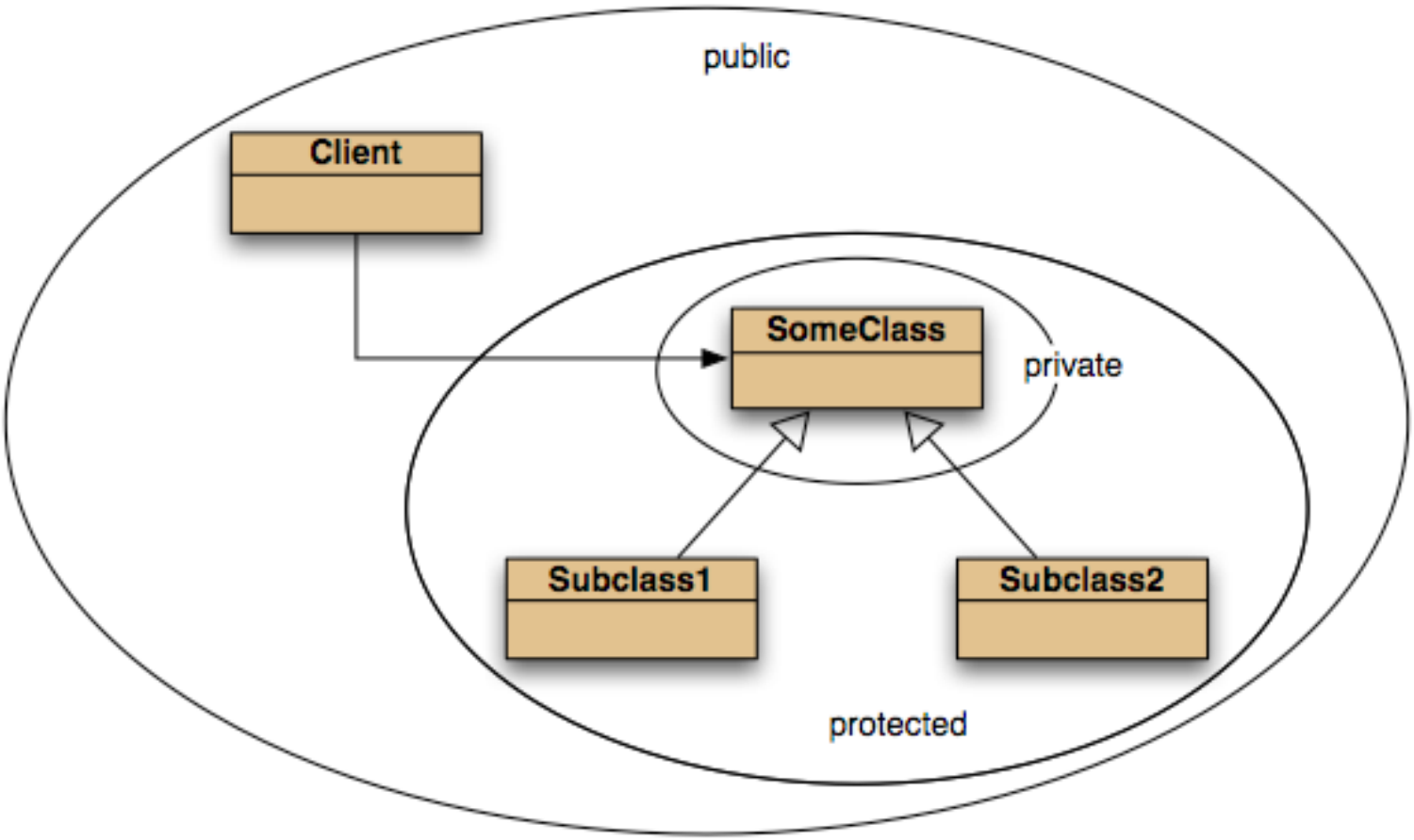
Studnet.java

```
public class Student
{
    ...

    public boolean equals(Object obj)
    {
        if(!(obj instanceof Student))
            return false;
        if(this == obj)
            return true;
        Student other = (Student) obj;
        return name.equals(other.name) &&
               id.equals(other.id) &&
               credits == other.credits;
    }

    ...
}
```

受保护的访问

- 任何类中受private访问控制符所修饰的字段和方法不能被其它类所访问，即使是其子类也不能访问，这一规定在有些时候过于严格。
- Java提供了专门的protected访问控制符，它所具有的访问权限介于public和private之间。
  - 用protected修饰的一个字段或方法，可以允许其（直接的或间接的）子类直接访问这个字段或方法。
  - 在不具有继承关系的其它类中不能访问用protected修饰的字段或方法。
- 为保证数据封装，Java的编程习惯通常是尽可能将所有字段定义为private的，但可以提供protected的访问/修改方法供指定范围内的其它类调用。

受保护的访问


public
Client
SomeClass
private
Subclass1
Subclass2
protected

本章小结

- 变量在源代码中所声明的类型是静态类型
  - Java编译器使用静态类型进行类型检查
- 变量中当前存储的对象的类型是动态类型
  - 程序运行时根据动态类型进行方法查找
- 父类中的方法能够被其子类所覆写
- 在子类的方法体中，super关键字可以用来调用父类中的相应方法
- 如果字段或方法被protected访问控制符所修饰，子类可以访问这些字段或方法，但是其他类不可以。