JAVA 第四章 对象与类

4.1 使用预定义类

*像C++一样，Java中也有很多预定义好的类，我们不必了解其中具体是怎么实现的，我们只需要知道方法名字以及参数就行，这也是我们在后面所要掌握的**封装库**的关键所在。*

*比如我们后面经常所用到的Date类，该类的作用是描述时间点的。我们可以通过**new Date() 来构造一个Date对象。***

当然了这是对于一般的类，还有一种类是抽象类，对应我们在C++中所学习的（可以参考：https://www.runoob.com/cplusplus/cpp-interfaces.html），我们在C++中先通过类的多态引进了虚函数的概念，而含有虚函数的类就被称为抽象类，该种类不能被用来实例化对象，只能作为接口使用，即用来继承其他的类。对于接口，我们可以利用接口来实现相应的功能，也就是我们在派生类中重写虚函数。

回到正题，如果我们在Java中遇到了抽象的类，一般都是用abstract来修饰的，我们就不能去实例化一个对象，但是我们可以通过抽象类中的方法（如果有合适的话）取得一个所需要类得实例，比如说Canlendar类，该类是Date类的替代品。（可以参考：https://blog.csdn.net/qq_40828705/article/details/103486834）

4.1.1 更改器和访问器

更改器方法**，我们构造完一个对象之后，如果调用相关的 方法后，对象的状态发生改变，那么说明该方法为更改器方法；相反，如果只访问对象而不去修改对象的方法称为**访问器方法。

C++注释： *在C++中，带有const后缀的方法称为**访问器方法**，即不能修改传进来的参数，没有声明为const的方法默认为**更改器方法**。但是，在Java语言中，访问器与更改器方法在语法上没有明显的区别。*

4.3 用户自定义类

4.3.1 多个源文件的使用

Java和C++一样，每一个程序可能有许多程序员自己编写的类，我们通常习惯将每一个类存放在一个单独的源文件中。

这样的话，我们编译源程序的时候，一种是使用通配符用Java编译器：

如果将许多个类分为很多个源文件的时候，我们不需要再像C++中的那样去再导入相关的类到主函数文件中，我们直接编译主函数所在的源文件即可，编译器会将其中涉及到的源文件编译。

```
javac Employee*.java
//该命令将Employee.java和EmployeeTest.java编译成类文件
```

或者直接键入：

```
javac EmployeeTest.java
```

该种方法是当Java编译器发现EmloyeeTest.java使用Employee类，他会查找名为Employee.class的文件。如果没有找到该文件的话，就会自动搜索Employee.java，然后对其进行编译。

4.3.2 构造器（C++中的构造函数）

构造器需要与类同名。在构造Employee类的对象的时候，构造器会运行，从而将实例字段初始化为所希望的初始状态。

***注意：**Java中的构造器与C++中的构造函数不一样，构造器总是结合new运算符来调用的。不可以对一个已经存在的对象调用构造函数来达到重新设置实例字段的目的。例如：*

```
james.Employee("James Bond",250000,1950,1,1);//Error
```

正确的构造器的调用方法：

```
Employee harry = new Employee("Harry Hacker",50000,1989,10,1);
```

注意：

·构造器与类同名；

·每一个类可以有一个以上的构造器；

·构造器可以有0个、1个、或者多个参数；

·构造器没有返回值；

·构造器总是伴随着new操作符一起调用；

4.3.2 用var声明局部变量

在Java中，如果可以从变量的初始值推断出他们的类型，那么可以用var关键字声明局部变量，而不需要再去指定类型。例如，可以不这样声明：

```
Employee hary = new Employee("Harry Hacker",50000,1989,10,1);
```

我们可以直接这样书写：

```
var harry = new Employee("Harry Hacker",50000,1989,10,1);
```

*这样写的好处就是可以避免重复写类型名Employee，我们直接用var就可以替代。所以有如下规则：**倘若无需了解任何Java API 就可以从等号右边明显的看出类型，在该种情况下，我们都可以去使用var表示法，不过我们不会对数值类型使用var，比如int、long或double，因为这样的话，0、0L和0.0之间的区别就没有了。***

4.3.3 隐式参数和显式参数

*显式参数就是方法后面的括号里面的数值；**隐式参数**，是出现在方法名字前面的**--Employee类型的对象--**。*

```
public void raiseSalary(double byPercent)
{
    double raise = this.salary * byPercent/100;
    this.salary += raise;
}
//这样书写的话，可以将实例字段和局部变量明显的区分开
```

4.3.4 封装的优点

警告：*注意不要编写返回可变对象引用的访问器方法。*

比如说：

```
class Employee
{
    private Date hireDay;
    private int age;
    ...
    public Date getHireDay()
    {
        return hireDay;//BAD
    }
    public int getAge()
    {
        return age;//Right
    }
}
```

Date 对象是可变的，这一点就破坏了封装性！请看下面这段代码：

```
Employee harry = . . .;
Date d = harry.getHireDay();
double tenYearsInMilliSeconds = 10 * 365.25 * 24 * 60 * 60 * 1000;
d.setTime(d.getTime() - (long) tenYearsInMilliSeconds);
// let's give Harry ten years of added seniority
```

出错的原因很微妙。d 和 harry.hireDay 引用同一个对象（请参见图 4-5）。对 d 调用更改器方法就可以自动地改变这个雇员对象的私有状态！

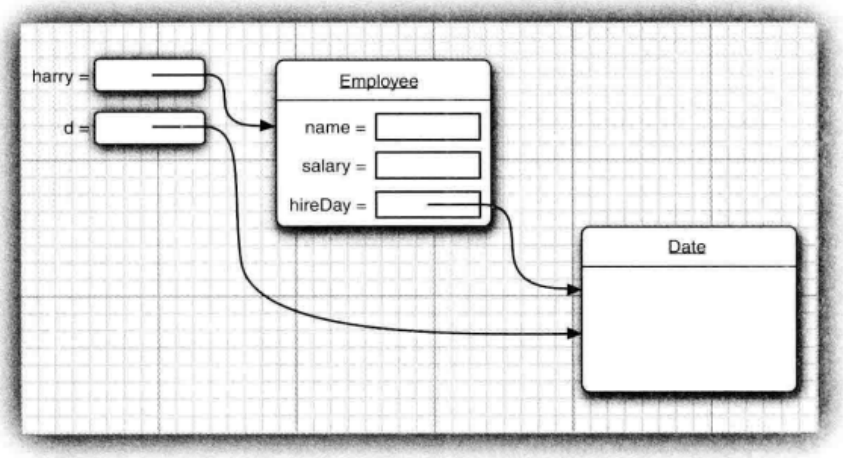


图 4-5　返回可变数据域的引用

如果需要返回一个可变对象的引用，首先应该对它进行克隆。对象克隆是指存放在另一个新位置上的对象副本。

```
class Employee
{
    ...
    public Date getHireDay()
    {
        return (Date) hireDay.clone();//Right
    }
}
```

4.3.5 基于类的访问权限

```
class Employee
{
    public boolean equals(Employee other)
    {
        return name.equals(other.name);
    }
}
```

一个方法可以访问所属类的任何对象（任何实例化对象的私有特性）。

4.3.6 私有方法

如果有时候，我们希望将一个计算代码分解为若干个独立的辅助关系。通常，这些辅助关系不应该成为公共接口的一部分，此时我们就需要将其设置为私有方法。

重点在于，只要方法是私有的，类的设计者就可以确信它不会在别处使用，所以可以将其删去。如果一个方法是公有的，就不能将其简单的删去，因为可能会有其他代码依赖与这个方法。

4.3.7 final实例字段

final修饰符 即修饰常量的，我们可以将实例字段定义为fianl，这样的字段必须在构造对象的时候进行初始化。必须确保在每一个构造器执行之后，这个字段的值已经设置，并且以后不能再修改这个字段。

*final修饰符对于类型为**基本类型或者不可变类的字段尤其有用。（如果类中的所有方法都不会改变其对象，这样的类就叫做不可变类。**例如，String类就是不可变的。）*

对于可变的类，使用final修饰符可能会造成混乱。例如下面字段：

```
private final StringBuilder evaluations;
```

它在Employee构造器中初始化为：

```
evaluations = new StringBuilder();
```

final 关键字只是表示存储在evaluations变量中的对象引用不会再指示另一个不同的StringBuilder对象。不过这个对象可以更改：

```
public void giveGlodStar()
{
    evaluations.append(LocalDate.now() + "Glod star!\n";
}
```

4.4 静态字段与静态方法

4.4.1 静态字段

*每一个类只有一个这样的字段，**该字段是这个类的所有实例共享的。**即使没有Employee对象，静态字段也是存在的。**它属于类，而不属于任何单个的对象。***

4.4.2 静态常量

静态变量使用的较少，但是静态变量却很常用。

4.4.3 静态方法

4.4.4 工厂方法

4.5 方法参数

*1.**按……调用（call by）***

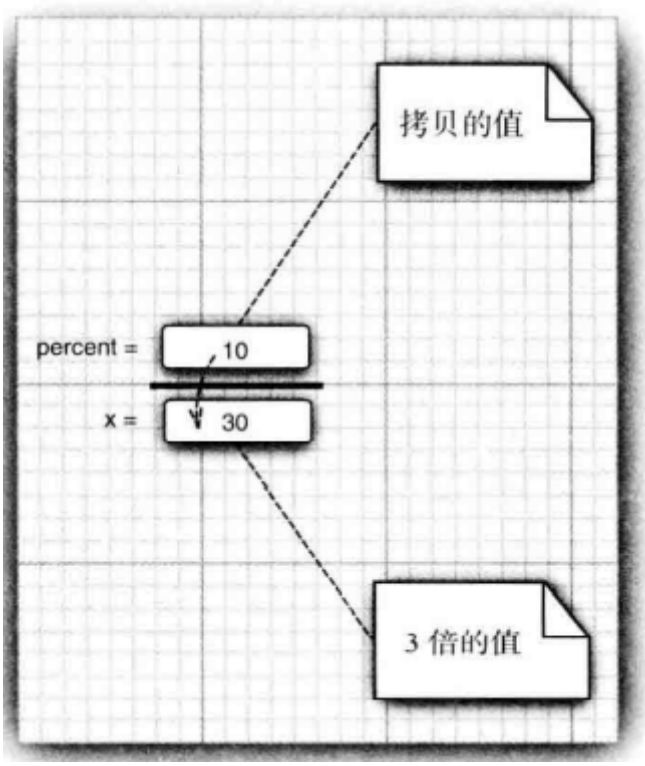
按值调用 *表示方法接受的是传进来参数的值，而不是其本身；*

按引用调用 *表示方法接受是传进来参数的变量地址；*

*2.Java程序设计语言**总是采用按值调用***

方法的得到的是所有参数的一个副本。具体来讲，方法不能修改传递给它的任何参数变量的内容。（和C++是差不多的）

```
1 public static void tripleValue(double x)//doesn't work
2 {
3     x = 3 * x;
4 }
```



3. 有两种类型的方法参数

·基本数据类型（数字、布尔值）

·对象引用（这个就是参数是那些对象，而不是一般的基本变量，这个对象引用相当于我们在C++中所学的引用&）

*但是有一点不用的是，Java中对对象采用的**并不完全**是按引用调用的，实际上对象引用也是按值传递的。*

（可以参考书上的swap方法代码）

4.总结

在Java中对方法参数可以做什么和不可以做什么：

·方法不能修改基本数据类型的参数（即数值型或布尔型）

·方法可以改变对象参数的状态

方法不能让一个对象参数去引用一个新的对象

4.6 对象构造

即我们在C++中所学习的构造函数，其中用到了重载的相关概念，使得一个类可以有许多个构造函数，然后这些构造函数根据不同的情况来构造出我们所需要的类。

·重载构造器（即重载构造函数

·用this(…)调用另一个构造器

·无参数构造器

·对象初始化块

·静态初始化块

·实例字段初始化

(ps:书写参数的技巧，参数前面添加一个a)

4.6.1 调用构造器的具体处理步骤：

1.如果构造器的第一行调用了另一个构造器，则基于所提供的参数执行第二个构造器；

2.否则，

a）所有的数据字段初始化为其默认值（0，false或null）；

b）按照在类中声明中出现的顺序，执行所有的字段初始化方法和初始化块；

3.执行构造器主体代码；

4.6.2 对象析构

在Java中，，Java会完成自动的垃圾回收，不需要人工回收内存，所以Java不支持我们在C++中所需要的析构函数。

（可以参考：https://blog.csdn.net/qq_37823003/article/details/107333386）

4.7 包