1 一个实例：Employee类

```
import java.time.*;

public class EmployeeTest
{
    public static void main(String[] args)
    {
        // 构造一个 Employee 数组
        Employee[] staff = new Employee[3];
        staff[0] = new Employee("Carl Cracker", 75000, 1987, 12, 15);
        staff[1] = new Employee("Harry Hacker", 50000, 1989, 10, 1);
        staff[2] = new Employee("Tony Tester", 40000, 1990, 3, 15);

        // 每个人的工资提高 5%
        for(Employee e : staff)
        {
            e.raiseSalary(5);
        }

        // 打印每个员工对象的信息
        for(Employee e : staff)
        {
            System.out.println("name=" + e.getName()
                + ",salary=" + e.getSalary()
                + ",hireDay=" + e.getHireDay());
        }
    }
}

class Employee
{
    private String name;
    private double salary;
    private LocalDate hireDay;

    public Employee(String n, double s, int year, int month, int day)
    {
        name = n;
        salary = s;
        hireDay = LocalDate.of(year, month, day);
    }
```

```
    public String getName()
    {
        return name;
    }

    public double getSalary()
    {
        return salary;
    }

    public LocalDate getHireDay()
    {
        return hireDay;
    }

    public void raiseSalary(double byPercent)
    {
        double raise = salary * byPercent / 100;
        salary += raise;
    }
}
```

在一个源文件中只能有一个公共类，非公共类数量不限，文件名必须与公共类的名字相同。在上面的例子中，有一个公共类`EmployeeTest`，因此源文件名为`EmployeeTest.java`。

2 剖析Employee类

2.1 实例字段和方法

这个类有三个实例字段：

```
private String name;
private double salary;
private LocalDate hireDay;
```

关键字`private`确保只有`Employee`类自身的方法能够访问这些实例字段，而其他类的方法不能访问。一般将实例字段标记为`private`，以确保封装不被破坏。

这个类包含一个构造器和4个方法：

```
public Employee(String n, double s, int year, int month, int day)
public String getName()
public double getSalary()
public LocalDate getHireDay()
public void raiseSalary(double byPercent)
```

这个类的所有方法都被标记为`public`。关键字`public`意味着任何类的任意方法都可以调用这些方法。

2.2 构造器

构造器有如下要点：

- 构造器与类同名。
- 每个类可以有一个以上的构造器。
- 构造器没有返回值。

- 构造器总是伴随着 `new` 操作符一起使用。
- 不能对一个已经存在的对象调用构造器来重新设置实例字段。

2.3 使用null引用

定义一个类时，最好清楚地知道哪些字段可能为 `null`。在上面的 `Employee` 类中，`salary` 是基本类型，不可能为 `null`；`hireDay` 字段在构造器中初始化为一个 `LocalDate` 对象，也不会为 `null`。而 `name` 字段可能为 `null`，如果调用构造器时使用的参数为 `null`，`name` 就会是 `null`。

对此有两种解决方法。“宽容型”方法是把 `null` 参数转换为一个适当的非 `null` 值：

```
if(n == null)
    name = "unknown";
else
    name = n;
```

在Java 9中，`Objects` 类对此提供了一个便利方法：

```
name = Objects.requireNonNullElse(n, "unknown");
// 如果 n 不为 null 则返回 n，如果 n 为 null 则返回第二个参数指定的默认对象
```

“严格型”方法直接拒绝 `null` 参数：

```
Objects.requireNonNull(n, "The name cannot be null");
// 如果 n 为 null，抛出 NullPointerException 异常，并给出指定的提示
name = n;
```

2.4 隐式参数与显式参数

方法用于操作对象以及存取它们的实例字段。调用方法时既需要给出参数，也需要指明对哪个对象进行操作。操作对象并没有在参数中列出，而是出现在方法名之前，称为隐式参数；方法声明中列出的参数称为显式参数。

在每一个方法中，关键字 `this` 指示隐式参数，默认情况下是隐含的。可以在方法中显式使用 `this` 关键字，将实例字段与局部变量区分开来。例如：

```
public void raiseSalary(double byPercent)
{
    double raise = this.salary * byPercent / 100;
    this.salary += raise;
}
```

2.5 封装性

`Employee` 类中的 `getName`、`getSalary`、`getHireDay` 方法是典型的访问器方法，它们只返回实例字段值，因此又称为字段访问器。

类的实例字段一般设置为 `private`，为了在类外能够获取或修改实例字段的值，需要提供公共的字段访问器方法和公共的字段更改器方法。这样做有如下好处：

- 当实例字段值出现错误时，只需要调试字段访问器和字段更改器，便于定位和解决错误。
- 可以在改变方法的内部实现时不影响其他代码。

> 注意：不要编写返回可变对象引用的访问器方法。
> 如果将`Employee`类中的`hireDay`字段改为`Date`类，就违反了这个原则：

```
public Employee
{
    private Date hireDay;
    ...
    public Date getHireDay()
    {
        return hireDay;
    }
    ...
}
```

> `LocalDate`类没有更改器方法，而`Date`类有一个更改器方法`setTime`。`Date`对象是可变的，这一点就破坏了封装性。例如：

```
Employee harry = ...;
Date d = harry.getHireDay();
double tenYearsInMilliSeconds = 10 * 365.25 * 24 * 60 * 60 * 1000;
d.setTime(d.getTime() - (long) tenYearsInMilliSeconds);
```

> 这时`d`和`harry.hireDay`引用同一个对象，对`d`调用更改器方法就会改变`harry`的状态。
> 如果需要返回一个可变对象的引用，应该对它进行克隆。

2.6 final实例字段

可以用`final`关键字定义常量实例字段。这样的字段必须在构造对象时初始化，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不可变类的字段尤其有用。对于可变的类，使用`final`修饰符可能会造成混乱。`final`关键字只是表示存储在对象变量中的对象引用不会再引用另一个不同的对象，而对象本身可以更改。

实例字段一般都定义为私有以确保封装性，而常量字段定义为公有并不会有问题，因为常量字段不可修改。