> 由 KiOii (*_EM_Cpper*)整理。 （KiOii (*_EM_Cpper*) makes this note.）

Objects and Classes

- 4.1 Introduction to Object-Oriented Programming
  - 4.1.1 Classes
  - 4.1.2 Objects
  - 4.1.3 Identifying Classes
  - 4.1.4 Relationships between Classes
- 4.2 Using Predefined Classes
  - 4.2.1 Objects and Object Variables
  - 4.2.2 The LocalDate Class of the Java Library
  - 4.2.3 Mutator and Accessor Methods
- 4.3 Defining Your Own Classes
  - 4.3.2 Use of Multiple Source Files
  - 4.3.3 Dissecting the Employee Class
  - 4.3.4 First Steps with Constructors
  - 4.3.5 Implicit and Explicit Parameter
  - 4.3.6 Benefits of Encapsulation
  - 4.3.7 Class-Based Access Privileges
  - 4.3.8 Private Methods
  - 4.3.9 Final Instance Fields
- 4.4 Static Fields and Methods
  - 4.4.1 Static Fields
  - 4.4.2 Static Constants
  - 4.4.3 Static Methods
  - 4.4.4 Factory Method
  - 4.4.5 The main Method
- 4.5 Method Parameters
- 4.6 Object Construction
  - 4.6.1 Overloading
  - 4.6.2 Default Field Initialization
  - 4.6.3 The Constructor with No Arguments
  - 4.6.4 Explicit Field Initialization --- 新操作
  - 4.6.5 Parameter Names
  - 4.6.6 Calling Another Constructor --- 新操作
  - 4.6.7 Initialization Blocks --- 新操作
  - 4.6.8 Object Destruction and the finalize Method
- 4.7 Packages
  - 4.7.1 Class Importation
  - 4.7.2 Static Imports
  - 4.7.3 Addition of a Class into a Pakage --- 新操作
  - 4.7.4 Package Scope
- 4.8 The Class Path
  - 4.8.1 Setting the Class Path
  - 关于package和classpath的使用例子总结
- 4.9 Documentation Comments
  - 4.9.1 Comment Insertion
  - 4.9.2 Class Comments
  - 4.9.3 Method Comments
  - 4.9.4 Field Comments

4.9.5 General Comments
4.9.6 Package and Overview Comments
4.9.7 Comment Extraction
4.10 Class Design Hints

Objects and Classes

In this Chapter

- **4.1 Introduction to Object-Oriented Programming** (*介绍 OOP*)
- **4.2 Using Predefined Classes** (*使用已经定义好的类*)
- **4.3 Defining Your Own Classes** (*定义你自己编写的类*)
- **4.4 Static Fields and Methods** (*静态字段和方法*)
- **4.5 Method Parameters** (*方法的形参*)
- **4.6 Object Construction** (*对象的构造函数*)
- **4.7 Package** (*包*)
- **4.8 The Class Path** (*关于类路径问题*)
- **4.9 Documention Comments** (*文档化注释*)
- **4.10 Class Design Hints** (*类的设计技巧*)

> 本章主要介绍了：
>
> - 介绍 OOP
> - 创建标准库类对象
> - 编写自己的类

4.1 Introduction to Object-Oriented Programming

- Java is Object-Oriented

  > 面向对象的程序是由对象组成的
  >
  > 数据抽象是一种目的：实现是被隐藏的，接口可被暴露的
  >
  > 封装技术是一种手段：实现数据抽象

4.1.1 Classes

Class （类）

- *define:* A **template** to make objects

  > 是我们用于创建对象群的模板

Instance (实例化)

- *define:* **construct** an **object** from a **class**

  > 由类构造出一个对象的过程：实例化

Encapsulation （封装）

- *define:* **imformation hiding**

> 概念上：数据隐藏技术

- *behavior:* **combining** data and behavior in one package,and **hiding** the implementation detail from the users of the object.

> 做法上：将数据和行为分开并放置于类中

Instance fields

- *define:* The bits of **data in an object** (注意区别于静态)

> 1.一个类实例出的特定对象具有特定字段值 （对象独立性）
>
> 2.特定字段值的集合是对象当前的状态
>
> 3.无论何时调用特定对象的方法都有可能修改对象的状态
>
> 站在实例对象的角度：状态和方法组成了实例对象

4.1.2 Objects

Three key characteristics of object

- ***The object's behavior***
  - *define:* do what? （**methods**）

    > 你能用这个对象来干嘛？你想让对象拥有什么方法？

- ***The object's state***
  - *define:* object react?

    > 当你调用对象的方法时，对象是如何反应的？

- ***The object's identity***
  - *define:* distinguished?

    > 对象是如何区别于其他拥有同样方法和状态的对象的？

4.1.3 Identifying Classes

> 识别类

- *wraning:* A simple rule of thumb in identifying classes is to look for nouns in the problem analysis.

> 方法是作为动词来看待
>
> 类是作为名词来看待

4.1.4 Relationships between Classes

The most common relationships between classes

- **Dependence(" uses-a ")** (*A依赖B*)
- **Aggregation(" has-a ")** (*B作为A的状态*)
- **Inheritance(" is-a ")** (*A继承了B*)

- *wraning:* Try to minimize the number of classes that depend on each other.

> 尽量减少类之间彼此依赖

UML

> Unified Modeling Language

> 这块内容先跳过: Violet UML Editor

4.2 Using Predefined Classes

4.2.1 Objects and Object Variables

- *constructor:*
  - *define:* A sepcial method to **construct and initialize objects**
  - *wraning*: Constructors always have the same name as the class name.

`new Date()`

- This expression constructs a new object
- The object is initia
- lized to the current date and time

`String s = new Date().toString();`

- use `toString` method to a newly constructed Date object

`Date birthday = new Date();`

- birthday----------->Data object

  > birthday类似于C++中的指针 (object variable)

- There is an important difference between objects and object variable

`Date deadline;`

- deadline doesn't refer to any object

  > deadline没有指向任何对象

`deadline = birthday;`

- Now both variable refer to the same object

  > 现在两个object variable都指向了同样的object

object variable is a reference to an object

- the object is stored elsewhere

initialization

- null value
  - *such as:* `Date d = null;`

> 没有引用任何对象

- calling new
  - *such as:* `Date d = new Date();`

  > 引用了对象

wraning

- In Java, you must use the **clone** method to get a complete copy of an object.

4.2.2 The LocalDate Class of the Java Library

Date Class

- represents a point in time

LocalDate class

- expresses days in the familiar calendar notation

> 日历表示法中表示天数

Wraning

- not use a constructor to construct objects of the **LocalDate** class

Use

- *construct:*
  - ***now*** method: `LocalDate.now();`

  > 使用静态方法来构造新对象，时间被初始化为创建时间

  - ***of*** method: `LocalDate.of(1999,12,31);`
  - *such as:* `LocalDate newYearsEve = LocalDate.of(1999,12,31);`

  > object variable: newYearsEve
  >
  > object : LocalDate.of(1999,12,31);

method

- ***getYear*** method
- ***getMonthValue*** method
- ***getDayOfMonth*** method
- **plusDays** method
  - construct a new object
  - given number of days away from the object

  > 加上天数获取到加上天数的新对象

```
LocalDate newYearsEve = LocalDate.of(1999,12,31);

int year = newYearsEve.getYear();        // 1999
int month = newYearsEve.getMonthValue(); // 12
int day = newYearsEve.getDayMonth();     // 31

LocalDate aThousandDaysLater = newYearsEve.plusDays(1000);
year = aThousandDaysLater.getYear();        // 2002
month = aThousandDaysLater.getMonthValue(); // 09
day = aThousandDaysLater.getDayofMonth();   // 26
```

> Date对象的方法 **getDay**，**getMonth**，**getYear** 已经被弃用了

4.2.3 Mutator and Accessor Methods

> 存取器方法 和 访问方法

- *wraning:* 真实修改对象的方法叫 mutator method

  > 类似newYearsEve.plusDays(1000)并没有修改newYearsEve对象

GreorianCalendar class

- In earlier version of java library
- **add** method: mutator method

  > 添加天数

- **get** method

```
GreorianCalender someDay = new GreorianCalender(1999,11,31);
someDay.add(Calendar.DAY_OF_MONTH,1000); // add

year = someDay.get(Calendar.YEAR); // 2002
month = someDay.get(Calendar.MONTH) + 1; // 09
day = someDay.get(Calendar.DAY_OF_MONTH);//26
```

> 在C++中，可以有const member function，用const来区分mutator和access，但是在java中不存在特殊的语法来区分

CalendarTest example

```
import java.time.*;
/**
*@version 1.0 2018-03-13
*@author KiOii(_EM_Cpper)
*/

public class CalendarTest
```

```
{
    public static void main(String[] args)
    {
        // Local date,now
        LocalDate date = LocalDate.now();
        // get month value (of year)
        int month = date.getMonthValue();
        // get day of month
        int today = date.getDayOfMonth();
        // set to start of month
        date = date.minusDays(today - 1);
        DayOfWeek weekday = date.getDayofWeek();
        int value = weekday.getValue();

        System.out.println("Mon Tue Wed Thu Fri Sat Sun");
        for(int i = 1;i < value;++i)
            System.out.print("    ");
        while(date.getMonthVale() == month)
        {
            System.out.printf("%3d",date.getDayOfMonth());
            if(date.getDayOfMonth() == today)
                System.out.print("*");
            else
                System.out.print(" ");
            date = date.plusDays(1);
            if(date.getDayOfWeek() == 1)
                System.out.println();
        }
        if(date.getDayOfWeek() != 1)
            System.out.println();
    }
}
```

4.3 Defining Your Own Classes

> To build a complete program,you combine several classes,one of which has a **main** method

The Simplest form

```
class ClassName
{
    field1
    field2
    //...

    constructor1
    constructor2
    //...
    method1
    method2
    //...
}
```

Define simplified version of an Employee class

```
class Employee
{
    // instance fields
    private String name;
    Private double salary;
    private LocalDate hireDay;
    // constructor
    public Emplyee(Sring n,double s,int year,int month,int day)
    {
        name = n;
        salart = s;
        hireDay = LocalDate.of(year,month,day);
    }
    // a method
    public String getName()
    {
        return name;
    }

    // more methods
}
```

> 每一个源文件都会被编译成一个.class文件
>
> 源文件中只有一个public类 并且 该类中含有一个main方法（程序执行入口点）

4.3.2 Use of Multiple Source Files

> place the Employee class into a file Employee.java
>
> place the EmployeeTes class into a file EmployeeTest.java

- invoke the Java compiler with a wildcard:
    - `javac Employee*.java`

> 通配字符的javac，编译所有匹配的文件

- compile the public class
  - `javac EmployeeTest.java`

  > 这样的话，编译器在EmployeeTest中发现EmployeeTest调用了Employee就会去寻找是否存在Employee.class，如果不存在的话就去寻找Employee.java并且会自动编译该文件生成Employee.class

4.3.3 Dissecting the Employee Class

> 刨析 Employee 这个类

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

  > All methods of this class are tagged as **public**.

public method

- *define:* **any method in any class can call the method**

private instance field

- *define:* **only method of the class** that can **access** these instance fields

4.3.4 First Steps with Constructors

> You can't apply a constructor to an existing object to reset the instance fields

- A constructor has the same name as class
- A class can have more than one constructor
- A constructor can take zero，one，or more parameters
- A constructor has no return value
- A constructor is always called with the new operator

  > 构造函数总是和new操作结合一起，比如new Date();

Wraning

- all Java objects are constructed on the **heap**
- constructor must be combined with **new**

> 还有，不要引入跟实例字段同名的local variables

4.3.5 Implicit and Explicit Parameter

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

- *inplicit parameter:*
  - 该方法隐藏了一个形参，类型为：the type of object which call the method，变量名为this关键字
  - 显示的形参是double byPercent

> 1、C++的成员函数可以在类外定义，Java的方法只能在类内定义
>
> 2、C++可以指定inline函数，Java的方法内联需要依赖于JVM的优化

4.3.6 Benefits of Encapsulation

field accessors

- getName method
- getSalary method
- ...

get and set the value of an instance field

- A private data field
- A public field accessor method
- A public field mutator method

accessor methods should not return reference to mutable objects

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

> 先拷贝，再返回拷贝对象存储的位置 （object.clone()做到）

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

4.3.7 Class-Based Access Privileges

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

> 可以直接使用同类型的对象的私有字段

4.3.8 Private Methods

> 当你希望将部分代码作为单独的助手方法时（help methods）
>
> 这些help methods 被 method使用，不想暴露给用户使用
>
> 例如单独的计算

4.3.9 Final Instance Fields

- *final fields:*
    - *wraning:* must be initialized when the object is constructed
    - *wraning:*

    ```
    private final StringBuilder evalutions;
    //
    evaluations = new StringBulider();
    //
    public void giveGoldStar()
    {
        evaluations.append(LocalDate.now() + ":Gold star!\n"); // ok
    }
    ```

    > 虽然evalutions是final实例字段，但是我们可以修改该字段引用的对象

> The final keyword merely means that the object reference stored in the evaluations variable will never again refer to different StringBuilder object.But the object can be mutated

4.4 Static Fields and Methods

4.4.1 Static Fields

- *wraning:* There is only one such field per class，is shared amony all instance of the class

  > 是类所拥有的，被类的所有实例对象所共享，不属于实例对象

4.4.2 Static Constants

- such as

```
public class Math
{
    //...
    public static final double PI = 3.14159265358979323846;
    //...
}
```

```
public class System
{
    //...
    public static final PrintStream out = ...;
    //...
}
```

> 注意：setOut可以修改System.out的值，这种操作不是java语言实现的（绕开访问机制），而是native method，这种操作我们不能模仿。

4.4.3 Static Methods

- *wraning:* Static methods are methods that do not operate on objects

  > 不需要对象才能使用，不能直接访问对象非静态字段，这意味没有implicit parameter（this）

- *wraning:* Static method can access a static field

  ```
  public static int getNextId()
  {
      return nextId; // nextId is a static field.
  }

  int n = Employee.getNextId();
  ```

  > 对象调用静态方法是合法操作。但是我们应该避免这种操作

Use static method in two situations

- method does not need to access the object state (状态是对象拥有的field)

> 当方法不需要访问对象状态时，需要的参数由显式形参来体现，可为static method

- method only needs to access static fields of the class

> 当方法只需要访问类的静态字段时

4.4.4 Factory Method

static factory method

- *define:* use static methods to construct objects
- *such as:* `LocalDate.now()、LocalDate.of()`

> 还有 NumberFormat等

```java
NumberFormat currencyFormatter = NumberForamt.getCurrencyInstance();
NumberFormat percentForamtter = NumberFormat.getPerentInstance();
double x = 0.1;
System.out.println(currencyForamtter.format(x)); // $0.10
System.out.println(percentFormatter.format(x));  // 10%
```

Why does not the NumberFormat class use a constructor instead?

- You can not give name to constructors

> 构造函数名需要和类同名，但是我们想要两种名字
>
> NumberFormat.getCurrencyInstance();
>
> NumberFormat.getPercentInstance();

- vary the type of the constructed object

> 使用构造函数你是不能改变构造的对象的类型的，然而实际上，针对DecimalForamt，它的factory method 返回了继承于DecimalFormat的子类对象。（后面章节介绍）

4.4.5 The main Method

> 好了，现在我们知道了，static的方法是不需要实例对象来调用的，因此main是static的。

- *wraning:* Every class can have a main method

> 每个类都可以有main方法，这是进行单元测试的好方法(unit test)，即直接从该类的main运行 （类似Employee单元：java Employee）
>
> 如果该类是较大应用程序的一部分，那么你启动那个较大的应用程序，该类的main不会被执行，放心吧。(类似Application项目 :java Application)

4.5 Method Parameters

- *call by value:* method **gets the value** that hte caller provides

- *call by reference:* method **gets the location** of the variable that the caller provides
- ***Wraning***: The Java programming language **always** uses **call by value**

two kinds of method parameters

- Primitive types(numbers,boolean values)

  > 基本类型传递时拷贝值，也就是拷贝值

- Object references

  > 类类型在传递时拷贝值，也就是引用拷贝，java的引用类似C++的指针，“指针模型”

What you can and cannot do with method parameters

- A method cannot modify a parameter of a primitive type

  > 一个方法不能够通过修改基本类型参数对外部造成影响

- A method can change the state of an object parameter

  > 一个方法不能修改外部对象引用，但是可以修改对象本身，即修改对象的状态是可行的

- A method cannot make an object parameter refer to a new object

  > 简单说就是引用是拷贝的，你无法修改外部引用

4.6 Object Construction

> Java offers a variety of mechanis for writing constructors

4.6.1 Overloading

- **empty StringBuilder object**

```
StringBuilder message = new StringBuilder();
```

- **specify an initial string**

```
StringBuilder todoList = new StringBuilder("To do:\n");
```

> This capability is called overloading.

- *define:* the same name but different parameters

> 函数调用时在同名函数中查找匹配函数的过程称之为重载解析

> To completely describe a method,you need to specify its name together with its parameter types.

- *signature of method*: 方法签名
  - *such as:*

```
indexOf(int)
indexOf(int,int)
indexOf(String)
indexOf(String,int)
```

  - *wraning:* The return type is not part of the method signature

> 函数重载：各种函数签名

4.6.2 Default Field Initialization

> If you don not set a field explicitly in a constructor,it is automatically set to a default value.

- numbers: 0
- boolean: false
- object reference: null

- *wraning:* different between fields and local variables

4.6.3 The Constructor with No Arguments

- *wraning:* If you write a class with no constructors whatsoever,then a no-argument constructor is privided for you.

  > 自动给定的构造函数将会执行默认初始化，跟C++是一样的，如果你定义了构造函数，那么就不会再为你生成构造函数，因此注意无参数的构造函数使用。

4.6.4 Explicit Field Initialization --- 新操作

- **such as**

```
class Employee
{
    private String name = "";
    //...
}
```

- *wraning:* This assignment is carried out **before the constructor executes**
- *wraning:* field can be intialized with a method call

```
class Employee
{
    private static int nextId;
    private int id = assignId();
    //...
    private static int assignId()
    {
        int r = nextId;
        nextId++;
        return r;
    }
    //...
}
```

4.6.5 Parameter Names

> 关于构造函数的参数名字规则

> 单字母

```
public Employee(String n,double s)
{
    name = n;
    salary = s;
    //...
}
```

> prefix

```
public Employee(String aName,double sSalary)
{
    name = aName;
    salary = aSalary;
}
```

> the same name

```
public Employee(String name,double salary)
{
    this.name = name;
    this.salary = salary;
}
```

4.6.6 Calling Another Constructor --- 新操作

> this 对象的另外一种含义

```
public Employee(double s)
{
    this("Employee #" + nextId, s); // call Employee(String,double)
    nextId++;
}
```

4.6.7 Initialization Blocks --- 新操作

- *two ways to intialize a date field:*
    - By setting a value in a constructor
    - By assigning a value in the declaration （在构造函数执行之前）
    - Intialization block （在构造函数执行之前）

    ```
    class Employee
    {
        private static int nextId;

        private int id;
        private String name;
        private double salary;

        // object intialization block
        {
            id = nextId;
            nextId++;
        }

    }
    ```

    > The initialization block run first，and then the body of the constructor is executed
    >
    > This mechanism is never necessary and is not common
    >
    > 这种机制是没有必要的，也不常见
    >
    > 如果想要使用这种机制的话，应该总是在字段定义之后放置初始化块。

- *when a constructor is called:*
    1. All data fields are intializing to their default values
    2. All field intializers and intialization blocks are executed
    3. If the first line of the constructor calls a second constructor，then the body of the second constructor is executed
    4. The body of the constructor is executed
- *initializa a static field:*
    - supply an intial value
        - `private static int nextId = 1;`
    - use a static initialization block

```java
static
{
    Random generator = new Random();
    nextId = generator.nextInt(10000); //  0 - 9999
}
```

> 当类首次被加载时，将会执行静态初始化，按照类中声明顺序执行

Wraning

> 初始化顺序优先级：

> 默认初始化进行 （defautl initialization）
>
> 类内赋值初始化（assignment a value in the declatation）

> 静态初始化块（static intialization block）

> 对象初始化块 （object initialization block）

> 构造函数执行 （constructor）

4.6.8 Object Destruction and the finalize Method

> Java does not support destructor

> Java does automatic garbage collection

- *wraning*: 有的对象如果使用了文件或者系统资源等等，则在不需要的情况下需要回收资源

finalize method

- *wraning*: The **finalize** method will be called **before** the garbage collector sweeps away the object.

  > finalize方法在gc之前被调用
  >
  > unsafe

> 后面的章节（7.2.5）将会介绍 使用 close 方法来处理资源释放问题

4.7 Packages

> Java allows you to group classes in a collection called a package

- *standare Java library*:
  - a number of packages
  - *such as:*
    - `java.lang`
    - `java.util`
    - `java.net`

    - All standare Java packages are inside the `java` and `javax` package hierarchies

> 从编译器的角度来看，嵌套的包之间没有任何关系
>
> 例如：java.util和java.util.jar没有任何关系，它们是独立的集合

4.7.1 Class Importation

> A class can use all classes from its own package and all public classes from other packages

- *access public classes in two ways:*
    - *full package:* `java.time.LocalDate today = java.time.LocalDate.now();`
    - *use import:* import语句需要放在package语句之后和源程序之前
        - *define:* give you a shorthand to refer to the classes in the package
        - you can import a specific class or the whole package

          `import java.util.Date;`

          `import java.util.*;`
        - you cannot use `import java.*;` or `import java.*.*;`

wraning

`import java.util.*;`

`import java.sql.*;`

> 这样的话使用Date会编译失败，它们都有Date类

如果你需要其中一个：屏蔽另外一个

```
import java.util.*;
import java.sql.*;
import java.util.Date;
```

如果你需要两个：给出完整的包名

```
java.util.Date dealLine = new java.util.Date();
java.sql.Date today = new java.sql.Date(/*..省略..*/);
```

> import是给编译器来处理引入名字的，编译后的字节码总是使用full package names去引用类
>
> 类比C++中的namespace和package，using 和import

4.7.2 Static Imports

- *define:* import static methods and fields
- *such as:* `import static java.lang.System.*;`

  > 这样去做的话，你可以直接用out，而不是System.out这样一串

4.7.3 Addition of a Class into a Pakage --- 新操作

> 打包语句放最顶部

```
package com.horstmann.corejava; //  (com\horstmann\corejava on windows)

public class Employee
{
    //...
}
```

> 如果没有指定package语句的话，将使用default package（has no package name）
>
> 注意：class 文件也被放入包目录中

一旦类中进行了import编译器会自动从相应包目录中寻找所需源文件并编译

> 注意区别

```
javac com/mycompany/PayrollApp.java
java com.mycompany.PayrollApp
```

4.7.4 Package Scope

> 没有指定private和public，则默认可被同包中的其他类访问（public in package）

4.8 The Class Path

> 通往类的路径必须和包名匹配
>
> 类文件也可以从存储在jar文件中
>
> jar文件压缩包含了多个类文件和子目录
>
> 第九章将会让我们了解到如何创建jar文件
>
> jar文件使用ZIP格式来组织

share class 共享类（使用文件下的类和jar）

1. Place your class files inside a directory,for example, ***/home/user/classdir***.

   > 如果你添加了类 ***com.horstmann.corejava.Employee***,Employee.class 必须放在
   >
   > ***/home/user/classdir/com/horstmann/corejava***

2. Place any JAR files inside a directory,for example, ***/home/user/archives***

   > 将jar文件放在文件目录中

3. Set the class path.

> c:\classdir; . ;c:\archives\archieve.jar
>
> . 是代表当前目录
>
> ; 是隔开操作
>
> 因此，classdir中的、和当前的、和archieves下的archieve.jar都被放入类路径。
>
> c:\classdir; . ;c:\archives*
>
> *是使用目录中下所有jar

- *wraning:* 运行时库是被默认放入类路径的

- *wraning:*

> javac 是会一直在当前目录下寻找的
>
> java 需要指定 " **.** "操作才会在当前目录下寻找

- *wraning:*

```
//如果只有这样的两个导入
import java.util.*;
import com.horstmann.corejava.*;
```

> 一旦我们使用某个类
>
> 编译器寻找类从顺序：
>
> java.lang (第一：因为这是被默认import的)
>
> java.util (按顺序来)
>
> com.horstmann.corejava
>
> 重复则报错

4.8.1 Setting the Class Path

> javac -d . name.java
>
> java -classpath class包路径;.;jar路径 java程序

关于package和classpath的使用例子总结

- 编译包
  - 例如：在当前目录下 有com文件，com文件下有mypackage文件，文件下有MyClass类
  - 即有类 com.mypackage.MyClass

    - 注意：我们需要在含包的目录下进行编译 (当前目录)
    - `javac com/mypackage/MyClass.java`

- 创建package
    - 假如我们有java源文件：MyClass.java,而且源码中指出了 `package com;`
    - 那么我们在该源文件目录下：`javac -d . MyClass.java`
    - 生成了包
- 运行当前目录下的package
    - java运行包，例如有包：com.MyClass
    - `java  com.MyClass`
- 使用当前目录下的package
    - 只有你在源文件中有import，编译器会自动在当前目录中找包（先找class，找不到找java源码编译）
- 使用其他目录下的包源码进行编译
    - 源程序MyClass使用其他目录下的包
    - 使用 -sourcepath
    - 例如，你想使用 C:\classdir下的包
    - `javac -sourcepath C:\classdir;.  MyClass.java`
- 使用其他目录下的包进行运行
    - 源程序包com.MyClass使用其他目录下的包
    - 使用 -classpath
    - 例如，你想使用 C:\classdir下的包
    - `java -classpath C:\classdir;.  com.MyClass`

4.9 Documentation Comments

> The JDK contains a very useful tool, called **javadoc**,that generated HTML documentation from your source files.

4.9.1 Comment Insertion

extract item

- package
- public classes and interface
- public and protected fields
- public and protected constructors and methods

- *wraning:* above the feature it describes
- *use:*

```
/**
*   free-form text
*   tags
*/
```

- *tags:* starts with **@**
    - *such as:*
        - @author
        - @param
- *wraning:* free-form text can use HTML modifiers
    - *such as:* <**em**> ... </**em**>

> 不要使用 <**h1**> or <**hr**>
>
> 使用{@code...}来标记代码
>
> 如果你使用了其他文件的链接或者图像，则必须把这些文件放入包含源文件的目录的子目录中，文件夹命名为 **doc-files**
>
> 例如：<**img src = "doc-files/uml.png" alt="UML diagram"**/>

4.9.2 Class Comments

```
/**
    A {@code Card} object represents a playing card,such
    as "Queen of Hearts".A card has a suit (Diamond,Heart,
    Spade or Clud) and a value (1 = Ace,2...10,11 = Jack,
    12 = Queen,13 = King)
*/
public class Card
{
    //...
}
```

4.9.3 Method Comments

- @param : 参数描述，可以使用HTML 标记
- @return : 返回值描述，可以使用HTML标记
- @throws : 异常描述

```
/**
*   Raises the salary of an employee.
*   @param byPercent the percentage by which to raise the salary
*   @return the amount of the raise
*/
public double rariseSalary(double byPercent)
{
    //...
}
```

4.9.4 Field Comments

- public field
  - generally means static constants

```
/**
*   The "Hearts" card suit
*/
public static final int HEARTS = 1;
```

4.9.5 General Comments

Be used in class

- @author : 你可以使用多个，标记多个作者
- @version : 版本描述

Be used in all documentation

- @since : 介绍了该特性的版本号，例如@since version 1.7.1
- @deprecated : 说明 class、method、variable等不应该被使用的注释
- @see ： 超链接
  - `package.class#feature label`
    - *such as:* `@see com.horstmann.corejava.Employee#raiseSalary(double)`

      > 链接到包的方法
  - `<a href="...">label</a>`
    - *such as:* `@see <a href = "www.horstmann.com/corejava.html">The Core Java home page</a>`
  - `"text"`
    - 会提示”see also“选项，提示内容是”text“
- 你可以在任何注释放入其他类或者方法的超链接
  - `{@link package.class#feature label}`

4.9.6 Package and Overview Comments

- Supply an HTML file named **package.html**

  > 所有的body标签中的内容会被提取
- Supply a Java file named **package-info.java**

  > 只能包含 Java专业的注释 `/** */`

4.9.7 Comment Extraction

1. 在包的目录下
2. 运行控制台

   > 几种方法

  - `javadoc -d docDirecory nameOfPackage`
  - `javadoc -d docDirectory nameOfPackage1 nameOfPackage2 ...`
  - `javadoc -d docDirectory *.java`

    > 当前目录（默认包）

  - `javadoc -link http://docs.oracle.com/javase/8/docs/api *.java`

4.10 Class Design Hints

- Always keep data private
- Always intialize data
- Don't use too many basic types in a class
- Not all fields need individual field accessors and mutators

  > 不是所有字段都需要独立的accessor 和 mutator

- Break up class that have too many responsibilities

  > 抓住机会，分解那些可以分解的类，使得概念上更加简单,需要经验来控制尺度

- Make the names of your classes and methods reflect their responsibilities

  > 让类和方法的名字能够反映它们的职责

- Prefer immutable classes

  > 如果你可以创建保持不变的类，那么是个更好的操作，处在于多线程中将更加强壮