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

Fundemental Programming Structures in Java

- 3.1 A simple Java Program
- 3.2 Comments
- 3.3 Data Types
  - 3.3.1 Integer Types
  - 3.3.2 Floating-Point Types
  - 3.3.3 The Char Type
  - 3.3.4 Unicode and the Char Type
  - 3.3.5 The Boolean Type
- 3.4 Variables
  - 3.4.1 Initializing Variables
  - 3.4.1 Constants - final
- 3.5 Operators
  - 3.5.1 Mathematical Functions and Contants
  - 3.5.2 Conversions between Numeric Types
  - 3.5.3 Casts
  - 3.5.4 Combining Assignment with Operators
  - 3.5.5 Increment and Decrement Operators
  - 3.5.6 Relational and boolean Operators
  - 3.5.7 Bitwise Operators
  - 3.5.8 Parenthess and Operator Hierarchy
  - 3.5.9 Enumerated Types
- 3.6 Strings - 貌似具有指针模型
  - 3.6.1 Substrings
  - 3.6.2 Concatenation
  - 3.6.3 Strings Are Immutable
  - 3.6.4 Testing String for Equality
  - 3.6.5 Empty and Null Strings
  - 3.6.6 Code Points and Code Units
  - 3.6.7 Thr String API
  - 3.6.8 Reading the Online API Documentation
  - 3.6.9 Building Strings
- 3.7 Input and Output
  - 3.7.1 Reading Input
  - 3.7.2 Formating Output
  - 3.7.3 File Input and Output - 终端是文件
- 3.8 Control Flow
  - 3.8.1 Block Scope
  - 3.8.2 Conditional Statements
  - 3.8.3 Loops
  - 3.8.4 Determinate Loops
  - 3.8.5 Multiple Selections - The switch Statement
  - 3.8.6 Statements That Break Control Flow
- 3.9 Big Numbers
- 3.10 Arrays
  - 3.10.1 The "for each" Loop
  - 3.10.2 Array Initializers and Anonymous Arrays
  - 3.10.3 Arrays Copying

3.10.4 Command-Line Parameters
3.10.5 Array Sorting
3.10.6 Multidimensional Arrays
3.10.7 Ragged Arrays

Fundemental Programming Structures in Java

> Java中的基本编程结构

In this chapter

- **3.1 A Simple Java Program** (*一个简单的Java程序*)
- **3.2 Comments** (*三种注释*)
- **3.3 Data Types** (*数据类型*)
- **3.4 Variables** (*变量*)
- **3.5 Operators** (*操作符*)
- **3.6 String** (*字符串*)
- **3.7 Input and Output** (*输入输出*)
- **3.8 Control Flow** (*控制流*)
- **3.9 Big Numbers** (*大数运算*)
- **3.10 Arrays** (*数组*)

3.1 A simple Java Program

- *code-look*

```
public class FirstSample
{
    public static void main(String[] args)
    {
        System.out.println("We will no use 'Hello World!'");
    }
}
```

- *wraning:* **Java is case sensitive**
- ***public*** *keyword:* **access modifier**
  - *purpose:* **control the level of access** other parts of a program have to this code

    > 控制程序的其他部分对代码的访问权限（级别）

The rules for class names

- *begin:* **letter**(字母)

Standard naming convention

- *class name:* **words begin with a uppercase letter**

- *such as:* ***FirstSample***

public class and file

- *wraning:* make the **file name** for the source code the same as the name of the **public class**
  - *such as:*
    - *file name:* **FirstSample.java**
    - *public class name:* **FirstSample**

main methord must be declared public

methods,not member function

- *in C/C++ :* we call **member functions**
- *in Java:* we call **methods** (*标准术语*)

main method does not return an "exit code" to the operating system

Java Statement

- *end with:* `;`
- *wraning:* **can multiple lines**

> 也就是说回车符号并不能代表语句的结束，可以多行

Java general syntax

- *define:* ***object.method(parameter)***
- example

```
{
    System.out.println("we will not use 'Hello,World!'");
}
// object:    System.out
// method:    println
// argument:  "we will not use 'Hello,World!'"
```

3.2 Comments

- *three way:*
  - `// comments`
  - `/* comments*/`
  - *documention automatically:*

```
/**
*
*
*/
```

3.3 Data Types

> Java is a strongly typed language

- **eight** *primitive types(八种基本类型)*
  - **four** of them are integer types
  - **two** floating-point number types
  - **one** character type
  - **one** boolean type

3.3.1 Integer Types

Type	Storage Requirement	Range
int	**4** bytes	just about 2 billion
short	**2** bytes	
long	**8** bytes	
byte	**1** byte	

- prefix:
  - `0x` or `0X`
  - `0`
  - *binary number:* prefix `0b` or `0B` (from **Java SE 7**)
- suffix:
  - `l` or `L`
- *wraning:* **no unsgined type**

3.3.2 Floating-Point Types

Type	Stoage Requirement	Range
float	**4** bytes	6-7位有效数字
double	**8** bytes	15位有效数字

- suffix:

    - `f` or `F`
    - `d` or `D`

Three constant floating-point values

- **Double.POSITIVE_INFINITY**（正无穷）
- **Double.GEGATIVE_INFINITY**（负无穷）
- **Double.NaN** (不存在)

```
if(x == Double.NaN) // error
if(Double.isNaN(x)) // ok
```

roundoff error

```
System.out.prinln(2.0 - 1.1); // 0.899999...
```

- use ***BigDecimal***

3.3.3 The Char Type

- *range:* **\u0000 ~ \uFFFF**

3.3.4 Unicode and the Char Type

UTF-16

- *define:* **two code units**
    - *first:* **U+D800 ~ U+DBFF**
    - *second:* **U+DC00 ~ U+DFFF**
- *wraning:* In Java,the char type describes **a code unit** in the UTF-16 encoding (**two bytes**)

    > 我们应该尽量避免使用char

3.3.5 The Boolean Type

- *two value:* **false** and **true**
- *wraning:* **cannot convert** between integers and boolean values

3.4 Variables

- *declaration:* ***type variable-name***
    - *such as:* `double salary;`
- *variable name:*
    - *begin with:* **letter** (Unicode just ok)
- *check Unicode char methods:* **isJavaIdentifierStart** and **isJavaIdenfierPart**

3.4.1 Initializing Variables

- *wraning:* You **must explicitly initialize** it by means of an assignment statement **before use it.**

  > 定义时最好初始化，使用时定义

- *wraning:* In Java,**no declarations are separate from definitions**

  > 声明和定义没有分开，不像C/C++

3.4.1 Constants - final

- *final* : denote a constant
  - *such as:* `final double CM_PER_INCH = 2.54;`
  - example

```
public class Constant
{
    public static final double CM_PER_INCH = 2.54;
    public static void main(String[] args)
    {
        //...
    }
}
// public的存在使得其他类也可以使用，通过:Constant.CM_PER_INCH
```

- *const:* **reserved Java keyword** (保留关键字)

3.5 Operators

- *forms:*
  - +、-、*、/、%
  - `15/2=7`、`15.0/2=7.5`、`15%2=1`
  - `10/0  // Exception`、`10.0/0 //INFINITY or NAN`
- *strictfp keyword:* strict computations(执行严格的浮点计算)

  > 严格的浮点计算和默认浮点计算的差别在于严格的浮点计算在当无法进行默认浮点计算时会导致溢出
  >
  > 方法指定了strictfp则方法体的所有浮点操作均为严格的浮点计算
  >
  > 类指定了strictfp则类的所有方法中浮点操作均为严格的浮点计算

3.5.1 Mathematical Functions and Contants

- **Math class**
  - static methods
    - *such as:* **sqrt(double)**
      - *use:* Math.sqrt

        > 由于是静态方法，于是我们可以采用 Class.method 的方式来使用

      - *code-look:* `double x = Math.sqrt(10.0);`
    - *such as:* **pow(double x,double y)** (=> x^y)

        - *such as:* **floorMod**
            - `-3 % 2 = -1`
            - `floorMod(-3, 2) = 1`

                > fllorMod总是将数字限制到第二个实参范围内，注意！第二个参数为负则结果为负
        - *such as:* **sin、cos、tan、atan、atan2、exp、log、log10**
    - *constant value:* **P、E**
- *wraning:* **StrictMath关注结果在各个平台的唯一性，但是性能不足于Math**

satic import使用之一

- *using name:* `import static java.lang.Math.*`

    > 类似于C++中的using namespace std效果，只是类似！

3.5.2 Conversions between Numeric Types

```
// 精度保留
char-------->int
byte-------->short--------->int-------->long
int--------->long
int--------->double
float------->double
// 精度可能丢失
int--------->float
long-------->double
long-------->float
```

3.5.3 Casts

- *use:* **(Type)**
    - *such as:*

        ```
        double x = 9.997;
        int nx = (int)x;
        int nx2 = (int)Math.round(x); // 获取离x最近的整数
        ```
- *wraning:* **(byte)400 将得到 44**，原因是截断了数据

3.5.4 Combining Assignment with Operators

- *such as:* **+=、-=、/=、%=**
- *code-look:*

    ```
    int x = 0;
    x += 3.5;   // x = (int)(x + 3.5)

    x += 4;     // x = x + 4
    ```

3.5.5 Increment and Decrement Operators

```
int m = 7;
int n = 7;
int a = 2 * ++m;  // now : a is 16,m is 8
int b = 2 * n++;  // now : b is 14,n is 8
// the same to C/C++
```

3.5.6 Relational and boolean Operators

- *such as:* **==、!=、&&、||、！、?：**
  - the same to C/C++

3.5.7 Bitwise Operators

- *such as:* **&、|、^、~、>>、<<**
  - the same to C/C++
  - `int fourthBitFromRight = (n & 0B1000)/0B1000;`
- *such as:* **>>>**
  - *wraning:* Java没有unsigned类型，因此在右移的时候有两种可选操作，第一种就是>>,这意味着最高的符号位不动，还有一种是>>>，即符号位清空的移动,全部补0

3.5.8 Parenthess and Operator Hierarchy

Operators	Associativity
[]、 .、 ()	--->(Left to right)
!、~、++、--、+（一元）、-（一元）、() Cast、**new**	<---
***、/、%**	--->
+、-	--->
<<、>>、>>>	--->
<、<=、>、>=、instanceof	--->
==、!=	--->
&	--->
^	--->
|	--->
&&	--->
||	--->
?:	<---
=、+=、-=、*=、/=、%=、&=、|=、^=、<<=、>>=、>>>=	<---

- *wraning:* Java 没有**逗号运算符**，但是可以进行如下操作
  - `int a = 0,b = 0;`

3.5.9 Enumerated Types

- *define way:*
  - `enum Name{A,B,C};`
- *use:*
  - `Name me = Name.A;`

3.6 Strings - 貌似具有指针模型

3.6.1 Substrings

```
String greeting = "Hello";
String s = greeting.substring(0,3);  // s is "Hel"
```

3.6.2 Concatenation

```
String s1 = "Exp";
String s2 = "Wit";
String s3 = s1 + s2;
int i = 10;
String s4 = s1 + i;    // "Exp10"
```

- *wraning:* **every object can be converted to a string**

join method

```
String all = String.join("/","S","M");
// multiple arguments,multiple string together
// all is "S/M"
```

- *parameter:*
  - *first:* **separater(分隔符)**
  - *else:* **strings**

3.6.3 Strings Are Immutable

- *Wraning:* We **cannot change a character** in an existing string.

> 我们不能拆分String对象成单个char来考虑，只能拆分成多个String对象来考虑
>
> 因此，如果我们想把greeting的值("Hello")变成("Help")的话，我们可以这么来做:
>
> greeting = greeting.substring(0,3) + "p";

- *wraning:* Java实现了自动垃圾回收，如果一个内存块不再需要，那么它最终会被回收。

3.6.4 Testing String for Equality

- **equals** method

```
s.equals(t);
"Hello".equals(greeting);  // ok
```

- **equalsIgnoreCase** method

```
"Hello".equalsIgnoreCase("hello");
```

- **== operator**
  - 对于两个String对象而言，==操作比较的是两个对象所处的位置差异（类似指针模型）
  - 因此我们不能用该操作来比较字符串是否是相等的
- **compareTo** method
  - 这是对C语言的strcmp最解决的模拟

```
if(greeting.compareTo("Hello") == 0) // 判定相等
```

3.6.5 Empty and Null Strings

- **length** method
  - *return the length of the string*
- *Empty string:* **""**
  - *length:* **0**
- **null** string:**null**
  - null 说明了该String对象没有指向任何位置，而empty string有指向的空间，只是空间没有值，null似乎是一种指针模型下空指针的值。
  - *use:*

```
if(str != null && str.length() !=0 )
{
    //...
}
```

3.6.6 Code Points and Code Units

> 代码点（字符） 和 代码单元（类似 **\u2122** 双字节char值）

- *wraning:*
  - （sequences of **char** values）Java字符串被实现为char值序列
  - （**char** data type is a code units) char数据类型是一种代码单元，用于在UTF-16编码中对**Unicode代码点**进行重新编码。
  - （代码单元----对应----代码点）
    - 常用的**Unicode code point**可以被一个code unit标识
    - 其他的**Unicode code point**需要被两个code unit标识（**requiring two code units in the UTF-16 encoding**）
- *wraning:* **length** method get the number of code units （char值数量）

> length 方法是获取代码单元(code units)的数量

- **codePointCount** method （真正的字符数量）
  - get the number of code points (这个才是获取代码点数量的方法)

> codePointCount 可以获取代码点数量，也就是真正的字符串长度，适应于所有的Unicode字符组成的字符串

- **charAt** method
  - get the code point at position
  - *code-look:*

```
char first = greeting.charAt(0);
char last  = greeting.charAt(greeting.length() - 1);
```

> charAt是以char值大小为基本单位的，因而范围是 **0 ~ length() - 1**

- 如何以 CodePoints为单位偏移?
  1. 通过代码点来偏移：`int index =  offsetByCodePoints(0,i);`
  2. 获取： `int cp = codePointAt(index);`
- 如何以 固定单位 进行偏移?
  - 首先，我们可以对齐，然后进行 "流"化
  - 具体实现:

```
int[] codePoints = str.codePoints().toArray();
```

  - 如何由数组再由String对象管理?

```
String str = new String(codePoints,0,codePoints.length()); // construcor
```

  - *wraning:*
    - **codePoints** method
      - yields a "stream" of **int** values

3.6.7 Thr String API

- *url:* https://docs.oracle.com/javase/9/docs/api/java/lang/String.html

3.6.8 Reading the Online API Documentation

3.6.9 Building Strings

> 有时候你需要进行字符串的构建，比如你从键盘输入或文件中的单词。使用字符串连接是非常没有效率的，因为每次连接都构造一个新的String对象

Using the StringBuilder class

1. construct an empty string builder（构造一个空的字符串构建器）

   `StringBuilder builder = new StringBuilder();`

2. add another part,call the **append** method (进行附加)

```
builder.append(ch);  // append a singel character
builder.append(str); // append a string
```

3. building the string,call the **toString** method

   `String completedString = builder.toString();`

3.7 Input and Output

> 本章探讨基于控制台应用程序的输入和格式化输出

3.7.1 Reading Input

- System.out
  - "standard output stream" (标准输出流：attached to 控制台窗口)
- System.in
  - "standard input stream"(标准输入流)

the order of reading console Input

1. construct a **Scanner** that is attached to **System.in**

   `Scanner in = new Scanner(System.in);`

2. use various methords of the **Scanner** class to read input
   - example

```
System.out.println("What is your name?");
String name = in.nextLine();                // reads a line of input
```

> program----------->System.out=>Console Window
>
> **program <=System.in**----------<-----Scanner object <-----Some object(console or file)
>
> 就目前情况而言，我们想要进行输入，需要通过Scanner的对象去关联，比如我们可以关联控制台窗口的输入。

- some **Scanner** method
  - **nextLine** : reads a line （读取一行,包括空格）
  - **next** : read a single word （读取单词）
  - **nextInt** : read a interger （读取int类型整数）
  - **nextDouble**: read a double (读取浮点数)
- **important wraning**:
  - **Scanner** class is defined in the **java.util** *package*
    - we should do that: `import java.util.*;`

Read a password

```
Console cons = System.console();
String usename = cons.readLine("User name:");
char[] passwd = cons.readPassword("Password:");
```

3.7.2 Formating Output

- **printf** method

```
double x = 10000.0/3.0;
System.out.printf("%8.2f",x);
// % : 为x占位
// 8 : 指出width: 8 characters ;  指出 右对齐
// .2 : 指出precision: 2 characters
```

- Conversion Character
  - **d、x、o、f**：十进制、十六进制、八进制、定点浮点
  - **e**：指数浮点
  - **g**：一般浮点（取 e和f的 表示占字符较短者）
  - **a**：十六进制浮点
  - **s**：String
  - **c**：char
  - **b**：boolean
  - **h**：散列码 （Hash code）
  - **tx or Tx**： Date and time (最好用**java.time** classes来代替)
  - **%、n**：百分号、平台相关分隔符
  - In addition,**flag** that control the appearence of the formatted output

```
System.out.printf("%,.2f",10000.0/3.0);
// 输出：3, 333.33
// 有 , 分隔符
```

Flags for **printf**

Flag	Purpose	Example
+	sign for positive and negative numbers	+3333.33
space	add a space before the positive numbers	| 3333.33|
0	add leading zeroes	-03333.33
-	left-justifies fiedld(左对齐)	|3333.33 |
(	enclose negative number	(3333.33)
,	add group sepatators	3,333.33
#	用于 f 或者 F时：includes decimal point	3333.
#	用于 x 或者 X时：add 0x or 0X	0xcafe
$	`%1$d %1$x`:1指出第一个参数,输出两个形式(该标志使得我们可以使用索引参数)	`159 9F`
<	`%d %<x`:the same to `%1$d %1$x`	`159 9F`

- **String.format** method

```
String message = String.format(“Hello,%s.Next year you'll be %d",name,age);
```

  - *define:* **Create** a formatted strin without printing it

- Conversion Character about (**tx or Tx**)

x	Type	Example
c	Complete data and time	Mon Feb 09 18:05:19 PST 2015
F	ISO 8601 data	2015-02-09
D	U.S formatted date(m/d/y)	02/09/2015
T	24-hour time	18:05:19
r	12-hour time	06:05:19 pm
R	24-hour time,no seconds	18:05
Y	Four-digit year(不足补0)	2015
y	Last two digits of the year(取年份的最后两位数字，不足补0)	15
C	First two digits of the year(取年份的前面两位数字，不足补0)	20
B	Full month name	February
b or h	Abbreviated month	Feb
m	Two-digit month(月份两位数字，不足补0)	
d	Two-digit day(日期两位，不足补0)	09
e	Two-digit day(日期两位，不补0)	9
A	Full weekday name(星期)	Monday
a	Abbreviated weekday name	Mon
j	Three-digit dat of year(年过去的天数，最大366,不足补0)	069
H	Two-digit hour(两位数小时，不足补0，最大23)	18
k	Two-digit hour(两位数小时，不补0，最大23)	18
`I`	Two-digit hour(两位数字小时，不足补0，最大12，最小01)	
J	Two-digit hour(两位数字，小时，不补0，最大12，最小1)	6
M	Two-digit minutes(两位数分钟，不足补0)	
S	Two-digit seconds(两位数秒数，不足补0)	

x	Type	Example
L	Three-digit milliseconds(三位数毫秒，不足补0)	
N	Nine-digit nanoseconds(九位数纳秒，不足补0)	
p	Morning or afternoon marker(上午或者下午标记)	pm
z	RFC 822 numeric offset from GMT	-0800
Z	Time zone(时区)	PST
s	Seconds since 1970-01-01 00:00:00 GMT	1078884319
Q	Milliseconds since 1970-01-01 00:00:00 GMT	1078884319047

- example

```
System.out.printf("%1$s %2$tB %2$te,%2$tY","Due date",new Date());
// print:Due date:February 9,2015
```

3.7.3 File Input and Output - 终端是文件

- **Input** （从文件输入）
  - use: **Scanner**
    - `Scanner in = new Scanner(Paths.get("myFile.txt"), "UTF-8");`

      > 如果路径含有反斜杆\,需要对其进行转义。即`\\`
      >
      > 例如：`"C:\\mydirectory\\myfile.txt"`
- **Output** （输出到文件）
  - use **PrintWriter**
    - `PrintWriter out = new PrintWriter("mFile.txt","UTF-8");`

      > 文件不存在则创建

IOException

```
public static void main(String[] args) throws IOExeception  //告诉编译器你知道可能会发生IOException
{
    Scanner in = new Scanner(Paths.get("myFile.txt"),"UTF-8");
    //...
}
```

3.8 Control Flow

> There is no goto.

3.8.1 Block Scope

> { /* Block Scope */}

- *wraning: 和C++不同的其中一点*

```
public static void main(String[] args)
{
    int n;
    {
        int k;
        int n;  //Error,cannot redefine n in inner block(无法在块内重定义块外变量)
    }
}
```

3.8.2 Conditional Statements

- *use:* **if** (*condition*) **statement**
- *use:* **if** (*condition*) **statement** **else** **statement**
- *use:* **if** (*condition*) **statement** **else if** (*condition*) **statement** ... **else** **statement**

> 可以多个else if
>
> 用法同C/C++

3.8.3 Loops

- *use:* **while** (*condition*) **statement**
- *use:* **do** **statement** **while** (*condition*) **;**

> 用法同C/C++

3.8.4 Determinate Loops

- *use:* **for** (**initialize** ; **test** *condition* ; **update**) **statement**

> 尽管用法和C++相同，但是三个部分的变量最好是相同的
>
> 例如:for(int i = 0;i < 10 ; ++i) { /*...*/}

3.8.5 Multiple Selections - The switch Statement

> 和 C/C++ 的 switch 语句一样

- *wraning:* 由于我们经常没有写break，我们可以告诉编译器来检查是否缺少break

```
java -Xlint:fallthrough Test.java
```

> 或者你可以标记方法，通过在方法附近用annotation注释 @SupportWraning("fallthrough")

- **case** lable

- byte、char、short、int
- Enum
- String (Java SE 7)

3.8.6 Statements That Break Control Flow

> goto为保留字

- *use:* **break ;**

  > 和 C/C++一样

- *use:* **break label ;**

  label :

  {

  {

  // ...

  if(condition) **break label;** // 跳转到 label 所在的块

  // ...

  }

  }

  // 跳转到这边继续执行

- *use:* **continue ;**

  > 和C/C++一样

- *use:* **continum label ;**

  > 回到循环开始

3.9 Big Numbers

> ***java.math*** package :**BigInteger** and **BigDecimal**

- **valueOf** method
  - *define:* turn ordinary number into a big number
  - *use :* `BigInteger a = BigInteger.valueOf(100);`
  - *wraning:* 现在，你无法使用类似 + or *的操作符了，而是需要相应的 method
    - **add** method: the same to +
    - **multiply** method: the same to *
    - **divide** method : the same to /
    - https://docs.oracle.com/javase/9/docs/api/java/math/BigInteger.html#BigInteger

    > 尽管String的 + 是重载的，但是java不允许我们程序员进行操作符重载，这和C++不同

3.10 Arrays

- *define:* a data structure that stores a collection of values of the same type.
- *access elem:* through integer **index** (0 ~ n-1)
  - *use:* **name[index]**
- *declare:* no elem
  - *use:* **Type[] name;**
- *create:* 分配元素
  - *such:* `int[] a = new int[100]; // 分配了100个int元素`

> 数字空间一旦创建完成，大小无法改动，改动得选择list

3.10.1 The "for each" Loop

- *use:* for **(variable : collection) statement**
  - such as

```
for (int elem : a)
    system.out.println(elem);
```

> 对elem进行修改不会影响原内容

3.10.2 Array Initializers and Anonymous Arrays

> 数字的初始化和创建匿名数组

- *initialize:* `int[] arr = {1,2,3,4,14,35}; // 不需要用new`
- *anonymous arrays:* `new int[] {12,53,76,12}; // 创建匿名数组对象`
  - *such as:*

```
int[] a = {1,2,3};   // a is a array:1,2,3
a = new int[] {4,5,6,7,8}; // a is new array:4,5,6,7,8
// 这一步等价于下面两步
// int[] anonymous = new int[] {4,5,6,7,8};
// a = anonymous;
```

- *wraning:* It is legal to have arrays of length 0.
  - *use:* **new elemType[0]**

> 长度为0的数组是合法的，并不和null一样

3.10.3 Arrays Copying

> int[] arrJ = new int[100]; // java
>
> int* arrC = new int[100]; // C++
>
> 两个是一样的
>
> arrJ也是指针的东西

- *wraning:* 既然int[]是指针，那么如果我们想真实的拷贝数组元素到新数组里面呢?

  > 不存在指针的dereference操作，我们需要调用Arrays的copyOf方法

  ```
  int[] copiedLuckyNumbers = Arrays.copyOf(luckyNumbers,luckyNumber.length);
  ```

  > 为什么需要长度参数? 这意味着我们可以扩展空间到新数组，相当于
  > 创建原空间+扩展空间==>进行内容拷贝===>返回位置到copiedLuckNumbers

3.10.4 Command-Line Parameters

- **main(String[] args)**
  - args: 接受控制台命令行上面的实参
- exmaple:

```
public class Message
{
    public static void main(String[] agrs)
    {
        if(args.length == 0 || args[0].equals("-h"))
        {
            System.out.print("Hello");
        }
        else if(args[0].equals("-g"))
        {
            System.out.print("Goodbye,");
        }
        for(int i = 1;i < args.length;++i)
        {
            System.out.print(" " + args[i]);
        }
        System.out.println("!");
    }
}
// 编译:
// java Message -g cruel world
// 运行:
// Goodbye,cruel world!
// args[0] :"-g"
// args[1] :"cruel"
// args[2] :"world"
```

3.10.5 Array Sorting

- **Arrays.sort** method

```
int[] a = new int[10000];
//...
Array.sort(a);    // tuned version of the QucikSort algorithm
```

3.10.6 Multidimensional Arrays

> 多维数组

- *use example:* `elemType[][] arr = new elemType[10][10];`
- *use for each:* 用范围for对二位数组操作

```
for(double[] row : a)
    for(double value : row)
        // do something
```

> 标准提供了打印数组时数组转字符串的方法toString和多维数组的deepToString方法

3.10.7 Ragged Arrays

> Java没有多维数组，而是由一位数组伪装的多维数组
>
> 因此我们可以这么做：`int[][] odds = new int[10][];`

- *wraning:* 类似于C++中

```
double** balnces = new double*[10]; // C++
for(i = 0;i < 10;++i)
    balances[i] = new double[6];
```

> The Next chapter covers **object-oriented** programming in Java