1 声明数组

在声明数组变量时，需要指出数组类型和数组变量的名字。例如：

```
int[] a;
```

这条语句只声明了变量 `a`，并没有将 `a` 初始化为一个真正的数组。应该使用 `new` 操作符创建数组：

```
a = new int[10];
```

在方括号中指定数组长度，数组长度可以是变量，也可以是 0。数组一旦创建，就不能更改长度。

可以在创建数组对象的同时提供初始值。例如：

```
int[] smallPrime = { 2, 3, 5, 7, 11, 13 };
```

这个语法中不需要使用 `new`，也不用指定长度。将初始值按顺序列在大括号中，中间用逗号分隔。最后一个值后面允许有逗号，便于为数组增加值。

还可以声明匿名数组：

```
new int[] { 17, 19, 23, 29, 31, 37 };
```

这会分配一个新数组并填入大括号中提供的值，数组大小设置为初始值个数。可以用这种语法重新初始化一个数组而无需创建新变量。例如：

```
smallPrime = new int[] { 17, 19, 23, 29, 31, 37 };
```

2 访问数组元素

数组下标从 0 开始，可以使用`数组名[下标]`的形式访问数组元素。如果访问越界，会引发“array index out of bounds”异常。

创建数组时，数组元素会默认初始化。数字数组的元素初始化为 0，`boolean`数组的元素初始化为`false`，对象数组的元素初始化为`null`。

要想获得数组中的元素个数，可以使用`数组名.length`。例如：

```
// 遍历数组元素
for (int i = 0; i < a.length; i++)
    System.out.println(a[i]);
```

3 `for each`循环

`for each`循环的格式为：

```
for(variable : collection) statement
```

定义一个变量`variable`用于暂存集合`collection`中的每一个元素，并执行 statement 中的语句。`collection`这一集合表达式必须是一个数组或者是一个实现了`Iterable`接口的类对象。例如：

```
int[] a = new int[] { 17, 19, 23, 29, 31, 37 };

for (int element : a)
    System.out.println(element);
```

4 数组拷贝

在 Java 中，允许将一个数组变量拷贝到另一个数组变量。例如：

```
int[] a = new int[10];
int[] b = a;
```

这时，两个变量将引用同一个数组。如果修改数组`b`中的元素，数组`a`中的元素也会同时被修改。

如果希望将一个数组的所有值拷贝到一个新的数组中去，就要使用`Arrays`类中的`copyOf()`静态方法：

```
int[] b = Arrays.copyOf(a, a.length);
```

第二个参数是新数组的长度。这个方法通常用来增加数组的大小：

```
a = Arrays.copyOf(a, 2 * a.length);
```

如果数组元素是数值型，额外的元素将被赋值为 0；如果数组元素是`boolean`型，将赋值为`false`。如果长度小于原始长度，则只拷贝前面的值。

5 命令行参数

`main`方法带有`String[] args`参数，表明`main`方法将接收一个字符串数组，可以通过命令行指定。

6 `Arrays` API

```
/* java.util.Arrays */
/* type 代表数组元素类型，可以是 int、long、short、char、byte、boolean、float 或 double */

// 返回包含 a 中元素的字符串，这些元素用中括号包围，并用逗号分隔
static String toString(type[] a)

// 返回与 a 类型相同的数组，长度为 length，数组元素为 a 的值
static type[] copyOf(type[] a, int length)

// 返回与 a 类型相同的数组，长度为 end-start，数组元素为 a 的值
// 如果 end > a.length，结果会填充 0 或 false
static type[] copyOfRange(typr[] a, int start, int end)

// 使用优化的快速排序算法对数组进行排序
static void sort(type[] a)

// 使用二分查找算法在有序数组 a 中查找值 v
// 若找到 v，返回下标；若找不到，返回一个负值 r，-r-1 是应插入的位置
static int binarySearch(type[] a, type v)

// 将数组的所有元素设置为 v
static void fill(type[] a, type v)

// 如果两个数组大小相等，并且下标相同的元素都对应相等，返回 true
static boolean equals(type[] a, type[] b)
```

7 多维数组

多维数组的声明和初始化与一维数组类似：

```
double[][] balances;
balances = new double[6][10]; // 第一个中括号是行数，第二个是列数

int[][] square =
    {
        { 16, 3, 2, 13 },
        { 5, 10, 11, 8 },
        { 9, 6, 7, 12 },
        { 4, 15, 14, 1 }
    };
```

数组初始化后，就可以使用多个中括号访问各个元素。

多维数组本质上属于“数组的数组”，把每一行的一维数组看做一个元素。例如上面的`balances`数组，它实际上有 6 个元素，每个元素又是长度为 10 的数组。表达式`balances[i]`引用数组的第`i`行，它本身也是一个数组，`balances[i][j]`引用这个数组的第`j`个元素。

可以使用多重循环遍历多维数组元素：

```
// 每一行是一个元素，balances.length 实际上是行数
for (int i = 0; i < balances.length; i++)
{
    // 对第 i 行的元素进行处理
    for (int j = 0; j < balances[i].length; j++)
    {
        //...
    }
}
```

一个`for each`循环不能自动处理二维数组的每个元素，它会把每一行当做一个元素，循环处理各行。要想访问二维数组的所有元素，需要使用两个嵌套的循环，例如：

```
for (double[] row : a) // row 提取一行作为一个元素
    for (double value : row) // value 提取这一行中的各个元素
        //...
```

由于可以单独地访问数组的某一行，所以可以让两行交换。例如：

```
double[] temp = balances[i];
balances[i] = balances[i + 1];
balances[i + 1] = temp;
```

还可以构造“不规则数组”，即数组的每一行有不同的长度。例如：

```
// 首先，分配一个数组包含这些行
int[][] odds = new int[NMAX + 1][];

// 接下来，分配这些行
for (int n = 0; n <= NMAX; n++)
    odds[n] = new int[n + 1];

// 之后，就可以采用通常的方式访问其中的元素了
```