ГЛАВА 10

# Массивы


В языке программирования Java *массивы* являются объектами (§4.3.1), создаются динамически и могут присваиваться переменным типа `Object` (§4.3.2). Все методы класса `Object` можно вызывать для массива.

Объект массива содержит ряд переменных. Количество переменных может быть равно нулю (в этом случае массив считается пустым). Переменные, содержащиеся в массиве, не имеют имен. Вместо этого обращение к ним осуществляется с помощью выражений доступа к массиву, использующих значения индекса, которые представляют собой неотрицательные целые числа. Эти переменные называются *компонентами массива*. Если массив имеет *n* компонентов, мы говорим, что *n* представляет собой длину массива; обращение к компонентам массива осуществляется с помощью целочисленных индексов от 0 до $n-1$ включительно.

Все компоненты массива имеют один и тот же тип — *тип компонента* массива. Если типом компонента массива является `T`, то тип самого массива записывается как `T[]`.

Значение компонента массива типа `float` всегда является элементом набора значений float (§4.2.3); аналогично значение компонента массива типа `double` всегда является элементом набора значений double. Значениям компонентов массива типа `float` не разрешено быть элементами набора значений float с расширенным показателем степени, не являющимися одновременно элементами набора значений float. Аналогично значениям компонентов массива типа `double` не разрешено быть элементами набора значений double с расширенным показателем степени, не являющимися одновременно элементами набора значений double.

Тип компонента массива сам может быть типом массива. Компоненты такого массива могут содержать ссылки на подмассивы. Если, начиная с любого типа массива, рассмотреть тип его компонентов, а затем (если он также представляет собой тип массива) тип компонентов этого типа и так далее, то в конце концов будет достигнут тип компонента, который не является типом массива. Он называется *типом элемента* исходного массива, а компоненты на этом уровне структуры данных называются *элементами* исходного массива.

Существует несколько ситуаций, в которых элемент массива может быть массивом: если тип элемента представляет собой `Object`, `Cloneable` или `java.io.Serializable`, то некоторые или все элементы могут быть массивами, поскольку любой объект массива может быть присвоен любой переменной этих типов.

# §10.1. Типы массивов

Типы массивов используются в объявлениях и выражениях приведения (§15.16).

Тип массива записывается как имя типа элементов, за которым следует некоторое количество пустых пар квадратных скобок `[]`. Количество пар скобок указывает глубину вложенности массива.

Длина массива частью типа не является.

Тип элемента массива может быть любым типом, примитивным или ссылочным. В частности, справедливо следующее.

- Разрешены массивы с типом интерфейса в качестве типа элемента.

  Элемент такого массива может иметь в качестве значения пустую ссылку или экземпляр любого типа, который реализует данный интерфейс.

- Разрешены массивы с типом абстрактного класса в качестве типа элементов.

  Элемент такого массива может иметь в качестве значения пустую ссылку или экземпляр любого не являющегося абстрактным подкласса этого абстрактного класса.

Супертипы типа массива описаны в §4.10.3.

> Отношение супертипа для типов массивов не совпадает с отношением суперкласса. Непосредственным супертипом `Integer[]` в соответствии с §4.10.3 является `Number[]`, но непосредственным суперклассом `Integer[]` является `Object` в соответствии с объектом `Class` для `Integer[]` (§10.8). На практике это не имеет значения, так как `Object` является супертипом для всех типов массивов.

# §10.2. Переменные массивов

Переменная типа массива хранит ссылку на объект. Объявление переменной типа массива не создает объект массива и не выделяет память для его компонентов. Оно создает только саму переменную, которая может содержать ссылку на массив.

Однако инициализирующая часть объявления (§8.3, §9.3, §14.4.1) может создавать массив, ссылка на который становится исходным значением переменной.

**Пример 10.2-1. Объявления переменных массивов**

```
int[]     ai;       // Массив int
short[][] as;       // Массив массивов short
short     s,        // Переменная типа short
          aas[][];  // Массив массивов short
Object[]  ao,       // Массив Object
          otherAo;  // Массив Object
Collection<?>[] ca; // Массив Collection неизвестного типа
```

Приведенные выше объявления не создают объектов массивов. Объявления переменных массивов, которые создают объекты массивов, показаны ниже.

```
Exception ae[]  = new Exception[3];
Object aao[][]  = new Exception[2][3];
int[] factorial = { 1, 1, 2, 6, 24, 120, 720, 5040 };
```

```
char ac[]       = { 'n', 'o', 't', ' ', 'a', ' ',
                    'S', 't', 'r', 'i', 'n', 'g' };
String[] aas    = { "array", "of", "String", };
```

Квадратные скобки `[]` могут встречаться в качестве части типа в начале объявления либо быть частью объявления определенной переменной, либо находиться в обоих местах одновременно.

Например:

```
byte[] rowvector, colvector, matrix[];
```

Это объявление эквивалентно следующему.

```
byte rowvector[], colvector[], matrix[][];
```

В объявлении переменной (§8.3, §8.4.1, §9.3, 9.4, §14.4.1, §14.14.2, §15.27.1), за исключением параметра переменной арности, тип массива переменной обозначается типом массива, находящимся в начале объявления, за которым следуют пары квадратных скобок, а затем *Identifier* переменной.

Например, объявление локальных переменных

```
int a, b[], c[][];
```

эквивалентно ряду объявлений

```
int a;
int[] b;
int[][] c;
```

Скобки в объявлениях разрешены как дань традиции C и C++. Однако общие правила объявлений переменных разрешают скобкам находиться как в типе, так и у идентификатора переменной, так что объявление локальных переменных

```
float[][] f[][], g[][][], h[];
```

эквивалентно ряду объявлений

```
float[][][][] f;
float[][][][][] g;
float[][][] h;
```

Мы не рекомендуем такую смешанную запись, когда пары скобок имеются в обеих частях объявления.

После создания объекта массива его длина никогда не меняется. Чтобы сделать переменную массива ссылающейся на массив иной длины, переменной следует присвоить ссылку на другой массив.

Одна переменная типа массива может содержать ссылки на массивы разной длины, потому что длина массива не является частью его типа.

Если переменная массива *v* имеет тип `A[]`, где *A* является ссылочным типом, то *v* может содержать ссылку на экземпляр любого типа массива `B[]`, что обеспечивает возможность того, что *B* может быть присвоен *A* (§5.2). Это может привести к исключению во время выполнения при позднем присваивании; этот вопрос рассматривается в §10.5.

# §10.3. Создание массива

Массив создается с помощью выражения создания массива (§15.10.1) или инициализатора массива (§10.6).

Выражение создания массива указывает тип элемента, количество уровней вложенности массивов и длину массива как минимум для одного из уровней вложенности. Длина массива доступна как переменная экземпляра `length`, объявленная как `final`.

Инициализатор массива создает массив и предоставляет исходные значения его компонентов.

# §10.4. Доступ к массивам

Обращение к компоненту массива осуществляется с помощью выражения доступа к массиву (§15.10.3), которое состоит из выражения, значение которого представляет собой ссылку на массив, за которой следует индексное выражение в квадратных скобках, как в записи `A[i]`.

Нумерация элементов любых массивов начинается с нуля. Массив длиной $n$ может быть индексирован целыми числами от 0 до $n-1$.

**Пример 10.4-1. Обращение к массиву**

```
class Gauss {
    public static void main(String[] args) {
        int[] ia = new int[101];
        for (int i = 0; i < ia.length; i++) ia[i] = i;
        int sum = 0;
        for (int e : ia) sum += e;
        System.out.println(sum);
    }
}
```

Вывод программы имеет вид

```
5050
```

Программа объявляет переменную `ia`, которая имеет тип массива элементов типа `int`, т.е. `int[]`. Переменная `ia` инициализируется как указывающая на новый объект, созданный с помощью выражения создания массива (§15.10.1). Выражение создания массива указывает, что в массиве должен быть `101` компонент. Как показано в исходном тексте, длина массива доступна посредством поля `length`. Программа заполняет массив целыми числами от `0` до `100`, суммирует их и выводит результат.

Массивы должны индексироваться значениями типа `int`; значения типа `short`, `byte` или `char` также могут использоваться в качестве значений индексов, так как над ними может быть выполнено унарное числовое повышение (§5.6.1) и они станут значениями типа `int`.

Попытки обращения к компонентам массива с помощью индекса типа `long` приводят к ошибке времени компиляции.

Все обращения к массивам проверяются во время выполнения; попытка использовать индекс меньше нуля или больший или равный длине массива приводит к генерации исключения `ArrayIndexOutOfBoundsException` (§15.10.4).

# §10.5. Исключение `ArrayStoreException`

Для массива типа `A[]`, где *A* является ссылочным типом, во время выполнения выполняется проверка присваиваний компоненту массива, чтобы гарантировать, что конкретное значение действительно может быть присвоено компоненту массива.

Если тип присваиваемого значения не совместим по присваиванию (§5.2) с типом компонента, генерируется исключение `ArrayStoreException`.

Если тип компонента массива недоступен во время выполнения (§4.7), виртуальная машина Java не может выполнить проверку, описанную в предыдущем абзаце. Поэтому выражение создания массива с недоступным во время выполнения типом элементов запрещено (§15.10.1). Можно объявить переменную типа массива, тип элемента которого недоступен во время выполнения, но присваивание результата выражения создания массива переменной обязательно вызовет предупреждение о непроверенном типе (§5.1.9).

**Пример 10.5-1. `ArrayStoreException`**

```
class Point { int x, y; }
class ColoredPoint extends Point { int color; }
class Test {
    public static void main(String[] args) {
        ColoredPoint[] cpa = new ColoredPoint[10];
        Point[] pa = cpa;
        System.out.println(pa[1] == null);
        try {
            pa[0] = new Point();
        } catch (ArrayStoreException e) {
            System.out.println(e);
        }
    }
}
```

Вывод этой программы имеет вид

```
true
java.lang.ArrayStoreException: Point
```

Переменная `pa` имеет тип `Point[]`, а переменная `cpa` имеет в качестве значения ссылку на объект типа `ColoredPoint[]`. Объект типа `ColoredPoint` может быть присвоен объекту типа `Point`; следовательно, значение `cpa` может быть присвоено переменной `pa`.

Обращение к этому массиву `pa`, например, при проверке `pa[1]` на равенство `null` не приводит к ошибке времени выполнения. Это связано с тем, что элемент массива типа `ColoredPoint[]` представляет собой `ColoredPoint`, а каждый

> `ColoredPoint` может заменять собой `Point`, поскольку `Point` является суперклассом для `ColoredPoint`.
>
> С другой стороны, присваивание массиву `pa` может привести к ошибке времени выполнения. Во время компиляции присваивание элементу `pa` проверяется, чтобы гарантировать, что присваиваемое значение представляет собой `Point`. Но поскольку `pa` хранит ссылку на массив `ColoredPoint`, присваивание корректно, только если тип присваиваемого значения во время выполнения представляет собой именно `ColoredPoint`.
>
> Виртуальная машина Java выполняет проверку таких ситуаций во время выполнения, чтобы гарантировать корректность присваивания; в случае некорректного присваивания генерируется исключение `ArrayStoreException`.

# §10.6. Инициализаторы массивов

*Инициализатор массива* может быть указан в объявлении поля (§8.3, §9.3) или локальной переменной (§14.4) или как часть выражения создания массива (§15.10.1) для создания массива и представления некоторых его исходных значений.

*ArrayInitializer*:
`{` *[VariableInitializerList] [,]* `}`

*VariableInitializerList*:
*VariableInitializer {, VariableInitializer}*

Далее для ясности следует повторение фрагмента из §8.3.

*VariableInitializer*:
*Expression*
*ArrayInitializer*

Инициализатор массива записывается как список выражений, разделенных запятыми, заключенный в фигурные скобки `{}`.

После последнего выражения списка может располагаться завершающая запятая, которая игнорируется.

Каждый инициализатор переменной должен быть совместим по присваиванию (§5.2) с типом компонентов массива, иначе генерируется ошибка времени компиляции.

Если тип компонента инициализируемого массива недоступен во время выполнения (§4.7), генерируется ошибка времени компиляции.

Длина создаваемого массива должна быть равна количеству инициализаторов переменных, непосредственно заключенных в фигурные скобки инициализатора массива. Память выделяется для нового массива именно этой длины. Если для массива не хватает памяти, вычисление инициализатора массива прерывается генерацией исключения `OutOfMemoryError`. В противном случае создается одномерный массив указанной длины, а каждый компонент массива инициализируется его значением по умолчанию (§4.12.5).

Затем сразу же выполняются заключенные в фигурные скобки инициализаторы переменных. Выполнение происходит слева направо в порядке их появления в исходном

тексте. *n*-й инициализатор переменной определяет значение $n-1$-го компонента массива. В случае прерывания выполнения инициализатора переменной по той же причине завершается выполнение инициализатора массива. Если все выражения инициализаторов переменных завершаются нормально, нормально завершается и инициализатор массива, давая значение вновь инициализированного массива.

Если тип компонента представляет собой тип массива, то инициализатор переменной, определяющий компонент, сам может быть инициализатором массива; таким образом, инициализаторы массивов могут быть вложенными. В этом случае выполнение вложенного инициализатора массива строит и инициализирует объект с помощью рекурсивного применения описанного выше алгоритма и присваивает его компоненту.

**Пример 10.6-1. Инициализаторы массивов**

```
class Test {
    public static void main(String[] args) {
        int ia[][] = { {1, 2}, null };
        for (int[] ea : ia) {
            for (int e: ea) {
                System.out.println(e);
            }
        }
    }
}
```

Вывод этой программы имеет следующий вид.

```
12
```

После этого происходит генерация исключения `NullPointerException` при попытке индексации второго компонента массива `ia`, который представляет собой ссылку `null`.

# §10.7. Члены массивов

Члены типа массива перечислены ниже.

- Поле `length`, объявленное как `public final`, которое содержит количество компонентов массива. Значение `length` может быть положительным или нулевым.
- Метод `clone`, объявленный как `public`, который перекрывает метод с тем же именем класса `Object` и не генерирует проверяемых исключений. Типом возвращаемого значения метода `clone` типа массива *`T`*`[]` является *`T`*`[]`.

  Клон многомерного массива строится на основании так называемого поверхностного копирования, т.е. создается только один новый массив. Подмассивы же являются общими.
- Все эти члены унаследованы от класса `Object`; единственный метод `Object`, который не является унаследованным, — метод `clone`.

Смотрите в §9.6.4.4 другую ситуацию, в которой различия между методами `Object`, объявленными как `public` и не как `public`, требуют особого внимания.

Таким образом, массив имеет те же `public`-поля и методы, что и следующий класс.

```
class A<T> implements Cloneable, java.io.Serializable {
    public final int length = X ;
    public T[] clone() {
        try {
            return (T[])super.clone();
        } catch (CloneNotSupportedException e) {
            throw new InternalError(e.getMessage());
        }
    }
}
```

Заметим, что приведение к `T[]` в рассмотренном выше примере генерировало бы предупреждение о непроверенном типе (§5.1.9), если бы массивы действительно были реализованы таким образом.

**Пример 10.7-1. Массивы клонируемы**

```
class Test1 {
    public static void main(String[] args) {
        int ia1[] = { 1, 2 };
        int ia2[] = ia1.clone();
        System.out.print((ia1 == ia2) + " ");
        ia1[1]++;
        System.out.println(ia2[1]);
    }
}
```

Вывод этой программы имеет следующий вид.

```
false 2
```

Он демонстрирует, что компоненты массивов, на которые ссылаются переменные `ia1` и `ia2`, являются разными переменными.

**Пример 10.7-2. Общие подмассивы после клонирования**

Тот факт, что подмассивы после клонирования многомерного массива являются общими, иллюстрируется следующей программой.

```
class Test2 {
    public static void main(String[] args) throws Throwable {
        int ia[][] = { {1,2}, null };
        int ja[][] = ia.clone();
        System.out.print((ia == ja) + " ");
        System.out.println(ia[0] == ja[0] && ia[1] == ja[1]);
    }
}
```

Вывод этой программы имеет следующий вид.

```
false true
```

Он демонстрирует, что массив `int[]`, который представляет собой `ia[0]`, и массив `int[]`, который представляет собой `ja[0]`, являются одним и тем же массивом.

# §10.8. Объекты `Class` массивов

Каждый массив имеет связанный с ним объект типа `Class`, разделяемый со всеми другими массивами с тем же типом компонентов.

Хотя тип массива не является классом, объект `Class` каждого массива действует так, как если бы выполнялось следующее:

- непосредственным суперклассом каждого типа массива был бы `Object`;
- каждый тип массива реализовывал бы интерфейсы `Cloneable` и `java.io.Serializable`.

**Пример 10.8-1. Объект `Class` массива**

```
class Test1 {
    public static void main(String[] args) {
        int[] ia = new int[3];
        System.out.println(ia.getClass());
        System.out.println(ia.getClass().getSuperclass());
        for (Class<?> c : ia.getClass().getInterfaces())
            System.out.println("Superinterface: " + c);
    }
}
```

Вывод этой программы имеет следующий вид.

```
class [I
class java.lang.Object
Superinterface: interface java.lang.Cloneable
Superinterface: interface java.io.Serializable
```

Здесь строка `"[I"` представляет собой сигнатуру типа времени выполнения для объекта `Class` “массив с типом компонентов `int`”.

**Пример 10.8-2. Разделяемые объекты `Class` массива**

```
class Test2 {
    public static void main(String[] args) {
        int[] ia = new int[3];
        int[] ib = new int[6];
        System.out.println(ia == ib);
        System.out.println(ia.getClass() == ib.getClass());
    }
}
```

Вывод этой программы имеет следующий вид.

```
false
true
```

В то время как `ia` и `ib` ссылаются на различные массивы, результат сравнения объектов `Class` демонстрирует, что все массивы, компоненты которых имеют тип `int`, являются экземплярами одного и того же типа массива (а именно — `int[]`).

# §10.9. Массив символов не является строкой

В языке программирования Java, в отличие от C, ни массив элементов типа `char` не является строкой `String`, ни строка `String` не является массивом элементов типа `char` с завершающим символом `'\u0000'` (символ `NUL`).

Объект `String` является неизменяемым, т.е. его содержимое никогда не изменяется, в то время как массив элементов типа `char` имеет изменяемые элементы.

Метод `toCharArray` в классе `String` возвращает массив символов, содержащий ту же последовательность символов, что и `String`. Класс `StringBuffer` реализует полезные методы для изменяемых массивов символов.