Тема 3. Типове данни

доц. д-р инж. Владимир Николов

25.2.2020 г.

Съдържание

Референтни типове

- ***Array*** *- масиви;*
  - **Особености на масивите;**
  - **Динамични масиви;**
  - **Копиране на масиви в Java;**
  - **Референциите на масивите в Java.**
- **Класове за низове в** java.lang.*
  - **String;**
  - **StringBuilder;**
  - **StringBuffer.**

Референтни типове

Дефиниция: ***Типове данни, които се създават САМО в динамичната памет (heap) и се предават чрез референция между методите.***

- ***class – класове разгледани в Л2***
- ***Array - масиви;***
- ***interface – интерфейси.***

Референтни типове

Дефиниция: *Array – масив:*

Вграден в езика клас за съхранение на константен брой стойности в една променлива. Следствия:

- ***Базов клас – Object, поддържа всички негови методи;***
- ***Име на класа:***

```
int[] arrInt = new int[5];
System.out.println(arrInt.getClass().getName()); => [I
```

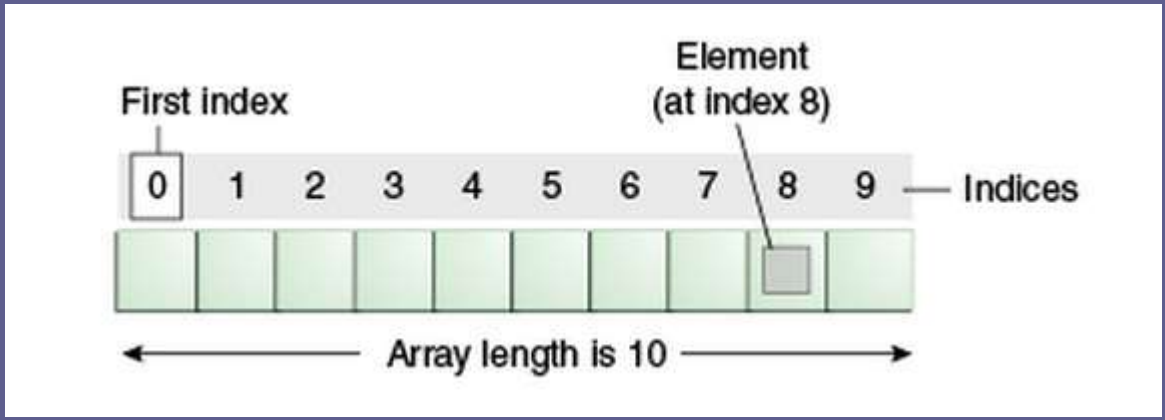
Име на класа масив: [

Тип на масива: I

двумерен масив?

Референтни типове *Array*

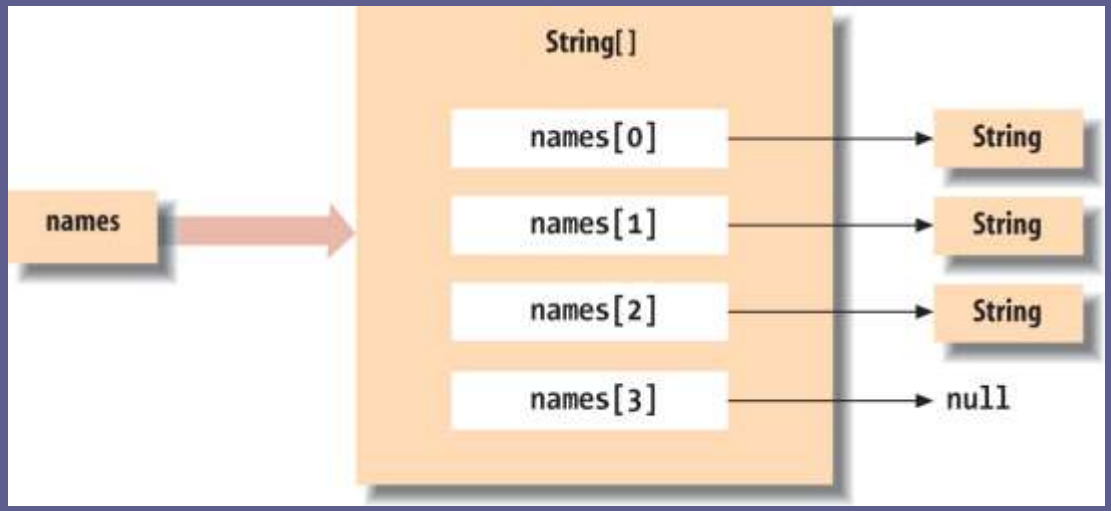
Организация на масив



Референтни типове *Array*

Масивите се представят чрез референции:

```
String names [] = new String [4];
names [0] = new String();
names [1] = "NameIndex1";
names [2] = someObject.toString();
// names[3] == null
```



Референтни типове *Array*

Особености на масивите

- При деклариране на масиви в C++ се създава памет за съхраняване на масива. При деклариране на масив в Java се декларира само референция към масива (не се създава паметта);
- Аналогът в C (C++) е масив от указатели към обекти, но указателите в C или C ++ са стойности на адреса.
  - Един набор от референции е концептуално подобно, но референциите допускат само присвояване "=";
- За да се създаде реална памет за съхранение на елементите трябва да се алокира експлицитно с оператора "new";

Референтни типове *Array*

- *C++*

int A[10]; // A е масив с големина 10 от типа (int)

A[0] = 5; // запис на данни в първия елемент на A

- Създаване на масив в *JAVA - синтаксис*

int [] A; // A е указател (null референция към масива)

A = new int [10]; // референция A се инициализира с
// масив с големина 10 от тип (int)

A[0] = 5; // запис на данни в първия елемент на A

Общ синтаксис:

<тип>[] <променлива> ;

<тип> [] <променлива>= new <тип> [<размер>];

Референтни типове *Array*

Съкратени с начална инициализация:

- Както в C++ и в Java могат да се инициализират стойности на масива със затворени във фигурни скоби елементи :

```
int [] primes = { 2, 3, 5, 7, 7+4 };
// primes е указател (референция към масива) с
      дължината 5 и стойности 2, 3, 5, 7, 11

byte [] bar = new byte[] { 1, 2, 3, 4, 5 }; //инициализация
```

Референтни типове *Array*

Нулева начална инициализация:

В Java се осигурява началната инициализация на елементите в създавания масив. Подразбират се нулеви стойности на типовете:

Тип	стойност
boolean	false
char	'\u0000'
byte, int, short, long, float, double	0
Референция	null

Референтни типове *Array*

- В Java е осигурена проверка за границите на масива;
- При индексация извън границите на масива - изключение (out-of-bounds);
- Статично поле length (само за четене)

Пример:

```
int [] A = new int[10]; //установява дължината
    ...  A.length ...  // може да се прочете ( 10)
    A = new int[20];
    ...  A.length ...        // нова стойност 20
```

Референтни типове *Array*

Достъп до елементите на масива с цикъл:

```
int[] numbers = {1,2,3,4,5,6,7,8,9,10};
    for ( int index = 0; index < numbers.length; index++ ) {
      System.out.println("Element value : " + numbers[index]);
    }
```

Еквивалентно с For-Every:

```
        for (int value : numbers) {
          System.out.println("Element value : " + value);
        }
```

Референтни типове *Array*

Масивите са динамични типове:

```
int [] A;

int [] A = new int [4];

int [][] A = new int[4][3];

int [][] A = new int[4][]; //null
A[1] = new int[4];
A[3] = new int[2];
```

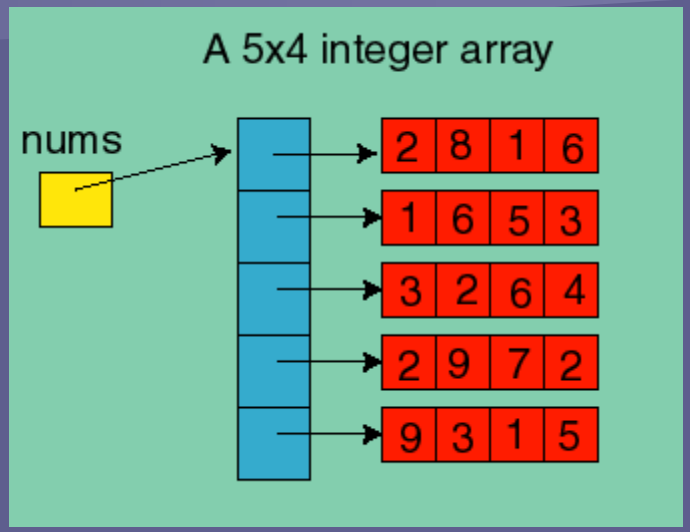
Референтни типове *Array*

Двумерни масиви:

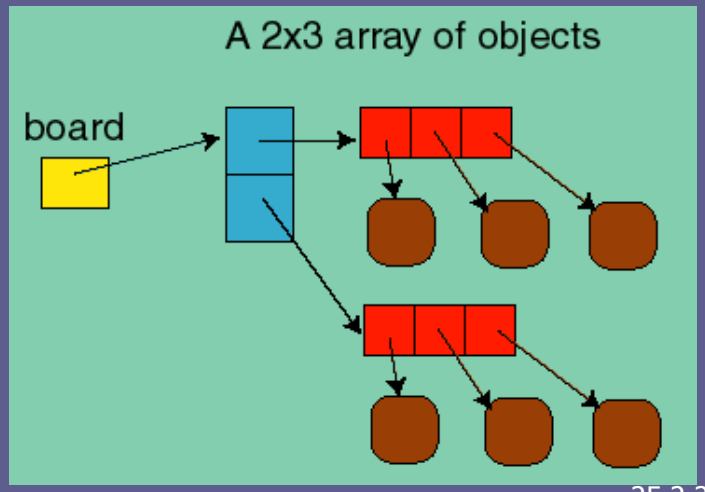
```
int [][] nums = new int[5][4];
nums[0][0]=2;
nums[0][1]=8;
...
```

Допуска се:

```
int [][] nums = new int[5][];
nums[1] = new int[4];
nums[3] = new int[2];
```



Масив от обекти:



Референтни типове *Array*

Копиране на масиви в Java:

- **Чрез clone на клас Object;**

Пример:

```
int[] numbers = {1,2,3,4,5,6,7,8,9,10};
System.out.println("Printing clone of the array:");
int numbersClone[]=numbers.clone();
for(int value:numbersClone)
System.out.println(value);
System.out.println("Are both equal?");
System.out.println(numbers==numbersClone);
```

резултат?

- **Чрез java.lang.System arraycopy**

Референтни типове *Array*

Копиране на масиви в Java.

Използва се методът на java.lang.System arraycopy . За изпълнението се използва синтаксис, подобен на println. Параметрите са 5:

src: масив source (масивът от който се копира)

srcPos: начален индекс на масива от който се копира

dst: масив destination (масивът към който се копира)

dstPos: начален индекс на масива, към който се копира

count: брой на стойностите за копиране

Референтни типове *Array*

Пример:

```
int [] A, B;
A = new int[10];
// зареждане на масива A
B = new int[5];
System.arraycopy(A, 0, B, 0, 5)  // копират се 5
   стойности от A в B
System.arraycopy(A, 9, B, 4, 1)  // копира се
// последната стойност на  A в последната на B
```

Референтни типове *Array*

Особености на копирането:

- Трябва да се осигури алокирана памет за масива към който се копира - runtime error;
- Масивът source трябва да има има достатъчно стойности, които да се копират, т.е. да има големина повече от srcPos+count;
- При прости типове се изисква типово съответствие на source и destination - не може например да се копира масив от int към масив от double или обратно;
- При референтни типове се допуска синтаксиса System.arraycopy(A, j, B, k, n) ако присвояването:

B[0] = A[0] е възможно.

Референтни типове *Array*

- Методът работи и при съвпадение на source и destination Например:

int [] A = {0, 1, 2, 3, 4}; // алокация

System.arraycopy(A, 0, A, 1, 4);

A ще съдържа: [0, 0, 1, 2, 3].

- Примери:

```
int [] A = new int[4];
int [] B = {0,1,2,3,4,5,6,7,8,9};
System.arraycopy(B,2,A,0,4);
```

Референтни типове *Array*

Примери:

```
int [] A = new int[4];
int [] B = {2,3,4};
System.arraycopy(B,0,A,0,4);
```

```
int [] A = new int[4];
int [] B = {0,1,2,3,4,5,6,7,8,9};
System.arraycopy(B,8,A,0,4);
```

Референтни типове *Array*

Примери:

Има ли грешки в кодовете?

```
int [] A = {1,1,1,1};
int [] B = {2,2,2};
System.arraycopy(A,0,B,1,2);
System.arraycopy(B,0,A,0,3);
```

```
//     int [] A = new int[4];
int [] B = {0,1,2,3,4,5,6,7,8,9};
System.arraycopy(B,0,A,0,10);
```

Проблеми с референциите на масивите в Java

- **Масивите са референтен тип и са наследници на Object, което трябва да се има впредвид при боравенето с тях - присвояване на референции:**

Примери:

```
String [] strings = new String [10];
Object [] objects = strings; // String [] като Object []
objects[0] = new Date(); // !!! Присвояване на друг
// грешка по време на изпълнение:
ArrayStoreException!
```

Проблеми с референциите на масивите в Java

- **Масивите по същество са референции към обектите, което трябва да се има впредвид при боравенето с тях (същност на оператор =):**

Присвояване на референции:

Примери:

```
int [] A = new int[3];
int [] B = new int[2];
A[0] = 5;
B = A;        //сочат една и съща памет!
B[0] = 2;  // A[0] = ?
```

Проблеми с референциите на масивите в Java

Какво се получава от действието на примерната програмната част?

```
// Има ли грешка от кода?
int A[5];
//
int [] A = new int[20];
int [] B = new int[10];
A = B;
A[15] = 0;
```

Проблеми с референциите на масивите в Java

Какво се получава от действието на програмната част?

```
// Пример 1
int [] A = {1,2,3};
int [] B;
B = A;
// Пример 2
int [] A;
A[0] = 0;
// Пример 3
int [] A, B;
B = 0;
```

Вградени типове за низове

Java осигурява два класа:

- String - за представяне на низове. Един обект от тип низ не може да бъде променен, след като е създаден.
  - Той може да бъде изтрит от heap - менажера, ако няма променливи, притежаващи референции към него, но не може да бъде променян (immutable);
- StringBuffer - за обработката им. Ако трябва да се промени на място (realloc), низа трябва да бъде обект на тип StringBuffer.

  Ефективност при конструктивни операции-поддържа се capacity() и length(). Динамично разширение при insert, append и др.

Вградени типове за низове

```
public final class String implements java.io.Serializable,
    Comparable<String>, CharSequence {
    // Вътрешно представяне на стойността
private final char value[];
// брой символи
private final int count;
private int hash; // хеш код (0)
    // отместване до първи индекс при операции: substring…
private final int offset;
…..
}
```

Лошо поведение при промени. Има подобрения в последните версии (1.7.XX) в поведението на substring

Вграден тип низ (String)

Низовете в Java се различават от масивите със символи на C++

Пакет: java.lang.String

Java включва класа *String*, **който е константен тип (immutable).**

String - особености

Особености и разлики на вътрешното представяне на типа String:

- Езикът C (C++) няма вградена поддръжка на низове. Стандартната техника на програмистите е да се използват символни масиви, завършващи с нула. Развитието на типа като клас – STL string;
- В Java низовете са имплементирани като обекти от класовете (String и StringBuffer), което означава, че са в ядрото на Java.

String - особености

Имплементацията на низ в Java, предимства:

- Начинът, по който се създават и се прави достъп до елементите на низовете, е уеднаквен и може да се прилага за всички низове на всички операционни системи;
- Тъй като низовите класове в Java са дефинирани като част от самия език, а не като библиотечно разширение (STL на C++), ***те функционират по дефинирания в езика начин***, независимо как са създадени;
- Класовете на Java за представяне на низове ***извършват проверки по време на изпълнение,*** което елиминира допускането на грешки и проблеми на тестването (сравнение C++);
- ***Използва се универсално кодиране - Unicode***

String - особености

- ***String не е еквивалент на масив от символи***, въпреки че може да се създаде обект *String* с използване на масив от символи.

✓ Следствие: Класът не бива да се заменя с масив от символи, тъй като ***String*** не може да се презапише по грешка на програмиста при предаването на стойността му като параметър на метод.

Пример за конструиране от масив от символи:

String quote = "**constant string...**";

- ***Вградена дължина***.

Пример:

```
int length = quote.length(  );
```

String - особености

- ***Вграден оператор за конкатенация*** на стрингове. Пример:

```
String name = "John " + "Smith";
String name = "John ".concat("Smith");
```

- ***Еквивалентни преобразувания*** с масив от символи и байтове:

```
char [] data=new char[]{ 'c', 'h', 'a', 'r', 'a', 'c', 't', 'e', 'r', 's' };
String characters = new String( data );
byte [] data = new byte [] { (byte)97, (byte)98, (byte)99 };
String abcSolaris = new String(data, "ISO8859_1");//Solaris
String abcWindows = new String(data, "CP1252 ");//Windows
```

String - особености

- ***Достъп до символите*** на стринга се осигурява от метод charAt(index):

```
String s = "testCharAt";
for ( int index = 0; index < s.length(  ); index++ )
   System.out.println( s.charAt( index ) );
```

- Обратно преобразуване към масив от символи.

```
char [] abcs =
   "abcdefghijklmnopqrstuvwxyz".toCharArray(  );
```

String - особености

- ***Базовите типове поддържат стрингово представяне***, осигурено от toString на Object:

```
String one = String.valueOf( 1 ); // integer, "1"
String two = String.valueOf( 2.384f );  // float, "2.384"
String notTrue = String.valueOf( false ); // boolean, "false"
```

- За предизвикване на преобразуването може да се използва празен стринг и конкатенация:

```
String two = "" + 2.384f;
String today = "" + new Date(  );
```

String - особености

- ***Нулевите референции*** при преобразуване предизвикват изключения (NullPointerException):

```
 Date date = new Date(  );
// дава стрингова интерпретация напр:
//"Sun Feb 11 05:45:34 CST 2008"
String d1 = String.valueOf( date );
String d2 = date.toString(  ); // еквивалентно
date = null;
d1 = String.valueOf( date );  // "null"
d2 = date.toString(  );  // NullPointerException
```

String - особености

- ***Сравняването*** на стрингове се осъществява с метода equals (equalsIgnoreCase):

```
String one = "FOO";
String two = "foo";
one.equals( two );              // false
one.equalsIgnoreCase( two );    // true
```

- Да не се използва сравняване посредством оператор ==, сравняващ референции на обектите например:

```
String foo1 = "foo";
String foo2 = String.valueOf( new char [] { 'f', 'o', 'o' }  );
foo1 == foo2          // сравняване на референции false!
foo1.equals( foo2 )  // сравняване на стойности =>true
```

String - особености

- сравняване посредством оператор == (intern()):

```
public class X { public static String strX = "hello"; }
public class Y { public static String strY = "hello"; }
public class Z { public static String strZ = "hell" + "o"; }
public class Test { // Уникален екземпляр на низ от глобалната памет
  public static void main( String[] args ) {
    System.out.println( X.strX == Y.strY ); // true
    System.out.println( X.strX == Z.strZ ); // true
    String s1 = "hel";    String s2 = "lo";
    System.out.println( X.strX == (s1 + s2 ) ); // false
    System.out.println( X.strX == (s1 + s2).intern() ); // true
  }
}
```

String – основни методи

Търсенето в стринговете се осигурява от вградените методи:

- **startsWith:**

```
String url = "http://foo.bar.com/";
if ( url.startsWith("http:") )  // true
```

- **indexOf( )** намира първото срещане на символ или подстринг в стринга и връща позицията спрямо началото или -1:

```
String abcs = "abcdefghijklmnopqrstuvwxyz";
int i_p = abcs.indexOf( 'p' );      // 15
int i_def = abcs.indexOf( "def" );   // 3
int i_edf = abcs.indexOf( "edf" );  // -1
```

- **lastIndexOf( )** за търсене от края на низа.

String – основни методи

- Java 5.0+ има допълнителен метод contains( ) с действие като substring на C++ :

```
String log = "There is an emergency in sector 7!";
if  ( log.contains("emergency") )
    pageSomeone(  );
// еквивалентно на
if ( log.indexOf("emegency") != -1 )
    pageSomeone(  );
```

String – основни методи

- За преобразуване на низове:

```
String str = "   abc   ";
str = str.trim(  );  // нов стринг "abc"
String down = "FOO".toLowerCase(  );      // нов стринг "foo"
String up   = down.toUpperCase(  );       // нов стринг "FOO"
String abcs = "abcdefghijklmnopqrstuvwxyz";
String cde = abcs.substring( 2, 5 ); // нов стринг "cde"
String xy = "xxooxxxoo".replace( "xx", "X" ); // нов стринг "XooXxoo"
```

- Съществуват методи, използващи регулярни изрази за анализ и преобразуване в съответствие с регулярни изрази

String - особености

- При последователни операции се преобразува към динамичен вариант:

```
String foo = "To " + "be " + "or";
```

Еквивалентен код:

```
String foo = new
        StringBuilder().append("To").append("be").append("or").toString();
```

StringBuilder и StringBuffer

Дефиниция: Динамичен вариант на String, който създава новите стрингове, разширявайки динамичната памет на съществуващия стринг (modifiable).

Дефиниция: Клас синоним за създаване и поддръжка на променяем стринг - java.lang.StringBuilder (Java 5+)

- Разлика между String и StringBuffer-Връща нов **String** обект при операциите, които изискват промяна на стринга, например:
  - substring( );
  - concat( );
  - replace( );
  - toLowerCase( ) toUpperCase( );
  - trim( );
  - valueOf( );

StringBuilder и StringBuffer

Специфични методи на StringBuffer:

- append( ) предефиниран за Object, String,char[], char[] с отместване и дължина, boolean, char, int, long, float, double;
- insert( ) предефиниран за Object, String,char[], char[], boolean, char, int,long, float, double
- reverse( );

```
// Примери на StringBuffer
public class ImmutableStrings {
  public static void main(String[] args) {
// при използване на String
    String foo = "foo";
    String s = "abc" + foo +"def" + Integer.toString(47);
    System.out.println(s);
// при използване на StringBuffer:
    StringBuffer sb = new StringBuffer("abc"); // създава String!
    sb.append(foo);
    sb.append("def"); // създава String!
    sb.append(Integer.toString(47));
    System.out.println(sb);
  }
}
```

StringBuilder и StringBuffer

- Използване на StringBuilder и StringBuffer

Неправилен код с използване на оператор за многократна конкатенация към String: + ...

```
while( (line = readLine( )) != EOF )
   text += line;
```

Реалокация:

```
StringBuilder text = new StringBuilder( );
  while( (line = readline( )) != EOF )
      text.append( line );
```

- Преобразуване към String:

```
String strText=text.toString( ) ;
```

Въпроси ?