CSE 11: Lecture 14

- Declaring and creating arrays
- Indexing
- Array initialization
- Arrays as arguments and as returned values
- Multidimensional arrays

(Reading: Savitch, Ch. 6)

Aggregate data types

- A variable of a *simple* data type contains only one data value
  - In Java, the primitive data types are simple data types
- A variable of an *aggregate* data type can contain several data values
  - In Java, a class is often an aggregate data type, because an object of a class can contain several instance variables
- *Arrays* are a particular kind of aggregate data type
  - Arrays are a convenience: they provide a nice way to deal with a bunch of variables all of the same type
  - This can be *very* useful

Arrays: elements, length, and indexing

- An array is: a sequence of variables, all of the same type
  - in Java, these variables can be reference-type pointers or primitive values
  - the pointers or primitive values are the *elements* of the array
  - these elements can be of any type (primitive types, classes... even arrays!), but all must be of the same type according to the Java type system

- An array contains a certain number of elements
  - this number is the *size* or *length* of the array
  - in Java, the size of an array is specified when the array is created, and cannot be changed

- The elements of an array are *indexed* by integers
  - in Java, indexes run from 0 up to the size of the array
  - an element is referred to using the name of a pointer to the array, together with the index of the element

Declaring arrays in Java

- An array is declared with a declaration statement of the form

```
<typename> <identifier>[];
```

or

```
<typename>[] <identifier>;
```

  - This declares `<identifier>` to be a pointer to an array that contains elements of type `<typename>`.

  - Examples:

```
int[]  arr;   // declares arr to be an array of ints
JButton[] b; // declares b to be an array of JButton pointers
boolean vals[];  // declares vals to be an array of booleans
```

Creating arrays in Java

- Arrays in Java are reference types. (You can think of an array as a special kind of object with special features, but an object nevertheless.)
- Like all reference types in Java, a mere declaration statement never creates an array
- To create an array in Java, you use **new**, the typename of elements in the array, and a bracketed integer expression that specifies the length of the array to be created:

```
new <typename> [ <length> ]
```

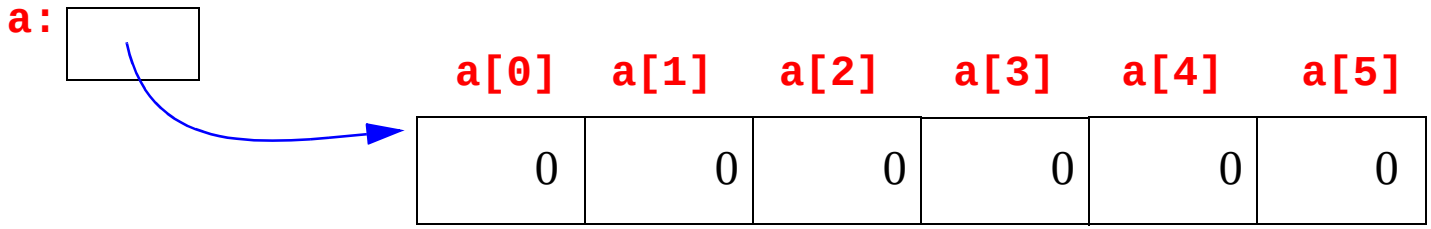
- The elements of the array are automatically initialized to default values: zero for numerical types; **false** for boolean; **null** for reference types
- Examples:

```
arr = new int[33];     // creates an array of 33 ints, all 0

b = new JButton[10];      // creates an array of 10 JButton
                          // pointers, all null (this creates
                          // no JButton objects!)

int s = 99;
vals = new boolean[s + 2];  // creates an array of 101 booleans,
                            // all false
```

Indexing arrays

- You refer to an an element of an array by using the name of a pointer to the array together with a bracketed int expression, called the *index*
- The index of the first element of the array is 0
- The use of any index value less than 0, or greater than or equal to the length of the array, is a runtime error in Java (an **`ArrayIndexOutOfBoundsException`** is thrown)

```
int[] a = new int[6];    // a points to an array of 6 ints
                         // initialized to 0
```



- **`a[0]`** refers to the first element of the array pointed to by **`a`** , **`a[1]`** refers to the second element of the array pointed to by **`a`**, etc.

Using arrays in Java

- The index of an array element can be any expression of int type with value in the proper range

```
int a[] = new int[6];  // create an array of 6 ints, named a

a[0]    // this expression refers to first element of a (index 0)

a[1+2]  // this expression refers to fourth element of a (index 3)

a[-1]          // NO!  array index can never be negative

a[6]           // NO!  array index should never be larger than
               //  size of array - 1

int j=3;

a[j]         // refers to fourth element of a (index 3)

a[j/2 - 1]   // refers to first element of a  (index 0)

double x=3.0;

a[x]   // NO!  array index MUST be int type, or a type Java
       // will automatically convert to int: byte, short, char
```

Using arrays in Java, cont'd

- An array element referred to with an array name and bracketed index can have its value used (an *rvalue*), or have its value assigned (an *lvalue*): it acts just like a variable!

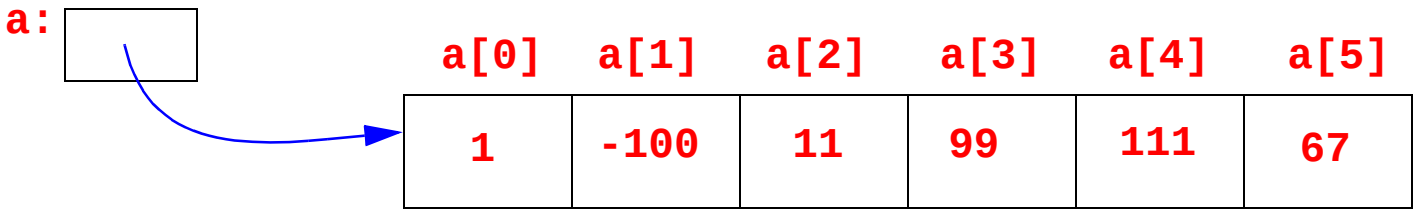
```
int i=2, j, a[];

a = new int[6];

a[0] = 333;         // assign 333 to first element of a

a[1] = i;           // assign value of i to second element of a

j = a[1];           // assign value of second element of a to j

a[2] = 3;           // assign 3 to a[2]

System.out.print(a[j]); // print out value of (j+1)st element of a

a[a[--j] + 1] = 444;    // what does this do?

System.out.print(a[3]); // print out value of 4th element of a

a[3]++;             // increment 4th element of a
```

Initializing arrays in Java

- Array elements in Java are initialized to default values (same as instance variables or static variables) if you do not initialize them explicitly

- You can initialize array elements to other values
  - One way is to use a comma-separated list of values enclosed in braces instead of **new**.
  - If you do it this way, it must be done in the declaration statement for the array only.

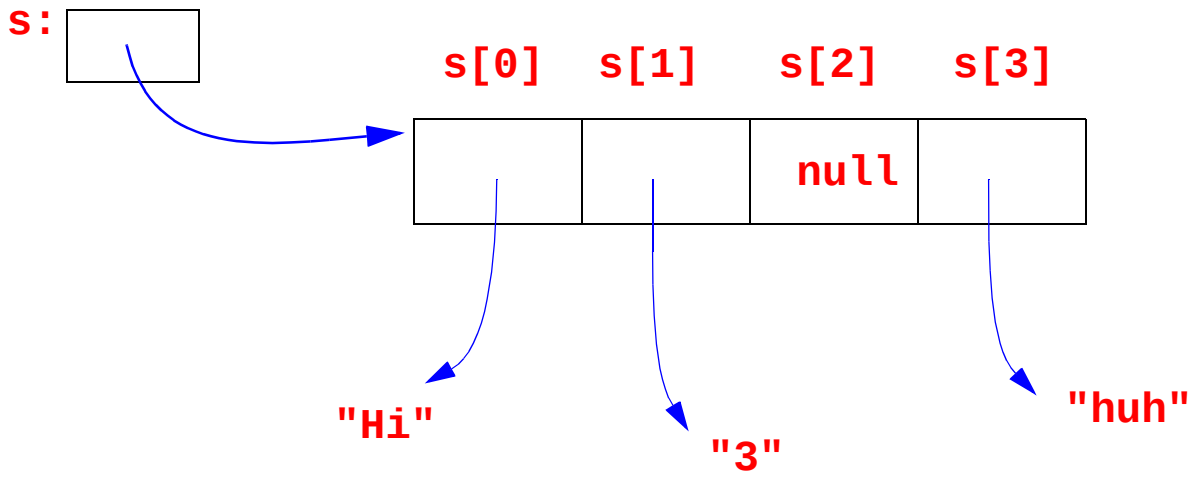
```
int j = 23, k = 44;
int[] a = {1, -100, j/2, 99, 111, j+k};
```



Initializing arrays of objects in Java

- You can also initialize elements of arrays of objects using the same approach

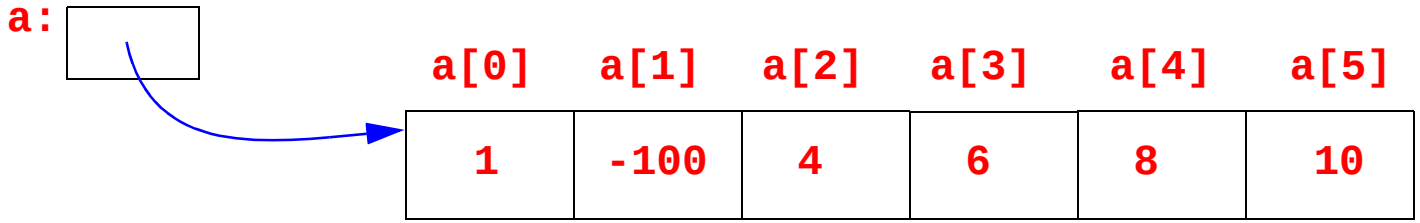
```
String[] s = {"Hi", Integer.toString(3), null, new String("huh")};
```



Initializing arrays in Java, cont'd

- Of course, you can always initialize array values explicitly, in a loop or one-by-one

```
int[] a = new int[6];
a[0] = 1;    a[1] = -100;
for(int i=2; i<6; i++) a[i] = 2*i;
```



The length instance variable for an array

- In Java, every array has a certain number of elements
- This is determined when the array is created, and cannot be changed
  - A pointer to an array can point to different arrays at different times; these arrays can be of different sizes; but each array has a fixed size specified when it is created
- The number of elements is also known as the size, or length of the array
- The length of any array can be determined by using its **`length`** public final instance variable:

```
int[] a = new int[6];

System.out.println( a.length );  // prints 6

double[] xx = {1.0, 2.0, 3.14159};

System.out.println(  xx.length  );  prints 3
```

- This is a really nice feature of Java arrays... they “keep track” of their own size information.

Passing arrays to methods

- Arrays can be arguments to methods
- In the method's formal parameter list, an array-type parameter must be declared as in an array variable declaration statement. Example:

```
// a method that takes an array of doubles as argument,
// and returns the average of the values in the array
public double average ( double[] val ) {
   double sum = 0.0;
   for(int i = 0; i < val.length; i++)
      sum += val[i];

   return sum / val.length;
}
```

- If a method takes an array argument, you specify the actual argument by just giving the name of the pointer to the array. Example:

```
double[] x = {1.0, 2.0, 3.0};
double avg = average( x );
System.out.println( avg );   // prints...
```

Arrays are reference types

- Arrays in Java are reference types, and they are passed to methods like objects:
  - they are passed by reference, not by value...
  - ...that is, the array is not copied, and the called method has access to the elements of the actual argument array

  Example on next slide.

Arrays as arguments: an example

```
   void swap(int x, int y) { // try to swap values of 2 int args
      int tmp = x;
      x = y;  y = tmp;
   }
   // swap element indx1 and element indx2 of array a
   void swap(int[] a, int indx1, int indx2) {
      int tmp = a[indx1];
      a[indx1] = a[indx2];  a[indx2] = tmp;
   }
-------------------------------------------------------------
   int i1 = 3; int i2 = 100;
   int[] arr = {i1, i2};

   swap(i1, i2);
   System.out.print(i1 + " " + i2); // prints...

   swap(arr,0,1);
   System.out.print(arr[0] + " " + arr[1]); // prints...

   System.out.print(i1 + " " + i2); // prints...
```

main's argument

- You have undoubtedly noticed that the main method in a Java program always has a header like

```
public static void main (String[] args) {
```

or

```
public static void main (String args[]) {
```

- Now you know this says that main has a formal parameter **`args`** that is an array of **`String`**s

- The elements of this array are the whitespace-delimited *command line arguments* that may have been typed in on the command line when the Java program was launched. Example on next slide

Command line arguments: an example

```
public class CLA {
   // a program that echoes its command line arguments
   public static void main (String[] args) {
      for(int i=0; i<args.length; i++) {
         System.out.println( args[i] ) ;
      }
   }
}
```

```
-----------------------------------------------------
% java CLA one   two three-and-four
one
two
three-and-four
```

Returning arrays from methods

- In Java, a method can return an array. As for other reference types, a pointer to the array is returned; the array is not copied
- As an example, here is a method that takes a String as argument, and returns the chars in the String as an array:

```
public char[] stringToChars(String s) {
   // create the char array we will return
   char[] ch = new char[s.length()];

   // set elements of the char array as required
   for(int i=0; i<ch.length; i++) {
      ch[i] = s.charAt(i);
   }

   // return a pointer to the char array
   return ch;
}
```

- (There is actually an instance method of the String class that does this: `public char[] toCharArray()`)

Multidimensional arrays in Java

- An array is a sequence of variables, all of the same type
  - these reference type pointers or primitive values are the *elements* of the array
  - these elements can be of any type (primitive types, classes... even arrays!), but they have to be of the same type

- So, it is possible in Java to have an array whose elements are themselves pointers to arrays
  - this is called an array of arrays, or a multidimensional array

- It is possible to have arrays of arrays of arrays, and arrays of arrays of arrays of arrays, etc., etc.
  - we will concentrate on 2-dimensional arrays: arrays whose elements are arrays of primitive type values or of non-array class type pointers

Declaring 2-dimensional arrays in Java

- An 2-dimensional array is declared with a declaration statement of the form

```
<typename> <identifier>[][];
```
or
```
<typename>[][] <identifier>;
```
or
```
<typename>[] <identifier>[];
```

  - This declares `<identifier>` to be the name of an array whose elements are arrays that contain elements of type `<typename>`.

  - All of these are legal, but the second form is probably clearest and most readable...

  - Examples:

```
int arr[][];   // declares arr to be an array of arrays of ints
Button[][] button;  // button is an array of arrays of Buttons
boolean[] vals[];  // vals is an array of arrays of booleans
```

Creating 2-dimensional arrays in Java

- As with all objects in Java, a mere declaration statement never creates an array of arrays
- To create a 2-d array in Java, you use **new**, the type of elements in the array, and two bracketed integer expressions that specify the dimensions of the array:

```
new <typename> [ <len1> ] [ <len2> ]
```

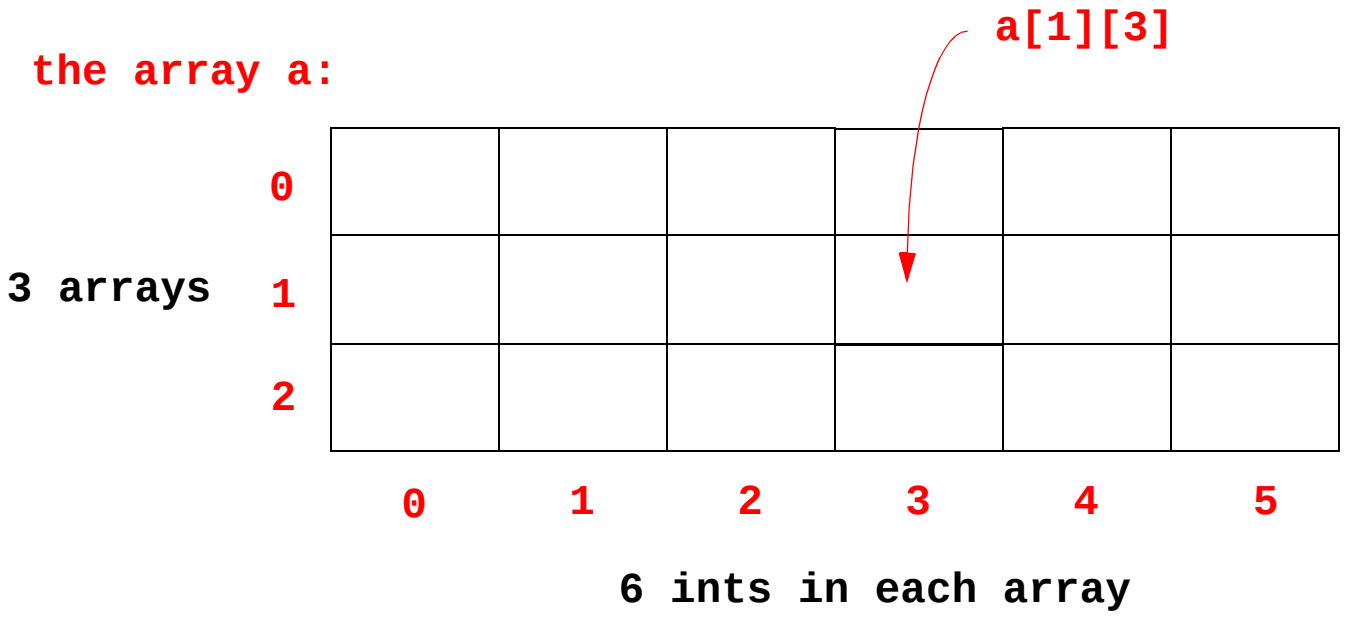
- **<len1>** is the number of arrays; **<len2>** is the number of elements in each array
- The elements of the arrays are automatically initialized to default values: zero for numerical types; **false** for boolean; **null** for class types
- Examples:

```
new int[25][4] // creates an array of 25 arrys of 4 ints each,
               // total 100 ints, all initially 0

new Button[10][10] // creates an array of 10 arrays of 10 Button
                   // pointers each, total 100 pointers, all
                   // null (this creates no Button objects!)

new boolean[40][50] // creates an array of 40 arrays of 50
                    // booleans each, total 2000 booleans,
                     // all initially false
```

Indexing 2-dimensional arrays

- You refer to an an element of a 2-d array by using the array name and two bracketed int indexing expressions
- The first index specifies which array, the second specifies the element of that array
- Convention: The first index indexes *rows* of the 2-d array, the second index indexes *columns*

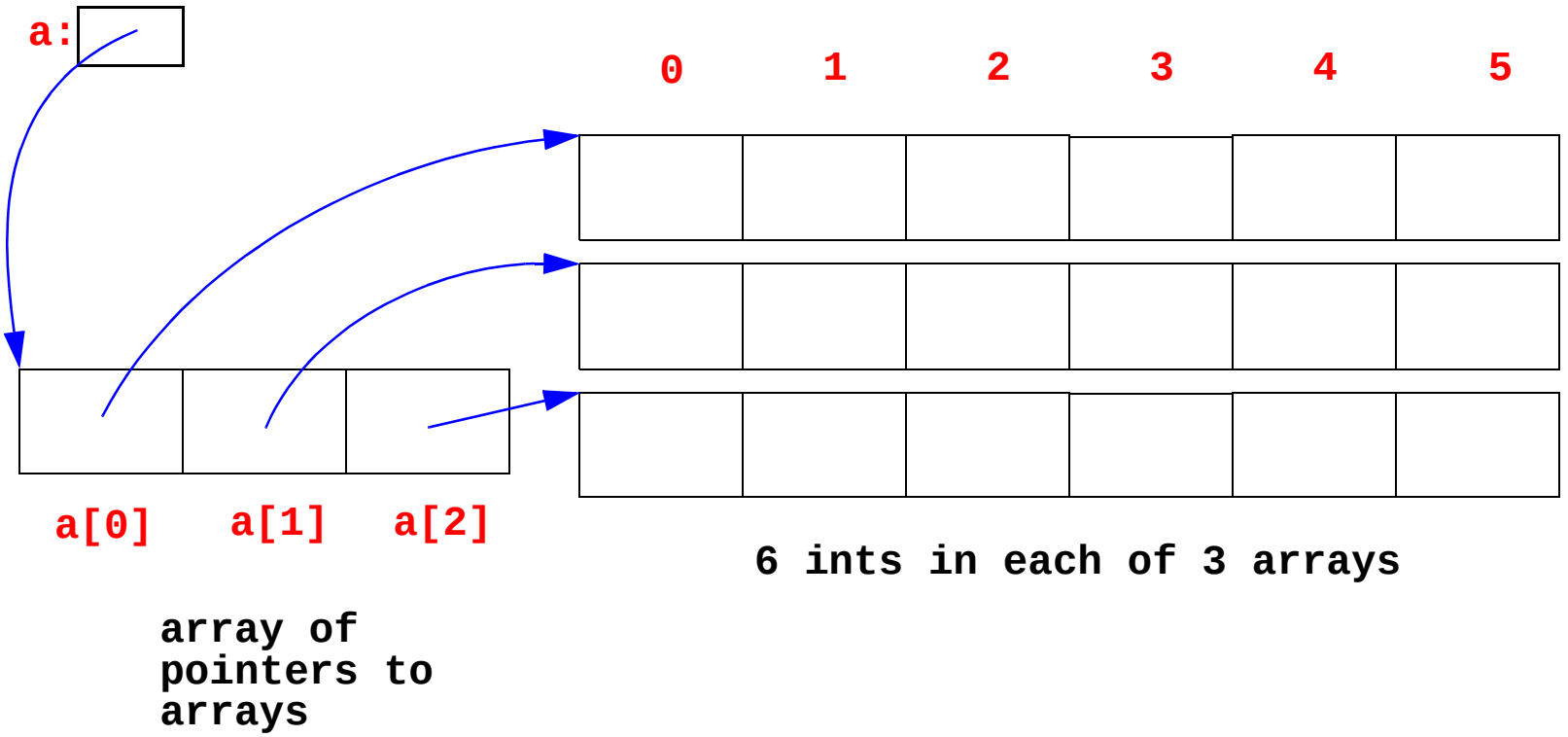
```
int[][] a = new int[3][6];   // a is an array of 3 arrays of 6 ints
```



Multidimensional arrays: a more detailed picture

- Creating an array of M arrays of N elements each actually creates M+1 arrays: one is an array of M pointers which are automatically initialized to point to the M N-element arrays. For example:

```
int[][] a = new int[3][6];   // a is an array of 3 arrays of 6 ints
```



Using multidimensional arrays in Java

- The index of an array element can be any expression with int value in the proper range
- As with 1-dimensional arrays, the use of any index value less than 0, or greater than or equal to the length of the array, is a runtime error

```
// create an array of 3 arrays of 6 ints each, named a
int a[][] = new int[3][6];

a  // A pointer of type int[][], i.e. array of arrays of ints
a.length  // what is the value of this expression?

a[0]  // this is the first element of a.  It is of type int[].
a[0].length  // what is the value of this expression?

a[0][0]  // this is the first element of the first element of  a.
        // It is an expression of type int.

a[3][0]  // out of bounds!
a[2][10]  // out of bounds!
a[1][-2]  // out of bounds!
```

Initializing multidimensional arrays in Java

- You can initialize array elements to non-default values
  - One way is to use a comma-separated list of values nested within braces instead of **new**.
  - If you do this, it must be done in the declaration statement for the array only.

```
// create a as an array of 2 arrays of 3 ints
int[][] a = {{1, 2, 3}, {4, 5, 6}};
```

the array a:

1	2	3
4	5	6

Initializing multidimensional arrays in Java, cont'd

- Of course, you can always set multidimensional array values explicitly, in a loop or one-by-one
- Nested for-loops can be handy for this.

```
int[][] a = new int[3][2];
for(int i=0; i<a.length; i++)
   for(int j=0; j<a[i].length; j++)
      a[i][j] = 2*i;
```

the array a:

0	0
2	2
4	4

Next time

- ✔ Comment translation
- ✔ Sorting
- ✔ Searching
- ✔ Partially filled arrays
- ✔ Parallel arrays
- ✔ Ragged and triangular arrays
- ✔ Arrays and the Java type system
- ✔ The Vector class

(Reading: Savitch, Ch. 6 and part of Ch 10)