For the pre-assignment. . .

The main method receives as a parameter an array of Strings from the command line.

```
public class SomeProgram {
   public static void main(String[] args) {
       System.out.println(args[0]);
       System.out.println(args[1]);
   }
}

...

> java SomeProgram ahoy aloha
ahoy
aloha
```

For the pre-assignment. . .

The String method `toCharArray` returns an array of chars equivalent to the String.

```
String str ="aloha";
char array = str.toCharArray();
System.out.println(array[0] + " " + array[1] + " " + array[2]);
```

...

```
a l o
```

CS 241 — Introduction to Problem Solving and Programming

Object-Oriented Programming

Arrays

Feb 23, 2005

Outline

- Review from last time: definition, syntax, etc
- How arrays are stored in memory
- Other issues with arrays
- Examples

Arrays

An array is an ordered collection of elements all of the same type.

Analogies:

- A composite type or way of formatting data
- A mathematical sequence
- A collection of data

Use of arrays

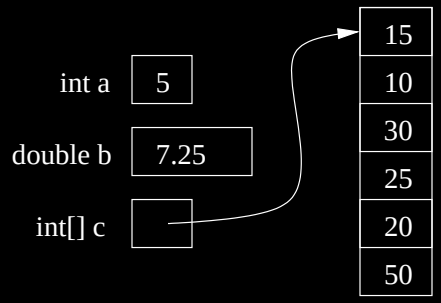
	Form	*Example*
Declaration:	*Base_Type*[ ] *Identifier*;	`int[] scores;`
Literal:	{ *E_0*, *E_1*, . . . *E_*$(n-1)$ }	`scores = { 10, 15, 20 }`
Indexed variable	*Identifier*[*Int_Expression*]	
as expression:		`int current = scores[1];`
in assignment:		`scores[1] = 30;`

Arrays in memory

Array variables are fundamentally different from other variables in how they are stored in memory.

Regular variables are containers for a value of their type.

Array variables are containers for a reference to a place in memory holding an array.



Arrays in memory: new

Create an array using the keyword `new`. (This is called creating or allocating an array dynamically, as opposed to using a literal, which would be statically.)

`new` *Base_Type*[*Int_Expression*]

```
int size = DocsIO.readint("How many scores? ");
int[] scores = new int[size];
for (int i = 0; i < size; i++)
    scores[i] = DocsIO.readint("Next score--> ");
```

When an array is created, its elements have default values; for ints, 0.

Arrays in memory: aliasing

Assigning an entire array to another array variable does not copy the array, but rather it copies the reference. The two variables then refer to the same array; a change to one is a change to the other.

```
int[] array1 = new int[5];
int[] array2 = array1;
array1[3] = 12;
System.out.println(array2[3]);
```

```
...

12
```

Arrays in memory: passing to methods

Passing an array to a method will copy the memory reference to the parameter, so changes in the method are changes to the original array.

```
        int[] array1 = new int[5];
        arrayMethod(array1);
        System.out.println(array1[3]);

...

    static void arrayMethod(int[] arr)  arr[3] = 12;

...

12
```

Null arrays

What if an array variable does not have an array in memory to point to? Then it is set to a value called null.

If you try to index on a variable that is null, you will get a `NullPointerException`. (Note this is a runtime error.)

```
int[] array1 = null;
System.out.println(array1[3]);
```

```
...

Exception in thread "main" java.lang.NullPointerException
        at SomeProgram.main(SomeProgram.java:10)
```

Length

Find the length of an array a using `a.length`. Notice, no parentheses. This is not a method.

```
for (int i = 0; i < scores.length; i++)
    System.out.println(scores[i]);
```

Out of bounds

If you try to use an index that does not exist, you will get an `ArrayIndexOutOfBoundsException`.

```
int[] array1 = new int[3];
System.out.println(array1[3]);
```

```
...

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at SomeProgram.main(SomeProgram.java:10)
```

Naming conventions

Our textbook says:

> Using a plural [name] seems to make sense, since the array holds more than one element. However . . . programs often reed better if they use a singular form, [because] when the array name is used in some sort of computation, the name refers only to one element.

```
int[] scores = new int[20];
...
System.out.println("The score is " + scores[2]);
```

or

```
int[] score = new int[20];
...
System.out.println("The score is " + score[2]);
```

Arrays as sequences and collections of data

Compute a section of the Fibonacci sequence. Use an array to represent/store the sequence so we can refer to an arbitrary element after it has been computed.

Write methods to compute the average, min, max, and range of a given set of data.

Arrays as sequences

```
static int[] makeFib(int n) {
    // An array to hold the sequence
    int[] fibSequence = new int[n];
    fibSequence[0] = 0;
    fibSequence[1] = 1;
    for (int i = 2; i < n; i++)
        fibSequence[i] = fibSequence[i-2] + fibSequence[i-1];
    return fibSequence;
}
```

Arrays as collections of data

```
static double average(int[] seq) {
    double sum = 0;   // The sum so far
    for (int i = 0; i < seq.length; i++)
        sum += seq[i];
    return sum / seq.length;
}
```

Arrays as collections of data

```
static int max(int[] seq) {
    int max = seq[0];   // The max so far
    for (int i = 1; i < seq.length; i++)
        if (max < seq[i])
            max = seq[i];
    return max;
}
```

Arrays as collections of data

```
static int min(int[] seq) {
    int min = seq[0];     // The min so far
    for (int i = 1; i < seq.length; i++)
        if (min > seq[i])
            min = seq[i];
    return min;
}
```

Arrays as structured data: time

Write a program to ask for two times (including date) and compare them. Represent a time as an array. Include methods to compare two times, ask the user to enter a time, and format a time nicely in a string.

Arrays as structured data: time

```
public static void main(String[] args) {
    System.out.println("First time");
    int[] time1 = enterTime();                // One time
    System.out.println("Second time");
    int[] time2 = enterTime();               // The other time
    String before = "";              // Blank or "not"
    if (!before(time1, time2))
        before = " not";
    System.out.println(toString(time1) + " is" + before + " before "
                       + toString(time2));
}
```

Arrays as structured data: time

```
static int[] enterTime() {
    int[] time = new int[6];
    time[0] = DocsIO.readint("Year? ");
    time[1] = DocsIO.readint("Month? ");
    time[2] = DocsIO.readint("Date? ");
    time[3] = DocsIO.readint("0=AM 1=PM? ");
    time[4] = DocsIO.readint("Hour? ");
    time[5] = DocsIO.readint("Minute? ");
    return time;
}
```

Arrays as structured data: time

```
static boolean before(int[] time1, int[] time2) {
    for (int i = 0; i < 6; i ++)
        if (i == 4) {
            if (time1[i] % 12 < time2[i] % 12)
                return true;
            else if (time1[i] % 12> time2[i] % 12)
                return false;
        }
        else {
            if (time1[i] < time2[i])
                return true;
            else if (time1[i] > time2[i])
                return false;
        }
    return false;
}
```

Arrays as structured data: time

```
static String toString(int[] time) {
    String toReturn = time[4] + ":" + time[5];
    if (time[3] == 0)
        toReturn += " AM";
    else
        toReturn += " PM";
    switch(time[1]) {
    case 1: toReturn += " January ";break;
    case 2: toReturn += " February ";break;
    case 3: toReturn += " March ";break;
    case 4: toReturn += " April ";break;
    case 5: toReturn += " May ";break;
    case 6: toReturn += " June ";break;
    case 7: toReturn += " July ";break;
    case 8: toReturn += " August ";break;
    case 9: toReturn += " September ";break;
    case 10: toReturn += " October ";break;
    case 11: toReturn += " November ";break;
    case 12: toReturn += " December ";break;
    }
    toReturn += time[2] + ", " + time[0];
    return toReturn;
}
```

Arrays as structured data: points

Write a program that will ask a user for two points and then plot them and calculate the distance. Represent a point with an array.

Arrays as structured data: points

```
public static void main(String[] args) {
    double[] point1 = new double[2];              // One point
    point1[0] = DocsIO.readdouble("x1: ");
    point1[1] = DocsIO.readdouble("y1: ");
    double[] point2 = new double[2];              // The other point
    point2[0] = DocsIO.readdouble("x2: ");
    point2[1] = DocsIO.readdouble("y2: ");
    plot(point1, point2);
    System.out.println("Distance: " + distance(point1, point2));
}
```

Arrays as structured data: points

```
static double distance(double[] point1, double[] point2) {
    double distance1 = point1[0] - point2[0];
    double distance2 = point1[1] - point2[1];
    return Math.sqrt((distance1 * distance1) +
                     (distance2 * distance2));
}
```

Summary

Understand the following concepts

- Array
- Array types
- Array literal
- Subscript / index
- Indexed variable
- `new`
- How arrays are stored
- `null` and null pointer exceptions
- Out of bounds exceptions