Arrays
Chris Piech
CS106A, Stanford University
What does this say?

Puzzle in Gold Bug by Edgar Allan Poe
## Changing Variable Types

### int to double?

```
int x = 5;
double xDbl = x;
```

### int to String?

```
int x = 5;
String xStr = "" + x
```

### String to int?

```
String xStr = "5"
int x = Integer.parseInt(x);
```

### String to double?

```
String xStr = "5.6"
double x = Double.parseDouble(xStr);
```

### Casting double to int

```
double x = 5.2;
int y = (int)x;
```

### GObject to GRect

```
GObject obj = getElementAt(5, 2);
GRect objRect = (GRect)obj;
```

### int to char

```
int diff = 'C'-'A';
char next = (char)'a' + diff;
```
Changing Variable Types

Enter arabic number: ๒๖
Western arabic translation: 42
Enter arabic number: ๒๖๐๐๐
Western arabic translation: 999000
Enter arabic number: ๒๖๐๐๐๐
Number Translation

Brāhmī

Indien (Gwālior)

Sanskrit-Devanāgarī (Indien)

Arabe occidental (Gobār)  Arabe oriental

Xle siècle (Apices)  XVle siècle (Dürer)

Piech, CS106A, Stanford University
Where are we?

- Karel the Robot
- Java
- Console Programs
- Graphics Programs
- Text Processing
- **Data Structures**
- Defining our own Variable Types
- GUIs

Piech, CS106A, Stanford University
A new variable type that is an object that represents an ordered, homogeneous list of data.

- Arrays have many *elements* that you can access using *indices*.

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>12</td>
<td>49</td>
<td>-2</td>
<td>26</td>
<td>5</td>
<td>17</td>
<td>-6</td>
<td>84</td>
<td>72</td>
<td>3</td>
</tr>
</tbody>
</table>

`#majorkey` of the day
Arrays Of Other Types

You can create arrays of any variable type. For example:

```java
double[] results = new double[5];
String[] names = new String[3];
boolean[] switches = new boolean[4];
GRect[] rects = new GRect[5];
```

- Java initializes each element of a new array to its default value, which is 0 for int and double, ‘\0’ for char, false for boolean, and null for objects.
You can create arrays of any variable type. For example:

```java
char[] oldSchoolString = new char[5];
```

- Java initializes each element of a new array to its `default value`, which is `0` for `int` and `double`, `'\0'` for `char`, `false` for `boolean`, and `null` for objects.
<table>
<thead>
<tr>
<th>Operation</th>
<th>Strings</th>
<th>Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make a new one</td>
<td>String str = &quot;abc&quot;;</td>
<td></td>
</tr>
<tr>
<td>Get length?</td>
<td>str.length()</td>
<td></td>
</tr>
<tr>
<td>Get element?</td>
<td>str.charAt(i)</td>
<td></td>
</tr>
<tr>
<td>Set element?</td>
<td>Not allowed</td>
<td></td>
</tr>
<tr>
<td>Loop?</td>
<td>for(int i = 0; i &lt; str.length(); i++)</td>
<td></td>
</tr>
</tbody>
</table>
# Data Structures

<table>
<thead>
<tr>
<th>Operation</th>
<th>Strings</th>
<th>Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make a new one</td>
<td>String <code>str = &quot;abc&quot;</code>;</td>
<td><code>int arr = new int[5];</code></td>
</tr>
<tr>
<td>Get length?</td>
<td><code>str.length()</code></td>
<td><code>arr.length</code></td>
</tr>
<tr>
<td>Get element?</td>
<td><code>str.charAt(i)</code></td>
<td><code>arr[i]</code></td>
</tr>
<tr>
<td>Set element?</td>
<td><em>Not allowed</em></td>
<td><code>arr[i] = 5;</code></td>
</tr>
<tr>
<td>Loop?</td>
<td><code>for(int i = 0; i &lt; str.length(); i++)</code></td>
<td><code>for(int i = 0; i &lt; arr.length; i++)</code></td>
</tr>
</tbody>
</table>

*note: there was previously a typo here*
Creating Arrays

```
type[] name = new type[length];
```

```
int[] numbers = new int[5];
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Java automatically initializes elements to 0.
• Like Strings, indices go from 0 to the array's length - 1.

```
for (int i = 0; i < 7; i++) {
    println(numbers[i]);
}
println(numbers[9]);       // exception
println(numbers[-1]);      // exception
```
name[index] = value;  // set element at index
\begin{itemize}
  \item Like Strings, indices go from 0 to the array's length - 1.
\end{itemize}

\begin{verbatim}
int[] numbers = new int[7];
for (int i = 0; i < 7; i++) {
    numbers[i] = i;
}
numbers[8] = 2;    // exception
numbers[-1] = 5;  // exception
\end{verbatim}

\begin{tabular}{|c|c|c|c|c|c|c|c|}
  \hline
  \textbf{index} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
  \textbf{value} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
  \hline
\end{tabular}
Q: What are the contents of numbers after executing this code?

```java
int[] numbers = new int[8];
numbers[1] = 3;
numbers[4] = 7;
numbers[6] = 5;
int x = numbers[1];
numbers[x] = 2;
numbers[numbers[numbers[4]]] = 9;
```

// 0 1 2 3 4 5 6 7
A. {0, 3, 0, 2, 7, 0, 5, 9}
B. {0, 3, 0, 0, 7, 0, 5, 0}
C. {3, 3, 5, 2, 7, 4, 5, 0}
D. {0, 3, 0, 2, 7, 6, 4, 4}
Similar to a String, you can get the length of an array by saying

```
myArray.length
```

Note that there are *no parentheses* at the end!

**Practice:**
- What is the index of the *last element* of an array in terms of its length?
- What is the index of the *middle element* of an array in terms of its length?
Just like with Strings, we can use an array’s length, along with its indices, to perform cool operations.
Just like with Strings, we can use an array’s length, along with its indices, to perform cool operations.

For instance, we can efficiently initialize arrays.

```java
int[] numbers = new int[8];
for (int i = 0; i < numbers.length; i++) {
    numbers[i] = 2 * i;
}
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
</tr>
</tbody>
</table>
Just like with Strings, we can use an array’s length, along with its indices, to perform cool operations.
For instance, we can read in numbers from the user:

```java
int length = readInt("# of numbers? ");
int[] numbers = new int[length];
for (int i = 0; i < numbers.length; i++) {
    numbers[i] = readInt("Elem " + i + ": ");
}
```
Just like with Strings, we can use an array’s length, along with its indices, to perform cool operations. Try it out! *sum up* all of an array’s elements.

```java
// assume that the user has created int[] numbers
int sum = 0;
for (int i = 0; i < numbers.length; i++) {
    sum += numbers[i];
}
println(sum);
```
Sometimes, we want to hardcode the elements of an array.

```java
int numbers = new int[7];
numbers[0] = 5;
numbers[1] = 32;
numbers[3] = 12;
...

// This is tedious!
```
Sometimes, we want to hardcode the elements of an array. Luckily, Java has a special syntax for initializing arrays to hardcoded numbers.

```
type[] name = { elements };
```

```
// Java infers the array length
int[] numbers = {5, 32, 12, 2, 1, -1, 9};
```
• An array’s length is **fixed**. You cannot resize an existing array:

```java
int[] a = new int[4];
a.length = 10; // error
```

• You cannot compare arrays with `==` or `equals`:

```java
int[] a1 = {42, -7, 1, 15};
int[] a2 = {42, -7, 1, 15};
if (a1 == a2) { ... } // false!
if (a1.equals(a2)) { ... } // false!
```

• An array does not know how to print itself:

```java
println(a1); // [I@98f8c4]
```
### Array Methods to the Rescue!

- The class `Arrays` in package `java.util` has useful methods for manipulating arrays:

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays.binarySearch(<code>array</code>, <code>value</code>)</td>
<td>returns the index of the given value in a sorted array (or &lt; 0 if not found)</td>
</tr>
<tr>
<td>Arrays.copyOf(<code>array</code>, <code>length</code>)</td>
<td>returns a new copy of array of given length</td>
</tr>
<tr>
<td>Arrays.equals(<code>array1</code>, <code>array2</code>)</td>
<td>returns <code>true</code> if the two arrays contain same elements in the same order</td>
</tr>
<tr>
<td>Arrays.fill(<code>array</code>, <code>value</code>);</td>
<td>sets every element to the given value</td>
</tr>
<tr>
<td>Arrays.sort(<code>array</code>);</td>
<td>arranges the elements into sorted order</td>
</tr>
<tr>
<td>Arrays.toString(<code>array</code>)</td>
<td>returns a string representing the array, such as &quot;[10, 30, -25, 17]&quot;</td>
</tr>
</tbody>
</table>
Arrays.toString accepts an array as a parameter and returns a string representation of its elements.

```java
int[] e = {0, 2, 4, 6, 8};
println("e is " + Arrays.toString(e));
```

Output:
```
e is [0, 14, 4, 6, 8]
```
Arrays as Parameters

• Arrays are just another variable type, so methods can take arrays as parameters and return an array.

```java
private int sumArray(int[] numbers) {
    ...
}
```

```java
private int[] makeSpecialArray(...) {
    ...
    return myArray;
}
```
• Arrays are just another variable type, so methods can take arrays as parameters and return an array.

• However, arrays are objects, so per A Variable Origin Story, an array variable box actually stores its location.

• This means changes to an array passed as a parameter affect the original array!
```java
public void run() {
    int[] numbers = new int[7];
    fillArray(numbers);
    println(Arrays.toString(numbers));
}

private void fillArray(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        arr[i] = 2 * i;
    }
}
```
Let’s write a method called `swapElements` that swaps two elements of an array. How can we do this?

What parameters should it take (if any)? What should it return (if anything)?

```java
private ??? swapElements(???) {
    ...
}
```
public void run() {
    int[] array = new int[5];
    ...
    swapElements(array[0], array[1]);
    ...
}

private void swapElements(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
}
public void run() {
    int[] array = new int[5];
    // Ints are primitives, so they are passed by value!
    // Their variable boxes store their actual values. So
    // changes to the parameter do not affect the original.
}

private void swapElements(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
}
public void run() {
    int[] array = new int[5];
    ...
    swapElements(array, 0, 1);
    ...
}

private void swapElements(int[] arr, int pos1, int pos2) {
    int temp = arr[pos1];
    arr[pos1] = arr[pos2];
    arr[pos2] = temp;
}
public void run() {
    int[] array = new int[5];
    swapElements(array, 0, 1);
}

private void swapElements(int[] arr, int pos1, int pos2) {
    int temp = arr[pos1];
    arr[pos1] = arr[pos2];
    arr[pos2] = temp;
}
Example: Reverse Array Program

```java
public void run() {
    int n = readInt("Enter number of elements: ");
    int[] intArray = createIndexArray(n);
    println("Forward: " + arrayToString(intArray));
    reverseArray(intArray);
    println("Reverse: " + arrayToString(intArray));
}
```

```
public String arrayToString(int[] array) {
    String str = "";
    for (int i = 0; i < array.length; i++) {
        if (i > 0) str += ", ";
        str += array[i];
    }
    return "[" + str + "]";
}
```

```
private void reverseArray(int[] array) {
    for (int i = 0; i < array.length / 2; i++) {
        swapElements(array, i, array.length - i - 1);
    }
}
```

```
private void swapElements(int[] array, int p1, int p2) {
    int temp = array[p1];
    array[p1] = array[p2];
    array[p2] = temp;
}
```

```
private int[] createIndexArray(int n) {
    int[] array = new int[n];
    for (int i = 0; i < n; i++) {
        array[i] = i;
    }
    return array;
}
```

```
Enter number of elements: 10
Forward: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Reverse: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```
A *cryptogram* is a puzzle in which a message is encoded by replacing each letter in the original text with some other letter. The substitution pattern remains the same throughout the message. Your job in solving a cryptogram is to figure out this correspondence.

One of the most famous cryptograms was written by Edgar Allan Poe in his short story “The Gold Bug.”

In this story, Poe describes the technique of assuming that the most common letters in the coded message correspond to the most common letters in English, which are E, T, A, O, I, N, S, H, R, D, L, and U.

Edgar Allan Poe (1809-1849)
Poe’s Cryptographic Puzzle

53‡‡ †305) ) 6*; 4826) 4‡•) 4‡) ; 806*; 48‡8 II 60) ) 85; 1‡( ; ‡*8 †83 (88) 5‡; 46 ( ; 88*96* ?; 8) ‡‡ ( ; 485) ; 5‡2: *‡ ( ; 4956*2 (5*4) 8 II 8*; 4069285); ) 6‡8) 4‡‡; 1(‡9; 48081; 8: 8‡ 1; 48‡85; 4) 485‡528806*81(‡9; 48; (88; 4 (‡34; 48) 4‡; 161; :188; ‡?;

AGFDDGASS6NTTH265HP0S6HOSTRO6NTHERDRT
6DSRATOE4TONEDEGEEEANDTH6RTERNH6N
UTBSNHORTSRASTANDB4N4RTHAM6NBRANCHRSRT
BNTHED55BAST6DSR5S5HOS4T55HT4NRERTBYR5
FTHBHEATSHRADASBBRSH6NBRONTTHKRTRTKR
QUCHTHER5HT66TYERTOUT

8 33
; 26
4 19
‡ 16
‡ 16
* 13
5 12
6 11
( 10
† 8
8 8
0 6
9 5
2 5
: 4
3 4
? 3
II 2
− 1
• 1

Piech, CS106A, Stanford University
The basic idea behind the program to count letter frequencies is to use an array with 26 elements to keep track of how many times each letter appears. As the program reads the text, it increments the array element that corresponds to each letter.

TWAS BRILLIG
To the code!