Changing Variable Types

**int to double?**

```java
int x = 5;
double xDb1 = x;
```

**int to String?**

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

**String to int?**

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

**String to double?**

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

**Casting double to int**

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

**GObject to GRect**

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

**int to char**

```java
int diff = 'C' - 'A';
char next = (char)\'a\' + diff;
```
Where are we?

• Karel the Robot
• Java
• Console Programs
• Graphics Programs
• Text Processing
• **Data Structures**
• Defining our own Variable Types
• GUIs
Winter is Coming!
Karel Wars

Karel Wars Episode VII:
Revenge of the SuperKarel

An evil SuperKarel is threatening to disturb the grid world once again.
What was once a peaceful planet has now been thrown into chaos.

By taking the turnRight() command hostage, SuperKarel has forced all other Karels to turnLeft() three times.

Thanks to Nick Troccoli for the awesome example
In Java, an array list is an abstract type used to store a linearly ordered collection of similar data values.

When you use an array list, you specify the type ArrayList, followed by the element type enclosed in angle brackets, as in ArrayList<String> or ArrayList<Integer>. In Java, such types are called parameterized types.

Each element is identified by its position number in the list, which is called its index. In Java, index numbers always begin with 0 and therefore extend up to one less than the size of the array list.

Operations on array lists are implemented as methods in the ArrayList class, as shown on the next slide.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list.size()</td>
<td>Returns the number of values in the list.</td>
</tr>
<tr>
<td>list.isEmpty()</td>
<td>Returns <code>true</code> if the list is empty.</td>
</tr>
<tr>
<td>list.set(i, value)</td>
<td>Sets the $i^{th}$ entry in the list to <code>value</code>.</td>
</tr>
<tr>
<td>list.get(i)</td>
<td>Returns the $i^{th}$ entry in the list.</td>
</tr>
<tr>
<td>list.add(value)</td>
<td>Adds a new value to the end of the list.</td>
</tr>
<tr>
<td>list.add(index, value)</td>
<td>Inserts the value before the specified index position.</td>
</tr>
<tr>
<td>list.remove(index)</td>
<td>Removes the value at the specified index position.</td>
</tr>
<tr>
<td>list.clear()</td>
<td>Removes all values from the list.</td>
</tr>
</tbody>
</table>
The Java **ArrayList** class is derived from an older, more primitive type called an **array**, which is a collection of individual data values with two distinguishing characteristics:

1. *An array is ordered.* You must be able to count off the values: here is the first, here is the second, and so on.
2. *An array is homogeneous.* Every value in the array must have the same type.

As with array lists, the individual values in an array are called **elements**, the type of those elements (which must be the same because arrays are homogeneous) is called the **element type**, and the number of elements is called the **length** of the array. Each element is identified by its position number in the array, which is called its **index**.
### list.size()
- Returns the number of values in the list.

### list.isEmpty()
- Returns `true` if the list is empty.

### list.set(i, value)
- Sets the \( i \)\(^{th} \) entry in the list to `value`.

### list.get(i)
- Returns the \( i \)\(^{th} \) entry in the list.

### list.add(value)
- Adds a new value to the end of the list.

### list.add(index, value)
- Inserts the value before the specified index position.

### list.remove(index)
- Removes the value at the specified index position.

### list.clear()
- Removes all values from the list.

### array.length

Arrays have fewer capabilities
Why use arrays?

- Arrays are built into the Java language and offer a more expressive selection syntax.
- You can create arrays of primitive types like `int` and `double` and therefore don’t need to use wrapper types like `Integer` and `Double`.
- It is much easier to create arrays of a fixed, predetermined size.
- Java makes it easy to initialize the elements of an array.
- Many methods in the Java libraries take arrays as parameters or return arrays as a result. You need to understand arrays in order to use those methods.
1. Fantastic style

2. You will be a better programmer if you understand your roots
What does this say?

53‡‡†305) ) 6*; 4826) 4‡•) 4‡); 806*; 48†8¶ 60) ) 85; 1‡( ; : ‡*8†83 (88) 5*†; 46 ( ; 88*96† ?; 8) *‡( ; 485) ; 5*†2: *‡( ; 4956*2 (5*−4) 8¶ 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; ‡?;
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*.

```
  index  0  1  2  3  4  5  6  7  8  9
     value 12 49 -2 26  5 17 -6 84 72  3
```

*length = 10*
Creating Arrays

```java
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.
Getting values

name[index]    // get element at index

• 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

<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>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
name[index] = value;  // set element at index
name[index] = value;  // set element at index

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

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

<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>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
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}
Arrays

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.
Getting “length”

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};
```
Limitations of Arrays

• 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]
```
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(array, value)</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(array, length)</td>
<td>returns a new copy of array of given length</td>
</tr>
<tr>
<td>Arrays.equals(array1, array2)</td>
<td>returns true if the two arrays contain same elements in the same order</td>
</tr>
<tr>
<td>Arrays.fill(array, value);</td>
<td>sets every element to the given value</td>
</tr>
<tr>
<td>Arrays.sort(array);</td>
<td>arranges the elements into sorted order</td>
</tr>
<tr>
<td>Arrays.toString(array)</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.

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

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

private 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;
}

Example: Reverse Array Program

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.
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.
To the code!
import acm.program.*;

/**
 * This program creates a table of the letter frequencies in a paragraph of input text terminated by a blank line.
 */

public class CountLetterFrequencies extends ConsoleProgram {

/* Private instance variables */
private int[] frequencyTable;

public void run() {
    println("This program counts letter frequencies.");
    println("Enter a blank line to indicate the end of the text.");
    initFrequencyTable();
    while (true) {
        String line = readLine();
        if (line.length() == 0) break;
        countLetterFrequencies(line);
    }
    printFrequencyTable();
}

/* Initializes the frequency table to contain zeros */
private void initFrequencyTable() {
frequencyTable = new int[26];
for (int i = 0; i < 26; i++) {
    frequencyTable[i] = 0;
}
}

/* Initializes the frequency table to contain zeros */
private void printFrequencyTable() {
// Print the frequency table
}

/* Initializes the frequency table to contain zeros */
private void countLetterFrequencies(String line) {
// Count the letter frequencies in the line
}

}
/* Counts the letter frequencies in a line of text */
private void countLetterFrequencies(String line) {
    for (int i = 0; i < line.length(); i++) {
        char ch = line.charAt(i);
        if (Character.isLetter(ch)) {
            int index = Character.toUpperCase(ch) - 'A';
            frequencyTable[index]++;
        }
    }
}

/* Displays the frequency table */
private void printFrequencyTable() {
    for (char ch = 'A'; ch <= 'Z'; ch++) {
        int index = ch - 'A';
        println(ch + " : " + frequencyTable[index]);
    }
}

/* Initializes the frequency table to contain zeros */
private void initFrequencyTable() {
    frequencyTable = new int[26];
    for (int i = 0; i < 26; i++) {
        frequencyTable[i] = 0;
    }
}

public class CountLetterFrequencies extends ConsoleProgram {
    public void run() {
        println("This program counts letter frequencies.");
        println("Enter a blank line to indicate the end of the text.");
        initFrequencyTable();
        while (true) {
            String line = readLine();
            if (line.length() == 0) break;
            countLetterFrequencies(line);
        }
        printFrequencyTable();
    }
}