Arrays

Scalar vs. Array Variables

- Variables we have seen and used so far can hold only one primitive value or object at a time
  - We call these scalars
- Sometimes we need to store multiple data elements under one name
- Java arrays allow us to store a collection of data
- Arrays in Java are objects with fields and methods

Declaring Arrays

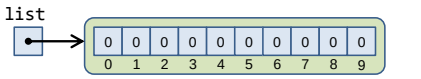
- Arrays store homogeneous data
- Square brackets [ ] indicate an array:

```
int[] list;       //  An array of integers

double[] nums;    //  An array of doubles

String[] words;   //  An array of strings
```

Creating an Array

- An array is a reference type
  - Must be created with new

```
int[] list;
list = new int[10];  //  list holds 10 integers
```



- An array's size is fixed once it is created
  - The reference can be reassigned, though

Array Length

- Every array has a public final length field which stores the number of elements in the array

```
int[] list = new int[100];
System.out.println (list.length);
```

Initializing Array Contents

```
//  Initializer list
int[] list = new int[] { 2, 4, 6 };

//  More simply
int[] list = { 2, 4, 6 };
```

Only possible at the point of declaration

Accessing an Element

- `[]` operator accesses an element within an array
- Beginning index is zero
- `list[2]` is the third element
- Display all the elements in array named `list`:

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

- Do **not** attempt to access an element at an index less than zero or greater or equal to the array's size!
  - Results in a run-time exception

"For each" for Simpler Iteration

- Display all the elements in array `list`:

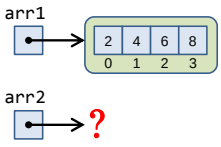
```
for (int i = 0; i < list.Length; i++) {
    System.out.println(list[i]);
}
```

- Alternate loop:

```
for (int elem : list) {
    System.out.println(elem);
}
```

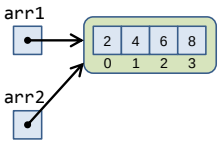
Array Aliasing

```
int[] arr1 = { 2, 4, 6, 8 },
      arr2;
```



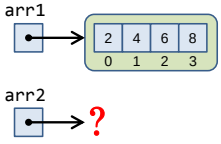
Array Aliasing

```
int[] arr1 = { 2, 4, 6, 8 },
      arr2;
arr2 = arr1;
```



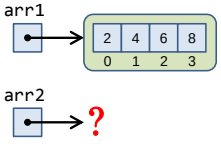
Copying an Array

```
int[] arr1 = { 2, 4, 6, 8 },
      arr2;
```



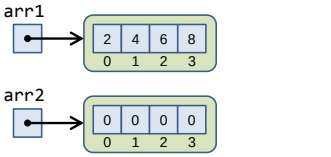
Copying an Array

```
int[] arr1 = { 2, 4, 6, 8 },
      arr2;
arr2 = new int[arr1.length];
```



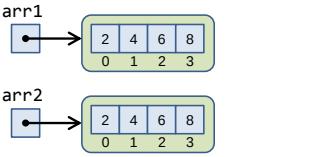
Copying an Array

```
int[] arr1 = { 2, 4, 6, 8 },
      arr2;
arr2 = new int[arr1.length];
for (int i = 0; i < arr2.length; i++) {
    arr2[i] = arr1[i];
}
```



Copying an Array

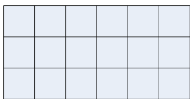
```
int[] arr1 = { 2, 4, 6, 8 },
      arr2;
arr2 = new int[arr1.length];
for (int i = 0; i < arr2.length; i++) {
    arr2[i] = arr1[i];
}
```



Multidimensional Arrays

- A one-dimensional (1-D) array has a length (think line)

- A two-dimensional (2-D) array has a length and width (think area)



- Higher-dimensional arrays (3-D+) are possible but rarely used

Sorting

- A common task: arrange elements of an array into ascending (or descending) order
- Many different sorting algorithms exist
  - Simple (slower) vs. complex (faster) sorts
- **Selection sort** is a simple to understand, simple to implement sorting algorithm
- **Quicksort** is the fastest known general-purpose sorting algorithm

Search

- Linear search
  - Start at first element
  - Examine all the elements in turn until you find the one you seek
- Binary search
  - Elements must be in sorted order
  - Check middle element
    - If it is the sought element, you are done
    - If the middle element is larger than the sought element, search the front half of the array
    - Otherwise, search the back half of the array
  - Much faster than linear search in general