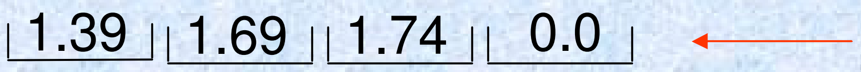
Arrays vs. ArrayList

- Compare and Contrast
- How to use each
- Example

What is an Array

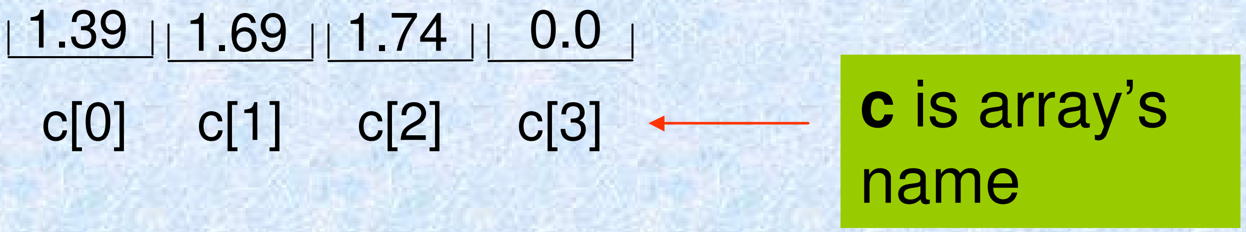
- An array is a block of consecutive memory locations that hold values of the same data type.
- Individual locations are called array's *elements*.
- When we say "element" we often mean the value stored in that element.



An array of **doubles**

What is an Array (cont'd)

- Rather than treating each element as a separate named variable, the whole array gets one name.
- Specific array elements are referred to by using array's name and the element's number, called *index* or *subscript*.



Arrays (built-in) vs. ArrayList

Array: A data structure that is a part of the Java language (built-in).

Holds a collection of the same data type.

ArrayList: A class that emulates an array.

Not part of the Java language, but is part of the library of classes that we can use.

Compare and Contrast

Array

- Can hold any type of data

- Once it is constructed, it is a fixed size.

```
Object [] a = new Object[100];

int [] b = new int[20];

Bug [] c = new Bug[17];
```

ArrayList

- Can hold only objects (no int, double, char, …)

- Can grow and shrink dynamically during run-time.

```
ArrayList<Object> a =
       new ArrayList<Object>(100);
// cannot do!!

ArrayList<Bug> c =
       new ArrayList<Bug>(17);
```

Array

- Access to data done directly.

ArrayList

- Access to data through methods of the class. (Gold sheet!!!)

```
Object x = a[i];
```

```
Object x = a.get(i);
```

```
Object y = new Object();
a[k] = y;
```

```
Object y = new Object();
a.set(i, y);
```

```
if(b[13] == 42)
{
        …
}
```

```
// Not possible!!
```

```
if(c[i].equals(c[i+1]))
{
        return true;
}
```

```
if(c.get(i).equals(c.get(i+1)))
{
        return true;
}
```

Overview of the “for each” loop

Purpose

The basic *for* loop was extended in Java 5 to make iteration over arrays and other collections more convenient.

This newer *for* statement is called the *enhanced for* or *for-each* (because it is called this in other programming languages).

Comparison of the “for each” loop

```
// Syntax of traditional for loop
for (int i = 0; i < arr.length; i++)
{
        type var = arr[i];
        body-of-loop
}
```

```
// Syntax of for each loop
for (type var : arr)
{
    body-of-loop
}
```

Example of the “for each” loop

```
// Syntax of traditional for loop
double[ ] ar = {1.2, 3.0, 0.8};
int sum = 0;
for (int i = 0; i < ar.length; i++)
{         // i indexes each element successively.
          sum += ar[i];
}
```

```
// Syntax of for each loop
double[ ] ar = {1.2, 3.0, 0.8};
int sum = 0;
for (double d : ar)
{         // d gets successively each value in ar.
          sum += d;
}
```

Drawbacks of the "for each" loop

1. **Only access. Elements can not be assigned to or removed from collection.**

2. **Only single structure. It's not possible to traverse two structures at once, eg, to compare two arrays.**

3. **Only single element. Use only for single element access, eg, not to compare successive elements.**

4. **Only forward. It's possible to iterate only forward by single steps.**

5. **At least Java 5. Don't use it if you need compatibility with versions before Java 5.**

Example: JavaBat Array1 and Array2