CS1001

Lecture 14

Overview

- Java Programming
- Arrays

Goals

- Understand the basics of Java programming
- Control Statements and Arrays

Assignments

- Brookshear: Ch 4, Ch 5 (Read)
- Read linked documents on these slides (slides will be posted in courseworks)

Arithmetic

- Operators: `+`, `-`, `/`, `*` , `%`
- The precedence of operators and parentheses work the same way as in algebra.
- **m % n** means the remainder when **m** is divided by **n** (e.g. 17 % 5 is 2).
- **%** has the same rank as `/` and `*`
- Same-rank binary operators are performed in order from left to right.

Arithmetic (cont'd)

- The type of the result is determined by the types of the operands, not their values; this rule applies to all intermediate results in expressions.
- If one operand is an int and another is a double, the result is a double; if both operands are ints, the result is an int.

Arithmetic (cont'd)

- Caution: if a and b are ints, then a / b is truncated to an int...

```
17 / 5  gives 3
 3 / 4  gives 0
```

- ...even if you assign the result to a double:

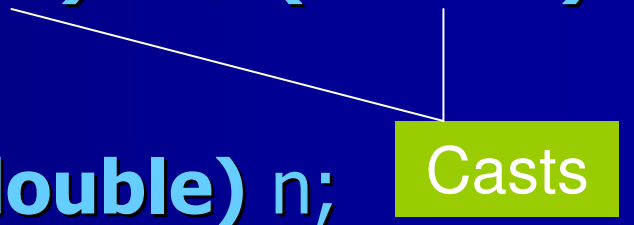
```
double ratio = 2 / 3;
```

The double type of the result doesn't help: ratio still gets the value 0.0.

Arithmetic (cont'd)

- To get the correct double result, use double constants or the *cast* operator:

```
double ratio = 2.0 / 3;

double ratio = 2 / 3.0;

double factor = (double) m / (double) n;

double factor = m / (double) n;

double r2 = k / 2.0;

double r2 = (double) k / 2;
```



Arithmetic (cont'd)

- Caution: the range for ints is from -2^{31} to $2^{31}-1$ (about $-2 \cdot 10^9$ to $2 \cdot 10^9$)
- Overflow is not detected by the Java compiler or interpreter

```
n =  8   10^n =  100000000   n! =       40320
n =  9   10^n = 1000000000   n! =      362880
n = 10   10^n = 1410065408   n! =     3628800
n = 11   10^n = 1215752192   n! =    39916800
n = 12   10^n = -727379968   n! =   479001600
n = 13   10^n = 1316134912   n! = 1932053504
n = 14   10^n =  276447232   n! = 1278945280
```

Arithmetic (cont'd)

- Use compound assignment operators:

```
a = a + b;  ⟶ a += b;
a = a – b;  ⟶ a –= b;
a = a * b;  ⟶ a *= b;
a = a / b;  ⟶ a /= b;
a = a % b;  ⟶ a %= b;
```

- Use increment and decrement operators:

```
a = a + 1;  ⟶ a++;
a = a – 1;  ⟶ a––;
```

Do not use these in larger expressions

Review:

- What is a variable?
- What is the type of variable that holds an object?

Review (cont'd):

- What is the range for ints?
- When is a cast to double used?
- Given

```
double dF = 68.0;
double dC = 5 /  9 * (dF – 32);
```

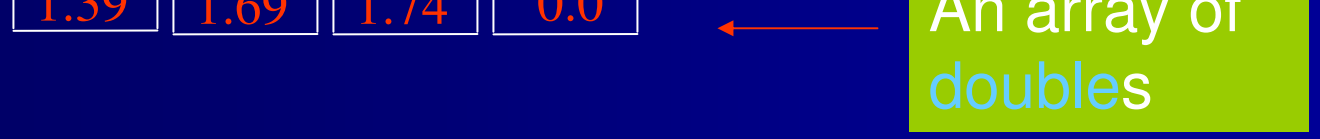
  what is the value of dC?
- When is a cast to int used?
- Should compound assignment operators be avoided?

Objectives:

- Learn about arrays and when to use them
- Learn the syntax for declaring and initializing arrays
- Learn how to access array's size and elements

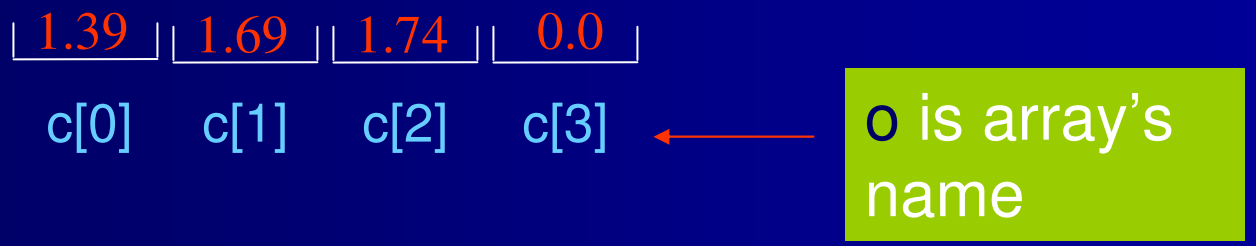
What is an Array

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



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***.



Indices (Subscripts)

- In Java, an index is written within square brackets following array's name (e.g., a[k]).
- Indices start from 0; the first element of an array a is referred to as a[0] and the *n*-th element as a[n-1].
- An index can have any int value from 0 to array's length – 1.

Indices (cont'd)

- We can use an int variable or any expression that evaluates to an int value as an index:

```
a [3]
a [k]
a [k – 2]
a [ (int) (6 * Math.random()) ]
```

Indices (cont'd)

- In Java, an array is declared with fixed length that cannot be changed.
- Java interpreter checks the values of indices at run time and throws IndexOutOfBoundsException if an index is negative or if it is greater than the length of the array – 1.

Why Do We Need Arrays?

- The power of arrays comes from the fact that the value of a subscript can be computed and updated at run time.

Before (no arrays):

```
int sum = 0;
sum += score0;
sum += score1;
…
sum += score999;
```

1000 times!

After (with arrays):

```
int n = 1000;
int sum = 0, k;

for (k = 0;  k < n;  k++)
    sum += scores[k];
```

Why Arrays? (cont'd)

- Arrays give ***direct access*** to any element — no need to scan the array.

Before (no arrays):

```
if (k == 0)
    display (score0);
else if (k == 1)
    display (score1);
else
…  // etc.
```

After (with arrays):

```
display (scores[k]);
```

Arrays as Objects

- In Java, an array is an object. If the type of its elements is anyType, the type of the array object is anyType[ ].
- There are two ways to declare an array:

```
anyType [ ]  arrName;
```

or

```
anyType  arrName [ ];
```

The difference becomes significant only when several variables are declared in one statement:

```
int [ ] a, b;   // both a, b are arrays
int a [ ], b;   // a is an array, b is not
```

Arrays as Objects (cont'd)

- As with other objects, the declaration creates only a reference, initially set to null. An array must be created before it can be used.
- There are two ways to create an array:

arrName = new *anyType* [length] ;

Brackets, not parens!

or

arrName = new *anyType* [ ] { val1, val2, …, valN };

Declaration and Initialization

- When an array is created, space is allocated to hold its elements. If a list of values is not given, the elements get the default values.

```
scores = new int [10] ;
                    // length 10, all values set to 0

words = new String [10000];
                   // length 10000, all values set to null
```

Initialization (cont'd)

- An array can be declared an initialized in one statement:

```
int scores [ ] = new int [10] ;    // length 10

private double gasPrices [ ] = { 1.49, 1.69, 1.74 };

String words [ ] = new String [10000];

String cities [ ] = {"Atlanta", "Boston", "Cincinnati" };
```

Initialization (cont'd)

- Otherwise, initialization can be postponed until later:

```
String words [ ];       // not yet initialized
...
words = new String [ console.readInt() ];

private double gasPrices [ ];     // not yet initialized
…
gasPrices = new double [ ]  { 1.52, 1.69, 1.75 };
```

Array's Length

- The length of an array is determined when that array is created.
- The length is either given explicitly or comes from the length of the {…} initialization list.
- The length of an array arrName is referred to in the code as arrName.length.
- length appears like a public field (not a method) in an array object.

Review:

- Why are arrays useful?
- What types of elements can an array have?
- How do we refer to an array's element in Java?
- What happens if an index has an invalid value?
- How do we refer to the length of an array?

Sorting

```
procedure Sort (List)
N ← 2;
while (the value of N does not exceed the length of List) do
    (Select the Nth entry in List as the pivot entry;
     Move the pivot entry to a temporary location leaving a hole in List;
     while (there is a name above the hole and that name is greater than the pivot) do
         (move the name above the hole down into the hole leaving a hole above the name)
     Move the pivot entry into the hole in List;
     N ← N + 1
    )
```

Alternate Sorting Algorithm (Selection Sort)

```
main(String[] args) {
    int inputSize = args.length;
    int[] sortedArray = new int[inputSize];
    for (i=0; i < inputSize; i++) {
        (1) Loop through args, find largest element and
        remember its index (ie for (int j = 0; …))
        (2) Put the largest element into the ith location of
    sortedArray;
        (3) Delete (set to -1) the ith element of args
}
```

http://www.ee.unb.ca/brp/lib/java/selectionsort/