Arrays

CS111 Lecture 17

Tuesday, April 4, 2000

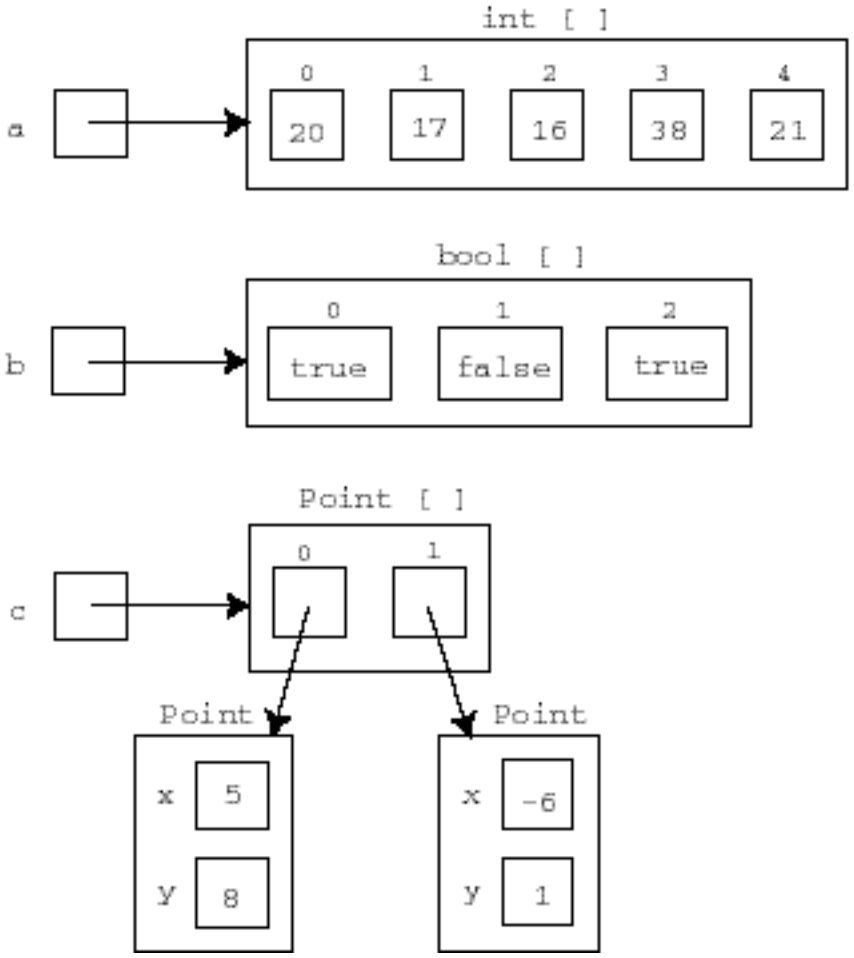
Arrays: Motivation

- Lists are great for representing sequences, but can only efficiently access elements in order from front to back.
- We often want an indexed sequence where we can efficiently access an element at any index.

Java Arrays: Summary of Properties

- **Fixed-length** indexed sequences of elements. Once created, the size of an array cannot change.
- Elements are **0-indexed** (i.e., indices start at 0, not 1).
- **Constant time access** to an element at any index.
- Arrays are **homogeneous** collections: all elements must have the same type. (No need for type casts seen in Object lists.)
- **Special concise syntax** for creating, accessing and updating arrays.
- **Dynamic (run-time) index checking** to ensure that indices of all array operations are in bounds. (The lack of such checking in C makes C particularly unsafe.)
- Arrays are objects that are **not instances of any class**.

Array Diagrams



Array Operations

- Return contents of variable an index:

```
a[2] → 17
if (b[1]) { then part} else {else part}
c[0].x → 5
```

Notes:

- ♦ a[2] is pronounced “a sub 2”
- ♦ `ArrayIndexOutOfBoundsException` for indices out of range

- Update contents of variable at an index:

```
a[0] = a[1] + a[2]; (* a[0] is now 33 *)
```

- Return length of array (number of slots, **not** highest index)

```
a.length → 5
b.length → 3
c.length → 2
```

Array Creation

- Arrays are created using the syntax **new** *type* [*size*]. E.g:

```
int [] a = new int[5]
```

- Slots in a new array contain a default value (0 for integer arrays, false for boolean arrays; the **null pointer** for object arrays).
- Must explicitly initialize slots to contain values other than default:

```
a[0] = 20;
a[1] = 17;
a[2] = 16;
a[3] = 38;
a[4] = 21;
```

- The {} syntax simplifies initialization. In Java 1.0.2, it can only be used when declaring an array variable:

```
int [] a = {20, 17, 16, 33, 21};
bool [] b = {true, false, true};
Point [] c = {new Point(5,8), new Point(-6,1)};
```

Arrays of Arrays

Arrays can contain other arrays as their elements. E.g.:

picture will be drawn in class

Array Functions: Summation

Typically manipulate arrays using for loops (but can also use while loops, tail recursion, non-tail recursion). E.g:

```
// Return the sum of all elements in integer array a.
public static int sum (int [ ] a) {
   int sum = 0;
  for (int i = 0; i < a.length; i++) {
     sum = sum + a[i];
   }
   return sum;
}
```

Array Functions: Doubling

```
// Double all of the elements in array a.
public static void doubleAll (int [ ] a) {
  for (int i = 0; i < a.length; i++) {
    a[i] = 2 * a[i];
  }
}
```

Array Functions: MinIndex

```
// Return the index of the minimum integer in the array, or -1 if the array
// is empty.
public static int minIndex (int [ ] a) {
  int minVal = Integer.MAX_VALUE;
  int minInd = -1;
  for (int i = 0; i < a.length; i++) {
    if (a[i] < minVal) {
      minVal = a[i];
      minInd = i;
    }
  }
return minInd;
}
```

Array Functions: minIndexBetween

```
// Return the index of the minimum integer in a[lo..hi],
// or -1 if the segment is empty.
public static int minIndexBetween (int [ ] a, int lo, int hi)
{
  int minVal = Integer.MAX_VALUE;
  int minInd = -1;
  for (int i = lo; i <= hi; i++) {
    if (a[i] < minVal) {
      minVal = a[i];
      minInd = i;
    }
  }
  return minInd;
}
```

Array Functions: Swap

```
// Swap the contents of a[i] and a[j]
public static void swap (int [ ] a, int i, int j) {
  int temp = a[i];
  a[i] = a[j];
  a[j] = temp;
}
```

Array Functions: Selection Sort

```
public static void selectionSort (int [ ] a) {
   int hi = a.length - 1;
   // Loop invariant:
   // All elements in a[0..i-1] are in their final sorted positions.
   for (int i = 0; i < hi; i++) {
      swap(a, i, minIndexBetween(a, i, hi));
   }
   // Last slot is not visited. Can you explain why?
}
```

Array Functions: Insertion

```
// Assume a[lo..hi-1] is in sorted order.
// Insert a[hi] into the segment a[lo..hi] in such a way to make it sorted.
public static void insertBetween(int [ ] a, int lo, int hi) {
   int val = a[hi];
   int index = hi;
   // Loop invariants:
   // All elements in a[lo..index-1] are sorted.
   // All elements in a[index+1..hi] are sorted and are >  val.
   while ((index > lo)
          && (a[index-1] > val)) { // Order of tests is crucial!
      a[index] = a[index-1]; // Shift value right = shift hole left.
      index = index - 1;
   }
  // Here, either (index == lo) or ((index > lo) && (a[index-1] <= val))
   a[index] = val;
}
```

Array Functions: Insertion Sort

```
public static void insertionSort (int [ ] a) {
    // Loop invariant:
    // All elements in a[0..i-1] are in sorted order.
    for (int i = 1; i < a.length; i++) {
        insertBetween(a, 0, i);
    }
}
```