Lecture 18

Arrays

Wednesday, April 4, 2012
4-6 PM in Duffield Atrium

Announcements for Today

Reading

- Sections 8.1 – 8.3
- PLive Lessons 7.5, 7.6

- **Prelim, April 17th 7:30-9:30**
  - Material up to next class
  - Review posted this weekend
  - Not the same as previous years
- **Conflict with Prelim time?**
  - Submit to Prelim 2 Conflict assignment on CMS
  - Do not submit if no conflict

Assignments

- A5 is due Thursday night
  - Keep reading Piazza
  - Should have worked on a method a day
  - Cannot give extensions
- A6 posted on Thursday
  - Get started immediately!
  - Prelim is same week it is due
  - If you get started right away, you will not have problems

Arrays

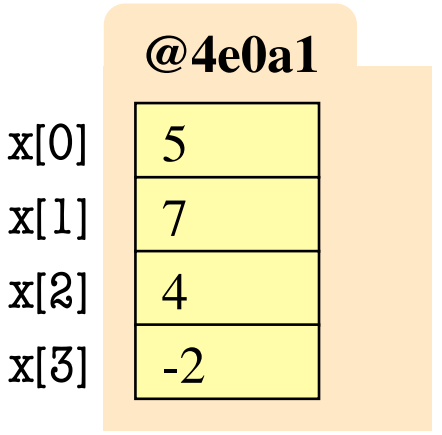
- **Array**: an object that holds a fixed number of values of the same type.
- Type of an array is written:

  <type>[] (e.g. **int**[])
- Declare a variable `x` that holds the name of an array of **int**s:

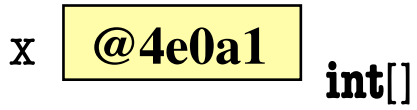
  <type> <name>; (e.g., **int**[] x;)
- Elements of array `x` are numbered:

  0, 1, 2, …, $n - 1$
- To refer to an element of an array:

  <var>[*<index>*] (e.g. x[3])



This array contains 4 values of type **int**

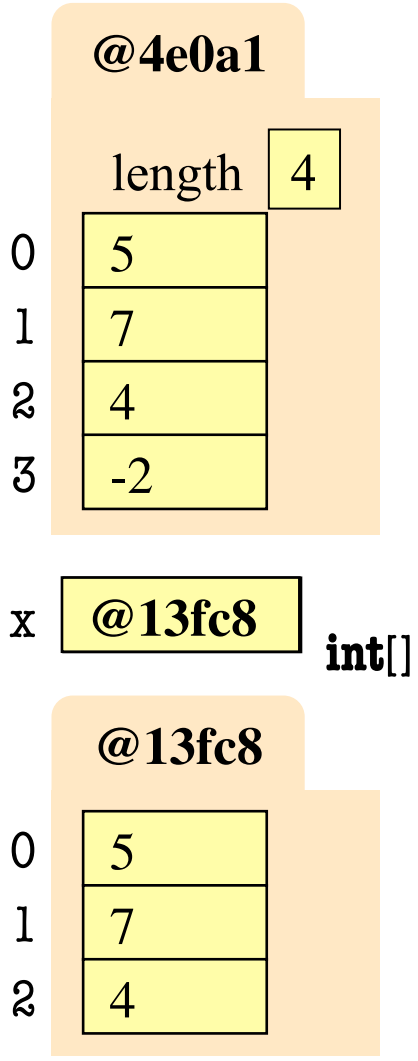


Arrays

- Array length is a field of the object

  x.length [**not** x.length()]
- The length field is **final**: it never changes after the array is created
- Length is not part of the array type
  - An **int**[] variable can be hold arrays of different lengths at different times
- Declaring x does not create array
  - As an object it starts out **null**
  - Need a special new-expression:

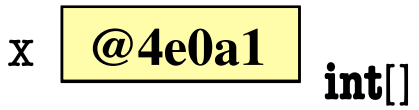
    new <*type*>[<*length*>]
    (e.g. x = **new int**[3];)



Overview of Array Syntax

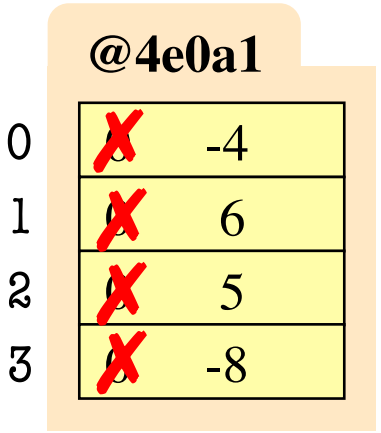
- int[] x;

Create a variable named **x** to hold an **int**[] value



- x = new int[4];

Create array object of length 4; put name in **x**

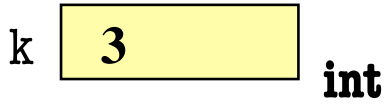


- x[2] = 5;
- x[0] = –4;

Assign 5 to element 2 and –4 to element 0

- int k = 3;
- x[k] = 2 * x[0];
- x[k–2] = 6;

Assign –8 to x[3] and 6 to x[1]



Arrays vs. Vectors vs. Strings

- **Declaration**

  `int[] a;`

  (contains **int**s)
- **Creation**

  `a = new int[n];`

  (size fixed forever)
- **Reference**

  `x = a[i];`
- **Change**

  `a[i] = x;`

Variables a[0], a[1], … are at successive locations in memory. Element type can be class or primitive type.

- **Declaration**

  `Vector<Integer> v;`

  (contains Integers)
- **Creation**

  `v= new Vector<Integer>();`

  (can be resized at will)
- **Reference**

  `x= v.get(i);`
- **Change**

  `v.set(i, x);`

Storage layout unspecified (but really, it is an array). Element type can only be a *class* type.

- **Declaration**

  `String s;`

  (contains **char**s)
- **Creation**

  `s = “foo”;`

  (contents fixed forever)
- **Reference**

  `c = s.charAt(i);`
- **Cannot Change**

Storage layout unspecified (but really, it is an array) Element type is always **char**.

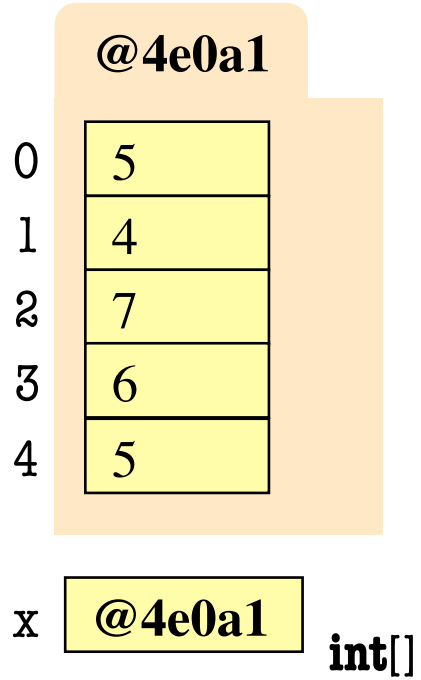
Array Initializers

- Initializing a newly created array:
  - `int[] c= new int[5];` ← create array of 5 ints initialized with default (0)
  - `c[0]= 5; c[1]= 4; c[2]= 7; c[3]= 6; c[4]= 5;` ← assign new values to elements
- Instead, use an array initializer: ← create array of 5 ints and initialize all elements
  - `new int[] { 5, 4, 7, 6, 5 }`

    no size goes here (implied by length of initializer list)

    types must agree with array's type
- In a declaration, short form is available:
  - `int[] c;`
    `c= new int[] { 5, 4, 7, 6, 5 };`
  - `int[] c= new int[] { 5, 4, 7, 6, 5 };`
  - `int[] c= { 5, 4, 7, 6, 5 };`

  all three do the same thing



Array Initialization Example

```
public class ArrayDemo {
  public static final String[] months=
     new String[]{"January","February","March","April","May",
                  "June","July","August","September","October",
                  "November","December"};
    /**Yields: the month name, given its number m
     *  Precondition: 1 <= m <= 12 */
    public static String theMonth(int m) {
        return months[m-1];
    }
}
```

e.g. ArrayDemo.theMonth(4)
returns months[3], or "April".

Variable months is:

static: object assigned is created only once

public: can be seen outside class ArrayDemo

final: it cannot be changed once initialized

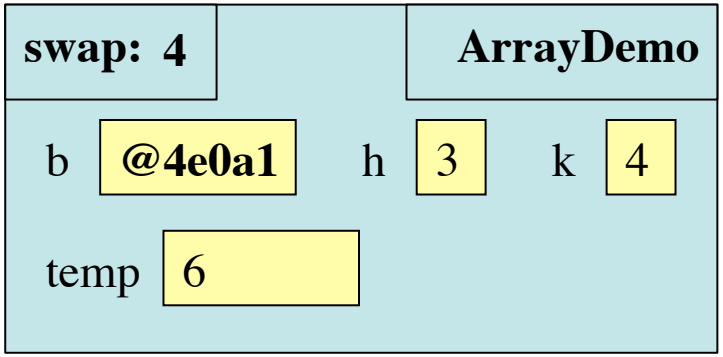
Procedure: Swap

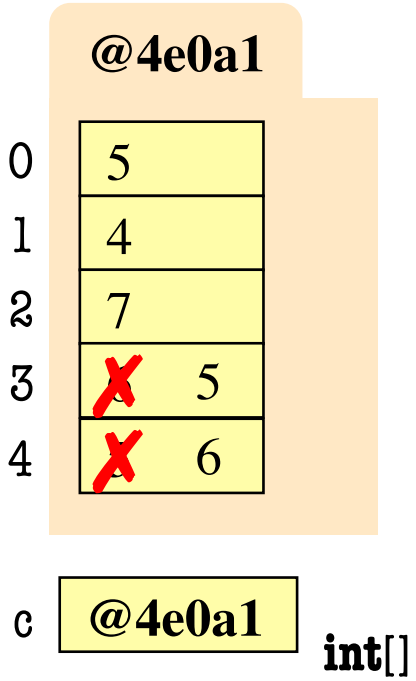
```
public class ArrayDemo {
   /** Procedure swaps b[h] and b[k] in b */
   public static void swap (int[] b, int h, int k) {
      int temp= b[h];
      b[h]= b[k];
      b[k]= temp;
   }
}

...

swap(c, 3, 4);
```

Swaps b[h] and b[k], because parameter b contains name of array.



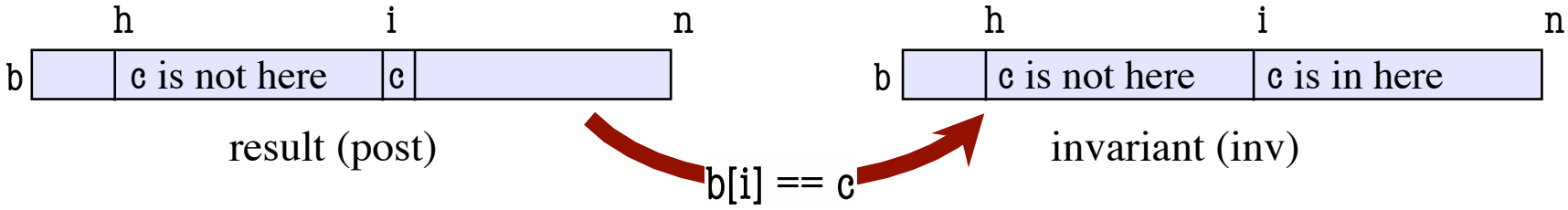


Array Algorithm: Linear Search

```
/** Yields: index of first occurrence of c in b[h..]
 *  Precondition: c is in b[h..] */
public static int findFirst(int c, int[] b, int h) {
    // Store in i the index of the first c in b[h..]
    int i= h;
    // inv: c is not in b[h..i–1]
    while (b[i] != c) {
        i= i + 1;
    }
    // post: b[i] == c and c is not in b[h..i-1]
    return i;
}
```

Analyzing the Loop

1. Does the initialization make **inv** true?
2. Is **post** true when **inv** is true and **condition** is false?
3. Does the repetend make progress?
4. Does the repetend keep **inv** true?



Array Algorithm: Loaded Dice

```
/**  Yields: a random int in 0..p.length-1; i is returned with probability p[i].
 *   Precondition: the entries of p are positive and sum to at least 1. */
public static int roll(double[] p) {
  double r= Math.random(); // r in [0,1)
  // Think of interval [0,1] as divided into segments of size p[i]
  // Store into i the segment number in which r falls.
  int i= 0;  double pEnd= p[0];
  // inv: r >= sum of p[0] .. p[i–1]; pEnd = sum of p[0] .. p[i]
  while ( r >= pEnd ) {
    pEnd= pEnd + p[i+1];
    i= i + 1;
  }
  // post: sum of p[0] .. p[i–1] <= r < sum of p[0] .. p[i]
  return i;
}
```

Analyzing the Loop

1. Does the initialization make **inv** true?
2. Is **post** true when **inv** is true and **condition** is false?
3. Does the repetend make progress?
4. Does the repetend keep **inv** true?