CS1110 28 October 2010 Arrays (secs 8.1-8.3)

Computer science has its field called computational complexity; mine is called computational simplicity. – Prof. Gries

On (computational) simplicity

We are trying to teach not just Java, but **how to think about problem solving**.

Most of us don't write perfect essays in one pass, and coding is the same: writing requires revising; programming requires revising.

If you are writing too much code —it gets longer and longer, with no end in sight: stop and look for a better way. If your code is getting convoluted and you have trouble understanding it: stop and look for a better way.

Learn to keep things simple, to solve problems in simple ways. This sometimes requires a different way of thinking.

A key point is to break a problem up into several pieces and do each piece in isolation, without thinking about the rest of them. Our methodology for developing a loop does just that.

1

```
/* day contains the number of days
   since ORIGINYEAR (1 Jan 1980)
*/
/* Set year to current year and day to
   current day of current year */
year = ORIGINYEAR; /* = 1980 */

while (day > 365) {
  if (IsLeapYear(year)) {
     if (day > 366) {
        day= day – 366;
        year= year + 1;
     }
  } else {
     day= day – 365;
     year= year + 1;
  }
}
```

Zune error
http://tinyurl.com/9b4hmy

On 31 Dec 2008, the Zune stopped working. Anger! On 1 Jan 2009 it worked.

Zune clock code keeps time in seconds since beginning of 1980. It calculates current day and year from it.

Does each iteration make progress toward termination? Not if day == 366!!

Example

year	day
1980	738
1981	372
1982	7

2

Array: object that stores lists of things.

Holds a *fixed* number of values of a declared type.

(So a0 will always hold 4 int values.)

The **type** of array a0 is **int**[]

Store its name in a variable (as always).

x a0 int[]

a0	
length	4
0	5
1	7
2	4
3	-2

Basic form of a declaration:

<type> <variable-name> ;

So, here is a declaration of x: **int**[] x ;

Does not create array, it only declares x.
x's initial value is **null**.

Elements of array are numbered: 0, 1, 2, …, x.length–1

3

Notes on array length

We write x.length, not x.length(), because length is a field, not a method.

Length field is **final:** an array's length (field or actual number of items) cannot be changed once the array is created.

We omit this field in the rest of the pictures.

a0	
length	4
0	5
1	7
2	4
3	-2

x a0 int[]

The length is not part of the array type, which is **int**[].

This means that an array *variable* can be assigned arrays of different lengths; x could later hold the name of a seven-item int array. (But not the name of a seven-item double array).

4

int[] x ;

x null int[]

x= **new int**[4]; Create array object with 4 default values, store its name in x

x a0 int[]

a0	
0	0
1	0
2	0
3	0

x[2]= 5;
x[0]= -4;

Assign 5 to array element 2; assign -4 to array element 0

x[2] is a reference to element number 2 of array x

a0	
0	-4
1	0
2	5
3	0

int k= 3;
x[k]= 2* x[0];
x[k-1]= 6;

Assign 2*x[0], i.e. -8, to x[3]
Assign 6 to x[2]

a0	
0	-4
1	0
2	6
3	-8

5

Array initializers

Instead of

int[] c= **new** int[5];
c[0]= 9; c[1]= 4; c[2]= 7; c[3]= 6; c[4]= 9;

Use an array initializer:

int[] c= **new int**[ ]{9, 4, 7, 6, 9};

a0	
9	
4	
7	
6	
9	

array *initializer*: gives initial values for the array items. Values must have the same type, in this case, **int**. Length of the array is the number of values in the list; so …

… must omit expression between brackets. *Sometimes*, can even omit the "new <type>[]" part (see pg 274).

6

Use of an array initializer

```
public class D {
  public static final String[] months= new String[]{"January", "February",
                "March", "April", "May", "June", "July", "August",
                "September", "October", "November", "December"};

  /** = the month, given its number m.
      Precondition: 1 <= m <= 12 */
  public static String theMonth(int m) {
    return months[m–1];
  }
}
```

months[m–1] is returned, since

months[0] = "January",
months[1] = "February",
…

Variable months is:

static: no reason to have each object contain one.

public: can be seen outside class D.

final: its value cannot be changed (Careful! you can still change the *elements* in the array whose name it permanently holds! e.g., illegal (except in the Interactions pane…) to say months= new String[] {"Lee"}, but legal to say months[0]= "Lee")

7

Differences between array and Vector ("classier", fbofw)

Declaration:	**int[]** a;	Vector v;
	Elements of a: **int** values	Elements of v: any Objects
Creation:	a= **new int**[n];	v= **new** Vector();
	Array always has n elements	Number of elements can change
Reference element:	a[e]	v.get(e)
Change element:	a[e]= e1;	v.set(e, e1);
	Array locations a[0], a[1], … in successive locations in memory. Access takes same time no matter which one you reference.	Can't tell how Vectors are stored in memory. Referencing and changing elements done through method calls
	Elements all the same declared type (a primitive type or class type)	Elements of any Object type (but not a primitive type). Casting may be necessary when an element is retrieved.
	Initialization shorthand exists. Class has no methods, can't be extended.	No special initialization. Class has methods, can be extended.

"Procedure" for swapping variable values

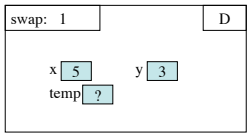
```
public class D {
  /** = Swap x and y */
  public static void swap (int x, int y) {
    int temp= x;
    x= y;
    y= temp;
  }
}
….
```

A call will NOT swap a and b. Parameters x and y are initialized to the *values* of a and b, and thereafter, there is no way to change a and b.

a 5 b 3

swap(a, b);

frame for call just after frame created and args assigned to parameters:



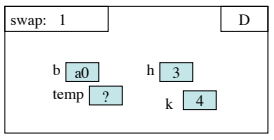
9

Procedure swap for swapping array elements

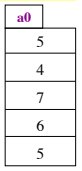
```
public class D {
  /** = Swap b[h] and b[k] */
  public static void swap (int[] b, int h, int k) {
    int temp= b[h];
    b[h]= b[k];
    b[k]= temp;
  }
}
….
  swap(c, 3, 4);
```

This does swap b[h] and b[k], because parameter b contains name of the array.

c a0



frame for call just after frame is created.



10

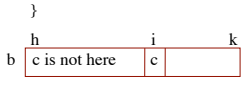
Linear search

```
public class D {
  /** = index of first occurrence of c in b[h..]
      Precondition: c is guaranteed to be in b[h..] */
  public static int findFirst (int c, int[] b, int h) {
    // Store in i the index of first c in b[h..]
    int i=  h;
    // invariant: c is not in b[h..i-1]
    while ( b[i] != c) {
      i= i + 1;
    }
    // b[i] = c  and c is not in b[h..i-1]
    return i;
  }
}
```

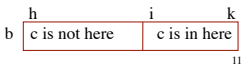
Remember:

h..h-1 is the empty range

Loopy questions:
1. initialization?
2. loop condition?
3. Progress?
4. Keep invariant true?



invariant

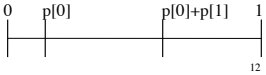


11

```
/** = a random int in 0..p.length-1, assuming p.length > 0.
    The (non-zero) prob of int i is given by p[i].
    Calls: roll(new double[] {.3, .7})
           roll (new double[]{33,.33,.34})*/
public static int roll(double[] p) {
  double r= Math.random(); // r in [0,1)
  /** Store in i the segment number in which r falls. */
  int i  = 0 ;    double iEnd= p[0];
  // inv: r is not in segments looked at (segments 0..i-1)
  //      and iEnd is the end of (just after) segment i
  while (   r not in segment i  ) {
                r >= iEnd
    iEnd= iEnd + p[i+1];
    i= i + 1;
  }
  // r is in segment i
  return  i;
}
```

Non-uniform randomness from uniform randomness: It's a kind of linear search!

1. init
2. condition
3. progress
4. invariant true



12