CS1110 25 Oct 2011: Arrays.

Reading: Secs. 8.1, 8.2, 8.3

Listen to the following lectures on loops on your PLive CD. They are only 2-3 minutes long, and each has an insightful message.

1. The 3 lectures on Lesson page 7-6 —read the whole page.
2. The 4 lectures in Lesson page 7-5.

Computational simplicity

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.

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

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.

A bug in the Zune

```
/* day contains the number of days since 1 Jan 1980 */
/* Set year and day to current year and day of 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;
   }
}
```

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

http://tinyurl.com/9b4hmy

Understanding the pieces of a loop

```
                               initialization
this will be true        →     // inv: <invariant>
before and after
each iteration                 while ( condition ) {
                                  repetend
                               }
want this to be          →     // R: <result assertion>
true at the end
```

- When developing the loop, how do we write the three pieces?
- When understanding a loop that someone gives us, how do we know the pieces are right?

4 Loopy Questions

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

Arrays

Array: an object that holds a fixed number of values of the same type.

(that array→ contains 4 values of type **int**)

The type of an array is written:

<type>[] —for instance, **int**[]

...so to declare a variable x that holds the name of an array of **int**s:

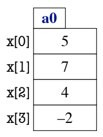
<type> <name> ; —e.g., **int**[] x;

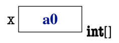
Elements of the *n*-element array x are numbered:

0, 1, 2, …, *n* – 1

To refer to an element of an array:

<var>[*<index>*] —for instance, x[3]





Arrays

The length of an array is a field of the array object.

x.length —not x.length()

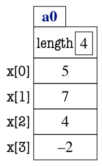
The length field is **final**: it never changes after the array is created. (We will omit in in pictures from here on.)

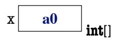
The length is not part of the array type. A variable of type **int**[] can be assigned arrays of different lengths at different times.

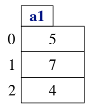
Declaring x does not create an array. An array type is an object type, so x can be **null**.

To create a new array, there is a special new-expression:

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







```
int[] x;
```

Create a variable named x that holds a value of type **int**[] (It is uninitialized)

```
x= new int[4];
```

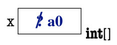
Create array object of length 4, store its name in x (Elements initialized to 0)

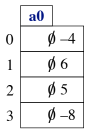
```
x[2]= 5;
x[0]= –4;
```

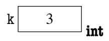
Assign 5 to array element 2 and –4 to element 0

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

Assign –8 to x[3] and 6 to x[2]







Arrays vs. Vectors vs. Strings

Declaration:	**int**[] a; contains ints	Vector<Integer> v; contains Integers	String s; contains chars
Creation:	a= **new int**[n]; size fixed forever	v= new Vector<Integer>(); can be resized at will	s= “foo”; contents fixed forever
Reference:	x= a[i];	x= v.get(i);	c= a.charAt(i);
Change:	a[i]= x;	v.set(i, x);	
	Variables a[0], a[1], … are at successive locations in memory. Element type can be class type or primitive type.	Storage layout not specified (but really, it is an array). Element type can only be a class type.	Storage layout not specified (but really, it is an array). Element type is always **char**.

Array initializers

To initialize the elements of 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:

```
new int[] { 5, 4, 7, 6, 5 }    ← create array of 5 ints and initialize all elements
```

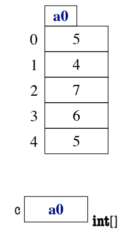
no size goes here (implied by length of initializer list)

types must agree with array’s type

When used in 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 D {

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

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

e.g. D.theMonth(4) returns months[3], which is "April".

Variable months is:

static: object assigned to it will be created only once.

public: can be seen outside class D.

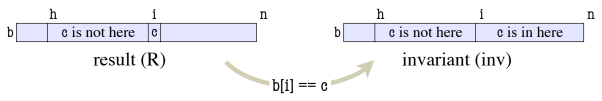
final: it cannot be changed.

Array algorithm: linear search

```
/** = 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;
  }
  // R: b[i] == c and c is not in b[h..i-1]

  return i;
}
```

4 Loopy Questions

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

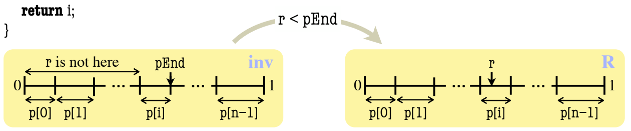


Array algorithm: loaded dice

```
/** = 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 the 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;
  }
  // R: sum of p[0] .. p[i−1] <= r < sum of p[0] .. p[i]

  return i;
}
```

4 Loopy Questions

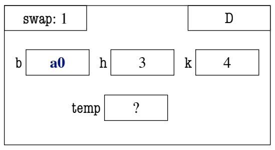
1. Does the initialization make **inv** true?
2. Is **R** always true when **inv** is true and **condition** is false?
3. Does the repetend make progress?
4. Does the repetend keep **inv** true?

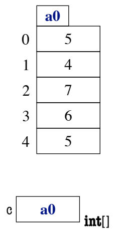


Procedure swap

```
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);
```

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


swap: 1
D
b
a0
h
3
k
4
temp
?


a0
0
1
2
3
4
5
4
7
6
5
c
a0
int[]