Lecture 4

Strings, Wrappers, & Containers

Announcements for This Lecture

Readings

- pp. 175–181
- Sections 2.5, 3.1.2-3.1.3
- Also Section 5.2
- PLive (optional):
  - Lesson 2-5
  - Lessons 5-1 and 5-2

Assignments

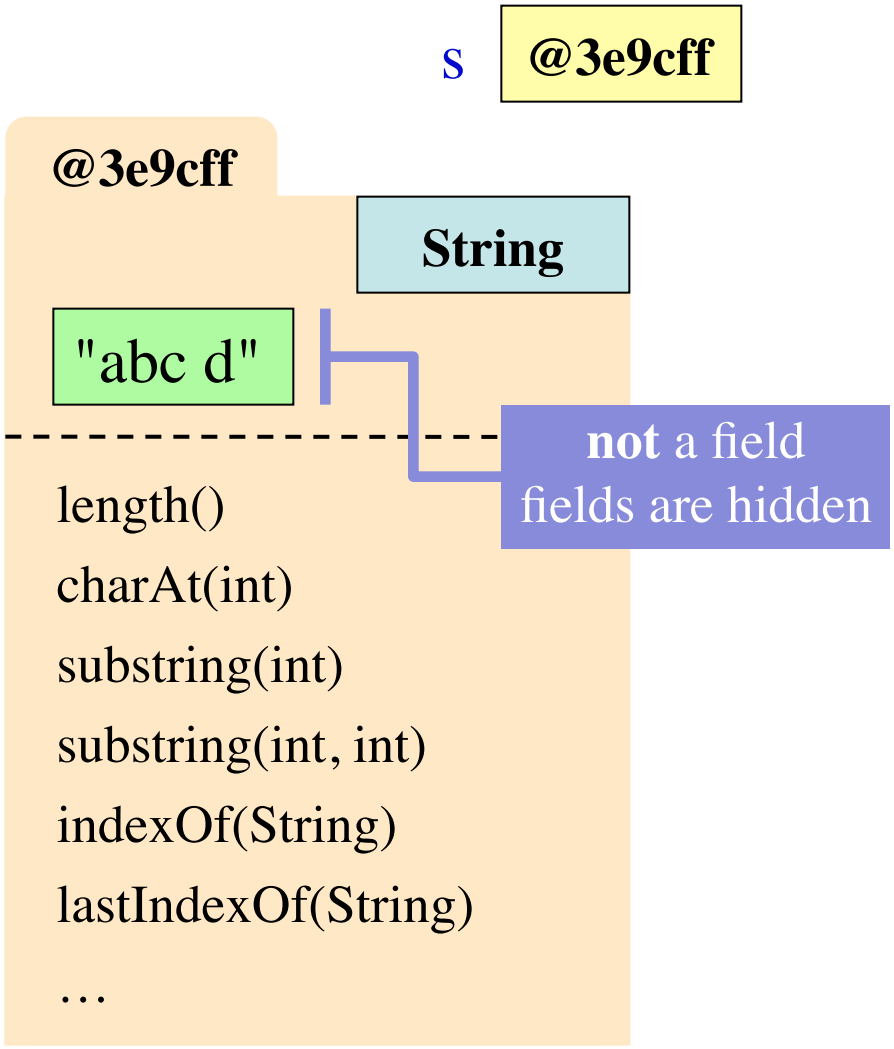
- Assignment 1 due FRIDAY!
  - Before Midnight!!!
  - Will get back by Sunday
  - Revise if you are told
- Start New Assignment
  - No code; written only
  - Meant to do while you revise
  - Scan and submit via CMS (or use a drawing program)

String is a Class; Quoted Text is an Object

- String s = "abc d";
- Indexed characters:

```
01234
abc d
```

- s.length() is 5
- s.charAt(2) is 'c'
- s.substring(2) is "c d"
- s.substring(1,3) is "bc"



String Has a Lot of Useful Methods

- String s = "abc d";
- Indexed characters:

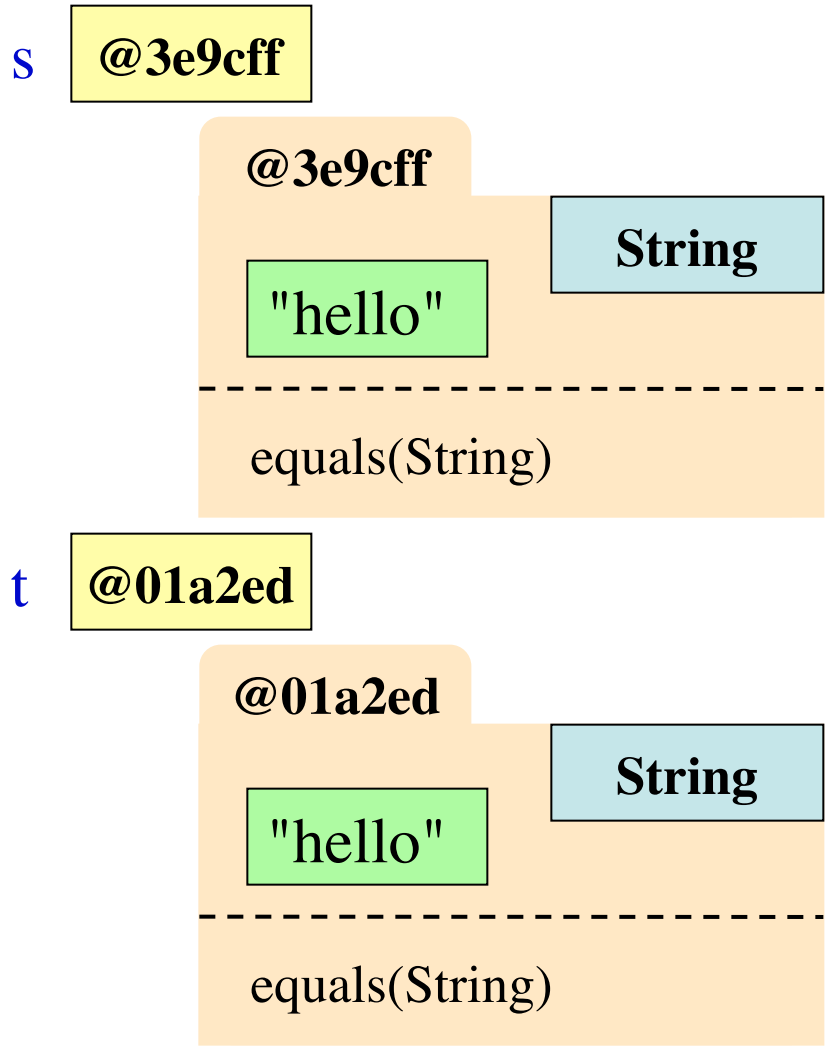
```
01234
abc d
```

- See text pp. 175–181
- Look in CD ProgramLive
- Look at API specs for String

- s.substring(2,4) is "c " **(NOT** "c d")
- s.substring(2) is "c d"
- " bcd ".trim() is "bcd"
  (trim beginning and ending blanks)
- s.indexOf("bc") is 1
  (index or position of first occurrence of in "bc" or -1 if none)

String Variables Hold Folder Names

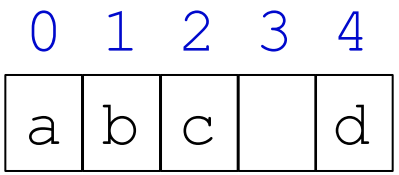
- Create two Strings
  - String s = "hello";
  - String t = "hello";
- Do not use == to test equality of s and t
  - s == t tests if same object
  - Not useful for Strings
- Use equals() instead
  - s.equals(t) tests if they have the same text



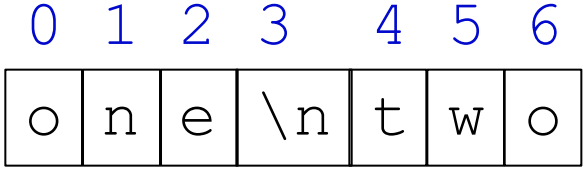
String as a Container

- Data arranged in a “list”
  - List of characters
  - Access characters by position, not field name
  - **Method**: charAt(int)
  - Position starts at 0
- How about other lists?
  - List of ints?
  - List of Points?

- String s = "abc d";



- String s = "one\ntwo";

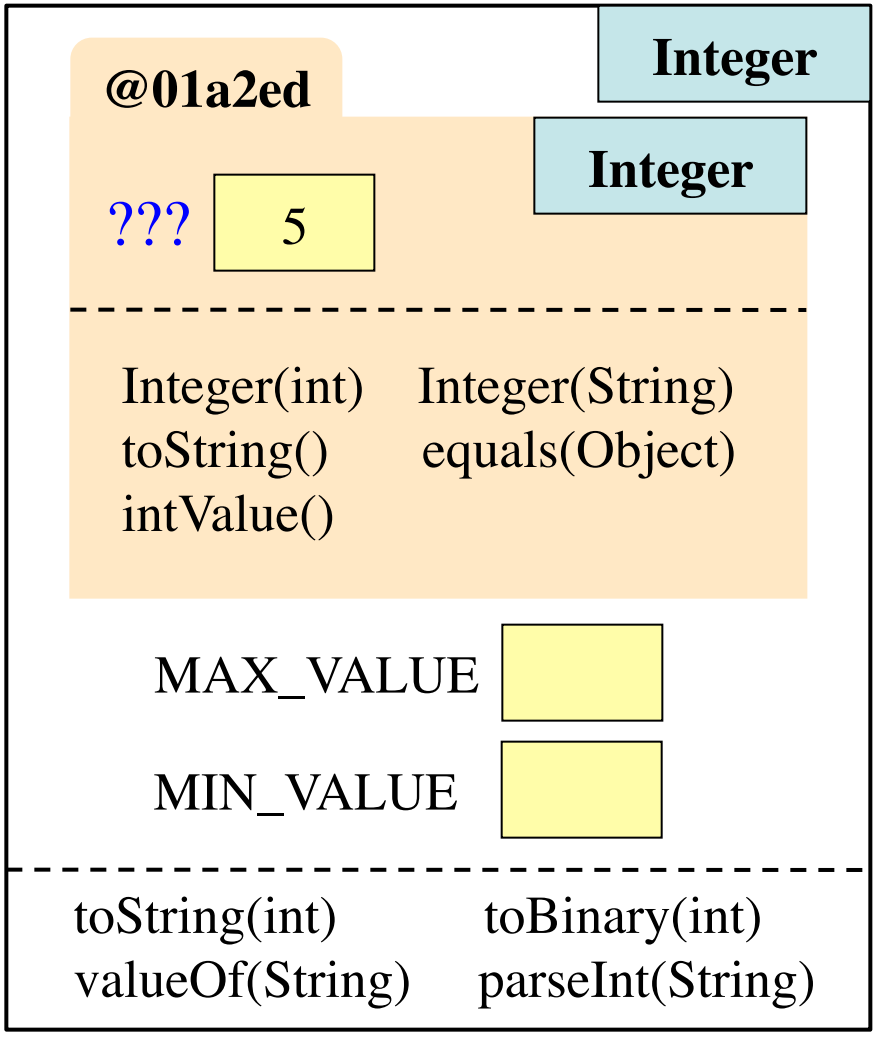


Containers

- **Container**: an object that holds a list of objects
  - **But cannot hold primitive values (e.g. int, double, etc.)!**
- Java has several container classes
  - The are all in package `java.util`
  - **Generic classes**: type depends on what is contained
  - Put contained type in < >
- **Example**: Vector
  - Vector<String>: Vector that holds String objects
  - Vector<Rhino>: Holds Rhino objects
  - Vector<Vector<String>>: ????
  - Vector<int>: **NOT ALLOWED!**

Wrappers: Turn Primitives into Objects

- Want Vector<int>
  - int is primitive type, not class
  - Need to convert an int value (e.g. 9) into an object
- Integer: a **wrapper class**
  - Contains or wraps one value
  - Value cannot be changed: it is *immutable*
- Many useful static features
  - Integer.MAX_VALUE
  - Integer.parseInt(String)



Each Primitive Type Has a Wrapper

- When you need to treat a primitive value as an object, then just wrap the value in an object of the wrapper class.

Primitive Type	Wrapper Class
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Each wrapper class has:

- Instance methods
  (e.g. equals, constructors, toString)
- Static variables and methods
  (for useful computations)

```
Integer k= new Integer(63);          int j= k.intValue();
```

You don't have to memorize the methods of the wrapper classes. But be aware of them. See Section 5.1 and PLive 5-1 and 5-2 for more.

Boxing and Unboxing

- Modern (post 1.4) Java boxes/unboxes
- **Boxing**: Automatically add a wrapper
  - Integer s = 4;
  - Same as Integer s = new Integer(4);
- **Unboxing**: Automatically remove a wrapper
  - int x = new Integer(4);
  - Same as int x = new Integer(4).intValue();
- Type is determined by the variable assigned

Boxing and Unboxing

- Modern (post 1.4) Java boxes/unboxes
- **Boxing**: Automatically add a wrapper
  - Integer s = 4;
  - Same as Integer s = new Integer(4);
- **Unboxing**: Automatically remove a wrapper
  - int x = s;
  - Same as int x = new Integer(4).intValue();
- Type is determined by the variable assigned

Avoid doing this; can be confusing

Example: Vector

- Create an empty vector instance (of Strings)

```
import java.util.Vector;
Vector vec = new Vector<Integer>();
```

- Add some strings to it

```
vec.add(new Integer(2));   // Adds 2 at position 0
vec.add(new Integer(7));   // Adds 7 at position 1
vec.add(new Integer(-3));  // Adds -3 at position 2
```

- Get the String at position 1

```
vec.get(1)                          // Function call, gives 7
```

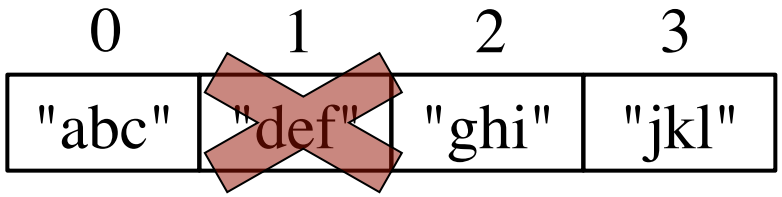
- Search vector for number 5

```
vec.indexOf(new Integer(5))  // Not found; gives -1
```

Vectors Can Add and *Remove*

- Do the following:

```
import java.util;
Vector vec = new
    Vector<String>();
vec.add("abc");
vec.add("def");
vec.add("ghi");
vec.add("jkl");
vec.remove(1);
```



- After all this, what is the value of vec.get(2)?

A: Function gives "abc"
B: Function gives "ghi"
C: Function gives "jkl"
D: I have no clue

Arrays

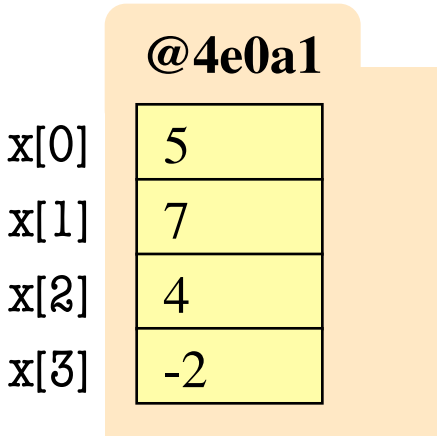
- **Array**: an object that holds a fixed number of values **of the same type**
- Type of an array is written:

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

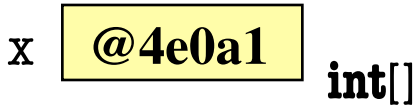
  <type> *<name>*; (e.g., **int**[] x;)
- Elements of array `x` are numbered:

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

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



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

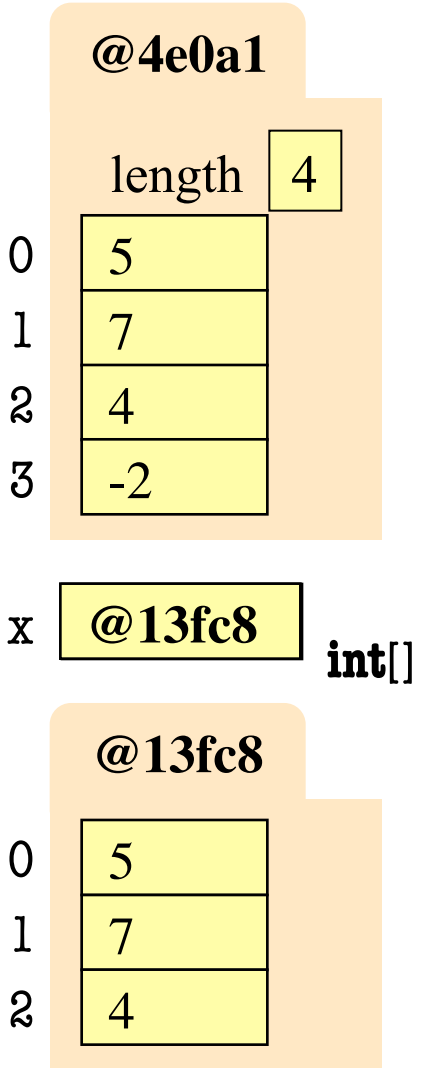


Arrays

- 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 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:

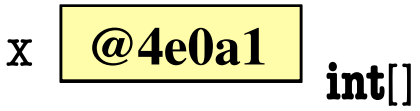
    new <*type*>[<*length*>]
    (e.g. x = **new int**[3];)



Overview of Array Syntax

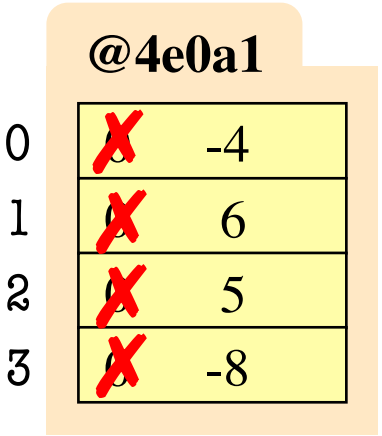
- int[] x;

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



- x = new int[4];

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

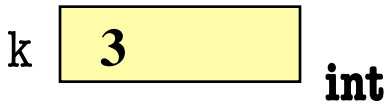


- x[2] = 5;
- x[0] = –4;

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;`

- **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);`

- **Declaration**

  `String s;`

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

  `s = "foo";`

  (contents fixed forever)
- **Reference**

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

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

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

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

Array Initializers

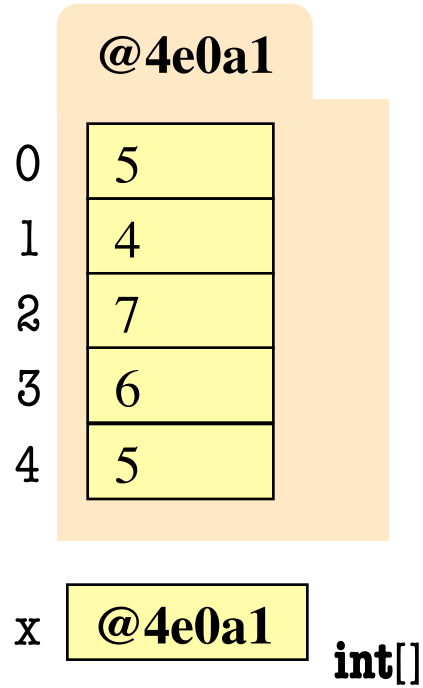
- Initializing a newly created array:
  - `int[] c= new int[5];` ← create array of 5 ints initialized with default (0)
  - `c[0]= 5; c[1]= 4; c[2]= 7; c[3]= 6; c[4]= 5;` ← assign new values to elements
- Instead, use an array initializer: ← create array of 5 ints and initialize all elements
  - `new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