Building Java Programs

Chapter 7

Lecture 7-3: File Output; Reference Semantics

reading: 6.4-6.5, 7.1, 4.3, 3.3

self-checks: Ch. 7 #19-23

exercises: Ch. 7 #5

Two separate topics

- File output
  - A lot like printing to the console
  - Pretty easy; needed for homework 7

- References
  - *Most difficult topic in the whole course*
  - Arrays (and next week objects) don't "work" how you probably think they do!
  - May or may not come up in your homework 7 solution

Output to files

- **`PrintStream`**: An object in the `java.io` package that lets you print output to a destination such as a file.
  - Any methods you have used on `System.out` (such as `print`, `println`) will work on a `PrintStream`.

- Syntax:

  `PrintStream` **name** `= new PrintStream(new File("`**file name**`"));`

  Example:

```
PrintStream output = new PrintStream(new File("out.txt"));
output.println("Hello, file!");
output.println("This is a second line of output.");
```

Details about `PrintStream`

`PrintStream` **name** `= new PrintStream(new File("`**file name**`"));`

- If the given file does not exist, it is created.
- If the given file already exists, it is overwritten.

- The output you print appears in a file, not on the console. You will have to open the file with an editor to see it.

- Do not open the same file for both reading (`Scanner`) and writing (`PrintStream`) at the same time.
  - You will overwrite your input file with an empty file (0 bytes).

System.out and PrintStream

- The console output object, `System.out`, is a `PrintStream`.

```
PrintStream out1 = System.out;
PrintStream out2 = new PrintStream(new File("data.txt"));
out1.println("Hello, console!");    // goes to console
out2.println("Hello, file!");       // goes to file
```

  - A reference to it can be stored in a `PrintStream` variable.
    - Printing to that variable causes console output to appear.
  - You can pass `System.out` as a parameter to a method expecting a `PrintStream`.
    - Allows methods that can send output to the console or a file.

PrintStream question

- Modify our previous Sections program from last lecture to use a `PrintStream` to output to the file `sections_out.txt`.

```
Section #1:
Sections attended: [9, 6, 7, 4, 3]
Student scores: [20, 18, 20, 12, 9]
Student grades: [100.0, 90.0, 100.0, 60.0, 45.0]

Section #2:
Sections attended: [6, 7, 5, 6, 4]
Student scores: [18, 20, 15, 18, 12]
Student grades: [90.0, 100.0, 75.0, 90.0, 60.0]

Section #3:
Sections attended: [5, 6, 5, 7, 6]
Student scores: [15, 18, 15, 20, 18]
Student grades: [75.0, 90.0, 75.0, 100.0, 90.0]
```

PrintStream answer

```
// Section attendance program
// This version uses a PrintStream for output.

import java.io.*;
import java.util.*;

public class Sections {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("sections.txt"));
        PrintStream out = new PrintStream(new File("sections_out.txt"));
        while (input.hasNextLine()) {    // process one section
            String line = input.nextLine();
            int[] attended = countAttended(line);
            int[] points = computePoints(attended);
            double[] grades = computeGrades(points);
            results(attended, points, grades, out);
        }
    }

    // Produces all output about a particular section.
    public static void results(int[] attended, int[] points,
            double[] grades, PrintStream out) {
        out.println("Sections attended: " + Arrays.toString(attended));
        out.println("Sections scores: " + Arrays.toString(points));
        out.println("Sections grades: " + Arrays.toString(grades));
        out.println();
    }
    ...
```

Prompting for a file name

- We can ask the user to tell us the file to read.
  - The file name might have spaces; use `nextLine()`, not `next()`

```
// prompt for input file name
Scanner console = new Scanner(System.in);
System.out.print("Type a file name to use: ");
String filename = console.nextLine();
Scanner input = new Scanner(new File(filename));
```

- What if the user types a file name that does not exist?

Fixing file-not-found issues

- `File` objects have an `exists` method we can use:

```
Scanner console = new Scanner(System.in);
System.out.print("Type a file name to use: ");
String filename = console.nextLine();
File file = new File(filename);

if (!file.exists()) {
    // try a second time
    System.out.print("Try again: ");
    String filename = console.nextLine();
    file = new File(filename);
}
Scanner input = new Scanner(file);  // open the file
```

Output:

```
Type a file name to use: hourz.text
Try again: hours.txt
```

Arrays as parameters and returns; values vs. references

reading: 7.1, 3.3, 4.3

self-checks: Ch. 7 #5, 8, 9

exercises: Ch. 7 #1-10

Swapping values

```
public static void main(String[] args) {
    int a = 42;
    int b = 64;

    // swap a with b (incorrectly)
    a = b;
    b = a;

    System.out.println(a + " " + b);
}
```

- What is wrong with this code? What is its output?

- The red code should be replaced with:

```
    int temp = a;
    a = b;
    b = temp;
```

A `swap` method?

- Does the following `swap` method work? Why or why not?

```
public static void main(String[] args) {
    int a = 42;
    int b = 64;

    // swap a with b
    swap(a, b);

    System.out.println(a + " " + b);
}

public static void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Value semantics

- Java parameters are initialized by *copying the value*
  - The parameter is a different variable
  - Assigning to a parameter variable has no effect on callers
  - So there is *no way* to write a swap method for two `int`s
  - We learned all this back when we learned parameters (although we didn't carefully discuss all the consequences at that time, to avoid overwhelming you with lots of information all at once)

Mini-exercise

- What does this print?

```
public static void main(String[] args) {
 int j = 10;
 squid(j);
 System.out.println(”in main - j = " + j);
}

public static void squid(int k) {
     System.out.println("starting squid - k = " + k);
     k = 20;
     System.out.println(”leaving squid - k = " + k);
}
```

Mini-exercise - answer

```
public static void main(String[] args) {
    int j = 10;
    squid(j);
    System.out.println(”in main - j = " + j);
  }

  public static void squid(int k) {
       System.out.println("starting squid - k = " + k);
       k = 20;
       System.out.println(”leaving squid - k = " + k);
  }
```

Output:

```
starting squid - k = 10
leaving squid - k = 20
in main - j = 10
```

Something different

```
public static void main(String[] args) {
    int[] a = new int[2];
    a[0] = 42;
    a[1] = 64;
    swap(a);

    System.out.println(a[0] + " " + a[1]);
  }

  public static void swap(int[] arr) {
      int temp = arr[0];
      arr[0] = arr[1];
      arr[1] = arr[0];
  }
```

- Prints `64 42` – this swap "works"
  - The question is why

This doesn't work

```
public static void main(String[] args) {
    int[] a = new int[2];
    a[0] = 42;
    a[1] = 64;
    swap(a);
    System.out.println(a[0] + " " + a[1]);
}

public static void swap(int[] arr) {
        int[] x = new int[2];
        x[0] = arr[1];
        x[1] = arr[0];
        arr = x;
}
```

- Prints 42 64

An analogy you know

1. Dan sends Alan an email attaching the file `funPic.jpg`. The file has a picture in it.
2. Alan gets the email and saves the file.
3. Then Alan changes his copy of `funPic.jpg` to have a different picture in it.

Is Dan’s file changed? No.

This is how parameters work:

- Dan’s file is like a variable in the caller
- Alan’s file is like a variable in the method called
- Changing the file’s contents is like assigning to a variable

Variables are like files in the analogy

A slightly different analogy

1. Dan sends Alan an email attaching a file that contains the URL `http://somesite.com/editFunPhoto.html`
2. Alan gets the email and saves the file.
3. Then Alan follows the link and uses it to change an online photo.

When Dan follows the link, does he see the new photo? Yes.

This is how arrays work:

- A file with a link in it is like a variable containing *a reference to an array*
- Updating an array affects all variables that refer to that array

Array variables are like files holding URLs in the analogy

The truth about `new`

- `new` doesn't actually return an array (or an object)

- It returns a "URL" for a new array (or a new object)
  - We call them "references" (or "addresses" or "pointers")

- A variable never "holds" an array (or an object)
  - It always holds a reference to an array (or an object)
  - So when we copy the reference (think URL), there are now two references *to the same array*

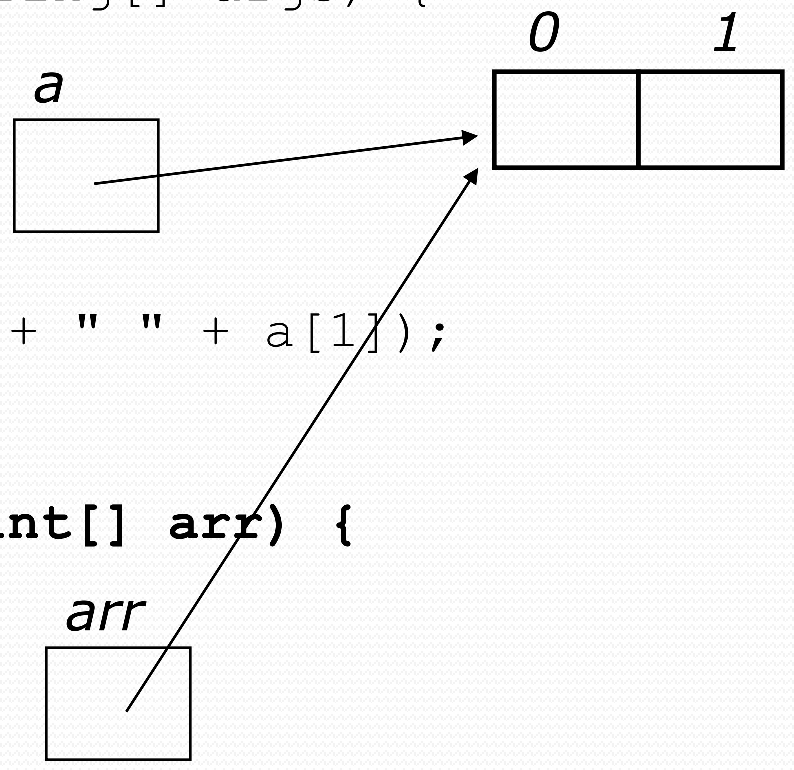
- `a[i] = 42;` follows the reference and updates the array
  - So any other references to the same array see the change

That explains everything

```
public static void main(String[] args) {
    int[] a = new int[2];
    a[0] = 42;
    a[1] = 64;
    swap(a);

    System.out.println(a[0] + " " + a[1]);
  }

  public static void swap(int[] arr) {
        int temp = arr[0];
        arr[0] = arr[1];
        arr[1] = arr[0];
  }
```



Arrows

- Of course, there aren't really "arrows" inside the computer
  - The array is at some "address" (think URL) and a variable referring to the array holds the "address"
  - You don't care what the address *is*; it isn't meaningful
    - You do care if two array variables hold the *same* address

- This is why printing an array variable doesn't work
  - `print` and `println` show the address

```
public static void main(String[] args) {
    int[] arr = {42, 64};
    System.out.println("array is at address: " + arr);
}
```

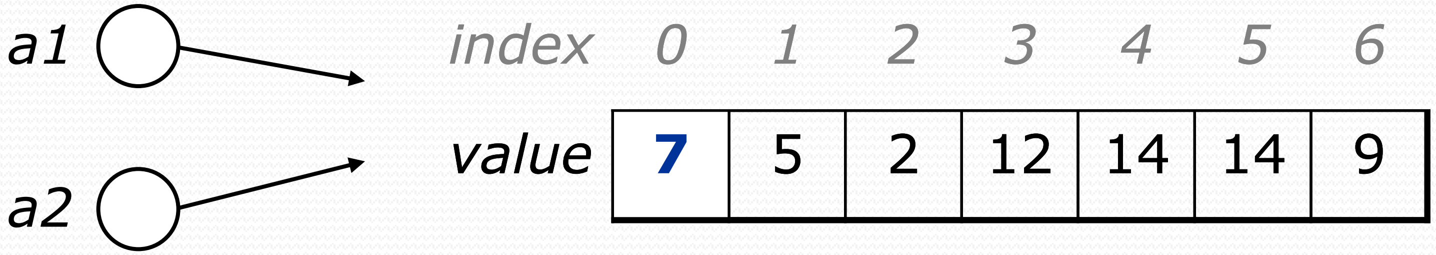
Our wrong example

```
public static void swap(int[] arr) { // WRONG
    int[] x = new int[2];
    x[0] = arr[1];
    x[1] = arr[0];
    arr = x;
}
```

1. `arr` refers to some array at some address #1
2. `x` refers to some new array at some address #2
3. we initialize the contents of the new array
4. we change `arr` to refer to the new array
   - but this has no effect on the caller

A non-parameter example

```
int[] a1 = {4, 5, 2, 12, 14, 14, 9};
int[] a2 = a1;        // refer to same array as a1
a2[0] = 7;
System.out.println(a1[0]);    // 7
```



Mini-exercise

- What does this print?

```
public static void main(String[] args) {
  int[] xs = new int[10];
  int[] ys = xs;
  xs[3] = 10;
  ys[4] = 20;
  System.out.println(ys[3] + ys[4]);
}
```

Mini-exercise - answer

```
public static void main(String[] args) {
   int[] xs = new int[10];
   int[] ys = xs;
   xs[3] = 10;
   ys[4] = 20;
   System.out.println(ys[3] + ys[4]);
}
```

- Output:

```
30
```

Mini-exercise #2

- What does this print?

```
public static void main(String[] args) {
    int[] xs = new int[10];
    int[] ys = xs;
    int[] zs = new int[10];
    xs[0] = 100;
    ys[0] = 50;
    zs[0] = 20;
    System.out.println(xs[0]);
    System.out.println(ys[0]);
    System.out.println(zs[0]);
}
```

Mini-exercise #2 - answer

```
public static void main(String[] args) {
    int[] xs = new int[10];
    int[] ys = xs;
    int[] zs = new int[10];
    xs[0] = 100;
    ys[0] = 50;
    zs[0] = 20;
    System.out.println(xs[0]);
    System.out.println(ys[0]);
    System.out.println(zs[0]);
  }
```

- Output:

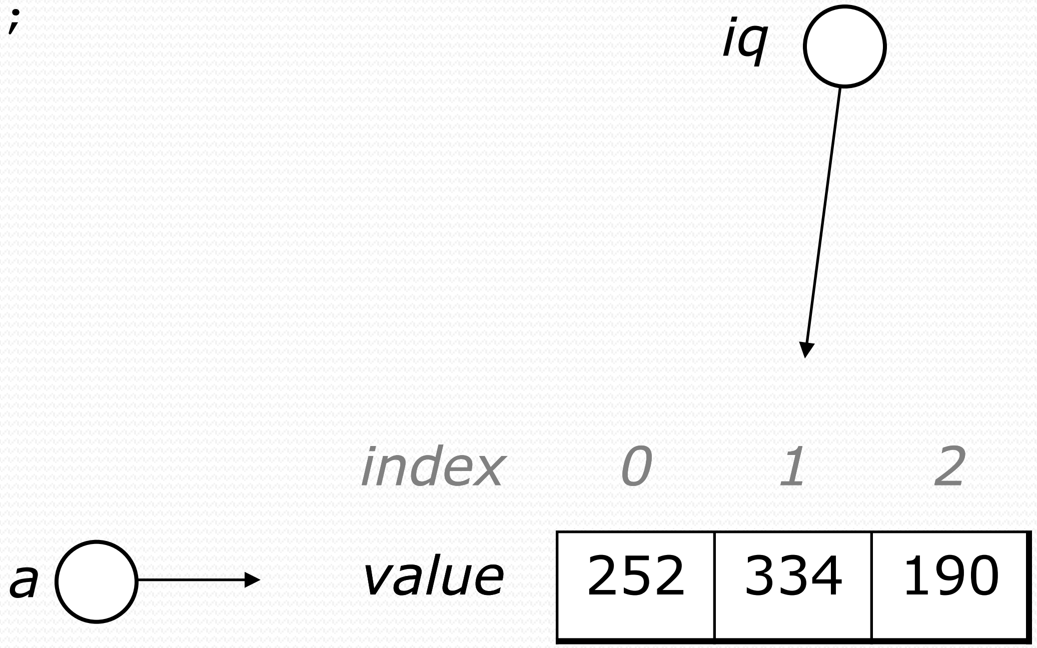
```
50
50
20
```

A useful parameter example

```
public static void main(String[] args) {
    int[] iq = {126, 167, 95};
    doubleAll(iq);
    System.out.println(Arrays.toString(iq));
}
public static void doubleAll(int[] a) {
    for (int i = 0; i < a.length; i++) {
        a[i] = a[i] * 2;
    }
}
```

- Output:

```
[252, 334, 190]
```


iq
index
0
1
2
a
value
252
334
190