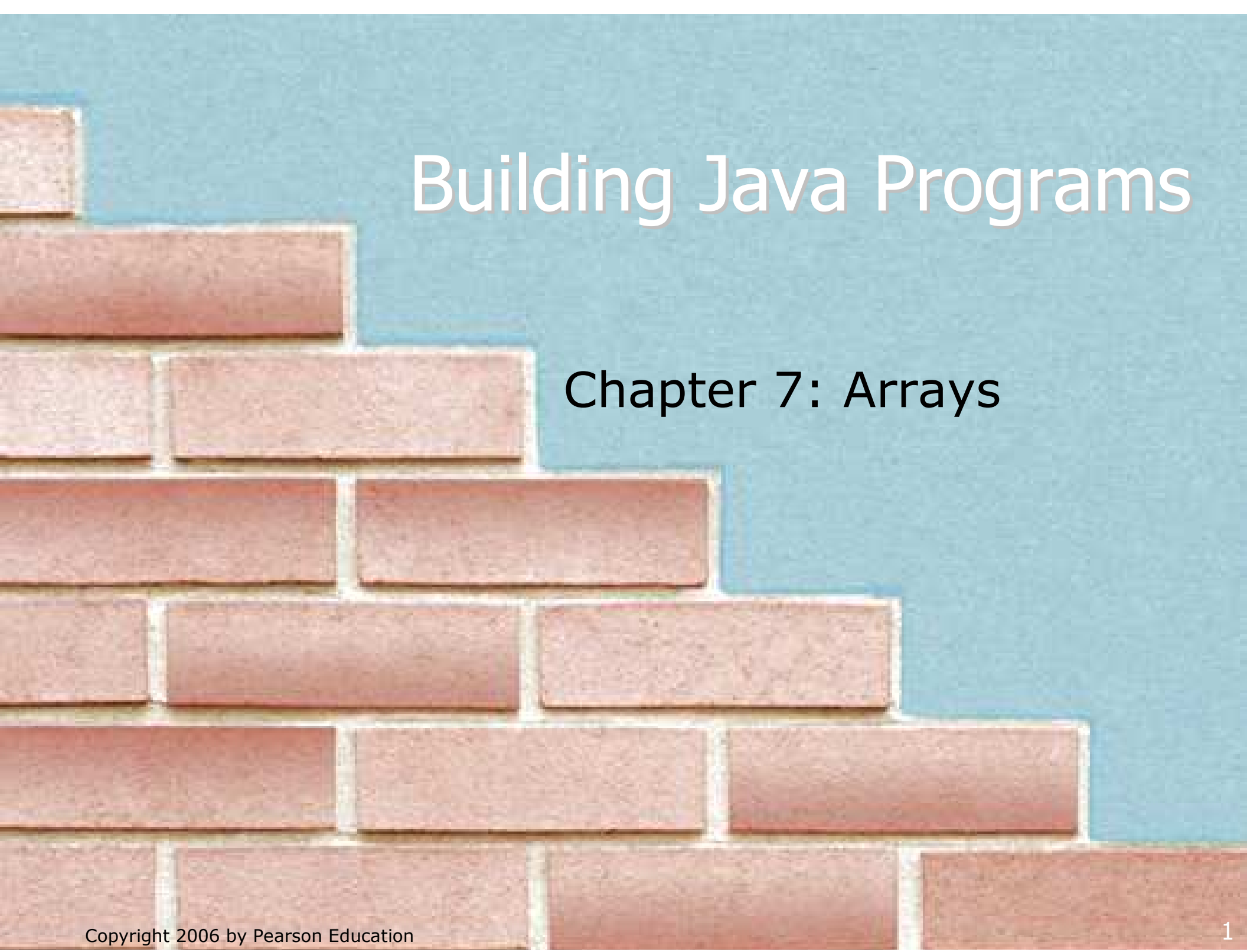
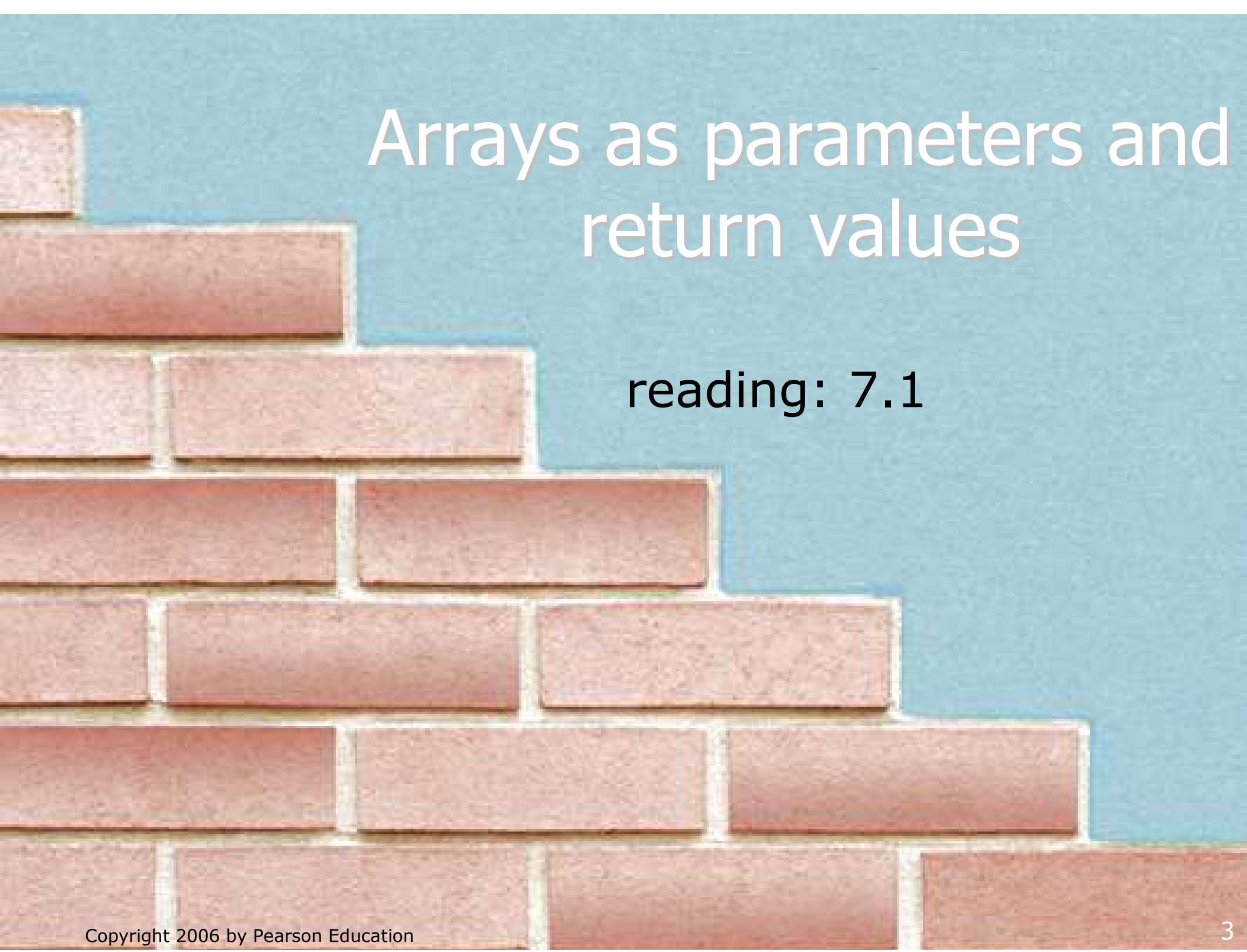


Lecture outline

- advanced array usage
  - arrays as parameters to methods
  - arrays as return values

- advanced file I/O
  - file output using `PrintStream`
  - fixing the file-not-found issue



Arrays as parameters and return values

reading: 7.1

Arrays as parameters

- An array can be passed as a parameter.

- Syntax (declaration):

```
public static <type> <name>(<type>[] <name>) {
```

  - Example:

```
public static double average(int[] numbers) {
```

- Syntax (call):

```
<method name>(<array name>);
```

  - Example:

```
int[] scores = {13, 17, 12, 15, 11};
double avg = average(scores);
```

Array parameter example

```
public static void main(String[] args) {
    int[] iq = {126, 84, 149, 167, 95};
    double avg = average(iq);
    System.out.println("Average = " + avg);
}

public static double average(int[] array) {
    int sum = 0;
    for (int i = 0; i < array.length; i++) {
        sum += array[i];
    }
    return (double) sum / array.length;
}
```

- Output:

```
Average = 124.2
```

Arrays passed by reference

- Arrays are objects.
  - When passed as parameters, they are passed by *reference.* (Changes made in the method will also be seen by the caller.)
- 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[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] *= 2;
    }
}
```

  - Output:

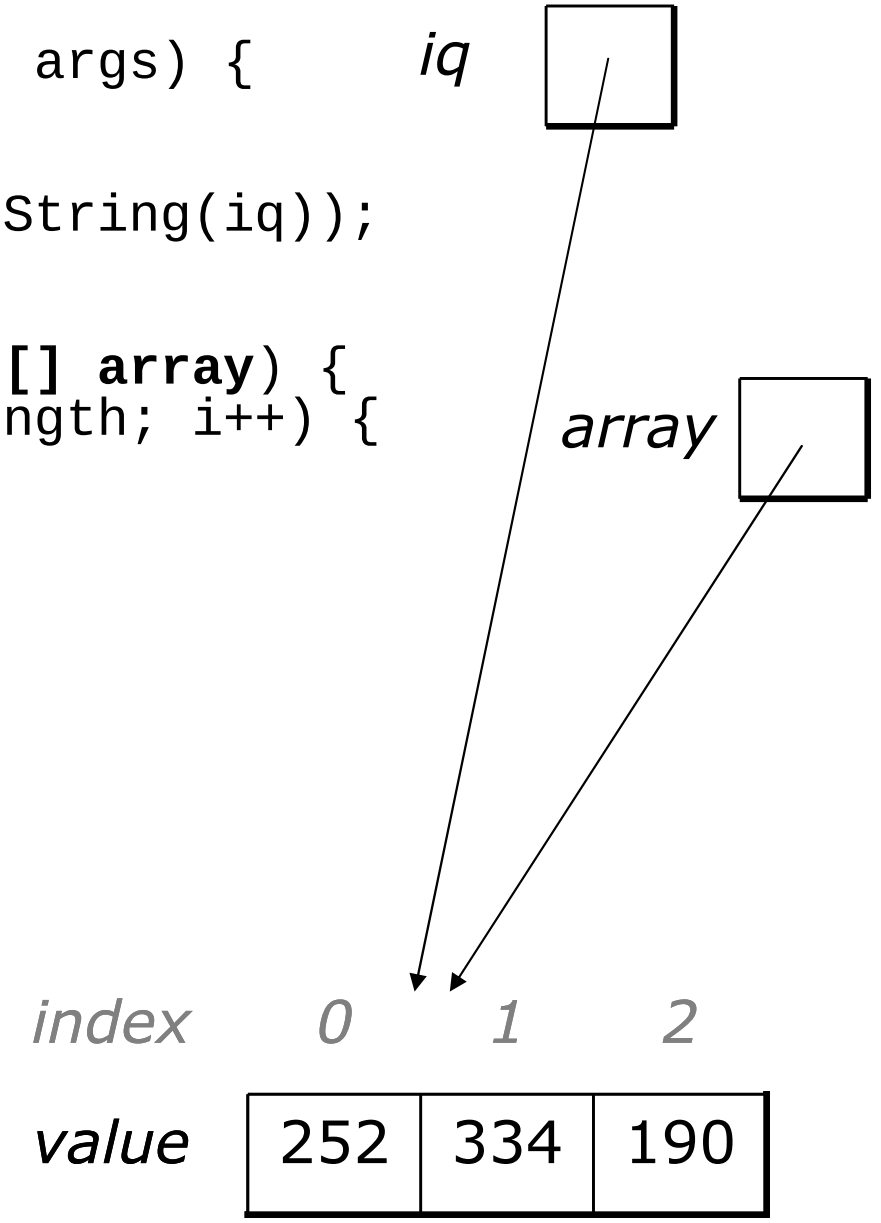
```
[252, 334, 190]
```

Array parameter diagram

```
public static void main(String[] args) {
    int[] iq = {126, 167, 95};
    doubleAll(iq);
    System.out.println(Arrays.toString(iq));
}

public static void doubleAll(int[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] *= 2;
    }
}
```

- Output:

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



Output parameters

- **output parameter**: An array or object passed as a parameter that has its contents altered by the method.
  - We can pass an array and the method can change its contents.
  - Example:

```
int[] nums = {5, -1, 3, 14, 8, 7};
Arrays.sort(nums);
System.out.println(Arrays.toString(nums));
Arrays.fill(nums, 42);
System.out.println(Arrays.toString(nums));
```

Output:

```
[-1, 3, 5, 7, 8, 14]
[42, 42, 42, 42, 42, 42]
```

Arrays as return values

- An array can be returned from a method.
  - Syntax (declaration):
    ```
    public static <type>[] <name>(<parameters>) {
    ```
  - Example:
    ```
    public static int[] countDigits(int n) {
        ...
    }
    ```
  - Syntax (call):
    ```
    <type>[] <name> = <method name>(<parameters>);
    ```
  - Example:
    ```
    int[] digits = countDigits(229231007);
    ```

Array return example

```
public static int[] countDigits(int n) {
    int[] counts = new int[10];
    while (n > 0) {
        int digit = n % 10;
        n = n / 10;
        counts[digit]++;
    }
    return counts;
}

public static void main(String[] args) {
    int[] tally = countDigits(229231007);
    System.out.println(Arrays.toString(tally));
}
```

Output:

```
[2, 1, 3, 1, 0, 0, 0, 1, 0, 1]
```

Array parameter questions

- Write a method named `average` that accepts an array of integers and returns the average of the element values.
- Write a method named `contains` that accepts an array of integers and a target value and returns whether the array contains the target value.
- Write a method named `roundAll` that accepts an array of `doubles` and rounds each to the nearest whole number.

- Improve the previous Histogram and Sections programs by making them use parameterized methods.

Array parameter answers

```
public static double average(int[] numbers) {
    int sum = 0;
    for (int i = 0; i < numbers.length; i++) {
        sum += numbers[i];
    }
    return (double) sum / numbers.length;
}

public static boolean contains(int[] values, int target) {
    for (int i = 0; i < values.length; i++) {
        if (values[i] == target) {
            return true;
        }
    }
    return false;
}

public static void roundAll(double[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] = Math.round(array[i]);
    }
}
```

Array parameter question

- Modify our previous Sections program to use methods for structure. Pass arrays as parameters and return.

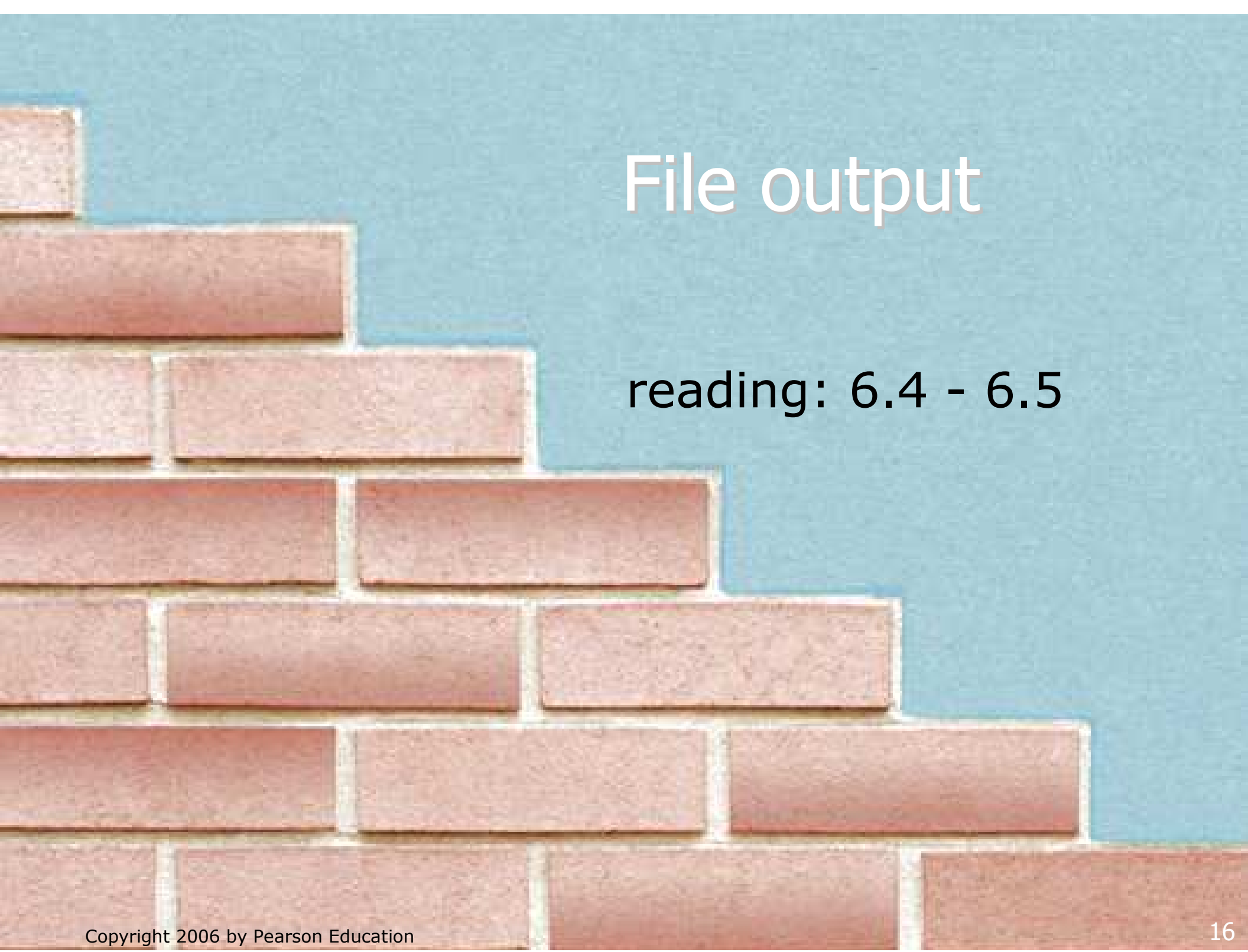
```
Section #1:
Sections attended: [9, 6, 7, 4, 3]
Student scores: [20, 18, 20, 12, 9]
Student grades: [100.0, 90.0, 100.0, 60.0, 45.0]

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]
```

Array param. answer

```
// This program reads a file representing which students attended
// which discussion sections and produces output of the students'
// section attendance and scores.
// This version uses methods for structure.

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"));
        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);
        }
    }

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

    ...
```

Array param. answer 2

```
...
// Counts the sections attended by each student for a particular section.
public static int[] countAttended(String line) {
    int[] attended = new int[5];
    for (int i = 0; i < line.length(); i++) {
        char c = line.charAt(i);
        // c == '1'  or  c == '0'
        if (c == '1') {
            // student attended their section
            attended[i % 5]++;
        }
    }
    return attended;
}

// Computes the points earned for each student for a particular section.
public static int[] computePoints(int[] attended) {
    int[] points = new int[5];
    for (int i = 0; i < attended.length; i++) {
        points[i] = Math.min(20, 3 * attended[i]);
    }
    return points;
}

// Computes the percentage for each student for a particular section.
public static double[] computeGrades(int[] points) {
    double[] grades = new double[5];
    for (int i = 0; i < points.length; i++) {
        grades[i] = 100.0 * points[i] / 20.0;
    }
    return grades;
}
}
```



Prompting for a file name

- We can ask the user to tell us the file to read.
  - We should use the `nextLine` method on the console `Scanner`, because the file name might have spaces in it.

```
// prompt for the 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);

while (!file.exists()) {
    System.out.print("File not found! 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
File not found! Try again: h0urz.txt
File not found! Try again: hours.txt
```

Output to files

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

- Printing into an output file, general syntax:

```
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.

Printing to files, example

- Example:

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

  - You can use similar ideas about prompting for file names here.

- Do not open a file for reading (`Scanner`) and writing (`PrintStream`) at the same time.
  - You could overwrite your input file by accident!
  - The result can be an empty file (size 0 bytes).

PrintStream question

- Modify our previous Sections program to use a `PrintStream` to output to the file `section_output.txt`.

  Contents of `section_output.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

```
// This program reads a file representing which
  students attended
// which discussion sections and produces output
  of the students'
// section attendance and scores, using methods
  for structure.
// 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("section_output.txt"));
        while (input.hasNextLine()) {   //
  process one section
```