Building Java Programs

Chapter 7

Lecture 7-3: Arrays as Parameters; File Output

reading: 7.1, 4.3, 3.3

self-checks: Ch. 7 #19-23

exercises: Ch. 7 #5

Section attendance question

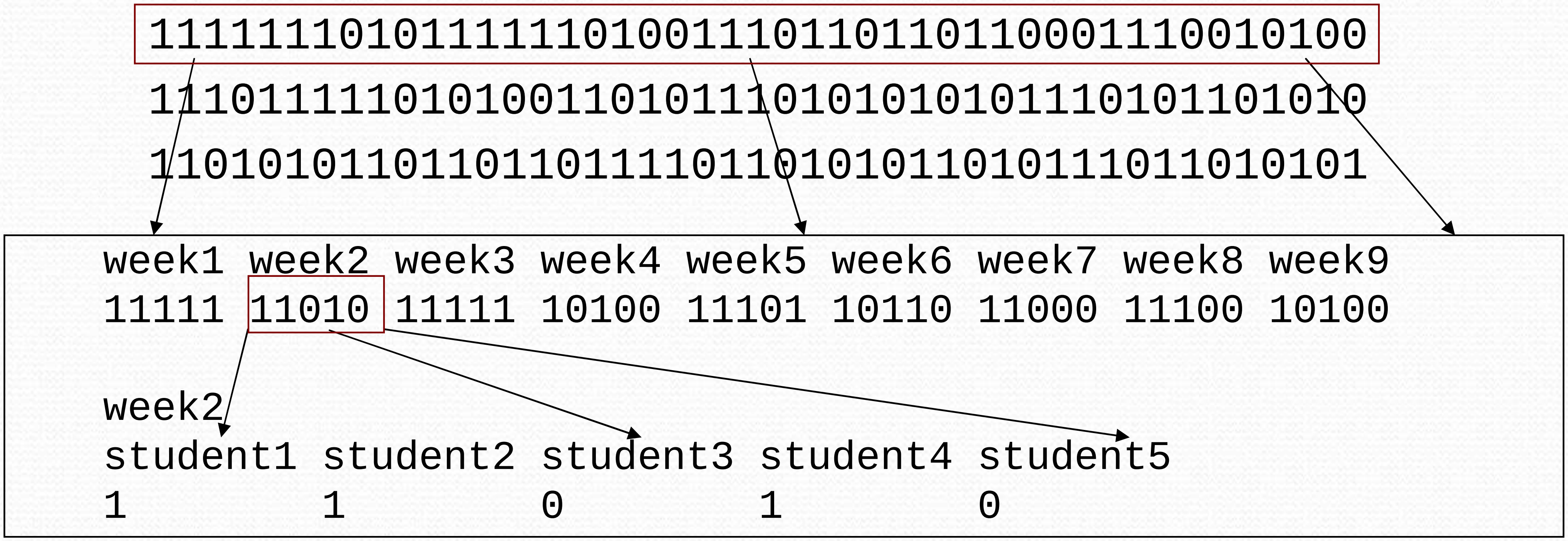
- Write a program that reads a data file of section attendance and produces the following output:

```
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]

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]

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]
```

  - Students earn 3 points for each section attended up to 20.

Section input file

- The input file contains section attendance data:



```
111111101011111101001110110110110001110010100
111011111010100110101110101010101110101101010
110101011011011011110110101011010111011010101
```

```
week1 week2 week3 week4 week5 week6 week7 week8 week9
11111 11010 11111 10100 11101 10110 11000 11100 10100

week2
student1 student2 student3 student4 student5
1        1        0        1        0
```

- Each line represents a section (5 students, 9 weeks).
  - 1 means the student attended; 0 not.

Data transformations

- In this problem we go from 0s and 1s to student grades
  - This is called *transforming* the data.
  - Often each transformation is stored in its own array.

- We must map between the data and array indexes.

  Examples:

by position	(store the *i* th value we read at index *i*)
tally	(if input value is *i*, store it at array index *i*)
explicit mapping	(count `'M'` at index 0, count `'O'` at index 1)

Section attendance answer

```
// This program reads a file representing which students attended which
// discussion sections and produces output of their attendance and scores.

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()) {
            String line = input.nextLine();      // process one section
            int[] attended = new int[5];
            for (int i = 0; i < line.length(); i++) {
                if (line.charAt(i) == '1') {     // c == '1'  or  c == '0'
                    attended[i % 5]++;           // student attended section
                }
            }
            int[] points = new int[5];
            for (int i = 0; i < attended.length; i++) {
                points[i] = Math.min(20, 3 * attended[i]);
            }
            double[] grades = new double[5];
            for (int i = 0; i < points.length; i++) {
                grades[i] = 100.0 * points[i] / 20.0;
            }
            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();
        }
    }
}
```

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 = 7;
    int b = 35;

    // 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?

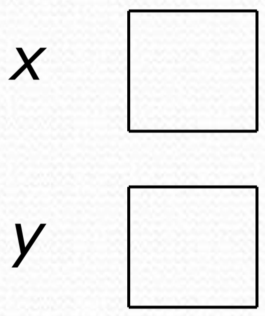
- Does the following swap method work? Why or why not?

```
public static void main(String[] args) {
    int a = 7;
    int b = 35;

    // 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 (primitives)

- **value semantics**: Behavior where values are copied when assigned to each other or passed as parameters.
  - When one primitive variable is assigned to another, its value is copied.
  - Modifying the value of one variable does not affect others.

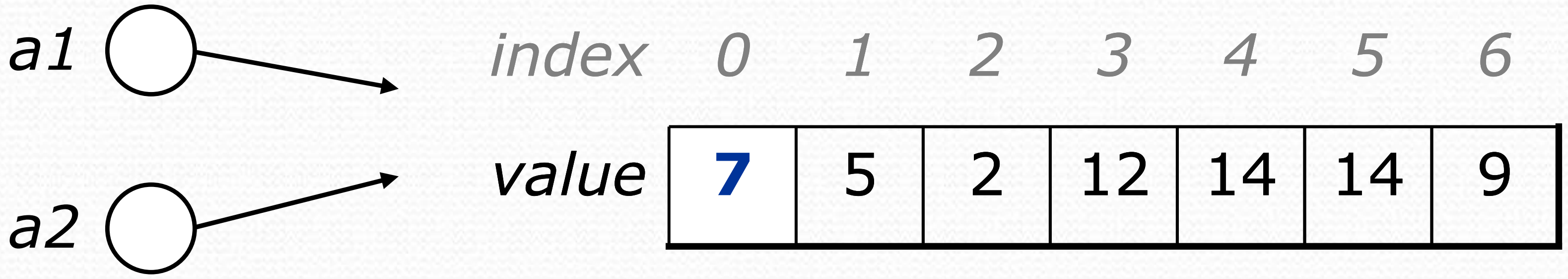
```
int x = 5;
int y = x;          // x = 5, y = 5
y = 17;             // x = 5, y = 17
x = 8;              // x = 8, y = 17
```



Reference semantics (objects)

- **reference semantics**: Behavior where variables actually store the address of an object in memory.
  - When one reference variable is assigned to another, the object is *not* copied; both variables refer to the *same object*.
  - Modifying the value of one variable *will* affect others.

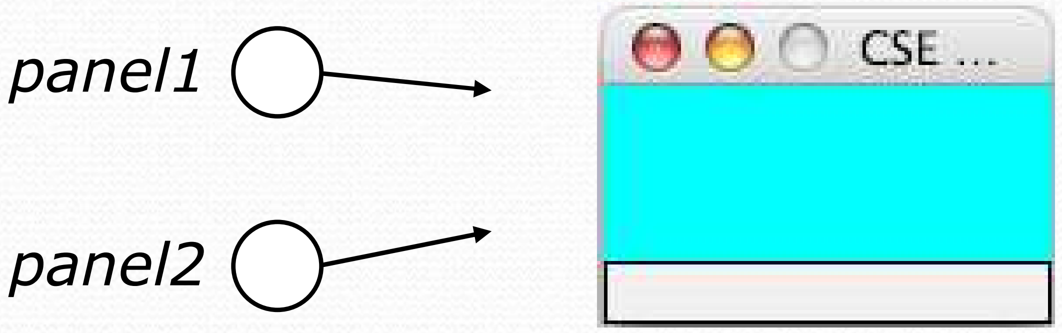
```
int[] a1 = {4, 5, 2, 12, 14, 14, 9};
int[] a2 = a1;      // refer to same array as a1
a2[0] = 7;
System.out.println(a1[0]);   // 7
```



References and objects

- Arrays and objects use reference semantics. Why?
  - *efficiency.* Copying large objects slows down a program.
  - *sharing.* It's useful to share an object's data among methods.

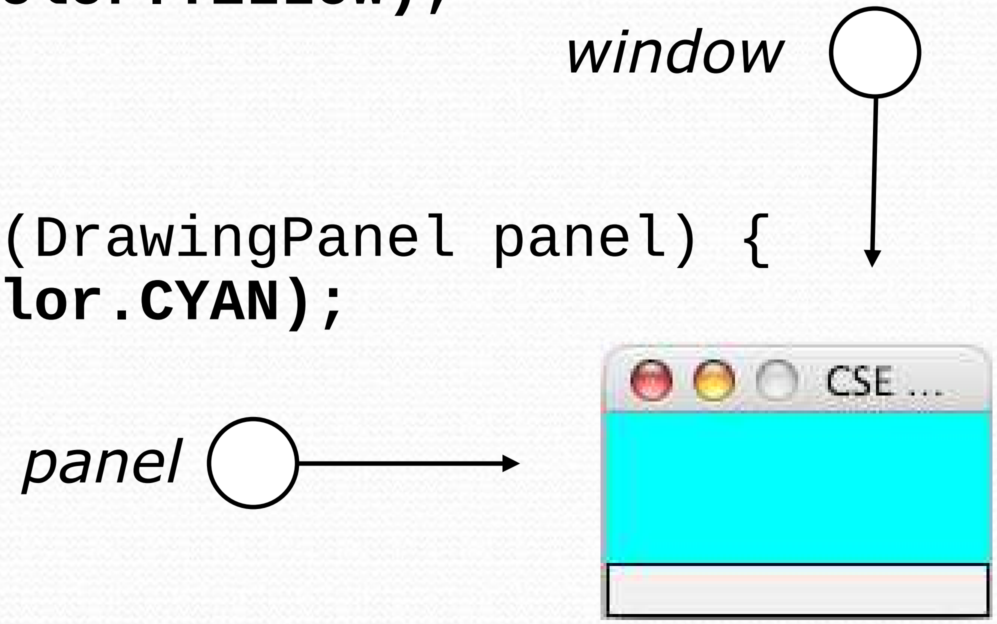
```
DrawingPanel panel1 = new DrawingPanel(80, 50);
DrawingPanel panel2 = panel1;    // same window
panel2.setBackground(Color.CYAN);
```



Objects as parameters

- When an object is passed as a parameter, the object is *not* copied. The parameter refers to the same object.
  - If the parameter is modified, it *will* affect the original object.

```
public static void main(String[] args) {
    DrawingPanel window = new DrawingPanel(80, 50);
    window.setBackground(Color.YELLOW);
    example(window);
}

public static void example(DrawingPanel panel) {
    panel.setBackground(Color.CYAN);
}
```



Arrays as parameters

- Declaration:

```
public static type methodName(type[] name) {
```

  - Example:

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

- Call:

```
methodName(arrayName);
```

  - 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
```

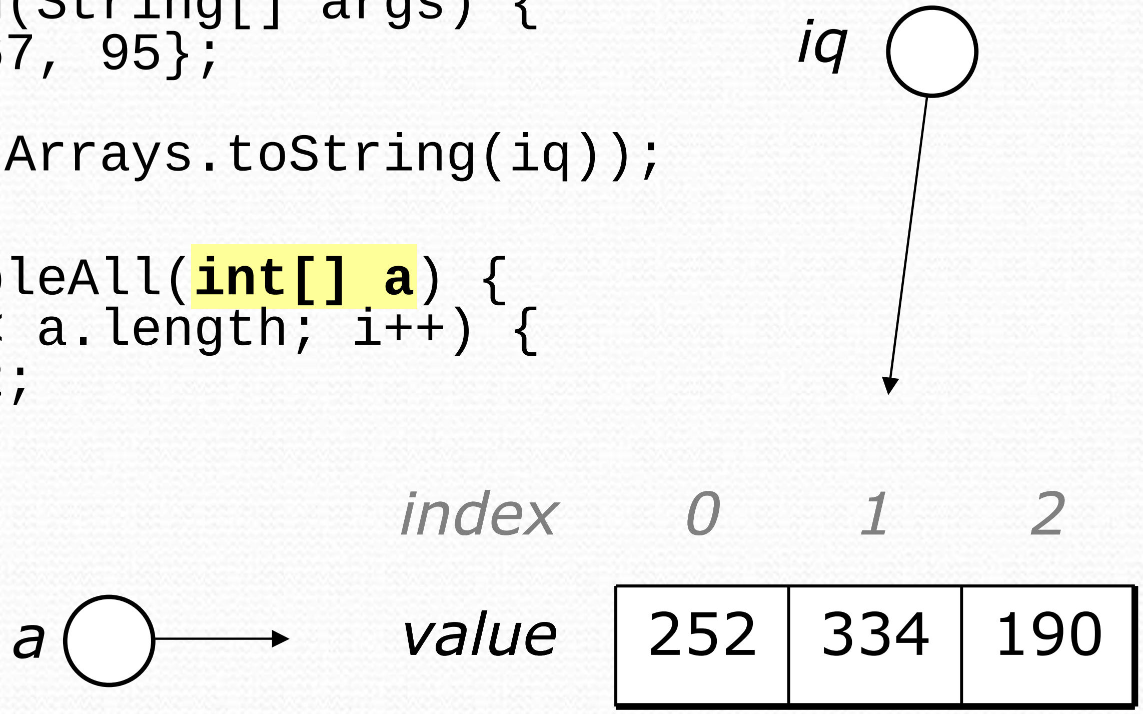
Arrays passed by reference

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

  - Output:

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



Arrays as return (declaring)

```
public static type[] methodName(parameters) {
```

- 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;
}
```

Arrays as return (calling)

type[] **name** = **methodName**(**parameters**);

- Example:

```
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 param/return question

- Modify our previous Sections program to use static methods that use arrays as parameters and returns.

```
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]

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]

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/return answer

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

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/return answer

```
    ...
    // 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;
    }
}
```

File output

reading: 6.4 - 6.5

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 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
```