Name: _____ Session: AM / PM
Circle one

Chapter 4 - Day 3

Files, Random Numbers, and the Capstone

WRITING FILES

Writing a file: `new PrintWriter("crew.txt")` opens a file for writing. If the file already exists, it is _____ first. `println` and `print` work just like `System.out`, but the text goes to the file. Written data waits in a buffer, and `close()` _____. A `main` that writes files needs `throws IOException` in its header, or the code won't compile.

Erase vs. append: To keep what is already in a file, open it with `new FileWriter("crew.txt", true)` and pass that to a `PrintWriter`. The `true` means new lines go at the _____ of the file. FW2 does this, so each run adds _____ line to crew.txt.

Term	Definition / memory cue
output file	
buffer	
throws IOException	
append	

FW1 Saving a crew list

`FW1_CrewManifest.java`

What changed: A for loop asks for each crew member's name and writes it to crew.txt, one name per line.

```
13 PrintWriter outputFile = new PrintWriter("crew.txt");
14 for (int i = 1; i <= crew; i++) {
15     System.out.print("Name of crew member " + i + ": ");
16     String name = keyboard.nextLine();
17     outputFile.println(name);
18 }
19 outputFile.close();
```

1. Line 17 sends each name to _____, not the screen
2. Line _____ closes the file and saves the names

Practice in pairs. Predict crew.txt after each run, in order.

Run	Input	Lines in crew.txt	Last line in crew.txt
FW1	2, Alan Shepard, Buzz Aldrin		
FW2	Sally Ride		
FW2	Mae Jemison		
FW3a	1, John Glenn		

READING FILES

Reading a file: `new File("crew.txt")` names the file, and `new Scanner(file)` reads from it instead of the keyboard. `nextLine`, `nextInt`, and `nextDouble` work the same way, and each read moves the read position _____. Call `close()` when you are done reading.

Term	Definition / memory cue
input file	
read position	
hasNext()	

Reading to the end: `while (inputFile.hasNext())` keeps reading until the file runs out, so the end of the file works like a _____. Reading a file that is not there crashes the program, so check `file.exists()` first.

FR2 Every line of a file

`FR2_ReadAllLines.java`

What changed: A hasNext() loop reads crew.txt one line at a time, numbering each name with a counter.

```
12 while (inputFile.hasNext()) {
13     String name = inputFile.nextLine();
14     count++;
15     System.out.println(count + ": " + name);
16 }
17 inputFile.close();
18 System.out.println("Crew members: " + count);
```

1. The loop stops when hasNext() is

2. On the empty crew.txt that FW3a left, line 18 prints

FR3 A total from a file

FR3_FuelFileTotal.java

What changed: Checks that fuel.txt exists, then adds up every number in it with an accumulator.

```
10 if (!file.exists()) {
11     System.out.println("fuel.txt was not found.");
12 } else {
13     Scanner inputFile = new Scanner(file);
14     double total = 0;
15     while (inputFile.hasNext()) {
16         double fuel = inputFile.nextDouble();
17         total += fuel;
18     }
19     inputFile.close();
20     System.out.printf("Total fuel: %.1f kg%n", total);
```

1. Without fuel.txt, the program prints

2. Line 16 reads each value with

Practice in pairs. Trace FR3 with fuel.txt (120.5, 80, 99.5, 100, one per line): one row per hasNext() check.

hasNext()	fuel	total after line 17
true	120.5	
	(loop ends)	(loop ends)

RANDOM NUMBERS

The Random class: `import java.util.Random;` and `Random rand = new Random();` make a random number generator. `rand.nextInt(n)` returns a whole number from 0 up to _____, never n itself. `rand.nextDouble()` returns a double from 0.0 up to, but not including, 1.0.

Term	Definition / memory cue
nextInt(n)	
nextDouble()	

Shifting a range: Adding to the result moves the whole range: `rand.nextInt(6)` gives 0 to _____, so RN1's `rand.nextInt(6) + 1` rolls a die from 1 to 6. For any range, use `rand.nextInt(high - low + 1) + low`. The number inside the parentheses is _____ are possible.

RN2 Ten rolls — RN2_RollTally.java

What changed: A for loop rolls ten times, counting the sixes and adding every roll to a total.

```
10 for (int roll = 1; roll <= 10; roll++) {
11     int die = rand.nextInt(6) + 1;
12     System.out.println("Roll " + roll + ": " + die);
13     total += die;
14     if (die == 6) {
15         sixes++;
16     }
17 }
```

1. The counter in this loop is

2. Total is always from 10 to

Practice in pairs. Fill in the lowest and highest value for the first four. For the last two, write the expression.

Expression	Lowest	Highest
`rand.nextInt(10)`		
`rand.nextInt(10) + 1`		
`rand.nextInt(51) + 50`		
`rand.nextInt(10) + 20`		
	1	100
	5	15

THE CHAPTER CAPSTONE

One program, the whole chapter: CAP_MissionLog flies missions, logs each orbit's fuel burn to mission_log.txt, then reads the log back. The fuel values are random, so predict the ranges and the structure, not exact numbers.

Practice in pairs, after the walkthrough: find the line where each topic starts in CAP-A to CAP-D below.

Topic	Line	Topic	Line
do-while loop		nested loop (the inner loop)	
priming read		closing the output file	
validation loop		hasNext() loop	
for loop with a bound the user typed		accumulator	
random number in a range			

CAP-A Setup and the validation loop

CAP_MissionLog.java

What changed: A do-while flies at least one mission. The orbit count is checked with a priming read and a while loop.

```
16 do {
17     missions++;
18
19     // Input validation: priming read, then a while loop
20     System.out.print("Mission " + missions + " orbits (1-5): ");
21     int orbits = keyboard.nextInt();
22     while (orbits < 1 || orbits > 5) {
23         System.out.print("Invalid. Orbits (1-5): ");
24         orbits = keyboard.nextInt();
25     }
```

1. Line 21 is the

2. The loop on line 22 repeats until orbits is from 1 to

CAP-B One pass per orbit

CAP_MissionLog.java

What changed: Each orbit gets a random fuel burn, a line in the log file, and a bar of # characters on the screen.

```
28 for (int orbit = 1; orbit <= orbits; orbit++) {
29     int fuel = rand.nextInt(41) + 60; // 60 to 100 kg
30     log.println(missions + " " + orbit + " " + fuel);
31
32     // Nested loop: one # for every 10 kg of fuel
33     System.out.print("Orbit " + orbit + ": ");
34     for (int bar = 1; bar <= fuel / 10; bar++) {
35         System.out.print("#");
36     }
37     System.out.println(" " + fuel + " kg");
38 }
```

1. fuel is a random number from 60 to

2. If fuel is 72, line 35 prints _____ # characters

CAP-C Fly another mission? — CAP_MissionLog.java

What changed: Asks whether to repeat. After the last mission, the log file is closed.

```
40      keyboard.nextLine(); // throw away the leftover newline
41      System.out.print("Fly another mission? (Y/N): ");
42      String answer = keyboard.nextLine();
43      again = answer.charAt(0);
44  } while (again == 'Y' || again == 'y');
45  log.close();
```

1. The do-while repeats when again is _____

2. Line _____ saves the log to the file

CAP-D Reading the log back — CAP_MissionLog.java

What changed: A hasNext() loop reads the log one line at a time, counting the orbits and adding up the fuel.

```
50  int entries = 0;
51  int totalFuel = 0;
52  while (logFile.hasNext()) {
53      int mission = logFile.nextInt();
54      int orbit = logFile.nextInt();
55      int fuel = logFile.nextInt();
56      entries++;
57      totalFuel += fuel;
58  }
```

1. Each pass of the loop reads _____ numbers

2. totalFuel is an

Practice in pairs. Trace lines 52-58 with this mission_log.txt: 1 1 72, 1 2 95, 2 1 64, 2 2 88 (one entry per line).

hasNext()	fuel	entries	totalFuel
true	72		
	(loop ends)	(loop ends)	(loop ends)