

NAME

DATE

PERIOD

UNIT 3 · TOPIC 3.1 · WORKSHEET

Keep What Matters

Pick the details, sort the attributes, name the steps, and catch the design that can't work.

About 35 minutes

10 questions

Cumulative through 3.1

TOPIC

3.1



DIRECTIONS

Work each puzzle in pencil. For Java code, write exactly what prints. Read past `public static` in method headers; it is explained in a later topic. Some puzzles include pieces you won't need.



THINK FIRST

A subway map leaves out almost every street. Is it *wrong*, or doing its job?

PART A · 3 QUESTIONS

A

Warm up your brain

1

PENCIL CHECK

Two track-team programs: the **Results app** ranks runners by race time; the **Jersey app** orders each runner a jersey with their name on the back. Write *keep* if the app needs the fact, or *hide* if it doesn't.

Fact about a runner	Results app	Jersey app
The runner's name		
Fastest time in the 100 m		
Shirt size		
Favorite pre-race snack		

2 YOU'RE THE JVM

A fitness app stores the length of a run as a whole number of meters. Run this code in your head. What prints?

```
int distanceInMeters = 2750;
int km = distanceInMeters / 1000;
int m = distanceInMeters % 1000;
System.out.println(km + " km " + m + " m");
```

A 2.75 km 750 m

B 2 km 750 m

C 2 km 75 m

D 2750 km 0 m



WATCH OUT

Units are part of what data means. To make the run one kilometer longer, `distanceInMeters + 1` adds one *meter*; it should be `+ 1000`. Names like `distanceInMeters` carry the unit so readers don't have to guess.

3 SORT IT OUT

A school district is designing a `SchoolBus` class from this description: "Each bus has its own bus number and keeps count of how many seats are taken. Every bus in the district uses the same dispatch phone number. A bus can pick up a rider. Inside the method that prints the route, a counter `i` steps through the stops. Some buses are painted with a mascot."

For each item, write **instance variable**, **class variable**, **behavior**, or **leave out**.

Item	Instance variable, class variable, behavior, or leave out?
<code>int busNumber</code>	
<code>String dispatchPhone</code>	
<code>int seatsTaken</code>	
<code>void pickUpRider()</code>	
<code>int i</code>	
<code>String mascotPicture</code>	

PART B · 2 PUZZLES

B Puzzles

4 CODE TILES

The bake sale planner has three helper methods. You can't see their bodies, and you don't need to: each comment says what the method does. Someone knocked main off the fridge. Put the tiles back in order so the program prints the target output. One tile is a decoy and isn't used.

BakeSale.java (helper methods)

```
// Returns how many cookies to bake: 2 per person.
public static int cookiesNeeded(int people)

// Returns how many trays the cookies fill, 12 per tray, rounded up.
public static int traysNeeded(int cookies)

// Prints the number of people, then the number of trays.
public static void printPlan(int people, int trays)
```

<code>int trays = traysNeeded(cookies);</code>	<code>printPlan(people, trays);</code>
<code>int people = 30;</code>	<code>int trays = traysNeeded(people);</code>
<code>int cookies = cookiesNeeded(people);</code>	

Target output

People: 30

Trays: 5

YOUR MAIN

5 SNIPPET POOL

A program used to print each row of a banner with its own copy of a loop. It now has one general method instead. Fill each numbered hole with a snippet from the pool so the program prints the target output. Each snippet is used at most once, and some aren't needed.

Banner.java

```
public class Banner {
    public static void main(String[] args) {
        printRow(4, "#");
        printRow(2, "=");
    }

    public static void printRow() {
        for (int i = 0; i < ; i++) {
            System.out.print();
        }
        System.out.println();
    }
}
```

THE POOL

Each snippet can be used only once, and not every snippet is needed.

int count, String mark	String mark, int count	int count
count	4	mark
"#"		

Target output

####

==



FAIR QUESTION

Three copies of the loop would work. What's wrong with copies?

Nothing, until something changes. Then every copy must be found and fixed, and a missed one is a bug that still compiles. One general method keeps the steps in one place: write once, call many times. That's *code reuse*.

PART C · 5 QUESTIONS

C Design and change

6 SAFE CHANGE?

Dozens of programs call this method. For each change, write **safe** (callers needn't be told) or **tell** (they must be), and why.

```
public static int boxesNeeded(int cans) {
    int boxes = cans / 6;
    if (cans % 6 != 0) {
        boxes++;
    }
    return boxes;
}
```

Proposed change	Safe or tell?	Why
Replace the body with return (cans + 5) / 6;		
Replace the body with return cans / 6;		
Add a second parameter, int perBox, to the header		
Replace the body with a loop that adds 6 to a total, counting boxes, until the total reaches cans		



KEY IDEA

Reading a class diagram

The next two puzzles plan a class before any code is written. A class diagram is a box with three parts: the class name on top, then the attributes (type and name), then the behaviors (method headers).

7 FACE-OFF

Tonight: Instance Variable and Class Variable argue about who matters more. Fill in the blanks.

INSTANCE VARIABLE	Every object gets its _____ copy of me. Change one bus's seat count and the other buses don't notice.
CLASS VARIABLE	There's only _____ of me, and every object _____ me. Change the phone number once, and _____ bus sees it.
INSTANCE VARIABLE	Fine. But if a bus's seat count were shared, picking up one rider would change the count on _____ the buses.
CLASS VARIABLE	So the test isn't whether every object <i>has</i> the data. It's whether every object needs the _____.

8 TWO TRUTHS AND A LIE

A class diagram for this description: *Each library card has its own card number and counts the books checked out on it. Every card has the same borrowing limit. A card can check out a book, return a book, and report whether it can borrow another.* Two claims are true and one is a lie. Circle the lie, then rewrite it so it's true.

LibraryCard
<pre>int cardNumber int booksOut int limit (class variable)</pre>
<pre>void checkOut() void returnBook() boolean canBorrow()</pre>

- **1** If the library raises `limit`, every card sees the new limit after one change.
- **2** `canBorrow()` needs no parameter, because the card already has `booksOut` and `limit` as attributes.
- **3** `booksOut` should also be a class variable, because every card has a number of books out.

THE LIE, REWRITTEN

9 DESIGN DETECTIVE

This diagram should match: *Each pet feeder holds some amount of food, in grams. A feeder can dispense a given number of grams and can report whether it is empty. It has **two** problems. Find each one and say how to fix it.*

PetFeeder
String color
void dispense() boolean isEmpty()
PROBLEM 1 AND ITS FIX
PROBLEM 2 AND ITS FIX

10 WHO AM I?

Some design ideas from this topic are playing a party game. Each gives a clue about itself. Name the idea.

Clue	I am...
I hide the details that don't matter for the problem and keep the ones that do.	
I let you use a method by knowing what it does, not how it does it.	
I let you think about data by its name and meaning, not by how it's stored.	
I break one big behavior into smaller methods, one for each step.	
I sit in a method header and let one method work for many different values.	