

NAME

DATE

PERIOD

UNIT 3 · TOPIC 3.5 · WORKSHEET

Give It Back

Trace mutators and accessors, rebuild a method that says yes or no, and catch the return that got away.

About 30 minutes

9 questions

Cumulative through 3.5

TOPIC

3.5



DIRECTIONS

Work each puzzle in pencil. When code prints, write exactly what it prints. Every class keeps its instance variables `private`; `main` reaches them only through the methods each class provides. Some puzzles include pieces you won't need.

PART A · 3 QUESTIONS

A

Warm up your brain



THINK FIRST

A method changes its `int` parameter to `0`. Does the variable the caller passed in become `0` too? Guess in the margin; question 8 settles it.

1

FACE-OFF

Tonight: an Accessor and a Mutator argue about who's more useful. Fill in the blanks.

ACCESSOR

I'm a _____ method. Call me and I hand back a _____ of an instance variable's value.

MUTATOR

I'm usually _____. I hand nothing back; I _____ the object's instance variables.

ACCESSOR

Whatever the caller does with my copy, the instance variable stays the _____.

MUTATOR

So to change it, the caller has to go through me, passing the amount as an _____.

2 PENCIL CHECK

A program runs `Counter c = new Counter(5);` and then the statements in the table, in order. Fill in count after each statement, and the value the method call hands back (write *nothing* for a method that returns nothing).

```
class Counter {
    private int count;
    public Counter(int start) {
        count = start;
    }
    public void add(int amount) {
        count = count + amount;
    }
    public int getCount() {
        return count;
    }
    public boolean isEven() {
        return count % 2 == 0;
    }
}
```

Statement	count after	The call returns
<code>c.add(3);</code>		
<code>int a = c.getCount();</code>		
<code>c.add(a);</code>		
<code>boolean e = c.isEven();</code>		
<code>c.add(-9);</code>		
<code>boolean f = c.isEven();</code>		

3 YOU'RE THE JVM

You are the Java Virtual Machine. `main` runs the two lines below the class. What prints?

```
class Lockbox {
    private int tries;
    public Lockbox() {
        tries = 0;
    }
    public int unlock(int code) {
        for (int guess = 1; guess <= 5; guess++) {
            tries++;
            if (guess == code) {
                return tries;
            }
        }
        return -1;
    }
}

// in main:
Lockbox box = new Lockbox();
System.out.println(box.unlock(3) + " " + box.unlock(9));
```

A 3 -1**B** 5 -1**C** 3 8**D** -1 -1

PART B · 3 PUZZLES

B Puzzles

4 CODE TILES

spend should subtract the cost only when the wallet has enough coins, and return whether it did. Build its body from five tiles so main prints the target output. One tile is a decoy.

```
class Wallet {
    private int coins;
    public Wallet(int start) {
        coins = start;
    }
    public boolean spend(int cost) {
        // your tiles go here
    }
    public int getCoins() {
        return coins;
    }
}
```

return true;	if (cost > coins) {	cost = coins - cost;
}	coins = coins - cost;	return false;

main runs	Target output
<pre>Wallet w = new Wallet(10); System.out.println(w.spend(4)); System.out.println(w.spend(9)); System.out.println(w.getCoins());</pre>	<pre>true false 6</pre>

THE BODY OF SPEND



FAIR QUESTION

Can a void method say return?

Yes, as `return;` with no value. It ends the method right there and sends control back to the caller. A mutator can use it to skip bad input: if the amount is negative, `return;` before changing anything. A void method can't return a *value*, because its header promised nothing comes back.

5 SNIPPET POOL

main runs `Thermostat t = new Thermostat(17.5);` and `t.raise(2.0);`, then prints `t.getTemp() + " " + t.isCold()`. Fill the holes from the pool so it prints `19.5 false`. A room is cold below `18.0` degrees.

```
class Thermostat {
    private double temp;
    public Thermostat(double start) {
        temp = start;
    }
    public void raise(double amount) {
        temp = temp + (1);
    }
    public (2) getTemp() {
        return temp;
    }
    public boolean isCold() {
        return (3);
    }
}
```

THE POOL

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

amount	temp	double	int
void	temp < 18.0	temp > 18.0	"cold"

6 YOU'RE THE COMPILER

You are the compiler. Each method is added to this Box class and called on new Box(7). Does the class compile? If so, write what printing the call shows. If not, say why.

```
class Box {
    private int size;
    public Box(int start) {
        size = start;
    }
    // one method from the table goes here
}
```

Method	Compiles?	Prints, or why not
1 public getSize() { return size; }		
2 public int half() { return size / 2.0; }		
3 public double half() { return size / 2; }		
4 public int grow() { size++; return size; size++; }		
5 public boolean big() { if (size > 10) { return true; } }		



WATCH OUT

A void method's call is a statement, not a value. If reset() is void, System.out.println(timer.reset()); and int left = timer.reset(); don't compile, because there's nothing to print or store. Call it on its own line: timer.reset();.

PART C · 3 QUESTIONS

C

Catch the mistake

7

ONE-TOKEN FIX

main runs `Score s = new Score(10);`, then `s.addBonus(5);`, then prints `s.getPoints()`. It prints 10, but it should print 15. Change exactly one token, and explain why it printed 10.

```
1  class Score {  
2      private int points;  
3      public Score(int start) {  
4          points = start;  
5      }  
6      public void addBonus(int bonus) {  
7          bonus = points + bonus;  
8      }  
9      public int getPoints() {  
10         return points;  
11     }  
12 }
```

THE CHANGE, AND WHY IT PRINTED 10

8 TWO TRUTHS AND A LIE

main runs `Coach c = new Coach();` and `int laps = 3;`, calls `c.train(laps);` twice, then prints `laps + " " + c.getTotal();`. One claim is a lie. Circle it and rewrite it so it's true.

```
class Coach {
    private int total;
    public Coach() {
        total = 0;
    }
    public void train(int laps) {
        laps = laps * 2;
        total = total + laps;
    }
    public int getTotal() {
        return total;
    }
}
```

- 1 The program prints 3 12.
- 2 On each call, the parameter laps starts at 3 and becomes 6.
- 3 Since both are named laps, doubling the parameter doubles main's laps too.

THE LIE, REWRITTEN

9 WHO AM I?

Some method words are playing a party game. Each gives a clue about itself. Name it.

Clue	I am...
My header has <code>void</code> , and my call stands alone as a statement.	
I hand back a copy of an instance variable's value and change nothing.	
I change an object's instance variables, and I'm often <code>void</code> .	
When I run, the method ends on the spot, even inside a loop.	
I start with a copy of the argument, so changing me never changes the caller's variable.	