

Teaching Unit 7: Methods, Classes, and Objects

7.1 Methods and Parameters

Solutions

7.1.1 - Trace

Determine the exact output. Explain the Java rule or state changes that produce it.

```
int n = 7;  
int copy = n;  
copy += 3;  
System.out.println(n + ":" + copy);
```

Output

7:10

The primitive value is copied into a distinct variable; modifying it does not change n.

7.1.2 - Write code

Write public static void bump(int[] values, int delta). Add delta to each array element. The array mutation must be visible to the caller.

One valid solution is shown below. Other implementations satisfying the contract are also valid.

```
public static void bump(int[] values, int delta) {  
    for (int i = 0; i < values.length; i++) values[i] += delta;  
}
```

Check your code

```
int[] a = {1,3}; bump(a,2); System.out.println(Arrays.toString(a));  
int[] b = {}; bump(b,2); System.out.println(Arrays.toString(b));
```

Expected output

```
[3, 5]  
[]
```

Teaching Unit 7: Methods, Classes, and Objects

7.2 Return Values

Solutions

7.2.1 - Trace

Determine the exact output. Explain the Java rule or state changes that produce it.

```
int a = 3;  
int b = 8;  
System.out.println(Math.max(a, b) + Math.min(a, b));
```

Output

11

The return values 8 and 3 become operands of the addition.

7.2.2 - Write code

Write public static int clamp(int value, int low, int high), assuming low <= high. Return the nearest boundary if value lies outside the interval; otherwise return value.

One valid solution is shown below. Other implementations satisfying the contract are also valid.

```
public static int clamp(int value, int low, int high) {  
    if (value < low) return low;  
    if (value > high) return high;  
    return value;  
}
```

Check your code

```
System.out.println(clamp(-1, 0, 10));  
System.out.println(clamp(5, 0, 10));  
System.out.println(clamp(11, 0, 10));
```

Expected output

0
5
10

Teaching Unit 7: Methods, Classes, and Objects

7.3 Classes and Constructors

Solutions

7.3.1 - Trace

Determine the exact output. Explain the Java rule or state changes that produce it.

```
int[] a = new int[2];  
int[] b = new int[2];  
a[0] = 6;  
System.out.println(a[0] + ":" + b[0]);
```

Output

6:0

Each new expression creates a distinct object with independent storage.

7.3.2 - Write code

Write a Counter class with a private int value, a constructor accepting its initial value, increment(), and getValue(). The methods operate on instance state.

One valid solution is shown below. Other implementations satisfying the contract are also valid.

```
public class Counter {  
    private int value;  
    public Counter(int initial) { value = initial; }  
    public void increment() { value++; }  
    public int getValue() { return value; }  
}
```

Check your code

```
Counter a = new Counter(2); Counter b = new Counter(10);  
a.increment();  
System.out.println(a.getValue() + ":" + b.getValue());
```

Expected output

3:10

Teaching Unit 7: Methods, Classes, and Objects

7.4 Object State and Encapsulation

Solutions

7.4.1 - Trace

Determine the exact output. Explain the Java rule or state changes that produce it.

```
ArrayList<Integer> a = new ArrayList<Integer>();  
a.add(5);  
ArrayList<Integer> b = a;  
b = new ArrayList<Integer>();  
System.out.println(a.size() + ":" + b.size());
```

Output

1:0

Reassigning a copied reference does not empty the original object.

7.4.2 - Write code

Write a Wallet class with private int balance initialized to 0, getBalance(), and deposit(int amount). Ignore nonpositive deposits.

One valid solution is shown below. Other implementations satisfying the contract are also valid.

```
public class Wallet {  
    private int balance;  
    public Wallet() { balance = 0; }  
    public int getBalance() { return balance; }  
    public void deposit(int amount) {  
        if (amount > 0) balance += amount;  
    }  
}
```

Check your code

```
Wallet w = new Wallet(); w.deposit(8); w.deposit(-3); w.deposit(0);  
System.out.println(w.getBalance());
```

Expected output

8