# HW7_Part1_Q2

## Code Analysis

### Initial Setup

```
int a = 7;
int b = 10;
ArrayList<Integer> arr = new ArrayList<>();
arr.add(1);
arr.add(2);
ArrayList<Integer> arr2 = arr;
arr2.add(3);
```

- `a` is initialized to 7, and `b` is 10.
- `arr` is an `ArrayList` initialized with `[1, 2]`.
- `arr2` references the same object as `arr`. Adding `3` to `arr2` also modifies `arr`. At this point, `arr` is `[1, 2, 3]`.

### Output 1

```
System.out.println("Test1:" + arr);
```

**Output**: `Test1:[1, 2, 3]`

### Method Call `A`

```
callStack.A(a);
System.out.println("Test2:" + a);
```

- `A` method takes a copy of `a`. The operation `int r = a + 4;` does not modify `a` in the `main` method.
- `a` remains `7`.

**Output**: `Test2:7`

### Method Call `B`

```
b = callStack.B(a);
System.out.println("Test3:" + a + ", " + b);
```

- `B` method modifies the local `b` and calls `B2` with `b + 1`. The return value of `B` is assigned to `b` in the `main` method.
  - Inside `B`, `b` becomes `7 + 1 = 8`. `B2` returns `9`, but it is not used.
  - `B` returns `8`, so `b` in the `main` method becomes `8`.

**Output**: `Test3:7, 8`

## Method Call `C`

```
callStack.C(arr);
System.out.println("Test4:" + arr);
System.out.println("Test5:" + arr2);
```

- `C` modifies the shared `arr` by adding `12`. Since `arr2` references the same object, the change is reflected in both.
- `arr` becomes `[1, 2, 3, 12]`.

**Output**:

```
Test4:[1, 2, 3, 12]
Test5:[1, 2, 3, 12]
```

## Method Call `D`

```
arr2 = callStack.D(arr);
System.out.println("Test4:" + arr);
System.out.println("Test5:" + arr2);
```

- `D` adds `20` and `30` to `arr`. The return value of `D` (the same `arr` object) is assigned to `arr2`, so `arr2` still references `arr`.
- `arr` becomes `[1, 2, 3, 12, 20, 30]`.

**Output**:

```
Test4:[1, 2, 3, 12, 20, 30]
Test5:[1, 2, 3, 12, 20, 30]
```

### Method Call `E`

```
arr2 = callStack.E(arr);
System.out.println("Test6:" + arr);
System.out.println("Test7:" + arr2);
```

- `E` adds `40` to the shared `arr`. Then, it reassigns `arr` to a new `ArrayList` containing `50`. However, this reassignment only affects the local `arr` within `E`. The original `arr` remains unchanged.
- `arr2` is assigned the new `ArrayList` created in `E`, which contains `[50]`.
- `arr` remains `[1, 2, 3, 12, 20, 30, 40]`.

**Output**:

```
Test6:[1, 2, 3, 12, 20, 30, 40]
Test7:[50]
```

## Final Output

```
Test1:[1, 2, 3]
Test2:7
Test3:7, 8
Test4:[1, 2, 3, 12]
Test5:[1, 2, 3, 12]
Test4:[1, 2, 3, 12, 20, 30]
Test5:[1, 2, 3, 12, 20, 30]
Test6:[1, 2, 3, 12, 20, 30, 40]
Test7:[50]
```