## Q Possible outputs of two threads with a synchronized stack

```
stack.push(1);                    stack.push(2);
int result = stack.pop();         int result = stack.pop();
stack.push(result);               stack.push(result);
println("Thread 1: " + result);   println("Thread 2: " + result);
```

**?**: What kinds of synchronization bugs are possible: data races, race conditions, both, or neither?

**Q1**: Give all of the possible outputs for these two threads assuming the stack is synchronized.

## Q Identify the concurrency bugs in this class

```
private Map<String, Double> balances = new HashMap<>();
public void withdraw(String name, double amount) {
    double balance = balances.get(name);
    if (balance < amount) throw new InvalidTransactionException();
    balances.put(name, balance - amount);
}
public void deposit(String name, double amount) {
    double balance = balances.get(name);
    balances.put(name, balance + amount);
}
```



Q1: Identify the concurrency bugs in this class. (Data races and race conditions.)

## Q Deadlock

**Deadlock** occurs when multiple threads are blocked on each other.

Give a bad interleaving for this synchronized BankAccount class.

```
synchronized void withdraw(int amount) { ... }
synchronized void deposit(int amount)  { ... }
synchronized void transferTo(int amt, BankAccount other) {
    this.withdraw(amt);
    other.deposit(amt);
}
```



## Parallel Stream Algorithms

```
int[] arr = {6, 4, 16, 10, 16, 14, 2, 8};
// Prepare to filter, but don't actually do any computation
Stream<Integer> stream = Arrays.stream(arr).parallel()
        .filter(x -> x > 10);
// Compute result in parallel and store in a new int[]
int[] result = stream.toArray(int[]::new);
```

| arr | 6 | 4 | 16 | 10 | 16 | 14 | 2 | 8 |
|---|---|---|---|---|---|---|---|---|

| result | 16 | 16 | 14 |
|---|---|---|---|

**Deadlock** occurs when multiple threads are blocked on each other.

**Q1**: Give a bad interleaving for this synchronized BankAccount class.