Homework 5

Mathew Kurian (mk26473)
Kapil Gowru (krg766)

Question 1

The Stack class has consensus number exactly two because:
- **Consensus Object**: The method T decide(T value) is called by each thread at most once. It is therefore *consistent* and *valid*.
- **Consensus Protocol**: A consensus solution that is both wait-free and lock-free.
- **Consensus Number**: A class C solves n-thread consensus if there exists a consensus protocol for any number of obejcts of class C and atomic registers. Therefore, a consensus number is the alrgest n for which class C solves n-thread consensus.

The protocol is wait-free since `decide()` contains no loops and `Stack` is inherently wait-free. There are two possible scenarios when a thread returns a value:
1. Both threads return their own input meaning both must have popped `WIN`, violating the `Stack` protocol.
2. Both threads return the other's input also violating the protocol.
In addition, the protocol states that at least one of the proposed values must be returned because its winning is value is written before `WIN` is popped.

```
class StackConsensus {
    Stack s;
    int [] proposed;
    StackConsensus(){
        s.push(LOSE);
        s.push(WIN);
    }
    T decide(T value){
        proposed[threadID] = value;
        if (s.pop() == WIN){
            return proposed[threadID];
        } else {
            return proposed[1-threadID];
        }
    }
}
```

To prove that the `Stack` class has a consensus number of exactly two (2), we must use three (3) threads A, B, C. We will also have a case analysis of the following: both A and B call `push()`, `pop()`, or A calls `push()` and B calls `pop()`.
1. Both A and B call `pop()`: In this case, there are two states possible. In the first one, A pops then B pops meaning this state is 0-valent. In the second state, B pops first and then A pops meaning this is 1-valent. Together this makes it impossible for thread C to get the correct value in both states.
2. Both A and B call `push()`: There are also two state possible in this case. In the first state, A pushes a and B pushes b and then proceed to pop the other's elements. In the other state, B pushes b and A pushes a, then proceed to pop their respective elements. In both of these cases, C will not be able to distinguish the states and thus cannot agree on a value.
3. A calls `push()` while B calls `pop()`: There are again two cases possible for this case. In the first state, A pushes a, B pops a, and A pops the uppermost value of the stack (if it exists). In the second state, B pops the uppermost value (if it exists), A pushes a, and A pops a. In both cases, C will not able identify the state of either A or B and thus will not know what thread to choose. In addition, we do not care what happens if an empty stack is popped, since that does not affect the state visible to C.