Solutions to Questions.

1. The following is my code snippet:

```java
monitor semaphoreEquivalent{
    int value = 0;

    public semaphoreEquivalent(int value){
        value = value;
    }

    public synchronized void Signal(){
        value++;
        notify(); //Java notify waiting thread
    }

    public synchronized void Wait(){
        while (value <= 0){
            wait();
            value--;
        }
    }
}
```

2. The following are my adjustments incorporating my semaphoreEquivalent implementation from above.

```java
function modify(){
    sem1.Wait()
    y = x + 2;
    sem1.Signal()
    sem1.Wait()
    x = x + y;
}

function change(){
    y = -55;
    sem1.Signal();
    sem1.Wait();
    y = y + 1;
    sem1.Signal()
}

main(){
    semaphoreEquivalent sem1 = new semaphoreEquivalent(0);
    x = 3; y = 4;
    co
    modify();
    change();
}
```
This works almost identically to the Semaphore implementation. This is due to the fact that the monitor is implemented to work just like a Semaphore. Essentially, value is initialized at 0 and so modify waits in a while loop. Change makes \( y = -55 \). Then the next line signals the monitor and modify is allowed to run while change is waiting. Modify then makes \( y = 5 \). It then sets change to run while waiting itself. Change makes \( y = 4 \) and frees modify to continue which then makes \( x = 7 \).