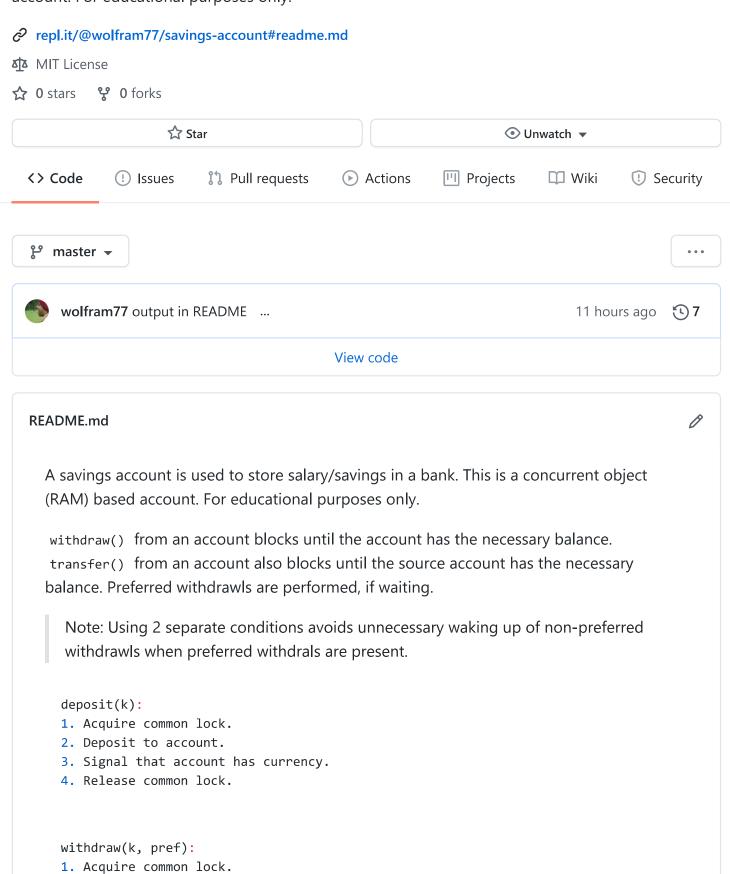
☐ javaf / savings-account

A savings account is used to store salary/savings in a bank. This is a concurrent object (RAM) based account. For educational purposes only.



2. Update pending preferred withdrawls. 3. Wait until sufficient balance available. 4. Withdraw from account. 5. If non-zero balance, then signal it. 6. Update pending preferred withdrawls. 7. Release common lock. transfer(k, from, pref): 1. Withdraw from source account. 2. Deposit to this account. ## OUTPUT Setting up accounts ... Starting transfers ... 6: [21] 100 from 3 4: [55] 100 from 5 7: [145] 100 from 1 0: [195] 100 from 9 5: [199] 100 from 6 2: [68] 100 from 8 1: [36] 100 from 6 3: [39] 100 from 0 9: [22] 100 from 3 8: [73] 100 from 2 3: [139] done 0: [195] done 4: [155] done 9: [122] done Boss donates 1000 to all 5: [1199] done 2: [1068] done 1: [1036] done 8: [1173] done 7: [1245] done

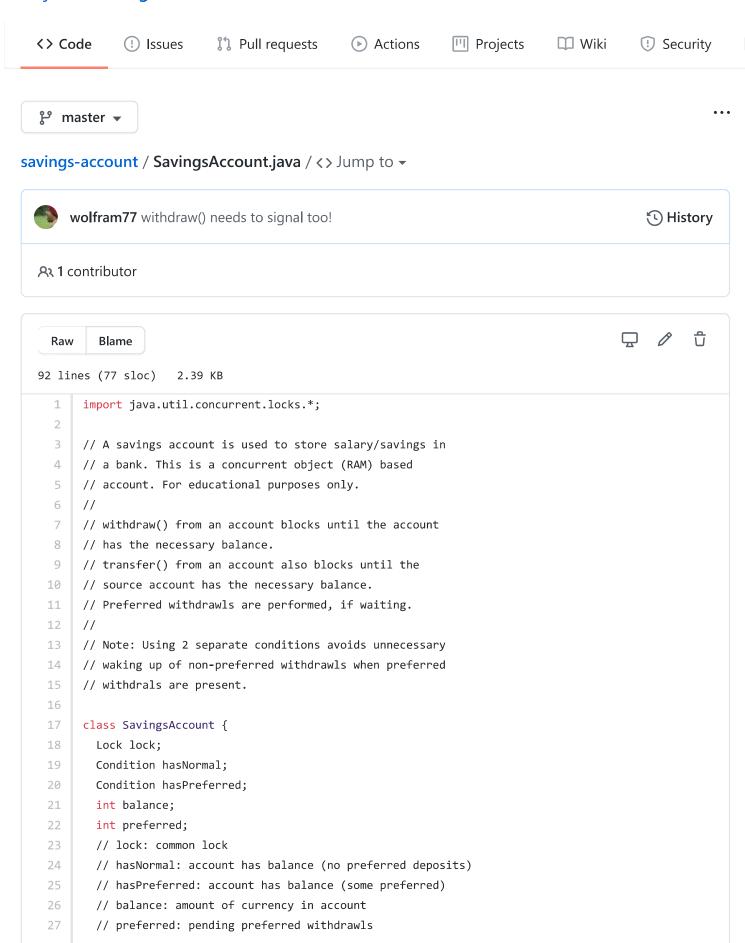
See SavingsAccount.java for code, Main.java for test, and repl.it for output.

references

6: [1021] done

• The Art of Multiprocessor Programming :: Maurice Herlihy, Nir Shavit

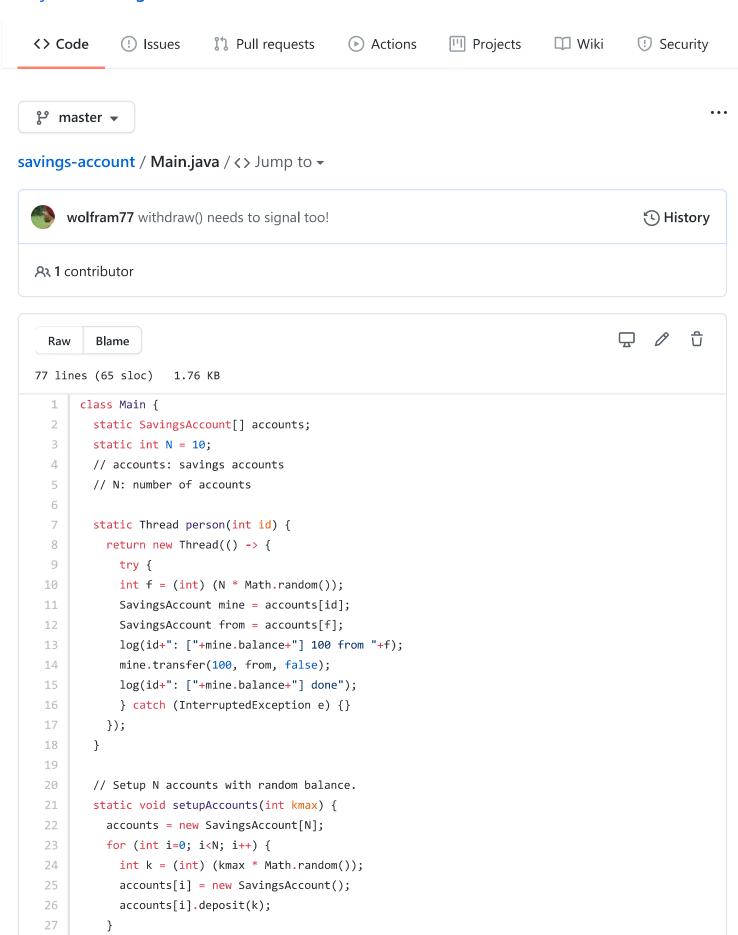
☐ javaf / savings-account



```
28
29
       public SavingsAccount() {
30
31
         lock = new ReentrantLock();
32
         hasNormal = lock.newCondition();
33
         hasPreferred = lock.newCondition();
34
         balance = 0;
         preferred = 0;
       }
37
38
39
       // 1. Acquire common lock.
40
       // 2. Deposit to account.
41
       // 3. Signal that account has currency.
       // 4. Release common lock.
42
43
       public void deposit(int k) {
44
         lock.lock(); // 1
45
         balance += k; // 2
46
         signal();
                    // 3
47
         lock.unlock(); // 4
48
       }
49
50
51
       // 1. Acquire common lock.
52
       // 2. Update pending preferred withdrawls.
53
       // 3. Wait until sufficient balance available.
54
       // 4. Withdraw from account.
       // 5. If non-zero balance, then signal it.
56
       // 6. Update pending preferred withdrawls.
57
       // 7. Release common lock.
58
       public void withdraw(int k, boolean pref)
59
       throws InterruptedException {
60
         lock.lock();
61
         if (pref) preferred++; // 2
62
63
         while (balance<k) await(pref); // 3</pre>
64
         balance -= k;
                                         // 4
65
         if (balance>0) signal(); // 5
         if (pref) preferred--; // 6
67
68
         lock.unlock();
                                  // 7
       }
70
71
72
       // 1. Withdraw from source account.
       // 2. Deposit to this account.
73
74
       public void transfer(int k,
75
       SavingsAccount from, boolean pref)
```

```
throws InterruptedException {
76
77
         from.withdraw(k, pref); // 1
         deposit(k);
                                 // 2
78
79
       }
80
81
82
       private void signal() {
         if (preferred>0) hasPreferred.signal();
83
84
         else hasNormal.signal();
85
       }
86
       private void await(boolean pref)
87
       throws InterruptedException {
88
         if (pref) hasPreferred.await();
89
90
         else hasNormal.await();
91
       }
92
     }
```

☐ javaf / savings-account



```
28
       }
29
       // Start all transfers (by person).
30
       static Thread[] startTransfers() {
31
32
         Thread[] t = new Thread[N];
33
         for (int i=0; i<N; i++)
34
           t[i] = person(i);
         for (int i=0; i<N; i++)</pre>
           t[i].start();
37
         return t;
38
       }
39
40
       // Deposit amount to all accounts.
41
       static void depositAll(int k) {
         for (int i=0; i<N; i++)</pre>
42
43
           accounts[i].deposit(k);
44
       }
45
46
       // Wait for all transfers to complete.
47
       static void awaitTransfers(Thread[] t) {
48
         try {
49
         for (int i=0; i<t.length; i++)</pre>
50
           t[i].join();
51
         } catch (InterruptedException e) {}
52
       }
53
54
       public static void main(String[] args) {
56
         log("Setting up accounts ...");
57
         setupAccounts(200);
58
59
         log("\nStarting transfers ...");
60
         Thread[] t = startTransfers();
61
         sleep(1000);
62
63
         log("\nBoss donates 1000 to all");
64
         depositAll(1000);
65
         awaitTransfers(t);
       }
67
68
       static void sleep(long ms) {
         try { Thread.sleep(ms); }
70
71
         catch (InterruptedException e) {}
72
       }
73
74
       static void log(String x) {
75
         System.out.println(x);
```

76 } 77 }