PROJECT (a general description)

CS589; Fall 2017
Due Date: **November 29, 2017**
Late project: **50% penalty**
After **December 4, 2017** the project will not be accepted.

Object-Oriented and State-Based Testing

The goal of this project is to test an *account* class that exhibits state behavior specified by the EFSM model. The source code of the class *account* is provided in a separate file.

Description of an account class:

An account requires a minimum balance of $500. If a balance is below a minimum balance in the account, a $20 fee is imposed on each transaction (withdraw, deposit). Before any account transactions can be performed on the account, operation *login()* must be issued followed by *pin()* operation. The *pin()* operation must contain the valid pin # (parameter *x*) that must be the same as the pin # provided in the *open()* operation (parameter *y*). It is allowed a maximum of *3* attempts to "provide" an invalid pin. An account can be locked. When an account is locked, no transaction can be performed on the account (except *unlock()* and *balance()* operations). For the simplicity of implementation, all deposit and withdraw transactions are performed in the whole dollar amounts only (no cents). The EFSM model for the *account* class is provided in a separate file. Notice that the EFSM model specifies the expected behavior of the *account* class.

The following operations are supported by the *account* class:

class *account*:

```
account()                   //constructor
int open(int x, int y, int z)    //sets balance to the value of x, pin number to
                            // the value of y, and an account # to the value of z
int login(int x)          // allows to login to the account, where x is an account #
int logout()              // allows to logout from the account
int pin(int x)            // provides pin # (parameter x)
int deposit (int d);      // deposits amount d to the account
int withdraw (int w);     // withdraws amount w from the account
int balance ();           // returns the value of the account balance
int lock (int x);         // locks an account where x is the lock #
int unlock (int x);       // unlocks an account when x equals to the correct lock #
```

Unless stated differently, each method (operation) returns 0 when the operation is successfully completed; otherwise, negative value -1 is returned.

TESTING

In this project the goal is to test the provided implementation (source code) of the *account* class. In order to test the *account* class, you are supposed to implement a testing environment that should contain a class driver to execute test cases. The following testing methods should be used:

1. Model-Based Testing. Use the provided EFSM model to test the *account* class. Design test cases for the *account* class so that all 2-transition sequences testing criterion (all transition-pairs) is satisfied based on the provided EFSM, i.e., all 2-transition sequences are exercised during testing.
2. Design a set of additional test cases so each default (ghost) transition in the EFSM is tested.
3. Use multiple-condition testing to design additional test cases to test predicates of conditional-statements in operations. Notice that if a predicate contains only a simple condition, the multiple-condition testing is equivalent to the branch testing for this predicate.
4. Execute all test cases designed in steps 1, 2, and 3. For each test case, determine the correctness/incorrectness of the test results. If for a given test case the results are incorrect (test failed), identify the cause of incorrectness (a defect) in the source code of the account class.

In the testing environment, you must introduce "testing-oriented" methods (in the *account* class) that will be used to watch "internal states" of an *account* object in order to determine the correctness/incorrectness of the results for test cases.

Note: As a tester, you are not supposed to modify the logic (source code) of any operation of the *account* class. In addition, notice that the source code under test may contain defects.

Sample test case:

Test #1: open(200,123,222), login(222), pin(111), pin(123), deposit(50), logout()

Notice when the EFSM model is “executed” on this test (sequence of events), the following sequence of transitions are traversed: T_1, T_2, T_3, T_8, T_{21}, T_9

The detailed description for the project report and deliverables will be presented later on.