Alex Avila
September 28, 2020
CS 3331 – Advanced Object-Oriented Programming – Fall 2020
Dr. Mejia
Programming Assignment 2

I confirm that the work of this assignment is completely my own. By turning in this assignment, I declare that I did not receive unauthorized assistance. Moreover, all deliverables including, but not limited to the source code, lab report and output files were written and produced by me alone.

1. Program Explanation

This assignment is an extension of the last assignment Programming Assignment 1 in which we needed to create a bank by reading the information of the user and storing everything in the Checking class and doing the logic of the messages in the Main file. This time in programming assignment 2 the purpose is to add more accounts for each user and more information to each individual customer and to make everything more organized we will need to create more classes.

To make certain classes not repeatable such as the Checking Saving and Credit classes that they are all Accounts we would use a technique called inheritance which will take all the attributes and functionality of an account to these three classes. Also because we will have many classes to deal with in this program then sometimes it may be hard to figure out where an error is if the program crashes, therefore, we will also use Test cases which will test individual methods around the program and if something fails the test cases will tell us what method is failing which test case which can be extremely helpful when debugging the program.

To make a good user experience of the customer using the bank system I made the user input the name of the file which should be a csv file with the information of the user formatted correctly. And then the user will choose to log in either as a Customer or as a Manager. To accomplish all this the tasks could be divided into creating all the classes containing their proper fields and relationships. Then using test cases we would need to test all the classes and methods accordingly. Knowing that everything works I then read the information of the file to create the instances of the appropriate classes and then display the menus for the customer and the manager.

2. What did I learn?

This assignment was packed with a lot of new concepts and techniques that I think would be useful once we get a job as programmers. Learning about how to construct many classes and make then have a relationship to one another made the code much cleaner and more expandable than in the previous assignment. Learning about inheritance to avoid repetition when we wanted to change something like the Account class was easy and convenient. Testing was also super helpful at the moment when something needed to be changed in the program and we wanted to

see of nothing broke while doing that we just needed to use the test cases to test everything at once.

My solution uses a few hash tables as that store references to instances and the accounts of each customer. The reason for these is to be able to access each customer by their name in constant time, however this may be a bit uneasy to read for people that are not that used to the program. Also my main has a while loop that will run until the users decides to exit the program, on this loop I called the appropriated method that calls and specific section of the menu for the user and this part of the program might have been improve because there are a lot of method calls in this loop.

I had the idea of calling each function inside the other function in order to not called the function from the for loop however, if I did that my function calls would only have keep making a huge recursion stack because the functions would never have ended. Since there were a lot of little thing to be completed for the lab the amount of time that it took me to complete it was around 20 hours

3. Solution Design

There were many requirements for this assignment such as separating the classes, reading from the file, creating the menus, test the classes and create a java do for the project, but every part could have been done individually. I first created all the classes in the way that the instructions wanted the classes and their specific methods and constructors. Since I wanted to focus mostly on the information that the csv file provided, I also created some constructors including only the given information such as constructors ignoring the interest rate since it is not giving in the information. Finally, each of these classes and function was able to be tested and see if they were able to be use in the program.

After creating all the classes I was able to read from the csv file and then use the constructors to create customers and store this customers into a hash table with their name as the key since the customers are going to sign in with their name I wanted that process to be fast. Then after finishing with all the menu functionality I created the log file and the updated bank information after all the interaction and once every coding part of the program was ready I started to create the Javadoc since I knew that nothing in the code would change that much.

4. Testing

To test my program, I used both black-box and white-box testing. As white box testing after creating all the classes and their respective logic I created many Junit files to test all the important methods on each class, this would allowed me to comfortably use my code and if something fails it would be really easy to spotted while debugging. As black box testing, I used the program many times trying to see if any of my inputs would break the program or if the output files would return what they are supposed to return. I think I tested my solutions a good amount of times but there is always the change that a little each case I did not think about might break the program.

5. Test results

For each important class in the system I created a test case class for that class in order to properly separate the test cases and easily spot the error in case something happens to my program.

The CustomerTest class testes the customer methods transfer which transfers money from one account to another. The tester tests all kind of input such as negative numbers or insufficient funds in one account.

```
public class CustomerTest {
   private Customer customer1;
   private Customer customer2;

   @Before
   public void setUp() throws Exception {
      customer1 = new Customer(
         "Mickey",
         "Mouse",
         "November",
         "1313 Disneyland",
         "(714) 781-4636",
         "000-00-0001",
         new Checking(
            1000,
            960.94
         ),
         new Savings(
            2000,
            3845.93
         ),
         new Credit(
            3000,
            -1549.33
         ));

      customer2 = new Customer(
         "Donald",
         "Duck",
         "September",
         "1313 Disneyland",
         "(714) 781-4636",
         "000-00-0002",
         new Checking(
            1001,
            1688.89
         ),
         new Savings(
            2001,
            1731.09
         ),
         new Credit(
            3001,
            -986.23
         ));
   }

   @After
   public void tearDown() throws Exception {
   }
```

```
   @Test
   public void transfer1() {
      assertTrue(customer1.transfer(customer1.getSavings(), customer1.getChecking(),
500));

      assertEquals(
         "savings should be 3345.93",
         3345.93,
         customer1.getSavings().getBalance(),
         0.001);

      assertEquals(
         "checkings should be 1460.94",
         1460.94,
         customer1.getChecking().getBalance(),
         0.001);

      assertTrue(customer1.transfer(customer1.getChecking(), customer1.getCredit(),
1000));


      assertEquals(
         "checkings should be 460.94",
         460.94,
         customer1.getChecking().getBalance(),
         0.001);

      assertEquals(
         "checkings should be -549.33",
         -549.33,
         customer1.getCredit().getBalance(),
         0.001);
   }

   @Test
   public void transfer2() {
      assertTrue(customer2.transfer(customer2.getSavings(), customer2.getCredit(),
900.89));

      assertEquals(
         830.2,
         customer2.getSavings().getBalance(),
         0.001);

      assertEquals(
         -85.34,
         customer2.getCredit().getBalance(),
         0.001);

      assertFalse(customer2.transfer(customer2.getSavings(), customer2.getCredit(),
100));

      assertEquals(
         830.2,
         customer2.getSavings().getBalance(),
         0.001);

      assertEquals(
         -85.34,
         customer2.getCredit().getBalance(),
         0.001);
   }
```

```
   @Test
   public void transfer3() {
      assertTrue(customer2.transfer(customer1.getSavings(), customer1.getChecking(),
3845.90));

      assertEquals(
         0.03,
         customer1.getSavings().getBalance(),
         0.001);

      assertEquals(
         4806.84,
         customer1.getChecking().getBalance(),
         0.001);

      assertFalse(customer2.transfer(customer1.getSavings(), customer1.getChecking(),
10));

      assertEquals(
         0.03,
         customer1.getSavings().getBalance(),
         0.001);

      assertEquals(
         4806.84,
         customer1.getChecking().getBalance(),
         0.001);
   }

   @Test
   public void transfer4() {
      assertTrue(customer1.getChecking().deposit(1000));
      assertEquals(1960.94, customer1.getChecking().getBalance(), 0.001);
      assertFalse(customer1.transfer(customer1.getChecking(), customer1.getCredit(),
1600));
      assertFalse(customer2.transfer(customer2.getSavings(), customer2.getCredit(),
1000));
   }

   @Test
   public void paySomeone1(){
      assertTrue(customer1.paySomeone(customer2, 100));
      assertEquals(860.94, customer1.getChecking().getBalance(), 0.001);
      assertEquals(1788.89, customer2.getChecking().getBalance(), 0.001);
   }

   @Test
   public void paySomeone2(){
      assertTrue(customer2.paySomeone(customer1, 1600.50));
      assertEquals(2561.44, customer1.getChecking().getBalance(), 0.001);
      assertEquals(88.39, customer2.getChecking().getBalance(), 0.001);
   }

   @Test
   public void paySomeone3(){
      assertFalse(customer1.paySomeone(customer2, 1000.50));
      assertEquals(960.94, customer1.getChecking().getBalance(), 0.001);
      assertEquals(1688.89, customer2.getChecking().getBalance(), 0.001);
   }
}
```

The Checking, Saving, and Credit files all test the deposit and withdraw methods since each of these classes inherited them from the account class and Credit is a little different since the balance is negative and the user is not allowed to withdraw from this class.

Checking

```
public class CheckingTest {
   private Checking checking;

   @Before
   public void setUp() throws Exception {
      checking = new Checking(1000, 360.24);
   }

   @After
   public void tearDown() throws Exception {
   }

   @Test
   public void deposit1() {
      assertTrue(checking.deposit(100));
      assertEquals(460.24, checking.getBalance(), 0.001);
   }

   @Test
   public void deposit2() {
      assertTrue(checking.deposit(10.50));
      assertEquals(370.74, checking.getBalance(), 0.001);
   }

   @Test
   public void deposit3() {
      assertFalse(checking.deposit(-50));
      assertEquals(360.24, checking.getBalance(), 0.001);
   }

   @Test
   public void withdraw1() {
      assertTrue(checking.withdraw(60));
      assertEquals(300.24, checking.getBalance(), 0.001);
   }

   @Test
   public void withdraw2() {
      assertFalse(checking.withdraw(-100));
      assertEquals(360.24, checking.getBalance(), 0.001);
   }

   @Test
   public void withdraw3() {
      assertFalse(checking.withdraw(500));
      assertEquals(360.24, checking.getBalance(), 0.001);
   }
}
```

Savings

```
public class SavingsTest {
   private Savings savings;
```

```
    @Before
    public void setUp() throws Exception {
        savings = new Savings(2000, 430.64);
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void deposit1() {
        assertTrue(savings.deposit(20));
        assertEquals(450.64, savings.getBalance(), 0.001);
    }

    @Test
    public void deposit2() {
        assertTrue(savings.deposit(20.50));
        assertEquals(451.14, savings.getBalance(), 0.001);
    }

    @Test
    public void deposit3() {
        assertFalse(savings.deposit(-50));
        assertEquals(430.64, savings.getBalance(), 0.001);
    }

    @Test
    public void withdraw1() {
        assertTrue(savings.withdraw(100));
        assertEquals(330.64, savings.getBalance(), 0.001);
    }

    @Test
    public void withdraw2() {
        assertFalse(savings.withdraw(-100));
        assertEquals(430.64, savings.getBalance(), 0.001);
    }

    @Test
    public void withdraw3() {
        assertFalse(savings.withdraw(500));
        assertEquals(430.64, savings.getBalance(), 0.001);
    }
}
```

Credit

```
public class CreditTest {
    private Credit credit1;

    @Before
    public void setUp() throws Exception {
        credit1 = new Credit(
            3001,
            -900.50
        );
    }

    @After
    public void tearDown() throws Exception {
```

```
    }

    @Test
    public void deposit1() {
        assertFalse(credit1.deposit(905.32));
        assertEquals(-900.50, credit1.getBalance(), 0.001);
    }

    @Test
    public void deposit2() {
        assertTrue(credit1.deposit(100.50));
        assertEquals(-800, credit1.getBalance(), 0.001);
    }

    @Test
    public void withdraw() {
        assertFalse(credit1.withdraw(100.50));
    }
}
```

Now as the for the black box testing also most the inputs work and menus showed the correct messages as well as the log file and the updated csv file. For example, if we want to sign in as Mickey and deposit $1000 in checking, then pay Donald $500 and make Donald withdraw 250 then this would be the result.

Console

After choosing deposit to checking

```
How much money do you want to deposit?
1000
Successfully deposited $1000.00 to Checking
```

After choosing Send money to person

```
Who would you like to send money to?
Donald Duck
How much money do you want to send?
500
Successfully transferred $500.00 to Donald Duck
```

After signing in as Donald and withdraw $250

```
Choose account to withdraw money from


    A. Checking

    B. Savings


A

How much money do you want to withdraw?

250

Successfully withdrew $250.00 from Checking
```

Transaction log

```
Disney Bank transaction file
Mickey Mouse deposited $1000.00 on Checking-1000. New balance: 41960.94
Mickey Mouse transferred $500.00 to Donald Duck. Checking-1000 balance: $1460.94
Donald Duck withdrew $250.00 from Checking-1001. New balance: 41938.89
```

BankOutput.csv

Mickey	Mouse	November 22 1943	000-00-0001	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1000	2000	3000	1460.94	3845.93
Donald	Duck	September 5 1983	000-00-0002	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1001	2001	3001	1938.89	1731.09
Minnie	Mouse	April 9 2010	000-00-0003	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1002	2002	3002	121.36	676.12
Goofy	Disney	July 2 1933	000-00-0004	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1003	2003	3003	1587.85	4811.16
Pluto	Disney	July 5 1941	000-00-0005	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1004	2004	3004	547.56	2316.37

Chip	Disney	January 12 1947	000-00-0006	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1005	2005	3005	1060.3	3286.32
Dale	Disney	November 28 1930	000-00-0007	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1006	2006	3006	301.9	1940.11
Cinderella	Disney	November 14 1977	000-00-0008	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1007	2007	3007	1061.07	2876.42
Ariel	Disney	September 15 1932	000-00-0009	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1008	2008	3008	19.45	3091.05
Jasmine	Disney	September 20 1932	000-00-0010	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1009	2009	3009	1745.05	4135.41
Aladdin	Disney	November 9 1992	000-00-0011	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1010	2010	3010	699.52	3517.98
Simba	Disney	July 25 1934	000-00-0012	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1011	2011	3011	1566.61	1570.65
Mufassa	Disney	July 29 1965	000-00-0013	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1012	2012	3012	958.83	3483.09
Aurora	Disney	January 21 1937	000-00-0014	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1013	2013	3013	1396.14	1644.06
Peter	Pan	November 11 1973	000-00-0015	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1014	2014	3014	1817.07	1405.99
Woody	Disney	July 2 1990	000-00-0016	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1015	2015	3015	1486.52	3904.7

Buzz	Lightyear	June 5 2003	000-00-0017	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1016	2016	3016	1969.33	3035.22
Jafar	Aladdin	December 11 1995	000-00-0018	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1017	2017	3017	803.62	3034.57
Ursela	Mermaid	January 9 1937	000-00-0019	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1018	2018	3018	855.38	2398.88
Beast	Disney	August 29 1969	000-00-0020	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1019	2019	3019	682.79	168.04
Belle	Disney	September 6 1928	000-00-0021	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1020	2020	3020	403.9	1412.94
Cruella	De Vil	September 22 1996	000-00-0022	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1021	2021	3021	966.03	4079.2
Scar	Lion	June 18 2017	000-00-0023	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1022	2022	3022	344.68	3510.62
Simba	King	March 13 2018	000-00-0024	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1023	2023	3023	1624.91	3397.5
Gaston	Disney	July 11 1943	000-00-0025	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1024	2024	3024	1181.1	414.81
Shere	Khan	September 15 1962	000-00-0026	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1025	2025	3025	1972.54	3912.97
Captaiin	Hook	December 6 1971	000-00-0027	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1026	2026	3026	997.05	1425.77

Kaa	Disney	November 10 1961	000-00-0028	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1027	2027	3027	1084.34	4665.51
Evil	Queen	February 18 1981	000-00-0029	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1028	2028	3028	955.49	4205.99
Pinocchio	Disney	July 3 1941	000-00-0030	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1029	2029	3029	1713.08	3767.19
Prince	Eric	September 28 1953	000-00-0031	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1030	2030	3030	114.21	1978.53
Quasimodo	Disney	November 4 1989	000-00-0032	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1031	2031	3031	1206.59	4790.61
Hercules	Disney	February 14 1966	000-00-0033	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1032	2032	3032	668.7	1602.05
Mulan	Disney	July 6 1966	000-00-0034	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1033	2033	3033	1096.88	285.32
Lilo	Disney	September 21 1969	000-00-0035	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1034	2034	3034	1844.34	146.14
Stitch	Disney	January 20 1954	000-00-0036	1313 Disneyland Dr Anaheim CA 92802	(714) 781-4636	1035	2035	3035	341.33	1858.86