19.01 Virtual Lecture Notes

John would like us to see what it means to work with money, so he has asked us to work with a basic Account class. We will be looking at assertions in the program, to see why they are there and what happens when they evaluate to false.

Before we begin, though, let us review assertions and exceptions. Remember that when adding assertions to your programs, you have several options. The first is to just add a comment stating the assertion. This does not actually test the assertion; it merely records the assertion for someone reading the code. Normally, you test for the assertion and then decide upon an action to perform if the assertion is false. If false, you could do nothing, or you could throw a runtime exception to tell Java to tell the user that an error has occurred. If you choose to throw an exception, you should make it include a meaningful message; otherwise, it will not be very effective.

First, let us define a class called **`account`**.

```
public class Account
{
   private NumberFormat fmt =
         NumberFormat.getCurrencyInstance();

   private final double RATE = 0.035;  // interest rate of 3.5%

   private long acctNumber;
   private double balance;
   private String name;

   //-----------------------------------------------------
   //  Sets up the account by defining its owner, account
   //   number,
   //  and initial balance.
   //-----------------------------------------------------
   public Account (String owner, long account, double initial)
   {
      name = owner;
      acctNumber = account;
      balance = initial;
   }
   //-----------------------------------------------------
   //  Returns the current balance of the account.
   //-----------------------------------------------------
   public double getBalance ()
   {
      return balance;
   }
```

```
   //-----------------------------------------------------
   //  Returns the account number.
   //-----------------------------------------------------
   public long getAccountNumber ()
   {
      return acctNumber;
   }

   //-----------------------------------------------------
   //  Returns a one-line description of the account as a
   //  string.
   //-----------------------------------------------------
   public String toString ()
   {
      return (acctNumber + "\t" + name + "\t" +
                          fmt.format(balance));
   }
}
```

At first glance, there do not appear to be any problems. But what if we follow John's rule that an account should not have a negative balance? Right now, that would mean that we should not allow an account to be created if it has a negative beginning balance. To stop the creation of an account, we need to do a check in the constructor to make sure that if a negative opening balance is provided, then no account will be created. Of course, we will return an appropriate message via an exception.

Before we do that, it must be mentioned that we do have two choices here. Either we return the error as stated before, or we choose to ignore the balance provided and set the balance to some initial value such as zero. Of course, the second option works only if we can assume accounts that are given a negative balance should be assigned a zero value. Since we do not know that, we will go with the exception option.

We will change the constructor to:

```
public Account (String owner, long account, double initial)
{
   name = owner;
   acctNumber = account;
   if (initial < 0)
     throw new RuntimeException("Initial Balance can't be less
                                    than zero");
    else
     balance = initial;
}
```

What this does is to check the value for **initial**, and if it is less than zero, a runtime exception error is thrown. A runtime exception is a generic exception that just says that an error has occurred during runtime.

Now we need a couple of other methods for our Account class. The first is:

```
   //-----------------------------------------------------
   //  Validates the transaction, then deposits the specified
   //  amount
   //  into the account. Returns the new balance.
   //-----------------------------------------------------
   public double deposit (double amount)
   {  String msg;

      if (amount < 0)  // deposit value is negative
      {
         msg = "Error: Deposit amount is invalid. ";
         msg = msg + acctNumber + "  " + fmt.format(amount));
         throw new IllegalArgumentException(msg);
      }
      else
         balance = balance + amount;

      return balance;
   }
```

In this instance, since amount is being tested and amount is an argument to the deposit method, we chose to test it to see if it was less than zero. Our assertion is that amount must be greater than or equal to zero to be a valid deposit. If it is less than zero, assertion is false. In that case, we throw an illegal argument exception. Why the difference? Why not just throw a runtime exception? First, realize that an illegal argument exception is a runtime exception. So we are throwing a runtime exception, just a more specific one. We pick the second one, as we believe it is more meaningful than the general runtime exception. When you program, you will have that choice, as well.

Now we add the last method we need:

```
   //-----------------------------------------------------
   //  Validates the transaction, then withdraws the specified
   //  amount
   //  from the account. Returns the new balance.
   //-----------------------------------------------------
   public double withdraw (double amount, double fee)
   {
      String msg;
       amount += fee;
```

```
      if (amount < 0)  // withdraw value is negative
      {

         msg = "Error: Withdraw amount is invalid.";
         msg = msg + " Account: " + acctNumber;
         msg = msg + " Requested: " + fmt.format(amount);
         throw new IllegalArgumentException(msg);
      }
      else
         if (amount > balance)  // withdraw value
                                // exceeds balance
         {

            msg = "Error: Insufficient funds.";
            msg = msg + " Account: " + acctNumber;
            msg = msg + " Requested: " + fmt.format(amount);
            msg = msg + " Available: " + fmt.format(balance);
            throw new RuntimeException(msg);
         }
         else
            balance = balance - amount;

      return balance;
   }
```

Since the first test involves an argument, we throw an illegal argument exception when there is an error. In the second case, an argument is used; however, it is compared to balance, which is not; therefore, it is better to throw the runtime exception. To help clarify this concept, think about it this way: the test must involve only arguments or literals, to make throwing an illegal argument exception better than the general runtime exception.

Finally, a program is needed to test this class. So here is a banking class:

```
public class Banking
{
   //-----------------------------------------------------------
   //  Creates some bank accounts and requests various services.
   //-----------------------------------------------------------
   public static void main (String[] args)
   {
      Account acct1 = new Account ("Ted Murphy", 72354, 102.56);
      Account acct2 = new Account ("Jane Smith", 69713, 40.00);
      Account acct3 = new Account ("Edward Demsey", 93757,
                                   759.32);

      acct1.deposit (25.85);
```

```
      double smithBalance = acct2.deposit (500.00);
      System.out.println ("Smith balance after deposit: " +
                          smithBalance);

      System.out.println ("Smith balance after withdrawal: " +
                          acct2.withdraw (430.75, 1.50));

      acct3.withdraw (800.00, 0.0);  // exceeds balance

      acct1.addInterest();
      acct2.addInterest();
      acct3.addInterest();

      System.out.println ();
      System.out.println (acct1);
      System.out.println (acct2);
      System.out.println (acct3);
   }
}
```

Note: Only the last operation, where it exceeds balance, actually generates an error. Unfortunately, the only way to test runtime exceptions, (which an illegal argument exception is, as well,) is to make errors and then run your program to test and see if the error testing works. So to fully see that the other two checks work, you will need to download the programs and modify them to make those mistakes. However, you should test for only one error at a time.

- Download the Account.java and Banking.java files to your module 19 demo programs directory and open them.
- Run the files and make sure you understand them before you continue.