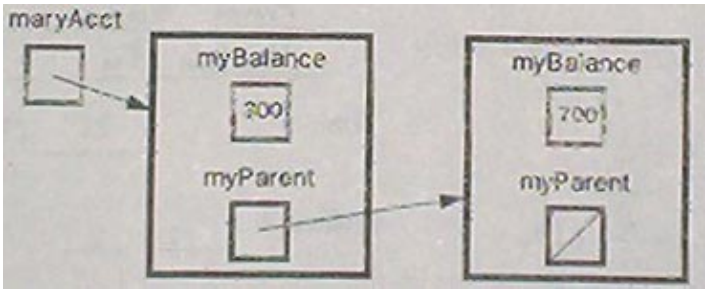
CS 61B, Spring 2001
Midterm #1
Professor Paul N. Hilfinger

Problem #0 (1 point, 1 minute)

Put your login name on each page. Also make sure you have provided the information requested on the first page.

Problem #1 (2 points, 4 minutes)

Write a single Java assignment statement that declares and assigns the variable *maryAcct* to have the value shown in the picture. Solutions of more than one statement may earn partial credit. The *Account* class definition from lab assignment 2 appears at the end of this exam.



Your solution:

_____

Problem #2 (8 points, 20 minutes)

The two parts of this problem both involve an *IslamicDate* class that is somewhat simpler than that used for homework assignment 2. An implementation of this class appears near the end of this exam.

Part a

Given below is an implementation of a *DateTester* class that could be used with the abbreviated *IslamicDate* class provided at the end of this exam. The statements in the body of the *main* method are not necessarily correct; errors include both compile-time and run-time errors.

For each statement, indicate whether it is OK or will result in an error by circling the appropriate word. You may assume that all methods in the abbreviated *IslamicDate* work correctly and that each line with an error is fixed before you consider the lines that follow it.

```
public class DateTester {
   public static void main (String [ ] args) {
      IslamicDate d1 = IslamicDate (1, 1);      OK   error
      System.out.println ("" + d1);             OK   error
      IslamicDate d2 = d1.makeTomorrow ();      OK   error
      IslamicDate d3 = null;                    OK   error
      System.out.println (d3.tomorrow ());      OK   error
   }
```

```
}
```

Part b
Give the Java code for a *DateTester* method *isLaterThan* that determines if the first of its argument dates is later than the second. The *isLaterThan* method will be used by adding the statement

```
System.out.println ("IS " + d1 + " later in the year than "
   + d2 + "? " + isLaterThan (d1, d2));
```

to the end of the (Corrected) *main* method in the *DateTester* class of part a. (As in part a, *DateTester* will be used with the abbreviated *IslamicDate* class, not the class you implemented in homework assignment 2. You aren't allowed to change the *IslamicDate* class.)

```
// REQUIRES: two nonnull IslamicDate references, representing dates
// in the same year.
// MODIFIES: nothing
// EFFECT: returns true if the first argument is later in the year
// than the second; returns false otherwise.
```

Problem #3 (5 points, 15 minutes)

Consider the following (incorrect) version of the *contains1MoreThan* method, similar to those you worked with in lab assignment 3. Assume that it's defined as part of a class *StringToCheck5*; the rest of the code from lab assignment 3 appears at the end of this exam.

```
// REQUIRES: s is not null.
// EFFECT: returns true when myString is the result of inserting
// exactly one character into s; returns false otherwise.

public boolean contains1MoreThan (String s) {
   if (s.length( )==0 {
      return true;
   } else if (myString.length( )==0 {
      return false;
   } else {
      StringToCheck5 remainder
         = new StringToCHeck5 (myString.substring(1));
      if (myString.charAt(0) == s.charAt(0)) {
         return remainder.contains1MoreThan (s.substring(1));
      } else {
         return remainder.contains1MoreThan (s);
      }
   }
}
```

Part a
Can this method crash when given an argument that satisfies the *REQUIRES* clause? Briefly explain.

Part b
Describe <u>all</u> pairs of Strings *myString* and *s* for which *contains1MoreThan* should return *false* but doesn't. For partial credit, you may give a <u>single</u> pair of Strings for which *contains1MoreThan* should return *false* but doesn't.

Problem #4 (4 points, 10 minutes)

Part a

Suppose that your lab partner has recoded the one-argument *Account* constructor to throw *IllegalArgumentException* if its argument is negative. (A listing of the *Account* class appears at the end of this exam.) Design a *main* program that tests the code. Your main program will attempt to initialize an *Account* with a negative balance and then print a suitable message about what happened.

```
public static void main (String [ ] args) {

}
```

Part b

Modify the one-argument *Account* constructor as described in part a so that it throws *IllegalArgumentException* if its argument is negative. Provide a suitable error message to initialize the exception.

```
// REQUIRES: balance >= 0.
// EFFECT: initializes a new Account object with the given balance
// and a null parent (i.e. no overdraft protection).
// Throws IllegalArgumentException if given a negative balance.
```

Framework of an abbreviated *IslamicDate* class

This class is similar to what you implemented for homework assignment 2, except that it supplies only the *toString*, *equals*, *tomorrow*, and *makeTomorrow* methods, and only one constructor.

```
public class IslamicDate {
       // REQUIRES: month is between 1 and 12, inclusive; day > 0;
       // day <= 30 if month is odd; day <= 29 if month is even.
       // EFFECT: initializes an IslamicDate object with
       // the given month and day.
       public IslamicDate (int month, int day) {
          myMonth = month;
          myDate = day;
       }

       // EFFECT: returns a String representation of this date.
       public String toString ( ) {
          return myMonth + "/" + myDay;
       }

       // EFFECT: returns true if this date represents the same
       // Islamic date as d; returns false otherwise.
       public boolean equals (IslamicDate d) {
          return (myMonth == d.myMonth) && (myDay == d.myDay);
       }

       // EFFECT: modifies this to represent the next calendar day.
       public void makeTomorrow ( ) {
          myDay++;
          if (isLegal ( )) {
```

```
            return;
          }
        myDay = 1;
        myMonth++;
        if (isLegal ( )) {
            return;
          }
        myMonth = 1;
      }

      // EFFECT: returns a date that represents the next calendar day.
      public IslamicDate tomorrow ( ) {
        IslamicDate d = new IslamicDate (myMonth, myDay);
        d.makeTomorrow ( );
        return d;
      }

      // EFFECT: returns true if this represents a legal Islamic date;
      // returns false otherwise.
      private boolean isLegal ( ) {
        // body of isLegal would go here.
      }

      private int myMonth;
      private int myDay;
}
```

Framework for the *Account* class used in lab assignment 2

```
public class Account {
      /**
       * This class represents a bank account whose current
       * balance is a nonnegative amount in US dollars, which may have
       * an auxiliary account to provide overdraft protection.
       */

      /**
       * REQUIRES: balance > 0.
       * EFFECT: Initialize an Account object with the given balance.
       */
      public Account (int balance) {
        myBalance = balance;
        myParentAccount = null;
      }

      /**
       * REQUIRES: balance > 0.
       * EFFECT: Initialize an Account object with the given balance,
       * and the given auxiliary account for overdraft protection.
       */
      public Account (int balance, Account overdraftAcct) {
```

```
            myBalance = balance;
            myParent = overdraftAcct;
        }

        /**
         * REQUIRES: amount >= 0.
         * MODIFIES: this.
         * EFFECT: Adds the given amount to this account's balance.
         */
        public void deposit (int amount) {
            myBalance += amount;
        }

        /**
         * REQUIRES: amount >= 0.
         * MODIFIES: this.
         * EFFECT: If this account's balance is at least the given amount,
         * deducts the amount from the balance and returns true.
         * If there is no overdraft protection, prints an error message
         * and returns false. Otherwise, requests a withdrawal of the given
         * amount from the parent account, and returns true if that withdrawal
         * succeeds and false otherwise.
         */
        public boolean withdraw (int amount) {
            // body of withdraw goes here
        }

        /**
         * EFFECT: Returns the number of dollars in the account.
         */
        public int balance ( ) {
            // body of withdraw goes here
        }

        private int myBalance;
        private Account myParent;
}
```

Lab3Tester.java

```
public class Lab3Tester {
    // Test teh various contains1MoreThan methods.
    public static void main (String [ ] args) throws Exception {
        ... (various calls to check)
    }

    // Call one of the StringToCheck contains1MoreThan methods
    // to see if the String tryLarger is the result of inserting
    // exactly one character into the String trySmaller.
    // The value of whichTest indicates which StringToCheck class
    // is used: whichTest == 1 means StringToCheck1, 2 means
```

```
   // StringToCheck2, etc.
   public static void check (String tryLarger, String trySmaller,
     int whichTest) {
       StringToCheck s = null;
       switch (whichTest) {
          ... (other cases go here)
       case 5:
          s = new StringToCheck5 (tryLarger);
          break;
       }
       ... (output statements that indicate the result of
              s.contains1MoreThan (trySmaller) go here)
     }
}
```

StringToCheck5.java

```
public class StringToCheck5 extends StringToCheck {

   public StringToCheck5 (String s) {
      super (s);
   }

   // Return true when myString is the result of inserting
   // exactly one character into s, and return false otherwise.
   public boolean contains1MoreThan (String s) {
      ...
   }
}
```

StringToCheck.java

```
public abstract class StringToCheck {

    // Constructor used by all of StringToCheck1, StringToCheck2, etc.
    public StringToCheck (String s) {
       myString = s;
    }

    // A method that any inheriting class has to supply.
    public abstract boolean contains1MoreThan (String s);

    // Accessible to any inheriting class.
    protected String myString;
}
```

Solutions!