The University College of the Cariboo
Computing Science Department

COURSE: COMP 113 **SEMESTER:** Fall 2004

TERM TEST #2 *Solutions*

Total Marks: 20 marks

[1 mark]

1. Write the method **Output()** above. It returns nothing and has only a single **String** parameter. Assume it uses the GUI to display the parameter.

```
public static void Output (String param)
{
   JOptionpane.showMessageDialog (null, param);
}
```

[1 mark]

2. Why is **Input()**, as shown above, a method that can not be written in Java?

The method seems to be able to accommodate multiple return types, yet with the same parameter datatype. The rules Java uses for determining which method to call, is based on the *method signature*: the method name and parameter list, but not the return type.

This makes sense, because in an assignment statement, the right-hand-side (RHS), where the method call is placed, must finish completely before placing the result in a variable during an assignment.

[1 mark]

3. Why would a class be created that has only *static* methods and instance variables?

As seen in the text book, and discussed in lecture, good examples are a utility class (such as SavitchIn) or special library (such as Math).

In these examples, the class is a collection of useful, independent methods.

[1 mark]

4. When used in an object declaration, what does the **new** keyword accomplish? Use a diagram of an example declaration and memory in your answer.

During a declaration, the **new** keyword returns a memory address that is assigned to a reference variable (which is the object identifier name). The reason it is called "new" is that it seems to accomplish two things: 1) *create a new object in the computer's memory*, and 2) *return the address of a "newly" allocated entity.*

[1 mark]

5. In a class definition, why are instance variables usually **private**?
   Through what techniques are the values of these variables visible and changeable?

   Instance variables are the unique aspect of objects, essentially the "characteristic" of the object. With such importance, the values of instance variables should be secured against unpredictable modifications *from outside the object*.

   By making them private, dedicated methods must be used to examine and change the instance variables. These methods ensure the variables always contain expected values: *accessors* (to examine, return the values) and *mutators* (to change the values).

[3 marks]

6. A programmer wishes to create a static method, or group of static methods, called **findSmaller()**, in an application program. The application just calls **findSmaller()** to find the smaller of 2 values supplied as arguments.

   The method(s) accommodate parameters of all possible datatype groups: two **int**egers, two **double**s, or two **char**acters. The return type is the same as the parameter type.

   The method does not explicitly define calls with two arguments of different type, such as the smaller of an **int**eger and a **double**.

   [2 marks]

   a) Write the method, or methods, that accomplish the above, ensuring that the appropriate datatype is also returned.

```
public static int findSmaller (int a, int b)
{
   int res = 0;
   if (a <= b)
      res = a;
   else    // b < a
      res = b;
   return (res);
}// findSmaller() - for int

public static double findSmaller (double a, double b)
{
   double res = 0;
   if (a <= b)
      res = a;
   else    // b < a
      res = b;
   return (res);
}// findSmaller() - for double

public static char findSmaller (char a, char b)
{
   char res = 0;
   if (a <= b)
      res = a;
   else    // b < a
      res = b;
   return (res);
}// findSmaller() - for char
```

[1 mark]

b) In addition, take a moment and describe what will occur in the application program, if an improper call is made to **findSmaller()** using arguments of mixed types,

i) **char** and **float** method using **double**s is called, the **char** and **float** are cast as a **double**

ii) **short** and **double** method using **double**s is called, the **short** is cast as a **double**

[5marks]

7. Define a **Rectangle** class that encapsulates the *length* and *width* of Rectangle objects (as **double** values)?

Although not required by the question, the following sets up the class definition:

```
public class Rectangle
{
   private double length;
   private double width;
   ...
```

[1 mark]

a) Write two constructors for class Rectangle: the default constructor and another constructor.
The default constructor defines a rectangle with length = 1 and width = 2.
The other constructor sets both the length and width of the rectangle to two parameter values.

```
public Rectangle ()
{
   length = 1.0;   width = 2.0;
}

public Rectangle (double lengthInit, double widthInit)
{
   length = lengthInit;   width = widthInit;
}
```

[1 mark]

b) Write the necessary accessor methods that retrieve a rectangle object's *length* and *width*.

```
public double getLength ()
{
   return (length);
}

public double getWidth ()
{
   return (width);
}
```

[1 mark]

c) Write a method **area()** that calculates and returns the area of the rectangle.

```
public double area()
{
   return (length*width);
}
```

[2 marks]

d) Write the **toString()** method that returns a *formatted* **String** with the dimensions of the rectangle alongside a small image of a rectangle (the image need not be accurate to the dimensions); for example: length 5.12 x width 7.013

```
*******
*     * W = 7.013
*******
   L = 5.120
```

```
public String toString ()
{
   DecimalFormat nice = new DecimalFormat ("0.000");
   String out="";


   out =       "*******\n";
   out = out + "*     * W = "+nice.format(width)+" \n";
   out = out + "*******\n";
   out = out + "   L = "+nice.format(length)+"\n";

   return (out);
}
```

For the questions below, use the following class definition for **BankAccount**, and the **main()** method,

```
public class BankAccount
{
      private String name;
      private double balance;
      private char   type;   // 'S'-savings, 'C'-chequeing
      private double interest;

      BankAccount (String nameClient, double balanceInit)
      {
          name = nameClient;
          balance = balanceInit;     // initial balance
          type = 'S';                // default: savings account
          interest = 2.1/12;         // deftault: 2.1% (over 12 months)
      }
      //... other methods

}

public static void main(String[] args)
{
     // create BA (BankAccounts)
   BankAccount nancyBA = new BankAccount("Nancy", 1000),
               gregBA  = new BankAccount("Greg", 4000,'C');

   //... rest of method
}
```

[1 mark1]

8. Write the member method **calcSavingsInt()** that has no parameters, and returns nothing.
   When called, this method adjusts the current object's balance by calculating the compound interest and adding it to the balance, but only if the account is of type 'S' (savings); otherwise, the balance is unchanged.

   (Hint: the formula is: new_balance = old_balance * (1+interest))

```
public void calcSavingsInt ()
{
   if (type == 'S')   // if account type is Savings
      balance = balance * (1+interest);
}
```

[2 marks]

9. Write the member method **transfer()** in the class **BankAccount** to make an electronic transfer.
   The *amount to transfer* and another recipient (a BankAccount object) are the two parameters.

   The action is to remove the *amount to transfer* from the current object, adding the value to the other object.
   The method ensures that the current object never results in a negative balance.

```
public void transfer (double amount, BankAccount other)
{
   if (amount > balance)     // if amount to transfer is greater than balance
      amount = balance;          // set amount to transfer *as* balance

   balance = balance - amount;  // reduce balance by amount

   other.balance = other.balance + amount;  // add amount to other object
}
```

[2 marks]

10. As it would be in the **main()** method, write the statement to transfer money between two objects.
    Call **transfer()** from **nancyBA** and transfer $500 to **gregBA**.

```
nancyBA.transfer(500,gregBA);
```

[2 marks]

11. Assume the class BankAccount has a method **toString()** which returns a formatted string of the BankAccount object's data.
    In the **main()**, what is the single Output() statement to call this method for the two objects, displaying the data of both in a single GUI dialog?

```
  // using the full GUI method and .toString() method
JOptionpane.showMessageDialog (null, nancyBA.toString()+", "+gregBA.toString() );

or

  // using the Output() method written in question 1
Output (nancyBA+", "+gregBA);       // which implicitly calls .toString() for each
```