

UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING
APS105 — Computer Fundamentals

Quiz #2 Solutions
November 15, 1999
6:10 p.m. – 7:50 p.m. (100 minutes)

Question 1 [10 marks]

A *hetero-literal* word pair is defined as a pair of words having no letters in common.

As examples, "TOM" and "CRUISE" form a hetero-literal word pair

"WAYNE" and "GRIZKY" do not (because of the E and Y in each word).

Write a static Java method called `isHetero()` that has two String parameters: `wordA` and `wordB`. The method should return a boolean value: `true` if `wordA` and `wordB` form a hetero-literal word pair, and `false` if they do not. You may use any of the methods in the String class. In particular, you may wish to use the method `charAt(n)` to obtain the character at position `n` in the String. You may assume that the strings contain only uppercase letters.

```
public static boolean hetero( String wordA, String wordB )
{
    for ( int i = 0; i < wordA.length(); i++ ) {
        char c = wordA.charAt(i);
        for ( int k = 0; k < wordB.length(); k++ )
            if ( c == wordB.charAt(k) )
                return false;
    }
    return true;
}
```

1

Question 2 [10 marks]

Consider the main method shown below:

```
public static void main (String[] args)
{
    double x1, y1, x2, y2, distance;
    x1 = 2.4;
    y1 = 1.5;
    x2 = 6.7;
    y2 = 3.0;
    distance = Math.sqrt( (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) );
    System.out.println( "Distance between points: " + distance );
}
```

a) [7 marks]

Write a complete class `CPoint` that could be used by the following main method to give exactly the same results as the main method show above. You should properly hide (i.e., encapsulate) any aspects of your class that are not directly used by the main program below.

```
public static void main (String[] args)
{
    CPoint p1 = new CPoint(2.4,1.5);
    CPoint p2 = new CPoint(6.7,3.0);
    double distance = p1.distanceFrom(p2);
    System.out.println( "Distance between points: " + distance );
}
```

b) [3 marks]

Add a method called `isOnBisector()` to the `CPoint` class. This method determines whether this point is on the *perpendicular bisector* of a line segment defined by two other `Point` objects, `p1` and `p2`, which are passed as parameters. That is, it should return a boolean value: `true` if this point is *equidistant* from `p1` and `p2`, and `false` otherwise. You may assume that computer arithmetic is perfectly precise.

```
// example usage of isOnBisector() method
CPoint p3 = new CPoint(5.0,1.1);
if ( p3.isOnBisector(p1,p2) )
    System.out.println( "Yes, p3 is on the bisector. " );
else
    System.out.println( "No, p3 is not on the bisector. " );
```

You should use the next page for your answer to this question.

2

Use this page for your answer to Question 2.

```
class CPoint
{
    private double x, y;

    public CPoint ( double a, double b )
    {
        x = a;
        y = b;
    }

    public double distanceFrom ( CPoint p )
    {
        double xx = x - p.x;
        double yy = y - p.y;
        return Math.sqrt ( xx*xx + yy*yy );
    }

    public boolean isOnBisector ( CPoint p1, CPoint p2 )
    {
        return (distanceFrom(p1) == distanceFrom(p2));
    }
}
```

3

Question 3 [10 marks]

Assume that a linked list has nodes that have been defined by the following:

```
class Elem
{
    public int value;
    public Elem next;
}
```

Write a Java method that will be invoked by a call of the form `x.printMerged(y)`, where `x` and `y` are both linked lists of type `Elem`. The contents of each list are already sorted in strictly ascending order. The method should print *in ascending order* all of the values contained by the union of both lists, one value per line. You may assume that the lists have no elements in common. In your solution, *do not sort or modify either list; write/call any other methods except* `System.out.println()`, *or create any new objects.*

```
class List
{
    private Elem head;

    public void printMerged(List y)
    {
        Elem p = head;
        Elem q;
        if ( y == null )
            q = null;
        else
            q = y.head;
        while ( p != null || q != null ) {
            if ( p == null ) {
                System.out.println( q.value );
                q = q.next;
            } else if ( q == null ) {
                System.out.println( p.value );
                p = p.next;
            } else if ( p.value <= q.value ) {
                System.out.println( p.value );
                p = p.next;
            } else {
                System.out.println( q.value );
                q = q.next;
            }
        }
    }
}
```

4

Question 4 [10 marks]

Lacsap's triangle is a triangular array of integers in which each row begins and ends with the row number, and each number in the interior of the triangle is the sum of two the numbers on either side in the previous row. The diagram below shows the first five rows of Lacsap's triangle.

```
      1
     2 2
    3 4 3
   4 7 7 4
  5 11 14 11 5
```

Write a static Java method `lacsap()`, that takes a single parameter `int n`. It should print the first `n` rows of Lacsap's triangle, with one row per line and one space between the numbers in a row. For simplicity, assume that `n >= 1` and do not try to print the triangle in the symmetrical form shown above.

Hint: you will need to use arrays to store your results.

```
class Lacsap
{
    public static void main(String[] args)
    {
        lacsap( StdIn.getInt() );
    }

    public static void lacsap( int n)          // a 2-D array solution
    {
        int a[][] = new int[n+1][n+1];

        // set the edges of the triangle
        for (int i = 1; i <= n; i++) {
            a[i][1] = i;
            a[i][i] = i;
        }

        // compute the interior of the triangle
        for (int i = 3; i <= n; i++)
            for (int j = 2; j < i; j++)
                a[i][j] = a[i-1][j-1] + a[i-1][j];

        // print the triangle
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++)
                System.out.print(a[i][j] + " ");
            System.out.println();
        }
    }
}
```

5

```
public static void lacsap( int n )          // another solution
{
    if( n >= 1 ) {
        int[][] ans = new int[n][n];
        ans[0][0] = 1;
        System.out.println( 1 );

        // compute each row
        for ( int i=1; i < n; i++ ) {
            final int row = i+1;
            ans[i][0] = row;
            System.out.print( ans[i][0] );

            // compute the row interior
            for ( int k=1; k < row-1; k++ ) {
                ans[i][k] = ans[i-1][k-1] + ans[i-1][k];
                System.out.print( " " + ans[i][k] );
            }

            ans[i][row-1] = row;
            System.out.println( " " + ans[i][row-1] );
        }
    }

    public static int[] lacsap( int n )          // a recursive solution
    {
        int[] ans = new int[n];
        if( n == 1 ) {
            ans[0] = 1;
            System.out.println( 1 );
            return ans;
        } else {
            int[] prevRow = lacsap(n-1);
            ans[0] = ans[n-1] = n;
            for ( int i=1; i<n-1; i++ ) {
                System.out.print( ans[i-1] + " " );
                ans[i] = prevRow[i-1]+prevRow[i];
            }
            System.out.println( ans[n-2] + " " + ans[n-1] );
            return ans;
        }
    }
}
```

6

Question 5 [10 marks]

A polynomial of the form

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_ix^i + \dots + a_nx^n$$

can be evaluated more efficiently using a technique known as *Horner's rule*. Horner's rule avoids computing the numerous x^i values explicitly, since this is redundant and slow, by rewriting and evaluating $p(x)$ in the form shown below

$$p(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + \dots + x \cdot (a_i + \dots + x \cdot (a_n) \dots)))$$

Write a **recursive** method `horner()`, which evaluates a polynomial using Horner's rule and returns the numeric value of $p(x)$. The parameter `double[] a` holds the coefficients a_i in an array, `double x` is the value at which the polynomial is to be evaluated, and `int i` is a marker you must use to keep track of your recursion. *If you do not use recursion, you will receive a maximum of 5 marks.*

```
public static double horner (double[] a, double x, int i)
{
    final int n = a.length-1;
    if( i == n )
        return a[n]; // base case
    else
        return a[i] + x * horner(a,x,i+1);
}
```