UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1 - OBJECT-ORIENTED PROGRAMMING

Friday 13th May 2011

09:30 to 11:30

Convener: J Bradfield
External Examiner: A Preece

INSTRUCTIONS TO CANDIDATES

1. **Note that all questions are compulsory.**
2. **Different questions may have different numbers of total marks. Take note of this in allocating time to questions.**
3. **This is an Open Book exam.**

Question 1

In each of parts (a)–(c) below, you will be asked to supply the body to a method inside a class. There will be a separate class for each part, named `OneA`, `OneB` and `OneC` respectively. You will be given a skeleton file for each of these classes, and the skeleton will contain the appropriate method declaration. You should add your definitions of the methods at the points marked as follows:

```
// ADD CODE HERE
```

(a) The *harmonic mean* H of the positive real numbers $x_1, x_2, ..., x_n$ is defined to be

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}.$$

For example, the harmonic mean of 1, 2, and 4 is $\frac{3}{\frac{1}{1}+\frac{1}{2}+\frac{1}{4}} = \frac{12}{7}$

In the class `OneA`, implement the static method

```
double harmonicMean(int[] nums)
```

to compute this function. You can assume that any argument array taken by this method will have non-zero length.

Expected behaviour:

```
harmonicMean(new int[] {1, 2, 3}) -> 1.6363636363636367
harmonicMean(new int[] {1, 2, 4}) -> 1.7142857142857142
harmonicMean(new int[] {3, 5, 7, 9}) -> 5.080645161290322
```

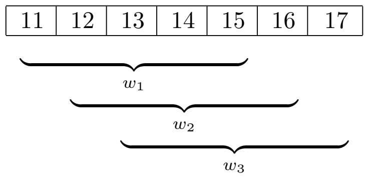
The method that you define only needs to agree with the output shown above to the first two decimal places. [*10 marks*]

(b) A *simple moving average* (SMA) is commonly used with time series data to smooth out short-term fluctuations and highlight longer-term trends or cycles. Given a series of numbers and a fixed window size w, the moving average is obtained by taking the arithmetic mean of the first w items; the start point of the window is then shifted forward one place, creating a new sequence of w items, which in turn is averaged. This process is repeated over the entire data series.

Here is an example of a 5-day moving average evolving over three days of stock-market closing prices:

Daily closing prices:	11, 12, 13, 14, 15, 16, 17		
First window of 5-day SMA:	$\frac{11+12+13+14+15}{5}$	=	13
Second window of 5-day SMA:	$\frac{12+13+14+15+16}{5}$	=	14
Third window of 5-day SMA:	$\frac{13+14+15+16+17}{5}$	=	15

Using the same example data, the following diagram illustrates how the five-day window moves (where w_1 is the window at step 1, w_2 is the window at step 2, and so on):



In the class `OneB`, implement the following static method:

```
double[] movingAvg(double[] data, int len)
```

This takes as input data an array of type `double[]`, together with an integer `len` which determines the window size of the sequence to be averaged. You can assume that $0 <$ `len` $\leq$ `N`, where `N` is the length of the input array. The method returns a new array of type `double[]` which contains the averaged data and is of length `N - (len - 1)`. That is, given an input array such as {`11.00, 12.00, 13.00, 14.00, 15.00, 16.00, 17.00`} and a `len` value of 5, `movingAvg()` should return the array {`13.00, 14.00, 15.00`} of length $7 - (5 - 1)$. It is important that your method should return a completely new array, and *not* directly modify the input array.

Expected behaviour:

```
movingAvg(new double[] {3, 3, 6, 2, 8, 9}, 3) ->
                       {4.00, 3.67, 5.33, 6.33}

movingAvg(new double[] {22.27, 22.19, 22.08, 22.17, 22.18,
                        22.13, 22.23, 22.43}, 5) ->
                       {22.18, 22.15, 22.16, 22.23}
```

Note that in these examples, we have truncated the doubles in the output array to two decimal places. However, the function that you define is not required to truncate doubles in this way. [*15 marks*]

(c) You are given a file of web spam links containing URLs such as the following:

```
www.TablaoCordobes.com
www.GROUPON.co.uk/Edinburgh+Coupons
```

All the URLs in question start with a *domain name* consisting of `"www"` followed by a dot (period) and one or more dot-separated labels such as `"TablaoCordobes"` and `"com"`. The domain name may be optionally followed by a 'path component' such as `"/Edinburgh+Coupons"`. Each URL is on a line by itself in the file, though some lines of the file contain arbitrary text and no URL.

Your task is to extract the URLs, identify the domain name, reverse it, and place it on a list. For example, given a text which contains, amongst others, the URL `"www.GROUPON.co.uk/Edinburgh+Coupons"`, your code should produce a list, one of whose elements is the string `"uk.co.GROUPON.www"`. Such a list could be used as part of a system to blacklist known spamming sites.

The task is broken down into subparts, each of which involves implementing the body of a static method within the class `OneC`.

(i) Implement the static method

```
String[] getDomainLabels(String url)
```

which given a string of the form `"A.B.C"` or `"A.B.C/D"` returns an array of type `String[] {"A", "B", "C"}`. If you want to use the `split()` method of `String` objects, you should be aware that the dot punctuation character (or period) has a special interpretation (namely as a regular expression 'wild-card') when it appears as the argument of this method. You should block this special interpretation by preceding the dot with two escape characters, i.e., use `split("\\.")`.

Expected behaviour:

```
getDomainLabels("www.GROUPON.co.uk/Edinburgh+Coupons") ->
{"www", "GROUPON", "co", "uk"}
```

[5 marks]

(ii) Implement the static method

```
void reverseArray(String[] labels).
```

Given an input array `labels` of type `String[]`, this should **modify the array in-place** so that the order of elements is reversed.

Expected behaviour:

```
String[] labels = {"www", "GROUPON", "co", "uk"};
reverseArray(labels);
// labels == {"uk", "co", "GROUPON", "www"}
```

[4 marks]

(iii) Implement the static method

```
String arrayToString(String[] labels)
```

which converts an array of type `String[]` into an output string where each element of the array appears in order, separated from following elements by a dot character.

Expected behaviour:

```
arrayToString(new String[] {"com", "TablaoCordobes", "www"})
                            ->  "com.TablaoCordobes.www"
```

[*5 marks*]

(iv) Implement the static method

`ArrayList<String> textToReversedDomains(String filename)`

whose details are described below.

To begin with, the method takes a filename as argument, and reads the file line-by-line. You should use the command `StdIn.redirectInput(filename)` to read in the file. In order to process the file line-by-line, use a loop of the form `while (!StdIn.isEmpty()){...}`. Then inside the loop, get each line of the file with the command `String line = StdIn.readLine()` (of course, you can use any variable name you like instead of `line`).

For each line in the input file, first check whether it is a URL, i.e., starts with the string `"www."`. If it does, then call `getDomainLabels()` on the line to create an array `labels` of domain labels. Next, call `reverseArray()` to reverse the `labels`, and call `arrayToString` to concatenate the elements of `labels` into a dot-separated string. Finally, this resulting string is stored as an element in a list of type `ArrayList<String>`, which should be returned by your method.

The file `spam.txt` is supplied for you to test out your methods. The first few lines of the file look as follows:

```
Hotel Regina Barcelona
www.reginahotel.com
Restaurant Barcelona
www.TablaoCordobes.com
Flights Barcelona Madrid
...
```

The corresponding initial sequence of the `ArrayList<String>` returned by your function should print out as follows (where lines have been wrapped to fit into the display):

```
[com.reginahotel.www, com.TablaoCordobes.www,
 com.vueling.www, uk.co.TravelRepublic.www,
 com.Travel-Betterdeals.Barcelona.www, ...]
```

[*5 marks*]

(d) Create a class `QuestionOneTester` with a single `main()` method. Inside `main()`, add calls to the static methods

```
OneA.harmonicMean()
OneB.movingAvg()
OneC.textToReversedDomains()
```

that you implemented for parts (a)–(c) above, in order to test that your implementations produce the correct results. (Remember that for a client program to call a static method from an external class, the method name must be qualified by the class name, as shown.) You are recommended to have your tests simply print out the value of the methods for some appropriate input arguments. Write one such test for each of the three methods specified. In the case of `OneC.textToReversedDomains()`, supply the filename `"spam.txt"` as the argument. [*6 marks*]

The files that you must submit for this question are the following:

(a) `OneA.java`

(b) `OneB.java`

(c) `OneC.java`

(d) `QuestionOneTester.java`

Question 2

This question focuses on developing data structures for 2D points, vectors and polygons. We start with points.

(a) As a simplification, we will take the coordinates of points to be integers rather than doubles. Here is the API for the `Point` data type:

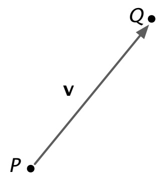
`public class Point`

`Point(int x, int y)`	*constructor*
`Point()`	*constructor*
`int getX()`	*the `x` coordinate of the point*
`int getY()`	*the `y` coordinate of the point*
`Point makeMax(Point p)`	*the maximization of the current point and `p`*
`Point makeMin(Point p)`	*the minimization of the current point and `p`*

Implement the class `Point` so that it meets the specified API. Note that the no-argument constructor for `Point` initializes both coordinates to 0.

The 'maximization' and 'minimization' methods for points require explanation: these each take as input a pair of points and return a new point that is the coordinate-wise maximum (respectively, minimum) of the inputs. That is, if point $P_0 = (x_0, y_0)$ and $P_1 = (x_1, y_1)$, then define $maximize(P_0, P_1)$ as the new point $Q = (max(x_0, x_1), max(y_0, y_1))$. Similarly $minimize(P_0, P_1) = (min(x_0, x_1), min(y_0, y_1))$. These two functions will be used below when we look at bounding boxes for polygons. [*7 marks*]

(b) Consider a vector $\mathbf{v} = \overrightarrow{PQ}$:



We will encode $\mathbf{v}$ as a pair of integers $(xdisp, ydisp)$, representing the displacement from P to Q. In 2D space, if P is the point $(0, 0)$, then $(xdisp, ydisp)$ will be the same as the coordinates of the endpoint Q. However, more generally, we also want to consider cases where the start of the vector is not $(0, 0)$. So if vector $\mathbf{v} = \overrightarrow{PQ}$ is represented as the displacement $(xdisp, ydisp)$, then the coordinates (P_x, P_y), (Q_x, Q_y) of P and Q, respectively, satisfy the equation $(Q_x, Q_y) = (P_x + xdisp, P_y + ydisp)$.

Here is the API for the `Vector` data type:

`public class Vector`

`Vector(int xdisp, int ydisp)`	*constructor*
`int getXDisp()`	*the* ***xdisp*** *of the vector*
`int getYDisp()`	*the* ***ydisp*** *of the vector*
`double magnitude()`	*the magnitude of the vector*
`Point translate(Point p)`	*translate* ***p*** *by the vector*

Implement the class `Vector` so that it meets the specified API.

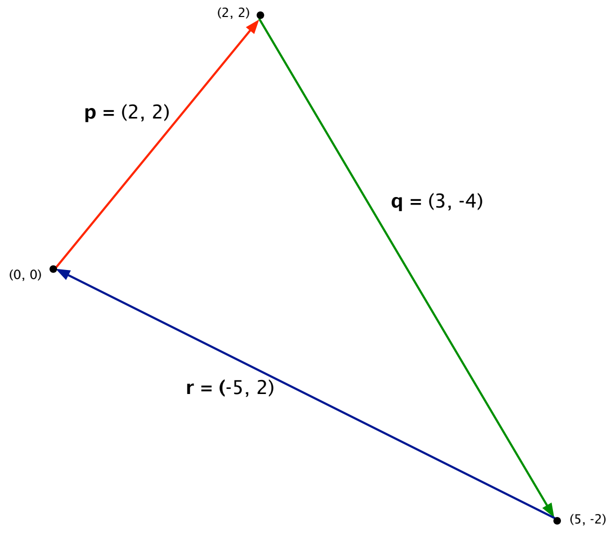
The *magnitude* $|\mathbf{v}|$ of vector $\mathbf{v}$ is defined by

$$|\mathbf{v}| = \sqrt{xdisp^2 + ydisp^2}$$

Given a point $P = (x, y)$ and vector $\mathbf{v}$ defined by $(xdisp, ydisp)$, the result of *translating* P *by* $\mathbf{v}$ is the point $Q = (x + xdisp, y + ydisp)$. [*10 marks*]

(c) A polygon K will be represented as a list of 2D vectors placed nose to tail. That is, given $K = [\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n]$, the startpoint of vector $\mathbf{v}_i$, for $0 < i \leq n$, is set as the endpoint of vector $\mathbf{v}_{i-1}$. We will restrict attention to polygons where the startpoint of $\mathbf{v}_0$, i.e., the first vector in the list composing K, has coordinates $(0, 0)$.

Recall that we are representing vectors as pairs of integers $(xdisp, ydisp)$. The polygon corresponding to the list of vectors $[(2, 2), (3, -4), (-5, 2)]$ can be visualized as follows:



Taking the startpoint of $\mathbf{p}$ as the origin $(0, 0)$, we see that the displacement of $\mathbf{p} + \mathbf{q} + \mathbf{r}$ is equal to the origin, and hence the polygon is *closed*.

Here is the API for the `Polygon` data type:

`public class Polygon`

	`Polygon(Vector[] vectors)`	*constructor*
`Vector[]`	`getVectors()`	*return the list of vectors in the polygon*
`double`	`perimeter()`	*the perimeter of the polygon*
`boolean`	`isClosed()`	*is this polygon closed?*
`double`	`area()`	*the area of the polygon*
`Point[]`	`boundingBox()`	*the maximal extent of the polygon along the x and y axes*

The task of implementing `Polygon` is broken down into sub-tasks.

A file `PolygonTester.java` is provided to help you test your code. This contains a static method `drawPolygon()` which, given a `Polygon` instance, will display the polygon using the `StdDraw` library.

(i) Define the required instance variables for the class `Polygon`, declare the constructor given in the API above, and define the instance method `getVectors()`. [*6 marks*]

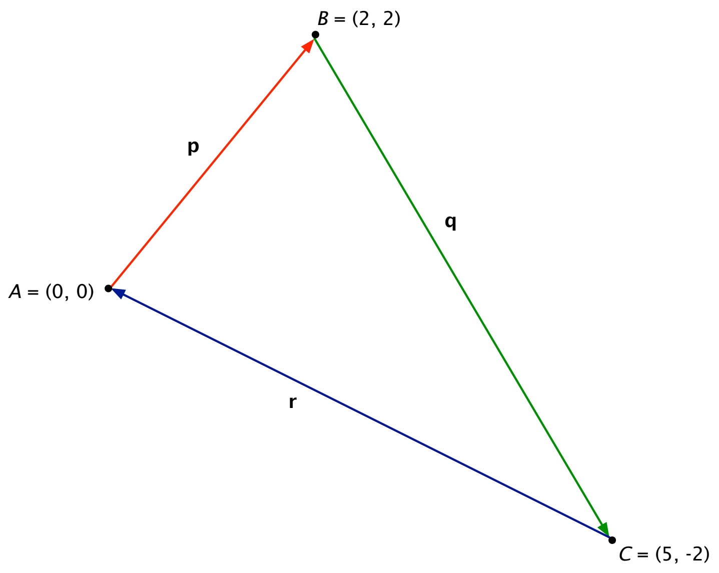
(ii) Define the instance method `perimeter()`. This returns a `double` corresponding to the sum of the magnitudes of the vectors that compose the polygon. [*7 marks*]

(iii) Define the instance method `isClosed()`. This returns `true` if the endpoint of the final vector in the list of vectors is equal to the startpoint of the initial vector, and returns `false` otherwise. Hint: use the `translate()` method of the `Vector` class, and remember that we are assuming that the startpoint of the initial vector is $(0, 0)$. [*7 marks*]

(iv) Define the instance method `area()`. Let $K = [\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n]$ be a closed (non-self-intersecting) polygon, and let $P_0, P_1, \ldots, P_{n+1}$ be the vertices of K, corresponding to the startpoints and endpoints of vectors in $[\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n]$. Since K is closed, $P_{n+1} = P_0$. Then the area A of K is defined by

$$A = \frac{1}{2}\sum_{i=0}^{n}(x_i y_{i+1} - x_{i+1} y_i). \tag{1}$$

To see what's going on here, consider again the example polygon we used earlier, where now the vertices have been labeled:



Given these input values, we can calculate A as follows:

$$\begin{aligned} A &= \frac{1}{2}((A_x \times B_y - B_x \times A_y) + (B_x \times C_y - C_x \times B_y) + (C_x \times A_y - A_x \times C_y)) \\ &= \frac{1}{2}((0 \times 2 - 2 \times 0) + (2 \times -2 - 5 \times 2) + (5 \times 0 - 0 \times -2)) \\ &= \frac{-14}{2} \end{aligned}$$

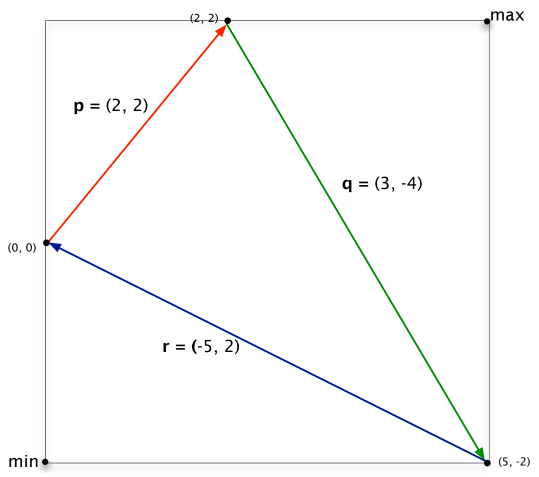
Thus, given some vector $\mathbf{v} = \overrightarrow{P_i P_{i+1}}$ in the list of vectors composing the polygon, the terms of the sum in equation (1) correspond to

$$\mathtt{P_i.getX()} * \mathtt{P_{i+1}.getY()} - \mathtt{P_{i+1}.getX()} * \mathtt{P_i.getY()}.$$

As we saw in the example, according to (1) the value A will be negative if the vertices of the polygon are arranged in clockwise order. However, define your method so that it returns the **absolute value** of A.

If the polygon is not closed, `area()` should return 0.0. *[10 marks]*

(v) Define the instance method `boundingBox()`. This returns an array `limits` of type `Point[]`, with length 2. Given a list `pointList` of points corresponding to vertices of the polygon, `limits[0]` is the minimization point of `pointList` while `limits[1]` is the maximization point of `pointList`. As an example, these points are labeled *min* and *max* in the figure below, and have coordinates $(-5, -4)$ and $(3, 2)$ respectively.



[*3 marks*]

The files that you must submit for this question are the following:

- `Point.java`
- `Vector.java`
- `Polygon.java`

Final Checklist

Here is a complete list of all the files required for this exam:

```
OneA.java
OneB.java
OneC.java
QuestionOneTester.java
Point.java
Vector.java
Polygon.java
```