UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08014 INFORMATICS 1 - OBJECT-ORIENTED PROGRAMMING

Wednesday 14th May 2014

14:30 to 17:30

INSTRUCTIONS TO CANDIDATES

1. **Note that all questions are compulsory.**
2. **Remember that a file that does not compile, or does not pass the simple JUnit tests provided, will get no marks.**
3. **This is an Open Book exam.**

Convener: J. Bradfield
External Examiner: C. Johnson

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

Question 1

In this question you will implement part of a simple runners' training application.

(a) Implement class `Length`. To allow for some runners measuring their runs in distance units (kilometres or miles) and others in time units (minutes), this has a private integer attribute `number` and a private `String` attribute `units`, to represent the length of a run by the number of whatever unit is being used. Your code must ensure that `number` is non-negative, and that `units` is one of "km", "miles", "minutes". (Do *not* use an enum.)

Your class must have a zero-argument constructor that sets `number` to 0 and `units` to "km", and getters and setters for its attributes. If a setter is given an argument that is not acceptable, it should simply do nothing: that is, it must not change the object, but it must not report this in any way as a mistake.

Your class must also have a method `convert` taking a `String newUnits` and a `double rate` and returning nothing. The parameter `rate` represents the number of *old* units that are equivalent to 1 *new* unit. If `newUnits` is one of the acceptable units, the `convert` method must change the `units` field of the object to be `newUnits` and change the `number` field to be the appropriately converted amount (rounded to the nearest integer, with .5s rounded up in the usual way).

Again, if the `newUnits` argument is not acceptable, the method should simply do nothing. You may assume, and need not check, that `rate` > 0.

Finally the `toString` method should return "40 minutes" if `number` is 40 and `units` is "minutes", and so on (with a single space character between the number and the units, as shown).

In summary `Length` has this public interface, with behaviour as explained above:

```
public class Length
-------------------------------------------
        Length()
    int getNumber()
   void setNumber(int n)
 String getUnits()
   void setUnits(String s)
   void convert(String newUnits, double rate)
 String toString()
-------------------------------------------
```

[25 marks]

(b) You are given, in file `ExerciseSession.java`, the code of a class `ExerciseSession` with the following public interface:

`public class ExerciseSession`	
`ExerciseSession(String type, String place)`	*constructor*
`String getType()`	*getter*
`void setType(String type)`	*setter*
`String getPlace()`	*getter*
`void setPlace(int place)`	*setter*
`String toString()`	*return "Type (place)" e.g. "Running (Edinburgh Meadows)"*

Your task is to implement the class `GymSession` to extend `ExerciseSession`. This class should have an additional private attribute `machines` of type `HashMap<String, Length>`, representing which gym machines were used in the session and how much, in terms of the Length class you implemented in part (a). In the constructor of `GymSession` you must initialise `machines` with an initial capacity for 8 machines. Your constructor must, like the constructor of `ExerciseSession`, take a `String type` and a `String place`, and must pass these to `ExerciseSession`'s constructor.

Remember that you will need the line

```
import java.util.HashMap;
```

at the top of your file.

Provide a public method `addMachine`, taking a `String` representing the name of the machine and a `Length` representing how much it is to be used, that adds a machine to `machines`, if necessary replacing the existing machine of that name. You should assume that the `Length` object is being maintained elsewhere, e.g., updated if the intended length changes; make sure you keep a reference to, not a copy of, the given `Length` object.

Override the method `toString()` so that it:

(a) first gives the type and place of the exercise session exactly as `ExerciseSession`'s `toString()` does;

(b) then on a new line by itself gives "Machines:";

(c) then lists the machines, each on a new line, by giving the key string of the machine, then one space, then the length as given in the corresponding value.

Take care that `toString()`'s output does *not* end with a new line.

For example, if the gym session is "Gym" at "The Pleasance", and the machines are "Treadmill" and "Cross-trainer", each with length 10 minutes, then sending `toString()` to an object representing this gym session should result in:

```
Gym (The Pleasance)
Machines:
Treadmill 10 minutes
Cross-trainer 10 minutes
```

(or:

```
Gym (The Pleasance)
Machines:
Cross-trainer 10 minutes
Treadmill 10 minutes
```

– the order in which machines are listed does not matter).

In summary `GymSession` has this public interface, with behaviour as explained above:

```
public class GymSession
-----------------------------------------------------
        GymSession(String type, String place)
   void addMachine(String name, Length length)
 String toString()
-----------------------------------------------------
```

[*25 marks*]

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

- `Length.java`
- `GymSession.java`

Question 2

Let $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ be lists of integers. In this question you will write code to test two inequalities: the Cauchy-Schwarz inequality and the Arithmetic Mean-Geometric Mean inequality. (It does not matter whether you already know what these are or not.)
Note: it is expected that you may want to use one or more methods of the Math class that you may not have used before. You should consult the Java documentation whenever you need to. **Hint:** you are likely to find a method that takes one number to the power of another useful.

(a) Create a public class `Inequalities`.

In the class `Inequalities`, implement the public static method

```
int dotProduct(int[] a, int[] b)
```

that, if `a` and `b` have the same length n, returns $\Sigma_{i=1}^{i=n} a_i b_i$, that is, the sum of the products of the array elements taken in order. (You might remember this as the dot product of `a` and `b` regarded as vectors, hence the name of the function, but you are not required to remember any Linear Algebra for this question.)

If `a` and `b` do not have the same length, return 0.

Expected behaviour:

```
dotProduct(new int[] {2, 1}, new int[] {3, 4})
```

should return 10, because this is $2 \times 3 + 1 \times 4$.

```
dotProduct (new int[] {2, 1, 3}, new int[] {3, 4})
```

should return 0, because the input arrays have different lengths. [*10 marks*]

(b) In the class `Inequalities`, implement the public static method

```
int[] concatenate (int[] a, int[] b)
```

that does not modify its arguments, but returns a new array whose elements are the elements of the first array in order, followed by the elements of the second array in order.

Expected behaviour:

```
int[] a = new int[] {5, 6};
int[] b = new int[] {1, 2, 3};
concatenate(a, b);
```

should return $\{5, 6, 1, 2, 3\}$. [*10 marks*]

(c) In the class `Inequalities`, implement the public static method

```
boolean cs(int[] a, int[] b)
```

that tests the Cauchy-Schwarz inequality on `a` and `b`. That is, it returns `true` if, and only if, both

- `a` and `b` have the same length, and
- lhs is less than or equal to rhs, where:
  - lhs is the square of (the `dotProduct` of `a` with `b`)
  - rhs is the product of (the `dotProduct` of `a` with itself) and (the `dotProduct` of `b` with itself).

Otherwise it must return `false`.

Expected behaviour:

```
cs (new int[] {2, 1}, new int[] {3, 4})
```

should return `true`, because both arrays have length 2, lhs is $((2\times3)+(1\times4))^2 = 100$, and rhs is $(2^2 + 1^2) \times (3^2 + 4^2) = 125$. [*10 marks*]

(d) In the class `Inequalities`, implement the public static method

```
boolean amgm(int[] a)
```

that tests the Arithmetic Mean-Geometric Mean inequality on `a`. That is, it returns `true` if, and only if, the arithmetic mean of the elements of `a` is greater than or equal to the geometric mean of the elements of `a`. Recall that:

- the arithmetic mean of n numbers is their sum, divided by n;
- the geometric mean of n numbers is the nth root of their product, that is, their product taken to the power $1/n$.

Expected behaviour:

```
amgm (new int[] {4, 9})
```

should return `true`, because the arithmetic mean is $(4 + 9)/2 = 6.5$ while the geometric mean is $\sqrt{(4 \times 9)} = 6$ [*10 marks*]

(e) Finally, in the class `Inequalities`, implement the public static `main` method that will exercise your methods. The user will give a list of arguments on the command line, all of which will be integers. **You are not required to provide any error handling in this part.**

The first argument, say n, is the length of each of two arrays. The next n arguments are the elements of the first array. The remaining n arguments are the elements of the second array. For example, if the user runs the program as

```
Inequalities 2 2 1 3 4
```

then you will be investigating the arrays `[2,1]` and `[3,4]`.

Once your `main` method has parsed the input and built the arrays, it must do two calculations and print their results to standard output. Use exactly the given format, and note that there is a single space after each colon.

(a) Calculate `cs` on the two given arrays, and print the result: in the example of user input given above we expect:

```
CS held: true
```

(b) Calculate `amgm` on the concatenation of the two given arrays, and print the result: in the example, we expect:

```
AMGM held: true
```

If you have done everything correctly, you will find that you get `true` in both cases provided the elements of the arrays are non-negative. If you have negative elements in the arrays, AM-GM may fail. [*10 marks*]

The file that you must submit for this question is:

- `Inequalities.java`

Final Checklist

You are reminded again that any file that does not compile, or does not pass the simple tests provided to you, will get no marks. You are advised to check this just before each submission you make.
Here is a complete list of all the files that you must submit for this exam:

```
Length.java
GymSession.java
Inequalities.java
```