CS1101 AY2007/8 Semester 1
Practical Exam (PE)

Important notes

- This is an **Open-Book** examination.
- Time allowed: **3 hours**.
- There are **2** exercises in this PE. Exercise 1 constitutes 30%; Exercise 2 70%.
- You are advised to spend some time thinking over the tasks to design your algorithms, instead of writing the programs right away.
- Manage your time well! Do not spend excessive time on any exercise or sub-task.

Exercise 1: Amnesty (Amnesty.java): 30% of total marks

A king grants partial amnesty to *n* prisoners, locked in individual prison cells that are numbered from 1 to *n*. To determine who is freed, the jailor makes *n* passes over the cells. First, the jailor unlocks each of the *n* cells on the first pass. Then starting with the second cell, he turns the key in every second cell on the second pass. On the third pass, he starts with the third cell, and turns the key in every third cell, and so on. In the last pass, the jailor turns the key in the *n*th cell. If a cell is unlocked, a turn of the key will lock it (and vice versa). When the jailor is finished, the prisoners can leave if their doors are unlocked.

Write a program **Amnesty.java** that reads in the number of prisoners in a jail and outputs the cell number of prisoners who are freed. You do not need to use ArrayLists but can use them if you choose.

Input Format

The input consists of a line that contains *n*, a positive integer (greater than or equal to 3) that indicates the number of prisoners in the jail.

Sample Runs

A sample run is shown below. The outputs are shown in **bold**.

```
$ javac Amnesty.java
$ java Amnesty
Enter the number of prison cells:
8
Prisoners at the following cells are freed:
1 4

$ java Amnesty
Enter the number of prison cells:
18
Prisoners at the following cells are freed:
1 4 9 16
```

A template program **Amnesty.java** is provided below. It is also available in your working directory.

```
// Amnesty.java
// Author:
// Matriculation number:
// Lecture group (X, Y or Z):
// Discussion group (1-8):

/**
 * This is the definition of the Amnesty class.
 */

import java.util.*;

class Amnesty {

    public static void main( String[] args ) {

        fill in your code here

    } // main

} // Amnesty
```

Marking Scheme: Total of 30 marks

1. Can the program be compiled? 5 marks
2. Use of array and correct processing of array? 5 marks
3. Style (including your particulars, use of constant, comments, spacing/indentation, choice of variable names): 5 marks
4. Correctness of result: 15 marks (5 test data sets; 3 marks for each data set.)
5. The marks for this task will constitute **30%** of the total marks.

Exercise 2: Taxi Estimator (Taxi.java): 70% of total marks

You are working with Singapore Tourist Bureau, and are helping their planning agency estimate the cost of (greatly simplified) taxi fares for groups of tourists and professionals. You are to create a program that can give an estimate of the taxi fare from one location to another given the number of people.

Another program has already created another class, `DistanceEstimator` which contains the following static method:

```
int estimateDistance(String origin, String destination)
throws UnknownLocationException
```

This method returns a distance in meters *m*, given `Strings` for the origin and destination. If the input arguments to the method are not known locations, the method will throw an exception.

Once you have obtained the distance, your program should calculate the fare schedule:

The first 1000 m or less: S$2.50
S$0.10 for every 200m
After 10 km – S$0.10 for every 175m

Write a program **Taxi.java** that reads in input consisting of the number of passengers, their origin and destination and calculates the estimated fares. There are 3 types of estimation of the cost. A template program **Taxi.java** is provided on the next page. It is also available in your working directory.

Type A.

You are to calculate the total estimated cost of the taxi ride(s) given the inputs of the number of people, and their source and destination using standard cabs. Standard cabs hold 4 passengers each.

Type B.

Calculate the total estimated cost of the taxi ride(s) using only Cab++'s, each of which hold 6 passengers. Note that fare schedule for the first 1000 m is different for Cab++'s. The fare after the first 1000 m remains the same as for standard Cabs.

The first 1000 m or less: S$5.00

Type C.

Calculate the lowest estimated cost of the taxi ride(s) using any combination of standard cabs and Cab++'s.

Input Format

Each line of input will specify a trip that requires an estimate. Each line is given as a 4-tuple, where each field is separated by a single space. The first field states which type of estimation is to be used, the second states the number of passengers *p*, the third the origin, the fourth the destination. All origins and destinations will be single alphabetic words with no space or punctuation. All test cases will use origins and destinations that are defined and known to the DistanceEstimator class. Your program terminates when a single '0' is entered on a separate line.

Note that in each problem you will not have more than 100 passengers, that is ($1 <= p <= 100$).

Sample Runs

Given that the static function calls and their return values:

```
estimateDistance(NUS,Vivocity) = 5000
estimateDistance(NUS,ChangiAirport) = 30000
```

A sample run is shown below. The outputs are shown in **bold**.

```
$ javac Taxi.java
$ java Taxi
Enter a trip specification: a 2 NUS Vivocity
Cost using standard cabs: 4.50

Enter a trip specification: b 2 NUS Vivocity
Cost using Cab++'s: 7.00

Enter a trip specification: c 2 NUS Vivocity
Lowest possible cost: 4.50

Enter a trip specification: a 10 NUS ChangiAirport
Cost using standard cabs: 55.50

Enter a trip specification: b 10 NUS ChangiAirport
Cost using Cab++'s: 42.00

Enter a trip specification: c 10 NUS ChangiAirport
Lowest possible cost: 39.50

Enter a trip specification: 0
Good-bye
```

When the terminating input line of '0' is found, the program should print "Good-bye" with a new line and exit.

Marking Scheme: Total of 70 marks

1. Can program be compiled? 5 marks
2. Correct use of helper class? 5 marks
3. Style (including your particulars, use of constant, comments, spacing/indentation, choice of variable names): 10 marks
4. Correctness of result: 50 marks (10 test data sets @ 5 each)
5. The marks for this task will constitute **70%** of the total marks.

```
// Taxi.java
// Author:
// Matriculation number:
// Lecture group (X, Y or Z):
// Discussion group (1-8):

/**
 * This is the definition of the Taxi class.
 */

import java.util.*;

class Taxi {

    public static void main( String[] args ) {

        fill in your code here

    } // main

} // Taxi
```