

PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science
DECEMBER EXAMINATIONS 2002

CSC 108H1 F
St. George and Erindale Campuses

Duration — 3 hours

Aids allowed: None.

Student Number: _____

Last Name: _____

First Name: _____

Lecture Time (circle one): 9am 10am 1pm 2pm 6pm

Instructor (circle one): Gary Baumgartner Alex Budanitsky Paul Gries

Do not turn this page until you have received the signal to start.
(In the meantime, please fill out the identification section above,
and read the instructions below.)

This examination consists of 7 questions on 16 pages (including this one). *When you receive the signal to start, please make sure that your copy of the examination is complete.* If you need more space for one of your solutions, use one of the blank pages near the end of the exam and *indicate clearly the part of your work that should be marked.*

Unless otherwise specified, you are allowed to use helper methods and comments are not required. However, comments can be helpful when assigning part marks.

Below are the abbreviations that you may use:

Abbreviation	Stands for
S.o.p	System.out.print
S.o.pln	System.out.println
I.pI	Integer.parseInt
D.pD	Double.parseDouble
JOP.sID	JOptionPane.showInputDialog
BR	BufferedReader
BR.rL	BufferedReader.readLine

1: _____/10
 # 2: _____/10
 # 3: _____/25
 # 4: _____/10
 # 5: _____/10
 # 6: _____/15
 # 7: _____/10
 TOTAL: _____/90

Good Luck!

Question 1. [10 MARKS]**Part (a)** [2 MARKS]

Write your student number **legibly** in the space provided on every page of this exam.

Part (b) [4 MARKS]

Write a **static** method **swap** that has two **String** parameters and swaps the values in those two parameters.

Part (c) [4 MARKS]

No matter what you do in the method, the method will not actually be useful to anyone. Briefly explain why.

Question 2. [10 MARKS]

Complete the following two methods. You don't need to catch `IOExceptions`, but you should declare them to be thrown in the method headers.

Part (a) [5 MARKS]

```
/**
 * Read a series of Strings from the file named f and return the shortest String,
 * or return null if the file is empty.
 * Precondition: f != null, and no two Strings in f have the same length.
 */
public static String getShortest(String f)
```

Part (b) [5 MARKS]

```
/**
 * Read a series of Strings from the file named f and return the second-shortest
 * String, or null if there is no second shortest.
 * Precondition: f != null, and no two Strings in f have the same length.
 */
public static String getSecondShortest(String f)
```

Question 3. [25 MARKS]

In Assignment 2, you wrote a program that had three classes: `GruelShop`, `Home`, and `Tenant`. `Tenants` knew which `Home` they lived in, and `Homes` kept track of their nearest `GruelShop`. In this question, you will add code to `Home` and rewrite `Tenant`'s `move` method according to later instructions. Do not modify `GruelShop`.

Below and on the opposite page are the relevant portions of the A2 solution. Unlike on A2, class `GruelShop` only has a price per gram (and no name or total income), class `Home` only has the number of grams in the fridge and the nearest `GruelShop`, and class `Tenant` only has a name, a `Home`, and an amount of money.

```
public class GruelShop {
    private double pricePerGram;

    public GruelShop(double g) {
        pricePerGram= g;
    }

    public int sellGruel(double d) {
        return (int) (d / pricePerGram);
    }

    public double getPricePerGram() {
        return pricePerGram;
    }
}
```

```
public class Tenant {
    private String name;
    private Home home;
    private double money;

    public Tenant(String n, double m, Home h) {
        name= n;
        home= h;
        money= m;
    }

    public String getName() { return name; }
    public Home getHome() { return home; }

    public boolean equals(Object o) {
        return ((Tenant) o).name.equals(name);
    }

    public void earnMoney(double d) {
        money += d;
    }

    public double getMoney() {
        return money;
    }

    public void move(Home h) {
        home= h;
    }

    public void buyGruel() {
        home.adjustGruelBy(
            home.getNearestGruelShop()
                .sellGruel(money));

        money= 0;
    }
}
```

```
public class Home {
    private int grams;
    private GruelShop nearestShop;

    /** This Home's list of Tenants. */

    public Home(String a, GruelShop gs) { nearestShop= gs; }
    public void adjustGruelBy(int g) { grams += g; }
    public int getGruelInFridge() { return grams; }
    public GruelShop getNearestGruelShop() { return nearestShop; }
}
```

Part (a) [2 MARKS]

Currently **Tenants** do not know the other **Tenants** in their **Home**. In the blank space provided in class **Home** above, declare and initialize a **Vector** that keeps track of a **Home's Tenants**. Pick a good name.

Part (b) [4 MARKS]

Write method **moveInto**, which belongs to class **Home**.

```
/** Move Tenant t into this Home. Precondition: t is not living in this Home. */
```

Part (c) [4 MARKS]

Write method **moveOutOf**, which belongs to class **Home**.

```
/** Move Tenant t out of this Home. Precondition: t is living in this Home. */
```

Part (d) [3 MARKS]

Write method `tenantIterator`, which belongs to class `Home`. Method `tenantIterator` returns an `Iterator` for the `Tenants` in the `Home`.

```
/** Return an Iterator of this Home's Tenants. */
```

Part (e) [4 MARKS]

Below, rewrite `Tenant`'s `move` method to make the `Tenant` tell their old `Home` that they are moving out, tell their new `Home` that they are moving in, and then move into the new one.

```
public void move(Home h) {
```

```
}
```

Part (f) [8 MARKS]

On the opposite page, write a class `LazyTenant` that is a subclass of `Tenant`. When told to `buyGruel`, a `LazyTenant` first checks whether there are any normal (non-lazy) `Tenants` living with them, and if the `LazyTenant` finds any non-lazy `Tenants`, tells one of them to `buyGruel`. If no non-lazy `Tenants` live with them, the `LazyTenant` buys gruel like a normal `Tenant`. Include a constructor. Write full JavaDoc for the class (note that you can do this even if you can't write a single line of code).

Hint: use the `LazyTenant`'s `Home`'s `Iterator`.

Write your `LazyTenant` class here.

Question 4. [10 MARKS]

Grezelky is a written human language. It looks a lot like English, but there are a few differences:

- All letters are lowercase.
- There is no punctuation.
- To convert from English to Grezelky, all consonants are repeated and an `o` is placed between the repeated pair. Vowels remain unchanged. (Vowels are the letters `a`, `e`, `i`, `o`, and `u`.)

For example, if the English is

```
"toronto is freezing"
```

then the Grezelky version is

```
"totororonontoto isos fofroreezozinongog"
```

A popular pastime among CS undergraduates at the University of Toronto has been to write Java code to convert English phrases to Grezelky phrases. Rumours abound that up to 700 students work on this at the same time!

Part of this English → Grezelky translator program is a method that accepts a `String` parameter that contains a message in English, and returns a `String` containing the corresponding Grezelky phrase.

Write that translator method below. You may assume that the parameter will only have lowercase letters and spaces; you do **not** have to write code to handle punctuation or capital letters.

Hint: declare a variable with value `"aeiou "` (notice the space at the end) and use `indexOf` on it to find out whether a character in the parameter is a vowel or a space, and therefore not a consonant.

Question 5. [10 MARKS]

In lecture, we looked at a class that modeled a library book holding. In this question, you'll consider a simple variation on that. A `LibraryHolding` has a title, a publisher, a year, and a number of copies owned by the library. It may also have an author, an editor, both an author and an editor, or neither (the absence(s) marked by `null`). Two `LibraryHoldings` are considered the same if all of their fields are equal, except possibly for the number of copies.

Below, write an `equals` method for this class that overrides `Object`'s `equals` method. You are allowed to write a single helper method for it.

```
public class LibraryHolding {
    private String title;
    private String author;
    private String editor;
    private String publisher;
    private int year;
    private int numCopies;
```

```
}
```

Question 6. [15 MARKS]

In a strange far-away land, a taxpayer pays both federal and provincial taxes. The exact amount depends on both taxable income and the number of dependent children. A taxpayer has the following personal information: a name, a Social Insurance Number, a taxable income, and a number of dependent children. Once created, a taxpayer's information is never modified.

Below are the income brackets a taxpayer may belong to; federal and provincial taxes are based on these brackets. The government changes the exact numbers every year; make sure they only need to change your program in one place to do this.

- Bracket 1: \$0 – \$29,850.00
- Bracket 2: \$29,850.01 – \$59,700.00
- Bracket 3: \$59,700.01 and up

The government often needs to get a **String** representation of a taxpayer, including the federal and provincial taxes owed. They don't ever need to access the name, income, Social Insurance Number, or number of dependent children separately, but they do need to be able to ask for the federal and provincial taxes as separate pieces.

Federal tax is calculated as follows:

- If the taxable income is in income bracket 1, the federal tax is 16% of the taxable income.
- If the taxable income is in income bracket 2, then the federal tax is 16% of \$29,850.00 plus 24% of the remaining income.
- If the taxable income is in income bracket 3, then the federal tax is (16% of \$29,850.00) + (24% of \$29,850.00) + (29% of the remaining income).

Provincial tax is calculated as follows:

- 41% of the federal tax, computed above.
- Provincial tax is reduced by the number of dependent children: \$328 per dependent child. If this deduction is greater than the basic provincial tax, then the provincial tax is zero, otherwise the provincial tax equals the basic provincial tax minus the deduction.

On the next page, write a class that describes a taxpayer and captures all the above pieces of information.

Write your class that describes a taxpayer on this page.

Question 7. [10 MARKS]

Write a method that takes a positive integer n and returns the following two dimensional array of integers; notice how the rows rotate:

1	2	3	..	n-1	n
2	3	4	..	n	1
:	:	:	:	:	:
n-1	n	1	2	..	n-2
n	1	2	..	n-2	n-1

This page is provided for rough work and any answers that didn't fit.

This page is provided for rough work and any answers that didn't fit.

Short Java API descriptions (all methods are public)**Object:**

```
boolean equals(Object o) // = "this Object is equal to o"  
String toString() // = a String representation of this Object
```

Integer:

```
static int parseInt(String s) // = s's value, as an int
```

Double:

```
static double parseDouble(String s) // = s's value, as a double
```

Boolean:

```
static boolean parseBoolean(String s) // = s's value, as a boolean
```

String:

```
String substring(int i, int j) // = the letters from i (inclusive) to j (non-inclusive)  
String substring(int i) // = the letters from i (inclusive) to the end  
int indexOf(String s) // = the index of s in this String; -1 if s is not a substring  
int indexOf(String s, int i) // = index of s after index i (inclusive); -1 if s not found  
int length() // = the number of characters in this String  
boolean equals(String s) // = this String has the same contents as s
```

JFrame:

```
JFrame() // An empty window with no title, not visible on the screen  
JFrame(String s) // An empty window with title s, not visible on the screen  
Container getContentPane() // this JFrame's content pane  
int getWidth() // this JFrame's width  
int getHeight() // this JFrame's height  
int getX() // this JFrame's horizontal coordinate  
int getY() // this JFrame's vertical coordinate  
void setSize(int w, int h) // set this JFrame's size to w wide and h high  
void setTitle(String s) // set this JFrame's title to s  
void setLocation(int x, int y) // set this JFrame's location to (x, y)  
void show() // make this JFrame visible  
void hide() // make this JFrame invisible  
void pack() // resize this JFrame according to its content pane's contents
```

JOptionPane:

```
static String showInputDialog(String m) // get input from the user, prompting with m  
static void showMessageDialog(Component c, Object message) // alert the user, with m
```

JTextArea:

```
JTextArea(int r, int c) // text area with r rows and c columns, and no initial text  
JTextArea(String s, int r, int c) // text area with r rows, c columns, and initial text s  
void setText(String s) // set the text in this text area to s  
void append(String s) // append s to the text in this text area  
String getText() // = the text in this text area
```

JButton:

```
JButton() // A button with no name  
JButton(String s) // A button named s  
String getText() // = this JButton's name
```

BufferedReader:

```
BufferedReader(Reader in) // Create a reader reading from 'in'  
// Return the next available line in the BufferedReader, or null if at end  
String readLine()
```

FileReader:

```
FileReader(String f) // Create a reader reading from a file named f
```

Container:

```
// add item i in location specified by loc, which is "North", "South", "East", "West", or  
// "Center". For example, add(new JButton("A"), "East") adds a new JButton in the east  
add(Component i, String loc)
```

Math:

```
static double abs(double a) // = the absolute value of a  
static double pow(double a, double b) // = a^b  
static double min(double a, double b) // = minimum of a and b  
static double max(double a, double b) // = maximum of a and b
```

Vector:

```
// add o at index i, shifting this[i..] to make room; always return true  
void add(int i, Object o)  
boolean add(Object o) // add o at the end; always return true  
void clear() // remove all elements  
boolean contains(Object o) // = "o is an element of this Vector"; uses o.equals  
Object get(int i) // = the element at index i  
// = the index of the first occurrence of o; -1 if none; uses o.equals  
int indexOf(Object o)  
int size() // = the number of items in this Vector  
boolean remove(int i) // remove the element at index i  
// remove the first occurrence of o (if present); return true iff o was removed;  
// uses o.equals  
boolean remove(Object o)  
Iterator iterator() // return an Iterator of the items in this Vector  
// return a String representation of this Vector in the form [e1, e2, ... eN]  
String toString()
```

Iterator:

```
boolean hasNext() // = "there are more elements to return"  
Object next() // = the next element
```

Total Marks = 90