PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER 2009 EXAMINATIONS

CSC 207H1F

Duration — 3 hours

Examination Aids: None

Student Number: |_|_|_|_|_|_|_|_|_|

Last (Family) Name(s): _____

First (Given) Name(s): _____

---

*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 *carefully*.)

---

This final 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.*

Comments and Javadoc are not required except where indicated, although they may help us mark your answers. They may also get you part marks if you can't figure out how to write the code.

You do not need to put `import` statements in your answers.

If you use any space for rough work, indicate clearly what you want marked.

MARKING GUIDE

# 1: _____/ 6

# 2: _____/ 6

# 3: _____/ 8

# 4: _____/ 5

# 5: _____/ 3

# 6: _____/ 6

# 7: _____/ 6

TOTAL: _____/40

*Good Luck!*



CONT'D...

## Question 1. [6 MARKS]

These are some lines from a file that gives imaginary enrolment data about some imaginary courses, with tab characters represented by "<T>":

```
ANA  <T>ANA300Y1<T>Y <T>Human Anat Histol        <T>LEC<T>0101  <T>Y<T>P  <T>N<T>0350<T>0325<T>000
ANA  <T>ANA301H1<T>S <T>Human Embryology         <T>LEC<T>0101  <T>Y<T>O  <T>N<T>0500<T>0500<T>054
ANA  <T>ANA498Y1<T>Y <T>Research Project         <T>LEC<T>0101  <T>N<T>E  <T>N<T>9999<T>0012<T>000
ANT  <T>ANT100Y1<T>Y <T>Intro Anthropology       <T>LEC<T>5101  <T>Y<T>P  <T>N<T>1250<T>1129<T>000
ANT  <T>ANT200Y1<T>Y <T>Intro Archaeology        <T>LEC<T>5101  <T>Y<T>O  <T>N<T>0300<T>0281<T>000
CSC  <T>CSC148H1<T>S <T>Intro to Comp Sci        <T>LEC<T>0102  <T>Y<T>P  <T>Y<T>0100<T>0061<T>000
CSC  <T>CSC148H1<T>S <T>Intro to Comp Sci        <T>LEC<T>5101  <T>Y<T>P  <T>Y<T>0120<T>0117<T>000
CSC  <T>CSC148H1<T>S <T>Intro to Comp Sci        <T>TUT<T>0101  <T>N<T>P  <T>N<T>0060<T>0059<T>000
CSC  <T>CSC148H1<T>S <T>Intro to Comp Sci        <T>TUT<T>0201  <T>N<T>P  <T>N<T>0048<T>0019<T>000
CSC  <T>CSC207H1<T>F <T>Software Design          <T>LEC<T>0101  <T>Y<T>P  <T>N<T>0120<T>0096<T>000
CSC  <T>CSC207H1<T>F <T>Software Design          <T>LEC<T>5101  <T>Y<T>P  <T>N<T>0085<T>0043<T>000
```

You can assume that the number of characters before the first tab character and between successive tab characters is always the same in every line of the file.

### Part (a) [2 MARKS]

Write a regular expression to match the lecture ("LEC") sections of fall-term courses with names beginning "Intro". Group the course codes ("XXX123..") and section numbers (for example, 5203) so that they can be extracted from the matched substring. Clearly indicate where you place blanks by with the "b" symbol.

### Part (b) [2 MARKS]

The third-to-last and second-to-last columns indicate the enrolment cap and the actual enrolment. Write a regular expression to match every line where the enrolment is at the cap.

### Part (c) [2 MARKS]

Write a regular expression to match any line where the course title is shorter than 10 characters, not including trailing blanks. For example, the course title for CSC148H1 is "Intro to Comp Sci". There is no need to group any part of the string, though you may do so if you find it helps.

(There are 30 characters in the course title area of each line. At least the first character of the title must be non-blank.)

## Question 2. [6 MARKS]

A `Queue` class typically has these methods:

- `public void enqueue(Object o) // add o at the end of the queue.`
- `public Object dequeue() // remove and return the front item of the queue.`
- `public Object front() // return the front item of the queue.`
- `public boolean isEmpty() // return whether there are any items in the queue.`

Assume you have declared this `Queue` class, including these methods. Which of the following would be appropriate implementation choices? If an implementation is reasonable, write "OKAY" in big block letters. If an implementation is not reasonable (because of violations of object-oriented principles or major style or correctness problems), *briefly* describe *one* problem with that implementation. You can reuse explanations.

- A `private final` instance variable with type `List<Object>` that refers to an instance of class `ArrayList<Object>`. No other variables or methods.

- Store the queue contents in a linked list. Have `front` and `tail` private instance variables with type `Node`, where `Node` is a private inner class with a value and a next pointer; the two `private` instance variables point to the front and tail `Node`s of the list. No other variables or methods.

- Store the queue contents in a linked list as described in the previous part. A linked list only has one first node, so use the Singleton pattern for class `Node` to make sure that there is only one such first node.

- A `private` instance variable with type `Map` that refers to an instance of class `HashMap`. The sequence of items are stored in the `Map`: each key is an item, and its value is the item that follows it in the queue. No other variables or methods.

- A `private static` variable with type `List<Object>` that refers to an instance of class `ArrayList<Object>`. No other variables or methods.

- A `private` instance variable with type `Object[]` that refers to an instance of class `Object[]`; two `private int` instance variables to store the front and end indices of the items of the queue; and whenever the array gets full, `enqueue` makes a new bigger array and copies the elements over. No other variables or methods.

## Question 3. [8 MARKS]

Consider this JUnit-like test class, and recall that `assert` throws an `AssertionError` if the boolean expression evaluates to `false`.

```
public class TestClass {

    public void testFail1() {
        assert false;
    }

    public void testZeroDivError() {
        int i = 3 / 0;
    }

    public void testPass() {
    }

    public void testFail2() {
        assert false;
    }

    public void extraMethod() {
    }
}
```

JUnit works using reflection.

Method `main` below instantiates whichever class is specified on the command line and calls method `runTests`.

```
    public static void main(String[] args) throws ClassNotFoundException {
        Class testClass = Class.forName(args[0]);
        runTests(testClass);
    }
```

On the following page, complete method `runTests` so that, for every method whose name begins with "test", it makes a new instance of the class, calls that test method, and counts how many tests pass, fail, or throw an unexpected exception of any kind. After all tests have been run, print a summary.

The `TestClass` above has one pass, two failures, and one error.

Hint: when you invoke a method using reflection, if the method you invoke throws any kind of exception (such as an `AssertionError`), Java will catch that exception and in turn throw an `InvocationTargetException`. To get the original exception, you need to catch the `InvocationTargetException` and call its `getCause` method, which will return the original exception so that you can inspect it.

```
    public static void runTests(Class c) {
        int passed = 0;
        int failed = 0;
        int errors = 0;









        System.out.println("Passed: " + passed
                + "\n Failed:  " + failed
                + "\n Errors:  " + errors);
    }
}
```

## Question 4. [5 MARKS]

Here is a really terrible program that actually compiles and runs.

```
import java.util.*;

public class Style {
    public int alison;
    public static ArrayList brian = new ArrayList();

    public static void main(String[] args) {
        Object[] carol = new Object[]{"Mozart", "Beethoven", "Darwin"};

        for (int david = 0; david < carol.length; david++) {
            brian.add(carol[david]);
        }
        readValues(brian);
    }

    public static int readValues(ArrayList carol) {
        for (Object brian : carol) {
            System.out.println(brian);
        }

        return 1;
    }
}
```

This is its output:

```
Mozart
Beethoven
Darwin
```

On the next page, state ten distinct things that are wrong with the style of this program.

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

## Question 5. [3 MARKS]

Discuss whether `JButton`'s listener approach is an example of the `Observer` pattern.

## Question 6. [6 MARKS]

Here is a class that iterates over all the divisors of k between 2 and k-1.

```
public class Divisors implements Iterator<Integer> {
    private int k;

    private int nextDiv = 1;

    public Divisors(int k) {
        this.k = k;
        setNextDiv();
    }

    public boolean hasNext() {
        return nextDiv < k;
    }

    private void setNextDiv() {
        for (int n = nextDiv + 1; ; n++)
            if (k % n == 0) {
                nextDiv = n;
                return;
            }
    }

    public Integer next() {
        int div = nextDiv;
        setNextDiv();
        return div;
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

Use `Divisors` to write a class `Primes` that iterates over the prime numbers from 2 upward. Hint: how can you use `Divisors` to tell whether a number is prime?

Ignore the fact that `int`s in Java must have a maximum value.

Use this page for your answer.

## Question 7. [6 MARKS]

Here is a program that reads from the standard input and writes to the standard output. It uses the "new" Java generics and other recent features.

On the next page, rewrite the program so that it does not use recent features of Java that would not run on a J2 Mobile Edition device. Caution: you will need to use a `Vector` instead of a `List`, and an `Enumeration` instead of an `Iterator`.

```
public class G {
    private static List<String> ls = new ArrayList<String>();

    public static void main(String[] args) {

        Integer[] nums = new Integer[args.length];
        int index = 0;
        for (String arg : args) {
            try {
                int i = Integer.parseInt(arg);
                nums[index] = i;
                index++; // Assume the array is big enough for the input.
            }
            catch (NumberFormatException nfe) {
                // The input line didn't contain a string representing an int.
                ls.add(arg);
            }
        }

        for (Integer i : nums)
            System.out.println(i);
        for (String s : ls)
            System.out.println(s);
    }
}
```



CONT'D...

Use this page for your answer.

*[Use the space below for rough work. This page will* **not** *be marked, unless you clearly indicate the part of your work that you want us to mark.]*



CONT'D...

*[Use the space below for rough work. This page will* **not** *be marked, unless you clearly indicate the part of your work that you want us to mark.]*

### Short Java APIs:

```
class Throwable
    // the superclass of all errors and exceptions
    Throwable getCause() // the throwable that caused this throwable to get thrown
    String getMessage() // the detail message of this throwable
    StackTraceElement[] getStackTrace() // the stack trace info
class Exception extends Throwable
    Exception(String m) // a new Exception with detail message m
    Exception(String m, Throwable c) // a new Exception with detail message m caused by c
class RuntimeException extends Exception
    // The superclass of exceptions that don't have to be declared to be thrown
class Error extends Throwable
    // something really bad
class Object:
    String toString() // return a String representation.
    boolean equals(Object o) // = "this is o".
interface Comparable:
    // < 0 if this < o, = 0 if this is o, > 0 if this > o.
    int compareTo(Object o)
interface Iterator:
    next() // the next item.
    hasNext() // = "there are more items".
class Collections:
    static max(Collection coll) // the maximum item in coll
    static min(Collection coll) // the minimum item in coll
    static sort(List list) // sort list
class Arrays:
    static sort(T[] list) // Sort list; T can be int, double, char, or Comparable
interface Collection:
    add(E e) // add e to the Collection
    clear() // remove all the items in this Collection
    contains(Object o) // return true iff this Collection contains o
    isEmpty() // return true iff this Set is empty
    iterator() // return an Iterator of the items in this Collection
    remove(E e) // remove e from this Collection
    removeAll(Collection<?> c) // remove items from this Collection that are also in c
    retainAll(Collection<?> c) // retain only items that are in this Collection and in c
    size() // return the number of items in this Collection
    Object[] toArray() // return an array containing all of the elements in this collection
interface Set extends Collection, Iterator:
    // A Collection that models a mathematical set; duplicates are ignored.
class HashSet implements Set
interface List extends Collection, Iterator:
    // A Collection that allows duplicate items.
    add(int i, E elem) // insert elem at index i
    get(int i) // return the item at index i
    remove(int i) // remove the item at index i
class ArrayList implements List
interface Iterable<T>:
    // Allows an object to be the target of the "foreach" statement.
    Iterator<T> iterator()
interface Iterator<T>:
    // An iterator over a collection.
    hasNext() // return true iff the iteration has more elements.
```



CONT'D...

```
    next() // return the next element in the iteration.
    remove() // removes from the underlying collection the last element returned. (optional)
interface Enumeration
    hasMoreElements() // return true iff the enumeration has more elements.
    nextElement() // return the next element in the enumeration
class Integer:
    static int parseInt(String s) // Return the int contained in s;
                          // throw a NumberFormatException if that isn't possible
    Integer(int v) // wrap v.
    Integer(String s) // wrap s.
    int intValue() // = the int value.
interface Map:
    // An object that maps keys to values.
    containsKey(Object k) // return true iff this Map has k as a key
    containsValue(Object v) // return true iff this Map has v as a value
    get(Object k) // return the value associated with k, or null if k is not a key
    isEmpty() // return true iff this Map is empty
    Set keySet() // return the set of keys
    put(Object k, Object v) // add the mapping k -> v
    remove(Object k) // remove the key/value pair for key k
    size() // return the number of key/value pairs in this Map
    Collection values() // return the Collection of values
class HashMap implement Map
class Scanner:
    close() // close this Scanner
    hasNext() // return true iff this Scanner has another token in its input
    hasNextInt() // return true iff the next token in the input is can be interpreted as an int
    hasNextLine() // return true iff this Scanner has another line in its input
    next() // return the next complete token and advance the Scanner
    nextLine() // return the next line as a String and advance the Scanner
    nextInt() // return the next int and advance the Scanner
class String:
    char charAt(int i) // = the char at index i.
    compareTo(Object o) // < 0 if this < o, = 0 if this == o, > 0 otherwise.
    compareToIgnoreCase(String s) // Same as compareTo, but ignoring case.
    endsWith(String s) // = "this String ends with s"
    startsWith(String s) // = "this String begins with s"
    equals(String s) // = "this String contains the same chars as s"
    indexOf(String s) // = the index of s in this String, or -1 if s is not a substring.
    indexOf(char c) // = the index of c in this String, or -1 if c does not occur.
    substring(int b) // = s[b .. ]
    substring(int b, int e) // = s[b .. e)
    toLowerCase() // = a lowercase version of this String
    toUpperCase() // = an uppercase version of this String
    trim() // = this String, with whitespace removed from the ends.
class System:
    static PrintStream out // standard output stream
    static PrintStream err // error output stream
    static InputStream in // standard input stream
class PrintStream:
    print(Object o) // print o without a newline
    println(Object o) // print o followed by a newline
class Vector:
    // A list of Objects; this API is for J2ME.
```

```
    addElement(Object obj) // Append obj
    Eumeration elements() // return an Enumeration of the elements in this Vector
    contains(Object obj) // Return true iff this Vector contains obj
    insertElementAt(Object obj, int i) // insert obj at index i
    setElementAt(Object obj, int i) // replace the item at index i with obj
    indexOf(Object obj) // Return the index of obj
    isEmpty() // Return true iff this Vector is empty
    elementAt(int i) // return the item at index i
    removeElementAt(int i) // remove the item at index i
    removeElement(Object obj) // remove the first occurrence of obj
    size() // return the number of items
class Class:
    static Class forName(String s) // return the class named s
    Constructor[] getConstructors() // return the constructors for this class
    Field getDeclaredField(String n) // return the Field named n
    Field[] getDeclaredFields() // return the Fields in this class
    Method[] getDeclaredMethods() // return the methods in this class
    Class<? super T> getSuperclass() // return this class' superclass
    boolean isInterface() // does this represent an interface?
    boolean isInstance(Object obj) // is obj an instance of this class?
    T newInstance() // return a new instance of this class
class Field:
    Object get(Object o) // return this field's value in o
    Class<?> getDeclaringClass() // the Class object this Field belongs to
    String getName() // this Field's name
    set(Object o, Object v) // set this field in o to value v.
    Class<?> getType() // this Field's type
class Method:
    Class getDeclaringClass() // the Class object this Method belongs to
    String getName() // this Method's name
    Class<?> getReturnType() // this Method's return type
    Class<?>[] getParameterTypes() // this Method's parameter types
    Object invoke(Object obj, Object[] args) // call this Method on obj
```

### Regular expressions:

Here are some predefined character classes:

| | |
|---|---|
| . | Any character |
| \d | A digit: [0-9] |
| \D | A non-digit: [^0-9] |
| \s | A whitespace character: [ \t\n\x0B\f\r] |
| \S | A non-whitespace character: [^\s] |
| \w | A word character: [a-zA-Z_0-9] |
| \W | A non-word character: [^\w] |

Here are some quantifiers:

| Quantifier | Meaning |
|---|---|
| X? | X, once or not at all |
| X* | X, zero or more times |
| X+ | X, one or more times |
| X{n} | X, exactly n times |
| X{n,} | X, at least n times |
| X{n,m} | X, at least n; not more than m times |