PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL/MAY EXAMINATIONS

CSC 207H1S

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 6 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:	/10
# 2:	/ 5
# 3:	/ 4
# 4:	/ 6
# 5:	/10
# 6:	/15
TOTAL:	/50

Good Luck!

CONT'D...

Question 1. [10 MARKS]

In this question, you must implement the iterator pattern.

Class `java.lang.Number` is an abstract class in Java 6. It is the superclass of classes such as `Integer` and `Double`.

The following class, `NumberList`, maintains a list of `Numbers`.

```
public class NumberList {
    private int numItems;
    private Number[] numbers;

    public NumberList(int size) {
        this.numbers = new Number[size];
        this.numItems = 0;
    }

    public void add(Number n) {
        this.numbers[this.numItems++] = n;
    }
}
```

Here is code that uses `NumberList`:

```
    public void printPairs() {
        NumberList numbers = new NumberList(50);
        numbers.add(4);
        numbers.add(5);
        numbers.add(6);

        for (Number n1 : numbers) {
            for (Number n2 : numbers) {
                System.out.println("" + n1 + n2);
            }
        }
    }
```

`printPairs` doesn't work because `NumberList` doesn't properly support the foreach loop. If it did work, `printPairs` would print this output, one per line: `44 45 46 54 55 56 64 65 66`.

Modify the `NumberList` code so that `printPairs` works. You can cross off or add code to `NumberList`, and you can also continue the class on the next page. You are welcome to define as many additional classes and methods as you like. Do not modify `printPairs`.

Continue your answer here:

Question 2. [5 MARKS]

Assume that `d` and `i` have been initialized like this:

```
double d = 22.5;
int i = 22;
```

Consider the following (somewhat silly) code, which compiles and runs in Java 6.

```
List<Number> theList = new ArrayList<Number>();
theList.add(d);
theList.add(i);
Number n = theList.get(0);
```

As you know, J2ME does not have generics and it does not have classes `Set` or `ArrayList` (or any of the Collections hierarchy). It also doesn't have a `Number` class, although the wrapper classes are available. Also, J2ME does not support autoboxing.

Class `Vector` *is* available in J2ME. The J2ME API for `Vector` is included on the last page of this exam.

Rewrite the four lines of code so that they only use features available in J2ME, and so that this print statement would have the same output in either version:

```
System.out.printf("%s %s\n", theList, n);
```

Question 3. [4 MARKS]

In lecture, you saw that the Singleton pattern could be implemented like this:

```
public class Singleton {
  private static Singleton instance = new Singleton();

  public static Singleton getInstance() {
      return instance;
  }

  private Singleton() {}
}
```

Now consider this code; remember that `final` means that you can no longer assign to variable INSTANCE.

```
public class Singleton {
    private Singleton() {}
    public final static Singleton INSTANCE = new Singleton();
}
```

Part (a) [1 MARK] Do you think this implements the Singleton pattern? Circle your answer:

YES NO

Part (b) [3 MARKS]

If you think it does implement the Singleton pattern, briefly explain which version you prefer and **why**. If you think it does not, briefly explain why not.

Question 4. [6 MARKS]

Part (a) [2 MARKS] Give a brief overview of the Scrum process you were told to use in CSC207H.

Part (b) [4 MARKS]

In two paragraphs or less, describe your experiences with Scrum. Address the following topics:

- Whether your team tried to use it.
- Which parts were effective.
- Which parts were not effective.
- What changes you would recommend.

You will not be marked on your opinions, but rather on how well they are described and justified.

Continue your discussion here, if you need the space. Please don't use the whole page unless your handwriting is huge.

Question 5. [10 MARKS]

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
```

Part (a) [2 MARKS]

Write a regular expression that matches any three-letter string ending in "at" except for "bat" and "cat".

Part (b) [2 MARKS]

Write a string that matches this expression: `([moogah]+)([^gries]*)([gries]{0,2})\2\1`

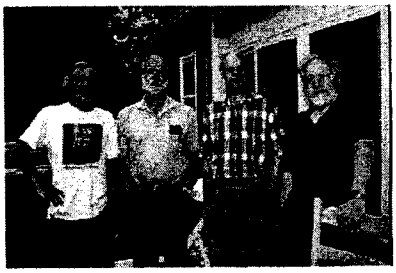
Part (c) [3 MARKS]

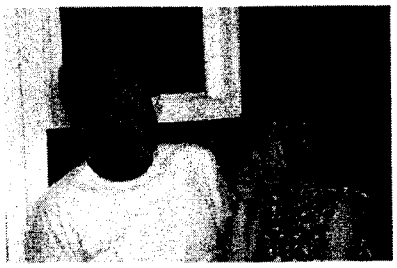
Write a regular expression that matches a two-digit number in the range 00–31. Shorter expressions will get more marks.

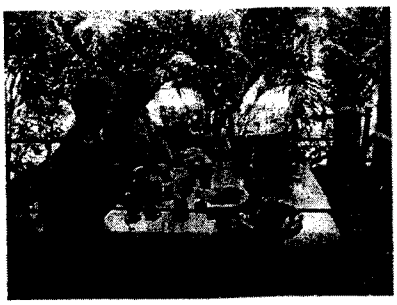
Part (d) [3 MARKS]

Write a regular expression that matches this filename format for backups of music files: a sequence of one or more lowercase letters and digits, followed by a hyphen, followed by 6 digits (the first digit of which must be either a 1 or a 2), followed by a period, followed by one of `mp3`, `m4a`, or `wav`, followed by the same 6-digit number as before, optionally followed by the extension `.bak`.

For up to 2 bonus marks on the exam, identify as many of the following people as you can.




CONT'D...

Question 6. [15 MARKS]

A client has asked you to write a set of Java classes to handle a booking system for meeting rooms. Here is their email:

> We require a system that will be used to book meeting rooms. All users of the system will be explicitly identified by a username, and will require a password to login. Users will belong to various groups. The groups have permissions to book certain rooms at certain times. The rooms have different seating capacities and audio/visual (A/V) equipment in them. When requesting a room, the booking user must specify who will be attending (which may be individuals or groups), and the A/V requirements. Users have an office location. The proximity of the meeting rooms to the various locations are stored by the system and used in scheduling meetings. Given a list of users, a gross timeframe (which includes earliest and latest dates, earliest and latest times during a day, and a preferred time), and A/V requirements, the system should provide suggestions (in order of desirability) for an appropriate meeting room and time, and then allow the user to book the room.

What to do: Design a set of classes with their methods and instance and static variables.

Here are some rules to keep you from getting hand cramps:

- **Don't implement any methods.** Just put empty curly braces unless a method is abstract.
- **Don't initialize any variables.** Just declare the type and the name.
- **We'll assume that all variables are private.** Unless you indicate otherwise.
- **We'll assume that all classes and methods are public.** Unless you indicate otherwise.
- **Don't write getter and setter methods for any instance variables.** We'll assume they exist. (You *do* need to write method headers for concepts that are more complex than getters and setters.)
- **Javadoc is not required.** But it will help us understand your design.

Follow the Java naming conventions and choose good names for your classes, variables, and methods.

If you think something is ambiguous, make a brief note about it and make a decision about how to interpret the ambiguity.

Continue your design here:

Continue your design here:

[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.]*

[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 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.
    next() // return the next element in the iteration.
    remove() // removes from the underlying collection the last element returned. (optional)
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
    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
```