PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2010 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 7 questions on 20 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: _____/ 8

2: _____/ 8

3: _____/12

4: _____/10

5: _____/ 5

6: _____/ 8

7: _____/ 5

TOTAL: _____/56

Good Luck!

CONT'D...

Question 1. [8 MARKS]

The following two methods use the "new" Java features.

On the next page, rewrite the program so that it does not use recent features of Java that would not run on a Java Mobile Edition device.

```
public List<String> readLines(String fileName) theLinesrows FileNotFoundException {

    List<String> theLines = new ArrayList<String>();
    Scanner sc = new Scanner(new File(fileName));
    while (sc.hasNextLine()) {
        String line = sc.nextLine().trim();
        theLines.add(line);
    }

    return theLines;
}

public List<String> merge(String fileName1, String fileName2) {

    try {
        List<String> lines1 = readLines(fileName1);
        List<String> lines2 = readLines(fileName2);

        List<String> mergedLines = new ArrayList<String>();
        mergedLines.addAll(lines1);
        mergedLines.addAll(lines2);

        return mergedLines;

    } catch(FileNotFoundException e) {
        System.err.println("One or more of the files does not exist.");
    }

    return null;
}
```

```
public Vector readLinesME(String fileName) throws IOException {

    InputStream is = getClass().getResourceAsStream(fileName);









}

public Vector mergeME(String fileName1, String fileName2) {









}
```

Question 2. [8 MARKS]

Part (a) [2 MARKS] Write a regular expression to match phone numbers of the form:

```
(XXX) XXX-XXXX
```

where each `X` is replaced by a single digit. Group the 3-digit area code so that it be extracted from the matched string. Clearly indicate where you place blanks with " ".

Part (b) [2 MARKS] A palindrome is a word that can be read the same way in either direction. For example, "kayak" and "noon" are both palindromes. Write a regular expression to match lowercase alphabetic palindromes of length 4 or 5.

Part (c) [4 MARKS]

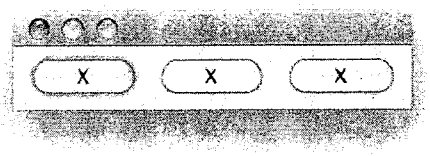
Each row in the table below gives first a regular expression, and secondly a string. Fill in the entry in the third column: either the part of the string that the regular expression matches or "NO MATCH" if there is no match.

You can assume that no regular expression or string has any white space before the first visible character or after the last. If your answer includes blanks, clearly indicate where you place blanks with " ".

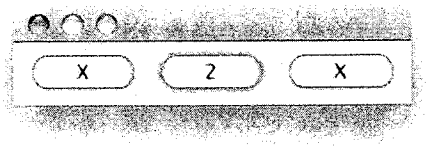
REGEX	STRING	MATCH
`^[a-z][\\w]*`	`x1z3y3`	
`\s\w\$`	`CSC207 Exam`	
`\w[^\s]\w`	`CSC207 Exam`	
`(\s[A-Z])\w{4}\1`	`Hi Hello Howdy`	

Question 3. [12 MARKS]

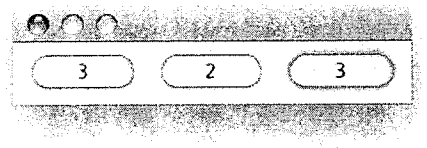
Recall the memory game you wrote in A2. In this question, you will write another GUI application: a slot machine game. The game involves generating 3 random numbers (between 1 and 3) and using buttons to display them. The numbers are initially hidden:



Once a button is clicked, it's number is revealed:



The game ends when all three are revealed (a winning game is one in which all three numbers, aka "slots", match):



Design and implement a set of classes to implement the slot game. As with A2, your user interface and slot game implementation should be separated. If the GUI were removed, the slot game implementation should still exist and be usable with a different interface.

Space for your answer to Question 3.

More space for your answer to Question 3.

Question 4. [10 MARKS]

Use the Iterator design pattern to implement a `ScrabbleHelper` class. A `ScrabbleHelper` has an array of words and a start letter, and iterates over the words from the array that start with that letter.

Here is an example of how `ScrabbleHelper` might be used:

```
public class Main {

    public static void main(String[] args) {
        String[] wordlist = new String[] {"ape", "bat", "pig", "ant", "ants"};

        ScrabbleHelper w = new ScrabbleHelper(wordlist, "a");
        while (w.hasNext()) {
            System.out.println(w.next());
        }
    }
}
```

The output is:

```
ape
ant
ants
```

Space for your answer to Question 4.

More space for your answer to Question 4.

Question 5. [5 MARKS]

This question uses the `ScrabbleHelper` class from the previous question. In particular, consider this `ScrabbleHelper` method:

```
/**
 * Return the next word beginning with the letter specified.
 * @return the next word
 * @throws NoSuchElementException - iteration has no more words
 */
public String next() { ... }
```

Write a JUnit test class to test `ScrabbleHelper`'s `next` method. Cases that are very similar to each other count only once. Include a comment describing each test case.

More space for your answer to Question 5.

Question 6. [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. Here is an example of running JUnit from the command line:
`java org.junit.runner.JUnitCore TestExample`

Method `main` retrieves the `Class` object for whichever class is specified on the command line (e.g., `TestExample` from the example above) 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 7. [5 MARKS]

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

```
import java.util.*;

public class Style {
    public int crosby;
    public static ArrayList luongo = new ArrayList();

    public static void main(String[] args) {
        Object[] iginla = new Object[]{"puck", "stick", "glove"};

        for (int toews = 0; toews < iginla.length; toews++) {
            luongo.add(iginla[toews]);
        }
        readValues(luongo);
    }

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

        return 1;
    }
}
```

This is its output:

```
puck
stick
glove
```

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.

[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 org.junit.Assert:
    static assertEquals(double expected, double actual, double delta)
    static assertEquals(java.lang.Object expected, java.lang.Object actual)
       (and other similar assertEquals methods)
    static assertFalse(boolean condition)
    static assertTrue(boolean condition)
    static fail()
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)
class Collections:
    static E max(Collection<E> coll) // the maximum item in coll
    static E min(Collection<E> coll) // the minimum item in coll
    static sort(List list) // sort list
    static shuffle(List list) // shuffles the list elements
class Arrays:
    static sort(T[] list) // Sort list; T can be int, double, char, or Comparable
interface Collection<E>:
    add(E e) // add e to the Collection
    clear() // remove all the items in this Collection
    boolean contains(Object o) // return true iff this Collection contains o
    boolean isEmpty() // return true iff this Set is empty
    Iterator iterator() // return an Iterator of the items in this Collection
    boolean remove(E e) // remove e from this Collection; boolean indicates success/failure
    boolean removeAll(Collection<?> c) // remove items from this Collection that are also in c
    boolean retainAll(Collection<?> c) // retain only items that are in this Collection and in c
    int size() // return the number of items in this Collection
    Object[] toArray() // return an array containing all of the elements in this collection
interface Set<E> extends Collection, Iterator:
    // A Collection that models a mathematical set; duplicates are ignored.
class HashSet implements Set
interface List<E> extends Collection, Iterator:
    // A Collection that allows duplicate items.
    add(int i, E elem) // insert elem at index i
    E get(int i) // return the item at index i
    E remove(int i) // remove the item at index i
class ArrayList implements List
```


CONT'D...

```
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.
    boolean hasNext() // return true iff the iteration has more elements.
    T next() // return the next element in the iteration.
    remove() // (optional) removes from the underlying collection the last element returned.
             // throws UnsupportedOperationException - if the remove operation is not supported
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<K, V>:
    // An object that maps keys to values.
    boolean containsKey(Object k) // return true iff this Map has k as a key
    boolean containsValue(Object v) // return true iff this Map has v as a value
    V get(Object k) // return the value associated with k, or null if k is not a key
    boolean isEmpty() // return true iff this Map is empty
    Set keySet() // return the set of keys
    boolean put(Object k, Object v) // add the mapping k -> v
    V remove(Object k) // remove the key/value pair for key k
    int size() // return the number of key/value pairs in this Map
    Collection values() // return the Collection of values
class HashMap implements Map
class InputStream:
    int read(byte[] b) // reads some number of bytes and stores them in b
                  // returns num bytes read or -1 if no bytes available
class Scanner:
    close() // close this Scanner
    boolean hasNext() // return true iff this Scanner has another token in its input
    boolean hasNextInt() // return true iff the next token in the input is can be
                         //interpreted as an int
    boolean hasNextLine() // return true iff this Scanner has another line in its input
    String next() // return the next complete token and advance the Scanner
    String nextLine() // return the next line as a String and advance the Scanner
    int nextInt() // return the next int and advance the Scanner
class String:
    String()  // constructs a new emtpy String
    String(byte[] bytes)  // constructs a new String using the array of bytes
    char charAt(int i) // = the char at index i.
    int compareTo(Object o) // < 0 if this < o, = 0 if this == o, > 0 otherwise.
    int compareToIgnoreCase(String s) // Same as compareTo, but ignoring case.
    boolean endsWith(String s) // = "this String ends with s"
    boolean startsWith(String s) // = "this String begins with s"
    boolean equals(String s) // = "this String contains the same chars as s"
    int indexOf(String s) // = the index of s in this String, or -1 if s is not a substring.
    int indexOf(char c) // = the index of c in this String, or -1 if c does not occur.
    String substring(int b, int e) // = s[b .. e)
    String toLowerCase() // = a lowercase version of this String
    String toUpperCase() // = an uppercase version of this String
    String trim() // = this String, with whitespace removed from the ends.
```

```
class StringBuffer:
   append(String)  // append the String to this sequence of chars
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
    Object get(int index) // return the element at the specified position
    boolean 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
    int indexOf(Object obj) // Return the index of obj
    boolean isEmpty() // Return true iff this Vector is empty
    Object elementAt(int i) // return the item at index i
    removeElementAt(int i) // remove the item at index i
    boolean removeElement(Object obj) // remove the first occurrence of obj
    int 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
interface ActionListener:
    void actionPerformed(ActionEvent e) // invoked when an action occurs
class JFrame:
    Container getContentPane() // the contentPane object for this frame
    Component add(Component comp)  // append comp to this container
    void pack() // size this window to fit its subcomponents
    void setVisible(boolean b) // shows or hides this window based on b
class JPanel:
    JPanel(LayoutManager layout) //a new JPanel with the given layout manager
    Component add(Component comp)  // append comp to this container
```

```
class JButton:
    void setText(String text) // set this button's text
    void addActionListener(ActionListener l) // add l to this button
class Container:
    Component add(Component comp)  // append comp to this container
interface LayoutManager:
    // interface for how to lay out containers
class FlowLayout implements LayoutManager:
    FlowLayout() // arranges components in a left-to-right flow
```

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\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