PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

AUGUST 2009 EXAMINATIONS

CSC 207H1Y
Instructor: Lung

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 examination consists of 4 questions on 15 pages (including this one).

Instructions:

- Check to make sure that you have all 15 pages.
- **Read the entire exam before you start.** Not all questions are of equal value, so budget your time accordingly.
- You do not need to add `import` lines or do error checking unless explicitly required to do so.
- You do not need to write comments or javadocs, although they may help us understand your code or enable us to give you partial marks.
- You do not need to catch exceptions in code you write unless explicitly required to do so.
- Helper functions are always allowed.
- If you use any space for rough work, indicate clearly what you want marked.

MARKING GUIDE

1: _____/10

2: _____/10

3: _____/10

4: _____/50

TOTAL: _____/80

Good Luck!

CONT'D...

Question 1. [10 MARKS]

For each code fragment below, state whether it is valid Java code (i.e., will not generate a compiler error) and, if not, why. Write your answer to the right of each code fragment. An ellipsis (...) means irrelevant code has been omitted.

Part (a) [2 MARKS]

```
public boolean isHello(String testString) {
    return testString == "Hello";
}
```

Part (b) [2 MARKS]

```
public int calculateValue(java.util.Stack calcStack) {
    ...
    throw new Exception();
    ...
}
```

Part (c) [2 MARKS]

```
public int returnFirst(int[] intAry) {
    if (intAry.length > 0) {
        return intAry[0];
    }
}
```

Part (d) [2 MARKS]

```
abstract class A {
    public abstract void a();
    public void b() {

    }
}

class B extends A {

}
```

Part (e) [2 MARKS]

```
public class Person {
    protected String name;
    ...
}

public class Actor extends Person {
    protected String stageName;

    public void adoptStageName() {
        name = stageName;
    }
    ...
}
```

Question 2. [10 MARKS]

Part (a) [1 MARK]

What is the name of your scrum master for phase III of the project?

Part (b) [4 MARKS]

How is the word *agile* ("able to move quickly and easily" – New Oxford American Dictionary, 2nd edition) an appropriate description of agile development? You may answer in point form.

Part (c) [5 MARKS]

An important aspect of software development is teamwork. Occasionally, things do not work out; for example, team members disappear – they may quit their jobs/burn out or otherwise suddenly stops communicating with the group. Based on what you've learned in this course and your experiences working in an agile development team, what precautionary measures would you take to minimize the impact of these situations? You may answer in point form.

Question 3. [10 MARKS]

Part (a) [2 MARKS]

A "fuzzy number" is an integer that contains an even number of 1s, 3s, and 7s, but no 8s or 0s. The following regular expression matches all fuzzy numbers (no comma separators as in 1,234,567) , but it also matches numbers that are *not* fuzzy. Give an example of a non-fuzzy number that is matched.

```
^(([^08]*[137]){2})+$
```

Part (b) [2 MARKS]

A comma-separated number groups numbers into groups of threes beginning from the right-most digit. Thus, a comma appears before the thousands digit, millions digit, billions digit, etc. The string "52,723" is one such number. Does the following pattern match only comma-separated integers? If not, give an example of a value that is falsely matched or falsely rejected. For the purposes of this question, the empty string and values like "00,207" are also comma-separated numbers. I.e., it is not problematic that these numbers are accepted by the pattern..

```
^(\d{0,3},)?(\d{3},)*(\d{3})?$
```

Part (c) [3 MARKS]

A "well-fed number" is an integer in which 7s appear but 9s do not. (Why? Because seven eight nine.) Write a regular expression that matches if and only if the string given is a well-fed number.

Part (d) [3 MARKS]

A deck of playing cards contains 52 cards. Each card has a value and a suit. Values range from the integers 2 through 10, jack, queen, king, and ace. The suit is one of hearts, diamonds, clubs, and spades. The values 2 through 9 are written using their integer form while ten, jack, queen, king, and ace are written using the initial letter of the word in upper case. The same goes for suits. The card can be represented using two characters – the first for its value and the second for its suit. The queen of hearts is thus QH, two of spades 2S, and ten of diamonds TD. Write a regular expression that matches if and only if the string given is a card represented in this form. Shorter answers are better.

Question 4. [50 MARKS]

In this question, you will write a simplified version of the UNIX `make` utility. The `make` utility is used to automate the process of compiling programs by executing commands based on files that have been updated. It reads in a file (a "makefile") that contains a list of targets, dependencies, and commands to execute:

```
target1: dependency1 dependency2 dependency3
    commands1
    commands2

target2: dependency4
    commands3
```

When `make` is executed, an argument is provided specifying a target. If the target (a file) has no dependencies or one of the dependencies has been modified more recently than the target, the commands are executed. Each dependency is the name of a file. Each dependency must be resolved by recursively resolving the dependencies of the target of the same name in the order that they appear (a sort of pre-order traversal). Targets and dependencies are separated by exactly one colon (":") and a space. Dependencies are separated by any number of spaces and tabs. Commands are indented by exactly four spaces and no blank lines may exist between commands and the target to which it belongs. Any number of blank lines may exist before a new target.

Consider the following makefile:

```
all:  MyProgram API
    echo "All done!"

MyProgram: GUI.class TextBasedInterface.class

GUI.class: GUI.java MainProgramLogic.java HelperClass.java
    javac GUI.java

TextBasedInterface.class:  TextBasedInterface.java MainProgramLogic.java HelperClass.java
    javac TextBasedInterface.java

API: docs/index.html

docs/index.html: MainProgramLogic.java HelperClass.java
    javadoc -o docs MainProgramLogic.java HelperClass.java
```

Assume that there are only `.java` files in the current directory.

Suppose the command `make GUI.class` is executed. First, since `GUI.java` and `MainProgramLogic.java` are not also targets, we simply check the last modified date of `GUI.java`, `MainProgramLogic.java` and `HelperClass.java`, the dependencies of `GUI.class`. If `GUI.class` is older than any of these files or does not exist (in this case, it does not exist), a (re-)compilation is necessary. Hence, the command `javac GUI.java` is executed, which will compile `GUI.java`.

Now, suppose the command `make MyProgram` is executed. First, we check the last modified date of `GUI.class` and `TextBasedInterface.class`. The target named `GUI.class` is checked first. Since it has just been compiled, `GUI.class` is up to date, so `make` does not recompile `GUI.java`. We then perform similar checks for `TextBasedInterface.class` as we did before for `GUI.class` and end up running the command `javac TextBasedInterface.java`. Lastly, since no file named MyProgram exists, we end up running the commands for the target MyProgram (there are none).

At this point, executing the command `make all` will have the effect of creating the javadocs and outputting the message "All done!" without recompiling any of the `.java files`. Now, if any of the `.java` files are updated and the command `make all` is executed, only the targets which depend on the modified files will be updated.

Assume the following class exists and is complete, aside from the `DependencyIterator` class:

```
public class Target {
   public String name;
   public String[] dependencies;
   public String[] commands;  // One command per element in the array

   /**
    * Class constructor for a Target with name and dependencies.
    */
   public Target(String name, String[] dependencies);

   /**
    * Adds an item to the list of commands to be executed if this target is out of date.
    * Note that the "commands" field will have a new reference after calling this method.
    * @param newCommand    The next command associated with this target.
    */
   public void addCommand(String newCommand);

   /**
    * Returns an Iterator that lists the dependencies of this Target.
    * @returns an Iterator for the dependencies of this Target.
    */
    public void DependencyIterator getIterator();

   /**
    * An iterator that visits every dependency of a target, including not only
    * direct dependencies, but all dependencies necessary to resolve the target.
    */
    public static class DependencyIterator implements Iterator;
}
```

Part (a) [10 MARKS]

Complete the following method according to its javadoc making use of regular expressions:

```
/**
 * Extract the information about a target from a line in the makefile.
 * If the line is not of the form
 *     target: dependency1 dependency2 ...
 * return null instead.
 * @param line    The line in the makefile to be parsed.
 * @returns a new Target with name and dependencies set or null
 *          if the line is not a target line.
 */
public Target extractTargetNameAndDependencies(String line) {
```

Part (b) [10 MARKS]

Complete the following method according to its javadoc:

```
/**
 * Reads in a makefile and returns a Map of Targets based on the contents
 * of the file.  The key of each entry in the Map is the name of the target and the
 * value is a Target object.
 * @param makefile  A File object for the makefile to be read.
 * @returns a Map of target names to Targets, one for each target in the makefile.
 */
public Map readMakefile(File makefile) {
```

Part (c) [7 MARKS]

Complete the following class according to its javadoc and fill in the javadoc for the constructor. Remember that this Iterator is returned by the Target.getIterator() method. Include any methods necessary.

```
/**
 * An iterator that visits every dependency of a target, including not only
 * direct dependencies, but all dependencies necessary to resolve the target.
 */
public static class DependencyIterator implements Iterator {
    /**









    */
    public DependencyIterator(Target target) {
```

Part (d) [8 MARKS]

For the following question, note that `Runtime.getRuntime().exec(command)` will execute the system command stored as a `String`. Complete the following method according to its javadoc:

```
/**
 * Resolve the Target named targetName and execute any commands necessary.
 * @params targetName    The name of the target to be executed.
 * @params targets       A mapping of target names (as Strings) to Targets.
 */
public static void resolve(String targetName, Map targets) {
```

Part (e) [5 MARKS]

Complete the main method so that it processes a makefile file named `Makefile` according to the description of the `make` command given at the beginning of this question:

```
public static void main(String[] args) {
```

Part (f) [10 MARKS]

Design **two** test cases to test the `make` command. For **each** test case, include a list of files along with file modification times and a valid makefile. Circle the name of the target to be provided to the `make` command for testing.

*[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.]*

Hopefully handy Java API notes:

```
class Object:
   String toString() // return a String representation of this object
   boolean equals(Object o) // returns true iff this object is ‘‘equal’’ to o
interface Comparable:
   int compareTo(Object o) // < 0 if this < o, =0 if this is o, > 0 if this > o
interface Iterator:
   Object next() // the next item.
   boolean hasNext() // returns true iff there are more items.
class Collections:
   static Object max(Collection coll) // maximimum item in coll
   static Object min(Collection coll) // 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(Object e) // add e to the Collection
   clear() // remove al lthe items in this Collection
   boolean contains(Object o) // return true iff this Collection contains o
   boolean isEmpty() // return true iff this Collection is empty
   remove(Object 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
   int size() // returns the number of itmes 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
   Object get(int i) // return the item at index i
   remove(int i) // remove the item at index i
class ArrayList implements List
class Integer:
   static int parseInt(String s) // return the int contained in s; throw a
      NumberFormatException if not possible
   Integer(int v) // A new Integer whose intValue() is v
   Integer(String s) // A new Integer whose intValue() is the number contained in String s
   int intValue() // the int value of this Integer
interface Map extends Collection:
   // An object that maps keys to values.
   boolean containsKey(Object k) // returns true iff this Map has k as a key
   boolean containsValue(Object v) // return true iff this Map has v as a value
   Object get(Object k) // return the value associated with k or null if k is not a key
   Set keySet() // retrun the set of keys
   put(Object k, Object v) // ad the mapping k -> v
   remove(Object k) // remove the key/value pairs 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 Scanner:
   close() // close this Scanner
   boolean hasNext() // return true iff this Scanner has another token in its input
```

```
   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
   String nextInt() // return the next int and advance the Scanner
class String:
   char charAt(int i) // return the char at index i
   boolean endsWith(String s) // returns true iff this String ends with s
   boolean startsWith(String s) // returns true iff this String begins with s
   boolean equals(String s) // returns true iff this String contains the same chars as s
   int indexOf(String s) // returns the index of s in this String or -1 if s is not a substring
   int indexOf(char c) // returns the index of c in this String or -1 if c does not occur.
   String substring(int b) // returns the bth character onwards of the String
   String substring(int b, int e) // returns the bth character up to, but not
      including the eth character
class File:
   File(String pathname) // Creates a new File instance.
   String getName() // Returns the name of the file or directory denoted by this file.
   long lastModified() // Returns the time that the file was last modified.  Time increases.
   boolean exists() // Tests whether the file exists.
```