PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER 2015 EXAMINATIONS

CSC 207 H1F
Instructor(s): Horton and Lung

Duration—3 hours

No Aids Allowed

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 read the instructions below *carefully.*

You must get 40% or above on this exam to pass the course (at least 32 out of 80); otherwise, your final course grade will be no higher than 47.
This Final Examination paper consists of 8 questions on 24 pages (including this one). *When you receive the signal to start, please make sure that your copy of the Final Examination is complete.*

- Legibly write your name and student number on this page.
- Legibly write your student number at the bottom of every odd page (except this one), in the space provided.
- If you use any space for rough work, indicate clearly what you want marked.
- In all programming questions you may assume all input is valid.
- You do not need to write Javadocs or internal comments.

# 1:	_____/18
# 2:	_____/ 8
# 3:	_____/10
# 4:	_____/ 8
# 5:	_____/10
# 6:	_____/ 8
# 7:	_____/10
# 8:	_____/ 8
TOTAL:	_____/80

Good Luck!

Question 1. [18 MARKS]

For the yes-no questions, there is no penalty for wrong answers, but no marks without a correct explanation.

Part (a) [2 MARKS]

You are writing a program to read in a file containing grades and perform analyses on it, such as determining the average grade and distribution of grades. This is the file format:

```
class size
student number, grade
student number, grade
...
```

The student numbers aren't needed by your program. Which would you recommend to store the grades (**circle** one): `int[]` `ArrayList`
Why?

Part (b) [2 MARKS]

Farley Ciorina made millions in the tech industry and now spends a lot of time camping in the middle of the woods, disconnected from the world. However, Farley occasionally wants to work on some of the code his company developed while on these trips. Is Subversion an appropriate tool for Farley to use while camping? (**circle** one) Yes No
Explain:

Part (c) [1 MARK]

The Java Collections framework has many interfaces including `Set` and `List`. Why was it appropriate to define these as interfaces rather than abstract classes?

Part (d) [2 MARKS]

Would it be a good idea for a Java `Iterator`'s `next()` method to return `null` to indicate the end of iteration? (**circle** one) Yes No
Explain:

Part (e) [2 MARKS]

Is there any circumstance under which it makes sense for a method to `catch` an exception it throws? (**circle** one) Yes No
Explain:

Part (f) [2 MARKS]

If you take another course that involves a programming assignment, will you use CRC cards before you begin programming? (**circle** one) Yes No
Explain:

Part (g) [3 MARKS]

Suppose you are interested in the correctness of method `froob()`. If you are given a set of test cases that cause every line of `froob()` to be executed at least once and all the tests pass, can you conclude that `froob()` is error-free? (**circle** one) Yes No
Give two reasons that support your answer:

Part (h) [2 MARKS]

Describe two things that an `Intent` object allows you to do in an Android app.

Part (i) [2 MARKS]

You are writing a class to implement a new Abstract Data Type called a "jumble". You want to make it a generic class, but your partner thinks it would be easier to just allow the methods for inserting and removing items to deal with instances of class `Object`. What does your design allow client code to do that your partner's design does not?

Question 2. [8 MARKS]

Part (a) [5 MARKS]

Pick one of the following two scenarios and consider the software development methodology you would use.

Scenario 1: Rainforest, Inc. is an online retailing giant. They have identified a number of shortcomings in their existing shipping software that employees use at Rainforest's warehouses. They are seeking to replace their existing system with a new one. Rainforest expects this software to work flawlessly as soon as it is installed in its warehouses.

Scenario 2: Wahoo?, Inc. is creating a new mobile application for its popular photo sharing service, Twinkl, in order to compete with new competitor FastGram. There are a few features that Wahoo thinks its new software will need in order to attract and retain customers. However, it is open to adding new features based on user testing. As such, Wahoo has found some volunteers to test the new application and provide feedback as needed over a six month period.

Circle the scenario you choose: Scenario 1: Rainforest Scenario 2: Wahoo?

List five software development methodologies (for example, would you use test-driven development? scrum?) or software tools you would use. Your answer should not involve the Java language in any way. For each item, explain the benefits it would bring, referring specifically to the scenario you chose.

Item 1: ______________________________

Benefit:

Item 2: ______________________________

Benefit:

Item 3: ______________________________

Benefit:

Item 4: ____
Benefit:

Item 5: ____
Benefit:

Part (b) [3 MARKS]

Describe three reasons why it is a good idea to specify an API for a class before writing any code.

Reason 1:

Reason 2:

Reason 3:

Question 3. [10 MARKS]

Part (a) [2 MARKS] Below is a class for Smurfs. Add the necessary code so that printing a `Smurf` will print their full Smurf name, which includes the middle names "Little Smurf". If your profs were Smurfs, printing them should yield `Jonathan Little Smurf Lung` and `Diane Little Smurf Horton`.

```
public class Smurf {
    private String firstName;
    private String lastName;

    public Smurf(String firstName, String lastName) {
        super();
        this.firstName = firstName;
        this.lastName = lastName;
    }









}
```

Part (b) [1 MARK] Define a new checked exception class named `TooFewNamesException`. It needs only one constructor with no arguments.

Part (c) [7 MARKS]

On the next page, complete the `SmurfWorld` class. After reading a Smurf's name, it should construct a `Smurf` object and print it. If the user enters fewer than two names or more than two names, the program should print a message. Here are three runs of the program that show its three kinds of behaviour:

```
Name: Diane Horton              Name: Diane          Name: Diane Percy Horton
Diane little Smurf Horton       Too little!          Too much!
```

Use a helper method that takes a string and attemps to construct and return a `Smurf` object. If the string contains fewer than two names, the helper should throw a `TooFewNamesException`, and if it has more than two names, it should throw a `TooManyNamesException` Assume that the necessary `import` statements have been written, and that `TooManyNamesException` has been defined, like and `TooFewNamesException`, as a checked exception.

```
public class SmurfWorld {
    // Helper method.
    private                                                   (String s)

        // Use blanks to split s into its component names.
        String[] names = s.split(" ");













    }

    public static void main (String[] args) {
        // Read a single line of input from the user and store it in String names.
        Scanner scanner = new Scanner( System.in );
        System.out.print( "Name: " );
        String names = scanner.nextLine();
        scanner.close();















    }
}
```

Question 4. [8 MARKS]

Using good object-oriented design, define any classes, interfaces and method signatures that are necessary in order to make the code below run. Do not fill in the bodies of the methods or define any fields. You also do not need to draw any UML diagrams.

```java
public class Main {
    public static void main(String[] args) {
        Calendar cal = new Calendar();

        // Schedule a daily repeating alarm for 6 am (6:00).
        Alarm wakeup = new RecurringAlarm(6, 0);
        wakeup.addDescription("Morning␣wake-up␣reminder.");
        wakeup.setVolume(Alarm.VOLUME_LOUD);
        cal.schedule(wakeup);

        // Schedule a one-time alarm for 1 pm (13:00) on 3 November, 2015.
        Alarm reminder = new OneTimeAlarm(13, 0, 3, 11, 2015);
        reminder.setVolume(Alarm.VOLUME_QUIET);
        cal.schedule(reminder);

        // Schedule a meeting for 1:30 pm (13:30) on 3 November, 2015.
        Event meeting = new Meeting("Meet␣with␣the␣mayor", 13, 30, 3, 11, 2015)
        meeting.setLocation("Bat␣Cave");
        cal.schedule(e);

        // Schedule a party for 6:00 pm (18:30) on 3 November, 2015.
        Event danceParty = new Party("Do␣the␣Harlem␣Shake", 18, 30, 3, 11, 2015);
        danceParty.setLocation("Dance␣Cave");
        danceParty.addDescription("Don't␣forget␣to␣bring␣a␣bowtie.");
        cal.schedule(danceParty);

        // Create a location named Bahen Centre at 40 St. George St.
        Location bahenCentre = new Location("Bahen␣Centre");
        bahenCentre.setLocation("40␣St.␣George␣St.");

        // Print to the screen directions between 40 St. George St. and the Bat Cave.
        WorldMap wm = new WorldMap();
        wm.printDirections(bahenCentre, meeting);

        // Print to the screen directions between the Bat Cave and the Dance Cave.
        wm.printDirections(meeting, danceParty);
    }
}
```

Begin your answer here:

Continue your answer here:

CONT'D...

Question 5. [10 MARKS]

Jonathan has lost his voice in the middle of the term and now has to say everything twice to be heard. He needs you to write a class for him called `TwoTwo`. These are the requirements:

- Items can be added to a `TwoTwo`. All items must be of the same type.
- The `TwoTwo` class must implement the iterator pattern, but when iterating through a `TwoTwo`, each item must be returned twice.

Jonathan has given you the program below to demonstrate. Your `TwoTwo` class must be written so that the program runs successfully.

You may assume that a `TwoTwo` never has items removed from it; therefore your `Iterator` may completely omit the optional `remove` method. You may also assume that a `TwoTwo` is never modified after an `Iterator` is created.

Hint: You do not need to actually make copies of any items elements in the `TwoTwo` – you can use a counter or something similar to track how many times an element has been returned.

```
public class Driver {
    public static void main(String[] args) {
        TwoTwo<String> twoTwoString = new TwoTwo<String>();
        twoTwoString.add("Hello");
        twoTwoString.add("CSC207");

        /*
         * Prints out
         *     Hello
         *     Hello
         *     CSC207
         *     CSC207
         */
        for(String s : twoTwoString) {
            System.out.println(s);
        }

        TwoTwo<Integer> twoTwoInteger = new TwoTwo<Integer>();
        twoTwoInteger.add(new Integer(207));

        /*
         * Prints out
         *     207
         *     207
         */
        for(Integer i : twoTwoInteger) {
            System.out.println(i);
        }

    }
}
```

Space for your answer to this question

cont'd...

Space for your answer to this question

Question 6. [8 MARKS]

Part (a) [1 MARK] Give a clear, concise, English description of the strings that match the following regular expression: `[0-9]|100|[1-9][0-9]`

Part (b) [2 MARKS] Give a clear, concise, English description of the strings that match the following regular expression: `(\\w*)\\s\\1`

Part (c) [1 MARK] For the following regular expression, which string(s) does it match? The regular expression: `[abcdef]+`

Circle all that apply

a. abcdef

b. abcdefabcdef

c. abc

d. abcabc

Part (d) [2 MARKS]

A standard deck of playing cards includes cards that have a value and a suit. The value is a number between 2 and 9, inclusive, or one of "jack", "queen", "king", "ace". The suit is one of "hearts", "diamonds", "clubs", "spades". Write a regular expression that recognizes a playing card written in this form: the card's value, a single space, the word "of", another space, and the card's suit. For example, your regular expression should match these strings:

- `jack of hearts`
- `3 of spades`

You may assume the strings being matched are all lowercase.

Part (e) [2 MARKS]

A letter grade is one of the uppercase letters from A through F followed optionally by a plus sign (+) or a minus sign (-), except for F, which is never followed by any sign. For example, the following are valid letter grades: `B+`, `C`, and `F`. However, the following are not valid letter grades: `b+`, `+C`, and `F-`. Write a regular expression that matches valid letter grades only.

Question 7. [10 MARKS]

The following program is valid Java, but it has at least fifteen different stylistic and/or design problems. Identify ten of these problems give a short but clear explanation of each. Write your answer directly on the code below. Note: Marks will be deducted for misidentifying problems.

```
/**
 * This class maintains the state of a bank account.
 */
public class Bank_Account {
    public float a;
    public int accountNumber;
    public ArrayList<Transaction> transactions;
    public static int totalTransactions = 0;
    public Boolean isPremiumAccount;

    /**
     * Class constructor specifying the account number; this account starts with
     * a balance of $0.
     * @param accountNumber This account's unique account number.
     * @param isPremiumAccount true iff this account is a premium
     account.
     */
    public Bank_Account(int accountNumber, Boolean isPremiumAccount) {
        try {
            this(accountNumber, isPremiumAccount, 0);
        } catch (Exception e) {
        }
    }

    /**
     * Class constructor specifying the account number and starting balance.
     * @param accountNumber This account's unique account number.
     * @param amount The starting balance of this account in dollars.
     * @param isPremiumAccount true iff this account is a premium account.
     */
    public Bank_Account(int accountNumber, Boolean isPremiumAccount, float amount) {


        this.accountNumber = accountNumber;
        this.a = amount;
        this.isPremiumAccount = isPremiumAccount;
        this.transactions = new ArrayList<Transaction>();
    }

    /**
     * Records a new transaction for this account.
     * @param transaction The new transaction to record.
     */
    public void addTransaction(Object transaction) {
        System.out.println("Debugging:␣transaction␣added.");
        transaction.add((Transaction) transaction);
        this.a = this.a + ((Transaction) transaction).amount;
        Bank_Account.totalTransactions = Bank_Account.totalTransactions + 1;
    }

    /**
     * Returns the number of transactions conducted using this account.
     */
    public int getNumTransactions() {
        return transactions.size();
    }

    /**
     * Deducts the premium service fee from this account.
     */
    public void deductPremiumFee() {
        transaction.add(new Transaction(-100));
        this.a = this.a - 100;
        Bank_Account.totalTransactions = Bank_Account.totalTransactions + 1;
    }

    /**
     * Deducts the standard service fee from this account.
     */
    public void deductStandardFee() {
        transaction.add(new Transaction(-5));
        this.a=this.a-5;
        Bank_Account.totalTransactions=Bank_Account.totalTransactions+1;
    }

    /**
     * Returns the total number of transactions conducted across all Bank_Accounts
     * @return Total number of transactions.
     */
    public int getTotalTransactions() {
        return Bank_Account.totalTransactions;
    }
}
```

Question 8. [8 MARKS]

Suppose you are to implement the back-end for an online quiz system called QuizMonkey. The user interface is someone else's job.

Some investors are coming in next week and you need to show them some working code. You have decided on this feature list:

- There is a set of multiple choice and numeric questions, which we call the "question bank".
- The question bank is loaded from a file when the program is launched, and saved between runs of the program.
- A multiple choice question has a set of answer options. Numeric questions have a minimum and maximum possible answer.
- All questions have question text (such as "How many classes can a Java class `extend`?") and a correct answer.
- A user can add questions to the question bank.
- Users are identified by their email address, which is unique.
- There is only one kind of user. Each user can add questions to the question bank, can create quizzes, and can take quizzes.
- A quiz contains one or more questions from the question bank.
- The system generates a unique integer, called the "quiz ID", for every quiz.
- When a user takes a quiz, the system must store their responses to the quiz questions.
- The system can report a user's score on a particular quiz.
- The system can report on question bank statistics: the total number of questions in the question bank, and for each particular question, the number of quizzes that include the question.

On the next two pages, create a CRC-card design for this feature list. You must use CRC-card notation, not UML. Do not write any Java code.

Begin your CRC design here.

Continue your CRC design here.

Use the space on this "blank" page for scratch work, or for any answer that did not fit elsewhere.
Clearly label each such answer with the appropriate question and part number.

Short Java APIs:

```
class Throwable:
    // the superclass of all Errors and Exceptions
    Throwable getCause() // returns the Throwable that caused this Throwable to get thrown
    String getMessage() // returns the detail message of this Throwable
    StackTraceElement[] getStackTrace() // returns the stack trace info
class Exception extends Throwable:
    Exception()          // constructs a new Exception with detail message null
    Exception(String m) // constructs a new Exception with detail message m
    Exception(String m, Throwable c) // constructs 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() // returns a String representation
    boolean equals(Object o) // returns true iff "this is o"
interface Comparable<T>:
    int compareTo(T o) // returns < 0 if this < o, = 0 if this is o, > 0 if this > o
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() // returns true iff the iteration has more elements
    T next() // returns the next element in the iteration
    void remove() // removes from the underlying collection the last element returned or
                  // throws UnsupportedOperationException
interface Collection<E> extends Iterable<E>:
    boolean add(E e) // adds e to the Collection
    void clear() // removes all the items in this Collection
    boolean contains(Object o) // returns true iff this Collection contains o
    boolean isEmpty() // returns true iff this Collection is empty
    Iterator<E> iterator() // returns an Iterator of the items in this Collection
    boolean remove(E e) // removes e from this Collection
    int size() // returns the number of items in this Collection
    Object[] toArray() // returns an array containing all of the elements in this collection
interface List<E> extends Collection<E>, Iteratable<E>:
    // An ordered Collection. Allows duplicate items.
    boolean add(E elem) // appends elem to the end
    void add(int i, E elem) // inserts elem at index i
    boolean contains(Object o) // returns true iff this List contains o
    E get(int i) // returns the item at index i
    int indexOf(Object o) // returns the index of the first occurrence of o, or -1 if not in List
    boolean isEmpty() // returns true iff this List contains no elements
    E remove(int i) // removes the item at index i
    int size() // returns the number of elements in this List
class ArrayList<E> implements List<E>
class Arrays
    static List<T> asList(T a, ...) // returns a List containing the given arguments


interface Map<K,V>:
    // 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) // returns true iff this Map has v as a value
    V get(Object k) // returns the value associated with k, or null if k is not a key
    boolean isEmpty() // returns true iff this Map is empty
    Set<K> keySet() // returns the Set of keys of this Map
    V put(K k, V v) // adds the mapping k -> v to this Map
    V remove(Object k) // removes the key/value pair for key k from this Map
    int size() // returns the number of key/value pairs in this Map
    Collection<V> values() // returns a Collection of the values in this Map
class HashMap<K,V> implements Map<K,V>
class File:
    File(String pathname) // constructs a new File for the given pathname
class Scanner:
    Scanner(File file) // constructs a new Scanner that scans from file
    void close() // closes this Scanner
    boolean hasNext() // returns true iff this Scanner has another token in its input
    boolean hasNextInt() // returns true iff the next token in the input is can be
                         // interpreted as an int
    boolean hasNextLine() // returns true iff this Scanner has another line in its input
    String next() // returns the next complete token and advances the Scanner
    String nextLine() // returns the next line and advances the Scanner
    int nextInt() // returns the next int and advances the Scanner
class Integer implements Comparable<Integer>:
    static int parseInt(String s) // returns the int contained in s
        throw a NumberFormatException if that isn't possible
    Integer(int v) // constructs an Integer that wraps v
    Integer(String s) // constructs on Integer that wraps s.
    int compareTo(Object o) // returns < 0 if this < o, = 0 if this == o, > 0 otherwise
    int intValue() // returns the int value
class String implements Comparable<String>:
    char charAt(int i) // returns the char at index i.
    int compareTo(Object o) // returns < 0 if this < o, = 0 if this == o, > 0 otherwise
    int compareToIgnoreCase(String s) // returns the same as compareTo, but ignores case
    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 a substring of this String: s[b .. ]
    String substring(int b, int e) // returns a substring of this String: s[b .. e)
    String toLowerCase() // returns a lowercase version of this String
    String toUpperCase() // returns an uppercase version of this String
    String trim() // returns a version of 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) // prints o without a newline
    println(Object o) // prints o followed by a newline


class Pattern:
   static boolean matches(String regex, CharSequence input) // compiles regex and returns
                                                             // true iff input matches it
   static Pattern compile(String regex) // compiles regex into a pattern
   Matcher matcher(CharSequence input) // creates a matcher that will match
                                       // input against this pattern
```

```
class Matcher:
   boolean find() // returns true iff there is another subsequence of the
                  // input sequence that matches the pattern.
   String group() // returns the input subsequence matched by the previous match
   String group(int group) // returns the input subsequence captured by the given group
                           //during the previous match operation
   boolean matches() // attempts to match the entire region against the pattern.
class Observable:
    void addObserver(Observer o) // adds o to the set of observers if it isn't already there
    void clearChanged() // indicates that this object has no longer changed
    boolean hasChanged() // returns true iff this object has changed
    void notifyObservers(Object arg) // if this object has changed, as indicated by
        the hasChanged method, then notifies all of its observers by calling update(arg)
        and then calls the clearChanged method to indicate that this object has no longer changed
    void setChanged() // marks this object as having been changed
interface Observer:
    void update(Observable o, Object arg) // called by Observable's notifyObservers;
        // o is the Observable and arg is any information that o wants to pass along
```

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]`
`\b`	`A word boundary: any change from \w to \W or \W to \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`