NAME (PRINT): ______________________________
Last/Surname First /Given Name

STUDENT #: ____________________ SIGNATURE: ______________

UNIVERSITY OF TORONTO MISSISSAUGA
DECEMBER 2009 FINAL EXAMINATION
CSC207H5F
Software Design

Wael Abou El Saadat
Duration - 3 hours
Aids: Open Book and Notes

The University of Toronto Mississauga and you, as a student, share a commitment to academic integrity. You are reminded that you may be charged with an academic offence for possessing any unauthorized aids during the writing of an exam, including but not limited to any electronic devices with storage, such as cell phones, pagers, personal digital assistants (PDAs), iPods, and MP3 players. Unauthorized calculators and notes are also not permitted. Do not have any of these items in your possession in the area of your desk. Please turn the electronics off and put all unauthorized aids with your belongings at the front of the room before the examination begins. If any of these items are kept with you during the writing of your exam, you may be charged with an academic offence. A typical penalty may cause you to fail the course.

Please note, you ***CANNOT*** *petition to* ***RE-WRITE*** *an examination once you have begun writing.*

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 exam consists of 5 questions (numbered 1 to 5) on 12 pages *(including this one, a tips sheet at the end, and 2 blank pages for your answers)*, printed on one side of the paper.
- Answer each question directly on the examination paper, in the space provided, and **use the reverse side of the page for rough work**. If you need more space for one of your solutions, use the reverse side of the page and indicate **clearly** the part of your work that should be marked.
- You do not have to comment your code but use **meaningful** variable names.
- Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0) so please write legibly.

1.	________	/ 18	(Short Questions)
2.	________	/ 20	(Abstract Data Types)
3.	________	/ 15	(Testing)
4.	________	/ 20	(Reflection and GUI)
5.	________	/ 27	(Design)
	________	/ 100	**TOTAL**

Question 1. Short Questions [*18 marks total*]

(a) [10 marks]

[T] or [F] A class that is abstract may not be sub-classed.

[T] or [F] The value of a final Java variable cannot be changed once it has been initialized.

[T] or [F] Every Java application has a main method that is static.

[T] or [F] A subclass can access private variables in its super class.

[T] or [F] Static methods do not require an instance of the class; they can be accessed through the class name.

[T] or [F] A single java try statement can have more than one catch statement.

[T] or [F] If a method has no parameters, then we don't need parentheses when we call it.

[T] or [F] The declaration statement Object o actually creates an object in memory.

[T] or [F] A Java class can define at most one constructor

[T] or [F] If a and b are two references of the same type, then a.equals(b) will return true if and only if a and b refer to the same object.

Continued on Page 3

(b) [8 marks] SVN

Assume you have checked out project *Test* from your *svn* repository to your local workspace and have since created the files *App.java, IO.java, and Test.java*, and put them in your local workspace. For each of the following svn-related actions specify the sequence of svn commands that must be executed. Also, specify what inputs each command would require to accomplish the task. Note: Syntax and order of inputs is not important.

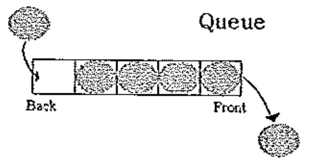
i. **[4 marks]** Ensure that your local workspace has the latest version of the files in your *svn* repository.

(ii) **[4 marks]** Ensure that your *svn* repository has exactly the same version of every file(new, modified, unchanged) in your local workspace.

Continued on Page 4

Question 2. Abstract Data Types [*20 marks total*]

(a) **[10 marks]** This question requires implementing a Queue (of Strings) with an array. Fill in the blanks in the code below (one constructor, three methods). Note that the constructor takes a parameter that is the maximum size of the queue. You may assume that the number of Strings in the queue will never be as large as this. But the amount of enqueueing and dequeueing by the client is not limited. You may want to draw a picture of the array, thinking carefully about what should happen when front or back approaches N. You may assume that the client will never dequeue from an empty queue.



```
public class ArrayQueue {
        private String [] a;
        private int N;      // the size of the ArrayQueue array
        private int back;   // where to put the next String
                            // to be enqueued
        private int front;  // where to get the next String
                            // to be dequeued
public ArrayQueue(int max) {



}
public boolean isEmpty() {



}
public void enqueue(String s) {



}
public String dequeue() {



}
}
```

Continued on Page 5

(b) [10 marks] The class declared below records information about teams of students working on a project. The class stores two maps: one that has a mapping from each student to the name of the group they belong to, and the other that has a mapping from each group name to the mark that group received on the project. Complete the method below that returns a new map where each student is a key and the value is the mark their group received. If a student is a member of a group that is not in the map of groups to marks, then they should receive a mark of 0.

```
public class GroupMarks {
       Map<String, String>  personToGroup;
       Map<String, Integer> groupToMark;

public Map<String, Integer> individualMarks() {
























  }
  }
```

Continued on Page 6

Question 3. Testing [*15 marks total*]

Fibonacci numbers are a set of numbers formed by adding the last two numbers to get the next in the series, except for the first 2 numbers, which can be specified to obtain different Fibonacci sequence. For instance, the following two groups are both Fibonacci numbers:

1 1 2 3 5 8 13……..
0 2 2 4 6 10 16 …….

Suppose you have a java method: **int Fibonacci(int first, int second, int n)**
where "first" and "second" are the initial two numbers and the method should return the nth Fibonacci number in the list. You may assume (1) n is a positive integer (2) "first" and "second" are both integers, and "second" is always greater or equal to "first". In other words, you do not need to test the above cases.

Write 5 useful, distinct test cases to test Fibonacci method. If you write more than 5 cases, we will only look at the first 5 cases. You do not need to write the whole Junit class, just the test methods. Assume any necessary packages are imported. Please either choose good methods name, or write brief comments to describe what you are testing.

Continued on Page 7

Question 4. Reflection and GUI [*20 marks total*]

(a) **[10 marks]** Reflection allows us to retrieve information about the source code from the binary file (.class). Write a java class *GenerateJUnitNullTestCases* that would take the name of any class file, and generate Java source code which contains unit test cases to test the methods in that class by passing nulls to each. You may assume that the class will always have a constructor which doesn't take any parameters. You may also assume that all the methods will take object parameters (*i.e. no primitive types being passed as parameters*)

An example invocation:

```
            java  GenerateJUnitNullTestCases  Point.class
```

will output the following on the command line:

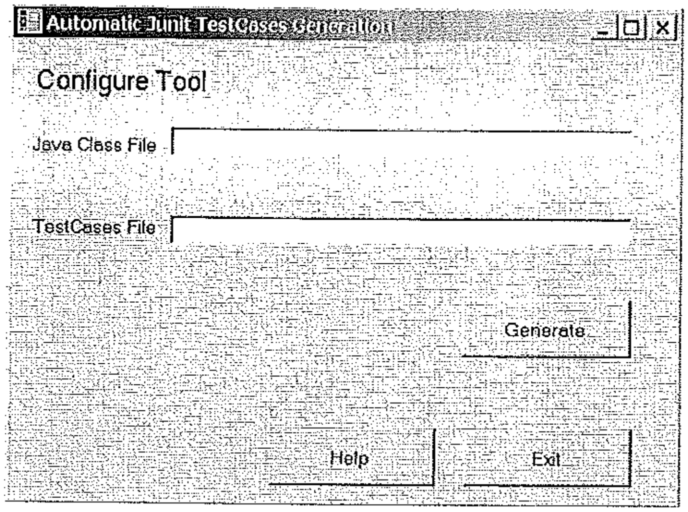
```
import junit.framework.TestCase;
public class PointTestCase extends TestCase {

   public void testNullSetX( )  {
        Point pnt;
        pnt = new Point( );
        pnt.setX( null );
        fail("Should raise a NullException");
   }

   public void testNullSetY( ) {
        Point pnt;
        pnt = new Point( );
        pnt.setY( null );
        fail("Should raise a NullException");
   }

   public void testNullSetXY( ) {
        Point pnt;
        pnt = new Point( );
        pnt.setXY( null , null );
        fail("Should raise a NullException");
   }
}
// -------------------------end of class----------------------------
```

Where the source code to produce Point.class must have been as follows: (note that your program does not have access to this source. You will have to use reflection to generate the above test cases).

```
public class Point {

        public Point( ) {…}

        public void setX( Object obj ){…}

        public void setY( Object obj ) {….}

        public void setXY( Object obj1, Object obj2 ){…}

   }
```

Continued on Page 8

(b) [10 marks] Write a Java Program using the AWT GUI library to create a GUI as illustrated below for the application you developed in part 4 (a). The text fields are for entering the class file (*e.g. Point.class*) and the class name of the source file containing test cases (*e.g.* `PointTestCase`) You can use any layout manager or null layout. Include the event handling methods and the calls to GenerateJUnitNullTestCases. Help will display guiding messages to the user under each of the text fields. Generate displays the output on the command line.



Continued on Page 10

Question 5. Design [*27 marks total*]

(a) **[2 mark]** One technique we have talked about to identify classes is to use nouns/verbs/adjectives from the problem-domain/statement. Briefly describe this technique.

(b) **[25 marks]** You have been hired to write a system for the UoT library! The goal is to keep track of who's checked out which books, and when the books are due. In particular, we want to make sure no student has more than 20 books at a time, and we want to send a detailed overdue notice to any student with late books. The supported functionalities are: checking out a book, returning a book, putting a hold on a book, sending email to a student for late book(s) and notifying a student that a book has arrived. When the application starts, it loads the books information from a text file with the following format, where each book is stored on a separate line:

Book-title,Book-Description,Book-number,borrower=student1-id,return-date=some-date, hold=student2-id,student3-id

If the book has not been borrowed => borrower= null
If no one has put a hold on a book => hold=null

and the application loads the students information from a file with the following format, where each student is stored on a separate line:
student-name,student-id,student-email

Specify the packages, classes in each package, and methods in each class. Use the Command interface as discussed in class as well as the builder and make sure that packages communicate with each via interfaces. Do *not* write any code - though if you find it convenient you can partially describe a member-variable/attribute using its type and a method by giving its signature.

Continued on Page 11

Tips Sheet

Junit methods:

static void assertTrue(boolean test)
static void assertFalse(boolean test)
assertEquals(expected, actual)
assertSame(Object expected, Object actual)
assertNotSame(Object expected, Object actual)
assertNull(Object object)
assertNotNull(Object object)
fail()

Map methods:

Method Summary	
void	`clear()` Removes all mappings from this map (optional operation).
boolean	`containsKey(Object key)` Returns `true` if this map contains a mapping for the specified key.
boolean	`containsValue(Object value)` Returns `true` if this map maps one or more keys to the specified value.
Set<Map.Entry<K,V>>	`entrySet()` Returns a set view of the mappings contained in this map.
boolean	`equals(Object o)` Compares the specified object with this map for equality.
V	`get(Object key)` Returns the value to which this map maps the specified key.
int	`hashCode()` Returns the hash code value for this map.
boolean	`isEmpty()` Returns `true` if this map contains no key-value mappings.
Set<K>	`keySet()` Returns a set view of the keys contained in this map.
V	`put(K key, V value)` Associates the specified value with the specified key in this map (optional operation).
void	`putAll(Map<? extends K,? extends V> t)` Copies all of the mappings from the specified map to this map (optional operation).
V	`remove(Object key)` Removes the mapping for this key from this map if it is present (optional operation).
int	`size()` Returns the number of key-value mappings in this map.
Collection<V>	`values()` Returns a collection view of the values contained in this map.

DONE