UNIVERSITY OF TORONTO

Faculty of Arts and Science

APRIL 2012 EXAMINATIONS

CSC207H1S

Instructor: Abbas Attarwala

Duration - 3 hours

<u>No Aids Allowed</u>

Question	Mark
1) Short Answers	/25
2) Regular Expressions	/15
3) Publish/Subscribe Design Pattern	/15
4) Subversion	/10
5) Iterator Design Pattern	/10
6) Generics	/15
7) Junit	/10

Total: **/100**

Question 1 (Short Answers). **[25]**

a) State an advantage and disadvantage to implementing the Observer design pattern. **[3]**

b) Briefly describe the difference between a Product Backlog and a Sprint Backlog. **[3]**

c) Write a regular expression to match Canadian postal codes. Each three-character block is separated by a single space. **[4]**

d) What are some of the advantages behind Java Exceptions? Is there any way Exceptions can be misused? **[3]**

e) Consider the following code:

```
public class File
{
        public static void iAMStaticFunction()
        {
                System.out.println("I am static function in
file");
        }
}
```

```
public class openPDF extends File
{
        public static void iAMStaticFunction()
        {
                System.out.println("I am static function in
pdf");
        }
}
```

```
public class openWord extends File
{
        public static void iAMStaticFunction()
        {
                System.out.println("I am static function of
Word");
        }
}
```

c.1) Give the output, and explain your answer in one line. **[4]**
File file1= new openWord();
file1.iAMStaticFunction();

c.2) Give the output, and explain your answer in one line. **[4]**
openPDF file2= new openPDF();
file2.iAMStaticFunction();

c.3) Give the output, and explain your answer in one line. **[4]**
File file3= new openPDF();
file3.iAMStaticFunction();

Question 2) Regular Expressions **[15]**

a) 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. **[7.5]**

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
x*yz	yzxxyz	
[A-Z]+(\w\s)+	Sam I am	
\s\w\$	CSC207 Exam	
cb*a*bc$	bcaaabc	
(\s[A-Z])\w{4}\1	Hi Hello Howdy	

b) You want to determine whether a user entered a North American phone number in a common format, including the local area code. These formats include 1234567890,123-456-7890, 123.456.7890, 123 456 7890, (123) 456 7890, and all related combinations. If the phone number is valid, you want to convert it to your standard format,(123) 456-7890, so that your phone number records are consistent. **[7.5]**

b.1) Write the regular expression that will match any of the above possible combination of phone numbers AND then write few lines of Java code that will convert valid phone numbers to the standard format.

Question 3)
Publish/Subscribe Design Pattern. [15]

class **Observable** has the following methods defined:
--void addObserver(Observer o);
--int countObservers();
--void deleteObserver();
--void deleteObserver(Observer o);
--boolean hasChanged(Observer o);
--void notifyObservers();
--void notifyObservers(Object arg);

AND the interface **Observer** is defined as following:

```
interface Observer
{
    public void update(Observable obs, Object obj)
}
```

a)

Using the **class observable** and the **interface Observer** you are to write the following software system. The software system is made up of the weather station that gets the actual weather data from sensors (humidity, temperature and pressure). The Weather Data object (that tracks the data coming from the Weather Station), and updates the UI displays that shows users the current weather conditions. You have following incomplete code at your disposal. **You must complete it, i.e. add code in incomplete methods, add missing methods and specify if the classes inherit or implement any of the above super class or interface.**

```
//YOU DO NOT NEED TO CHANGE THE WEATHERSTATION CLASS.
public class WeatherStation
{
  public static void main(String[] args)
  {
    WeatherData weatherData = new WeatherData();

    CurrentConditionsDisplay currentDisplay =
      new CurrentConditionsDisplay(weatherData);
    StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);
    ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);
    weatherData.setMeasurements(80, 65, 30.4f);
    weatherData.setMeasurements(82, 70, 29.2f);
    weatherData.setMeasurements(78, 90, 29.2f);
  }
}
```

//THIS IS INCOMPLETE CLASS. YOU MUST COMPLETE IT

```
public class WeatherData
{
   private float temperature;
   private float humidity;
   private float pressure;

   public WeatherData() { }

   public void measurementsChanged()
  {  //complete this method







  }

  public void setMeasurements(float temperature, float humidity, float pressure)
 {//complete this method







 }
}

public class CurrentConditionsDisplay
{
   Observable observable;
   private float temperature;
   private float humidity;

   public CurrentConditionsDisplay(Observable observable)
  {
      //THIS IS INCOMPLETE METHOD
       this.observable = observable;




   }
```

//1) THERE IS ONE KEY FUNCTION MISSING HERE. THIS FUNCTION THAT YOU WRITE HERE, MUST CALL THE display() FUNCTION BELOW.
//2) WRITE ANOTHER FUNCTION THAT ALLOWS IT TO UNSUBSCRIBE

```
   public void display()
  {
      System.out.println("Current conditions: " + temperature
         + "F degrees and " + humidity + "% humidity");
  }
}
```

Question 4 Subversion. **[10]**

A co-worker who is collaborating with you on a project has set up a Subversion repository for the two of you and has already added some files. You are using a computer on which SVN has already been properly set up.

Part (a) **[2]**

Assuming that your co-worker set the repository up at http://big.company.com/svn/trunk, write the SVN command(s) necessary to make a local copy of repository files on your machine.

Part (b) **[2]**

You have two files called todo.txt and notes.txt that need to be added to the version control system. Write the SVN command(s) necessary to get both of those files in to the repository that was set up by your co-worker.

Part (c). **[6]**

Assume that the repository now contains the following files file1.py, file2.java, todo.txt, and notes.txt. Assume also that you (User A) and your co-worker (User B) have up-to-date repositories. Assuming that the commands in the table are executed in order, fill in the table with YES or NO as appropriate, depending on whether the file notes.txt exists on the specified user's computer. Assume that no other commands are being executed except those indicated.

If you make any assumptions, please state them below the table.

User	Command	Local copy exists on A's computer after command is executed?	Local copy exists on B's computer after command is executed?	Copy exists in repository after command is executed?
A	svn remove notes.txt			
B	svn update			
A	svn commit			
B	svn update			

Question 5 Iterator Design Pattern. **[10]**
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, Numberlsit, 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.**

Solution to Question 5 comes in here:

Question 6) Generics: **[15]**

a) What is the motivation behind Generics? Do not tell me what Generics is. I need to know the motivation/intent behind it. **[2]**

b)

```
List words = new ArrayList();
words.add("Hello ");
words.add("world!");
String s = ((String)words.get(0))+((String)words.get(1))
assert s.equals("Hello world!");
```

b.1) Why are (String)words.get(0) and (String)words.get(1) unsafe operations? **[3]**

b.2) How can you fix the above unsafe operations at runtime? **[3]**

b.3) How can you fix the above unsafe operations at compile time? **[7]**

Question 7) **[10]**

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);
    }
}
```

Appendix:

```
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
```

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:

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, atleast n times
X{n,m}	X, atleast n; not more than m times