PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

AUGUST 2017 EXAMINATIONS

CSC 207 H1Y
Instructor(s): Shorser

Duration—2 hours

No Aids Allowed

You must earn at least 20 out of 50 marks (40%) on this final examination in order to pass the course. Otherwise, your final course grade will be no higher than 47%.

Student Number:	
Last (Family) Name(s):	
First (Given) Name(s):	
UTORid:	

Do **not** *turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

MARKING GUIDE

# 1:	______/10
# 2:	______/ 6
# 3:	______/10
# 4:	______/ 9
# 5:	______/ 6
# 6:	______/ 9
TOTAL:	______/50

This Final Examination paper consists of 6 questions on 15 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the paper is complete and fill in your name and student number above.*

- Comments and docstrings are not required except where indicated, although they may help us mark your answers.
- If you use any space for rough work, indicate clearly what you want marked.
- Do not remove pages or take the exam apart.

Good Luck!

Question 1. [10 MARKS]

A.	Throw an exception	H.	LinkedList	O.	`super();`
B.	UML diagram	I.	List	P.	`return isEmpty == true;`
C.	CRC card	J.	Encapsulation	Q.	`this.idCount = new Integer("5");`
D.	Casting	K.	Model-View-Controller	R.	Interface
E.	Generics	L.	Factory Method	S.	Abstract Class
F.	Array	M.	Strategy	T.	Default value
G.	ArrayList	N.	Empty Constructor	U.	Print the stack trace

The following are descriptions of items listed above. On the underscore beside each description, write the letter of the item it describes best. No two descriptions should be matched with the same item above.

Part (a) [1 MARK] ______ A method of representing object oriented code that is independent of programming language and allows you to easily see which classes contains which methods.

Part (b) [1 MARK] ______ A method of representing object oriented code that is independent of programming language and allows you to easily modify the functionality and position on the inheritance hierarchy of each class.

Part (c) [1 MARK] ______ A design pattern that makes it easier to isolate, test, and/or replace an algorithm.

Part (d) [1 MARK] ______ A design pattern that allows you to easily pass instances of a new subtype into a class without modifying said class.

Part (e) [1 MARK] ______ A Java object which stores a predetermined number of pointers that all point to memory address where values of the same type are stored.

Part (f) [1 MARK] ______ A list of method signatures that do not have a body and, potentially, some other instance methods which have been instantiated.

Part (g) [1 MARK] ______ An interface that allows an object to store multiple other objects of the same type in a specified order.

Part (h) [1 MARK] ______ Something done automatically or caused by the programmer when the execution of the code cannot be completed under the required circumstances.

Part (i) [1 MARK] ______ A temporary way to make your code compile under the following situation: you label a variable with a type that is higher up on the inheritance hierarchy than the constructor that was called to create the variable. Then you want to call a method that only exists in the same class as the constructor.

Part (j) [1 MARK] ______ The statement that the JVM automatically provides when you create a constructor in one class that does not specifically call the constructor of its parent class.

Question 2. [6 MARKS]

During the process of creating your Inventory System Project (the Project for this course), you and your group made many major decisions (at the class level or program-wide level) and local decisions (regarding implementation of methods, which variables to define, etc.). Think of a **major** decision that your group made, then changed later in order to improve your code.

Describe one such improvement, including:

- a description of the program before the improvement,
- the change(s) made to the program, and
- why the resulting program was better than before the change was made.

You should only describe **relevant** details of your project. Your description can include parts of the project that were coded by you and/or parts that were implemented by other group members.

If your group did not change any of your major decisions, then you can explain how you made a major decision in the first place, and how it contributed to your final design.

Question 3. [10 MARKS]

Write a class called `Registration` which can store variables of type `T`, where `T` can be replaced with `String`, `Person`, `Student`, etc.. It should be possible to instantiate `Registration` as follows:

```
Registration<Person> csc207registration = new Registration<Student>();
Registration<String> basketballTournament = new Registration<String>();
```

Variables that are stored to `Registration` can either be added to a private variable called `roster` or another private variable called `waitList`. At first, objects that are added to `Registration` should only be added to `roster`. The variable `roster` can store up to 100 instances of type `T`. Once it is full, instances of type `T` should be added to `waitList` instead, which can store as many instances as necessary.

It should be possible for another class to find out how many instances of type `T` are stored in `roster` and in `waitList` without directly accessing either variable. It should also be possible for another class to access the most recent instance added without using casting.

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, and refer to this answer on the original question page.

Question 4. [9 MARKS]

Identify <u>three</u> problems in the following code that will prevent it from compiling. Describe each problem and suggest a way to fix it.

```
public class ParentClass{
  private int pVar;
  public ParentClass(int x){
    pVar = x;
  }
  public void pMethod(){
  }
}
```

```
public class ChildClass{
  private int cVar;
  public ChildClass(){
  }
  public void cMethod(ParentClass p){
    System.out.println(p);
  }
```

```
public class Demo{
  public static ParentClass tunnelMethod(ChildClass c){
    return c;
  }

  public static void main(String[] args){
    ChildClass child = new ChildClass();
    ParentClass parent = new ParentClass();
    ParentClass both = new ChildClass();

    (tunnelMethod(child)).cMethod();
    both.cMethod(child);
    tunnelMethod(parent);
  }
}
```

Continued on next page...

1.

2.

3.

Student #: OVER. . .

Question 5. [6 MARKS]

Consider the following snippets of code. One or more incorporate the Dependency Injection design pattern. Circle the letter beside the snippet(s) that demonstrate Dependency Injection and **explain why this is the case for each.**

(A.)
```
public class ExamDemo{
  private Moogah obj1;

  public ExamDemo(Moogah arg){
    obj1 = arg;
  }
  ...
}
```

(B.)
```
public class ExamDemo{
  private Moogah obj1 = new Moogah();

  public ExamDemo(){
  }
  ...
}
```

(C.)
```
public class ExamDemo{
  private Moogah obj1 = new Moogah();

  public ExamDemo(Moogah arg){
    obj1 = arg;
  }
  ...
}
```

(D.)
```
public class ExamDemo{
  private Moogah obj1;

  public ExamDemo(){
  }

  public void setObj(Moogah arg){
    obj1 = arg;
  }
  ...
}
```

(E.) see next page for code...

CONT'D...

(E.) continued...

```
public class ExamDemo{
  private Moogah obj1;

  public ExamDemo(){
  }

  public void setObj(Moogah arg){
    obj1 = new Moogah();
    obj1 = arg;
  }
  ...
}
```

Question 6. [9 MARKS]

Part (a) [6 MARKS]

Write all substrings of `a2bbc1cc2bb12b2c` that satisfy the following regular expressions. If no such substrings exist, write "None".

(i) `bc?[12]c`

(ii) `a.*[12]`

(iii) `[a-z0-9]{2}(a|b|c)\1`

Part (b) [3 MARKS]

For this question, let us say that you receive a text file. The file contains information about network activity. Each line of the file contains the source (Internet Protocol) IP address and target IP address for one packet of information. Each IP address is written in binary using the IPv4 format with four sets of eight 1's and 0's that are separated by periods. An example line in the file looks like this:

```
10110101.00001001.10011001.00110011 --> 01101001.10001000.10110001.00000011
```

All lines in the file are formatted like this, including the arrow, or else they are written in English using only word characters. Write a regular expression that is satisfied only by lines in the file that are formatted as in the example and where the source and target IP both start with the same eight binary digits.

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
```

CONT'D...

```
                        //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

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, and refer to this answer on the original question page.

CONT'D...