PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2018 EXAMINATIONS

CSC 207 H1S
Instructor(s): Shorser, Sin

Duration—2 hours

No Aids Allowed

You must earn at least 24 out of 60 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.*

This Final Examination paper consists of 6 questions on 18 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 Javadoc 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.

MARKING GUIDE

1: _____/10

2: _____/10

3: _____/10

4: _____/10

5: _____/10

6: _____/10

TOTAL: _____/60

Good Luck!

OVER...

Question 1. [10 MARKS]

A.	Iterator	H.	Interface	O.	`==`
B.	Strategy	I.	Wrapper Class	P.	`equals`
C.	Observer	J.	Primitive Type	Q.	`toString`
D.	Constructor	K.	`Object`	R.	`print`
E.	Instance Method	L.	`Stack`	S.	`0`
F.	Static Method	M.	`List`	T.	`1`
G.	Abstract Class	N.	`String`	U.	`null`

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] _____ Java will provide each class with this, but only if it is not included in the code.

Part (b) [1 MARK] _____ Memory will be allocated for this before a java program begins execution and will remain allocated for this until the end of execution.

Part (c) [1 MARK] _____ This design pattern encapsulates an algorithm so that it can easily be changed or modified.

Part (d) [1 MARK] _____ A class `A` can implement this so that class `A` is guaranteed to contain all of the necessary methods, regardless of implementation.

Part (e) [1 MARK] _____ This cannot be instantiated, but it can be extended.

Part (f) [1 MARK] _____ The class you instantiate when you want to use the `equals` method on a primitive variable.

Part (g) [1 MARK] _____ The highest class in the inheritance hierarchy to have a `toString` method.

Part (h) [1 MARK] _____ These eight types do not have a `equals` method.

Part (i) [1 MARK] _____ This will compare the memory addresses of two instances of `String`.

Part (j) [1 MARK] _____ The default value of all primitive types.

Question 2. [10 MARKS]

Read each question carefully before circling the correct answer for each of the following.

Part (a) [2 MARKS] Given that the following git commands were executed in sequence while no one else was interacting with the remote repository, what changed in the remote repository as a result?

```
git push
git pull
((edit notes.txt here))
git add notes.txt
git commit -m "added a list of code that can now be deleted"
git push
```

(i) nothing

(ii) a potential conflict within notes.txt

(iii) a change to notes.txt with no conflict

Part (b) [2 MARKS] You see that a conflict occurred in `Demo.java`, in your remote git repository. Which of the following may have been the cause? Read all options before choosing the best one:

(i) You executed "`git push`" before "`git pull`".

(ii) You forgot to commit your changes before pushing them.

(iii) You made multiple commits without pushing them, during which time someone else pushed their changes to `Demo.java`.

Part (c) [2 MARKS] Which of the following is FALSE?

(i) You can extend `RuntimeException`.

(ii) You can throw an `Exception` from inside a `try` block.

(iii) Java will throw a direct subclass of `Exception` if your program attempts to perform an illegal arithmetic operation.

Part (d) [2 MARKS] Class `Widget` contains the following method:

```
public void run() {
  Foo var1 = new Foo();
  var1.writeToFile();
}
```

Class `Foo` contains a method with the following signature:

```
public void writeToFile throws IOException () { ... }
```

If everything else in the code has been debugged, then based only on these two methods, we know that

(i) the program will not compile.

(ii) the program will compile but may not run to completion.

(iii) the program will compile and run.

Part (e) [2 MARKS] One of the reasons you should never throw an instance of class `Exception` in a Java program is:

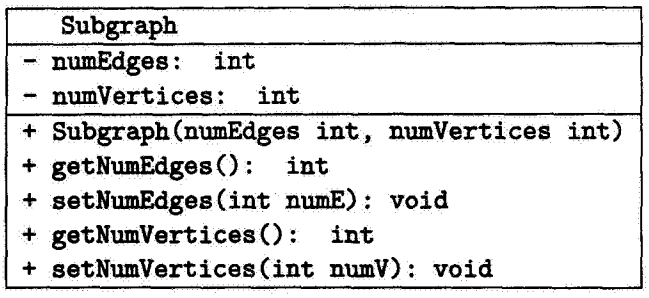
(i) the `catch` block for the exception will also catch instances of `RuntimeException` that you might not know about.

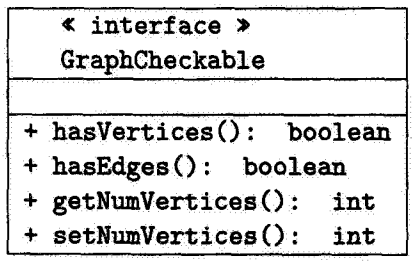
(ii) the class `Exception` is abstract, so you cannot call the constructor for it.

(iii) the compiler will not let you.

Question 3. [10 MARKS]

Part (a) [8 MARKS] Write the minimal amount of code required so class `Trigram` implements `GraphCheckable`, extends `Subgraph`, and admits three objects of generic type as arguments in the constructor. Those three objects are its "vertices". It has no "edges". `Trigram` should also contain at least one getter method to return its vertices.





Continues on next page ...

... Question 3 continues here.

Part (b) [2 MARKS] Write a main method that creates a `Trigram` that stores the following three instances of `String`: "`a`", "`b`", and "`c`" . Then call its **`hasVertices`** method and print the result to the screen.

Question 4. [10 MARKS]

Consider the Supplementary Code (see the end of this exam).

Part (a) [2 MARKS] Of all the design patterns we covered this semester, name one that can be used to improve this code.

Part (b) [2 MARKS] Suppose the code is re-written to implement the design pattern you named in part (a). List the names of all classes in the implementation with a brief description of what each one does and how it interacts with the other classes.

Examples of interactions: Is it instantiated inside another class? Does it extend another class? Does it call the constructor of another class inside one of its methods?

Part (c) [6 MARKS] Explain how your classes from part (b) implement the design pattern your named in part (a).

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 5. [10 MARKS]

As we discussed in class, the SOLID design principles are:

1. Single responsibility principle
2. Open/closed principle
3. Liskov substitution principle
4. Interface segregation principle
5. Dependency inversion principle

Part (a) [3 MARKS] More than one SOLID principle is violated by the Supplementary Code at the end of this exam. Choose one of the SOLID principles that is violated, write its name, and which lines are involved in the violation.

Part (b) [7 MARKS] Explain how you would fix the code so that it better implements the principle from part (a). You can write code to aid your explanation, if necessary.

Question 6. [10 MARKS]

Part (a) [3 MARKS]

Write the substring(s) of `aAaA.bB2B1` that satisfy each of the following regular expressions. If no such substrings exist, write "None".

(i) `[ab][AB](2)`

(ii) `(.){8}\w\d$`

(iii) `[a-z].+b?B[12]`

Part (b) [4 MARKS]

You are given a text file that contains a list of web addresses. Write a regular expression that will only match each web address that has all of the following properties:

1. The address starts with "`http://`" or `https://`".
2. Immediately after that it has "`www.`".
3. Following that, the domain name must contain only lowercase letters and digits, including at least one letter.
4. At the end of the domain name, there must be a `.ca`, or `.com`, followed by a forward slash.
5. Then, the path must contain the same subdirectory twice, separated each time by a forward slash.
6. Lastly, the file must end in `.shtml` or `.php`.

For example, your regular expression should match:

`http://www.example.ca/webstuff/webstuff/fun-with-webstuff.shtml`

Part (c) [3 MARKS]

Fill in each of the squares with a character so that each resulting horizontal string satisfies the regular expressions to its left and each vertical string satisfies the regular expression above it. Draw this symbol: ϕ in any box that you think should have a whitespace character.

	`([aeiou]){2}o`	`([l-p])\1\1x`	`v?[codepartv]?ty`
`(os|in)v+`			
`o?[left](.){2}`			
`[^xyz]r.*`			

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.

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

Supplementary Code For Questions 4 and 5

You may remove these pages to use as reference for Questions 4 and 5.

```
1  public class Location {
2
3    private int xcoord;
4
5    private int ycoord;
6
7    public Location(int x, int y){
8      xcoord = x;
9      ycoord = y;
10   }
11
12   public int getX(){
13     return xcoord;
14   }
15
16   public int getY(){
17     return ycoord;
18   }
19
20   public String toString(){
21     return 'Location: ' + xcoord + ' and ' + ycoord;
22   }
23
24   public boolean equals(Location other){
25     return other.xcoord==this.xcoord && other.ycoord==this.ycoord;
26   }
27 }
```

```
1  public class Path {
2
3    private Location start;
4    private Location end;
5    private int numLocations;
6
7    public Path(){
8      numLocations = 2;
9    }
10
11   public Path(int startX, int startY, int endX, int endY){
12     start = new Location(startX, startY);
13     end = new Location(endX, endY);
14     numLocations = 2;
15   }
16
17   public void addLocation(){
18     numLocations++;
19   }
20
21   public int getNumLocations(){
22     return numLocations;
23   }
24
25   public boolean equals(Path other){
26     return other.start.equals(this.start) && other.end.equals(this.end);
27   }
28 }
```

```
1  import java.util.ArrayList;
2
3  public class CurrentPath extends Path {
4
5    private ArrayList<Location> visitedLocations = new ArrayList<>();
6    private Location currLoc = new Location(0,0);
7    private int currIndex;
8
9    public CurrentPath(Location loc1){
10     currLoc = loc1;
11     visitedLocations.add(loc1);
12     super.addLocation();
13     currIndex = 0;
14   }
15
16   public void visitLoc(Location loc){
17     visitedLocations.add(loc);
18     super.addLocation();
19   }
20
21   public boolean hasNext(){
22     return (currIndex < visitedLocations.size()-1);
23   }
24
25   public Location next(){
26     if(this.hasNext()){
27       Location temp = currLoc;
28       currLoc = visitedLocations.get(++currIndex);
29       return temp;
30     }
31     else {
32       return currLoc;
33     }
34   }
35
36   public void printAll(){
37     while(this.hasNext()){
38       System.out.println(this.next().toString());
39     }
40     System.out.println(this.next().toString());
41   }
42
43   public boolean equals(CurrentPath cp){
44     return (this.currLoc).equals(cp.currLoc);
45   }
46 }
```