PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2017 EXAMINATIONS

CSC 207 H1S
Instructor(s): Shorser, Gries

Duration—2 hours

No Aids Allowed

You must earn at least 32 out of 80 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 8 questions on 19 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.

Marking Guide

# 1:	______/10
# 2:	______/10
# 3:	______/16
# 4:	______/ 6
# 5:	______/ 8
# 6:	______/ 6
# 7:	______/14
# 8:	______/10
TOTAL:	______/80

Good Luck!

Question 1. [10 MARKS]

All of the following statements apply to the Java programming language only. Indicate whether the statement is true or false for the Java Programming Language by circling **True** or **False** respectively.

Part (a) [1 MARK] Every time a class is instantiated, the constructor for class `Object` is called.
True / **False**

Part (b) [1 MARK] It is possible for a class to have two methods with the same name and different signatures.
True / **False**

Part (c) [1 MARK] It is possible to serialize an instance of a class without implementing any interfaces.
True / **False**

Part (d) [1 MARK] It is possible to use a static method without creating an instance of the class that contains it.
True / **False**

Part (e) [1 MARK] If `ClassB` extends `ClassA`, then no other class in the same program is allowed to extend `ClassA`.
True / **False**

Part (f) [1 MARK] `ClassC` does not inherit anything from `ClassA`. If `ClassB` extends `ClassA`, then `ClassB` cannot also extend `ClassC`.
True / **False**

Part (g) [1 MARK] If a method has return type `void`, it does not need to have a return statement.
True / **False**

Part (h) [1 MARK] A private method cannot be called directly by method `main`.
True / **False**

Part (i) [1 MARK] It is possible to create an object of type `List` that only contains objects of type `String`.
True / **False**

Part (j) [1 MARK] The "==" operator and method `equals` method can both be used to compare the values of two primitive variables.
True / **False**

Question 2. [10 MARKS]

Consider the following code. On the next page, you will write class TopThreeRanking.

```
public class Rankings {
  public static void main(String[] args) {
    System.out.println(TopThreeRanking.numObj);

    TopThreeRanking<String> ttr1 = new TopThreeRanking<>("yes", "no", "maybe");
    ttr1.inOrder();
    ttr1.reverseOrder();
    System.out.println(TopThreeRanking.numObj);

    TopThreeRanking<Integer> ttr2 = new TopThreeRanking<>(25, 16, 64);
    ttr2.inOrder();
    ttr2.reverseOrder();
    System.out.println(TopThreeRanking.numObj);

    TopThreeRanking<Character> ttr3 = new TopThreeRanking<>('C', 'A', 'B');
    ttr3.inOrder();
    ttr3.reverseOrder();
    System.out.println(TopThreeRanking.numObj);
  }
}
```

When the class TopThreeRanking is imported, the above code produces the following output:

```
0
yes
no
maybe
maybe
no
yes
1
25
16
64
64
16
25
2
C
A
B
B
A
C
3
```

Write code for class `TopThreeRanking` so that it creates the output described on the previous page when run with the given main method. Javadoc is not required.

Question 3. [16 MARKS]

Consider the code for `BigClass`, `LittleClass`, and `ExamDemo` at the end of this booklet.

Part (a) [12 MARKS] Write the output produced by this code.

Part (b) [4 MARKS] When the last exception is thrown, which methods are terminated? Write the names of the methods in the order that they terminate.

Question 4. [6 MARKS]

This question refers to BigClass, LittleClass, and ExamDemo at the end of this booklet. For each pair of statements, decide whether they will compile or not. Circle **Compiles** if they compile or **Does not compile** otherwise.

Part (a) [1 MARK]

```
LittleClass lc1= new LittleClass();
System.out.println(BigClass.doLittle(lc1));
```

This code: **Compiles** / **Does not compile**

Part (b) [1 MARK]

```
BigClass bc1 = new BigClass();
System.out.println(BigClass.doLittle(bc1.tunnelMethod(new LittleClass())));
```

This code: **Compiles** / **Does not compile**

Part (c) [1 MARK]

```
LittleClass lc2 = new LittleClass();
System.out.println(BigClass.doLittle(lc2.tunnelMethod(new BigClass())));
```

This code: **Compiles** / **Does not compile**

Part (d) [1 MARK]

```
BigClass bc1 = new BigClass();
System.out.println(BigClass.doLittle((BigClass) bc1.tunnelMethod(new LittleClass())));
```

This code: **Compiles** / **Does not compile**

Part (e) [1 MARK]

```
BigClass bc1 = new BigClass();
System.out.println(BigClass.doLittle((Object) bc1.tunnelMethod(new LittleClass())));
```

This code: **Compiles** / **Does not compile**

Part (f) [1 MARK]

```
LittleClass lc2 = new LittleClass();
System.out.println(BigClass.doLittle((LittleClass) lc2.tunnelMethod(new BigClass())));
```

This code: **Compiles** / **Does not compile**

Question 5. [8 MARKS]

Write CRC cards for a booking system that allows users to book a room for one hour. The system must have the following attributes:

1. All users of the system will be explicitly identified by a username and password.

2. Every user belongs to zero of more groups. Each group is associated with a set of rooms that any user in that group can book.

3. Each room has attributes such as seating capacity, equipment, etc.

4. When booking a room, the system should make sure that the people attending are fewer than or equal to the seating capacity.

5. Each user has an office location. The system stores the distance from each office to each room in order to rank the rooms by presence for that user.

6. Given an earliest time and a latest time, the system should be able to make a list of recommended rooms for a given user.

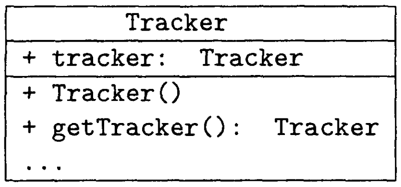
Use as many or as few CRC cards as you need. Indicate the page number(s) of your answer.

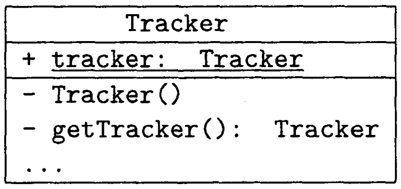
CONT'D...

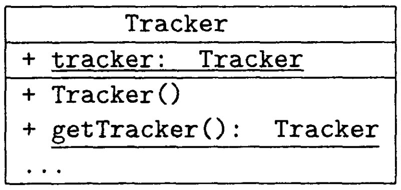
Question 6. [6 MARKS]

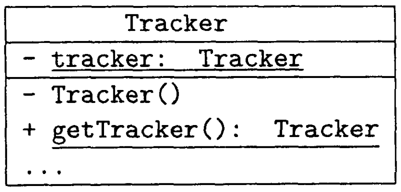
Consider the following problem: You have a class called **Tracker** which you want to instantiate once and only once in such a way that no other class can modify it or instantiate it.

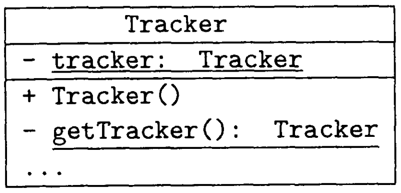
A solution to this problem is called the **Singleton Design Pattern**. Only one of the following UML diagrams provides a solution to this problem. Circle the correct solution and describe, beside it, which features of the class work together to solve this problem.











Question 7. [14 MARKS]

Part (a) [6 MARKS]

Write the substring(s) of `aBca11.2Bcccc` that satisfy each of the following regular expressions. If no such substrings exist, write "None".

(i) `a.{3}\d`

(ii) `3*2*B(([abc])\2)\1`

(iii) `(a|b|c)?(A|B|C)[abc][abc]`

Part (b) [4 MARKS]

Write a regular expression that is satisfied by any phone number that is written in the form `(xxx) xxx-xxxx` where the last three digits are the same as the digits in the parentheses. Your solution should contain fewer than 27 characters.

Part (c) [4 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.

	`(\sa)|e)+`	`[a-dsv-z]{2,3}.?`	`(a|e)\1\s`
`\s(se|ab)`			
`a?[winter]+`			
`.d.`			

Question 8. [10 MARKS]

Consider the following code. On the next page, identify ten ways that the code can be improved and explain how you would fix each. The improvements correct errors in the code or contribute to better style and design. We have included line numbers that you can use in your answers.

```
 1)  public class NotAClass {
 2)      public int letter ;
 3)
 4)      public static void main(String[] args) {
 5)         Object[] breakfast = new Object[]("Ontario", "Quebec");
 6)         Unrelated lunch = new Unrelated();
 7)         System.out.println((AlsoNotAClass)lunch.nameOfMethod());
 8)      }
 9)  }
10)
11)  public class AlsoNotAClass {
12)      int x;
13)      int y;
14)
15)      public AlsoNotAClass(int x, int y)
16)      {
17)          x = x;
18)          y = y;
19)      }
20)  }
21)
22)  public class Unrelated extends AlsoNotAClass {
23)      public Unrelated(){ }
24)
25)      public boolean nameOfMethod() {
26)           int april = (int) (Math.random() * 100);
27)           if (april < 50) {
28)               return true;
29)               throw new Exception();
30)           }
31)       }
31)  }
```

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

cont'd...

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.

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...

You can detatch this page, and all the following pages, if you like.

```
public class BigClass {

  public int ivar = 20;
  private static int svar = 2;

  public BigClass() {
    svar++;
  }

  public static int getSvar() {
    System.out.println("BigClass method");
    return svar;
  }

  public void printArg(int x) throws WrongMethodException {
    System.out.println(x);
    LittleClass temp = new LittleClass();
    temp.printOtherArg(x, 3);
  }

  public String toString() {
    return "BigClass variables: " + ivar + " " + svar;
  }

  public static BigClass doLittle(BigClass bc) {
    return bc;
  }

  public LittleClass tunnelMethod(LittleClass lc) {
    return lc;
  }

  public void noReturn(String s) {
    LittleClass temp = new LittleClass();
    temp.nextMethod(s);
  }
}
```

```
public class LittleClass extends BigClass {

  public int ivar = 10;
  private static int svar = 1;

  public LittleClass() {}

  public static int getSvar() {
    System.out.println("LittleClass method");
    return svar;
  }

  public void printOtherArg(int x, int y) throws WrongMethodException {
    System.out.println(y);
    throw new WrongMethodException();
  }

  public BigClass tunnelMethod(BigClass lc) {
    return lc;
  }

  public String toString() {
    return "LittleClass Variables: " + svar + " " + ivar;
  }

  public void nextMethod(String s) {
    throw new NullPointerException();
  }
}
```

```
public class WrongMethodException extends Exception {
}
```

```
public class ExamDemo {

  public static void main(String[] args) {
    System.out.println(BigClass.getSvar());
    BigClass obj1 = new BigClass();
    System.out.println(BigClass.getSvar());

    LittleClass obj2 = new LittleClass();

    System.out.println(BigClass.getSvar());
    System.out.println(obj1.ivar);
    System.out.println(obj2.ivar);
    System.out.println(obj1.toString());
    System.out.println(obj2.toString());

    try {
      obj1.printArg(7);
    } catch (WrongMethodException e) {
    }

    obj1.noReturn(new String());
    int q = BigClass.getSvar();
    System.out.println("Done");
  }
}
```

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