PLEASE HAND IN

UNIVERSITY OF TORONTO MISSISSAUGA DECEMBER 2008 FINAL EXAMINATION

CSC 207H5F
Instructor — Michelle Craig
Duration — 3 hours

PLEASE HAND IN

Examination Aids: Any handwritten or printed materials. No electronic aids.

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

*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 final examination consists of 6 questions on 18 pages (including this one). *When you receive the signal to start, please make sure that your copy of the examination is complete.*
There are two mostly-blank pages at the end of the exam that you may use if you run out of space.
You may be charged with an academic offence for possessing the following items during the writing of an exam unless otherwise specified: any unauthorized aids, including but not limited to calculators, cell phones, pagers, wristwatch calculators, personal digital assistants (PDAs), iPods, MP3 players, or any other device. If any of these items are in your possession in the area of your desk, please turn them off and put them with your belongings at the front of the room before the examination begins. A penalty may be imposed if any of these items are kept with you during the writing of your exam.
Please note, students are NOT allowed to petition to RE-WRITE a final examination.
Comments are not necessary unless specifically indicated in the question, you may abbreviate `System.out.println` as `S.o.p`, and you may assume that any necessary imports have been done.

MARKING GUIDE

# 1:	______/14
# 2:	______/20
# 3:	______/ 5
# 4:	______/14
# 5:	______/12
# 6:	______/ 5
TOTAL:	______/70

CONT'D ON PG 2

Question 1. [14 MARKS]

Part (a) [4 MARKS]

For each of the strings below, indicate whether or not it will match the regular expression `^b+c*\s*([a-z]+)\1((\d+A?)yes)$`. For each string that does match, indicate the sections of the string that matches group 2.

`"bll0yes"`	☐ match	☐ no match	group 2:
`"bbbcc ca99Ayes"`	☐ match	☐ no match	group 2:
`"bbc kkakka9yes"`	☐ match	☐ no match	group 2:
`"bcbc aa8Ayes"`	☐ match	☐ no match	group 2:
`"c Ayes"`	☐ match	☐ no match	group 2:
`"bcc catcat9yes"`	☐ match	☐ no match	group 2:

Part (b) [2 MARKS]

Sometimes when people are collaboratively editing a text file, they mark their comments to fellow authors, by adding a prefix on the comment lines. Write a regular expression to match lines of the file that begin with one of `[MWC]`, `[JC]`, or `[PG]`. Note that the square brackets are part of the prefix itself.

Your answer:

Part (c) [8 MARKS]

Your boss has given you a file shown on the right representing employee phone charges. He wants to know which individual had the highest charge, the value of that highest charge and also the total charges per department. A quick visual scan over the first few hundred lines of the file, reveals that the only department names are "sales", "development" and "admin". You also notice that the names can have any number of tokens and that while the file appears to consistently have a tab separating the name from the department, this isn't always the case. Some of the lines have only spaces and no tabs.

```
Bill Smith              sales          $450.31
Jane Barbara Patel      admin          $ 44.50
Chandrashekar           sales          $ 87.40
Mary Jane Van Hoot      development    $ 59
Tamir Heap              admin          $ 323
...
```

Complete the main method for the Java program on the next page so that it calculates what your boss requires. Assume that all the java imports have been correctly done.

CONT'D ON PG 3

```
public static void main(String[] args) throws IOException {
    // you fill in the pattern
  Pattern p = Pattern.compile("


  // declare and initialize any variables you need








  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
  String line = in.readLine();

  while (line != null) {
    Matcher m = p.matcher(line);
    if (m.matches()) {
      // extract the relevant parts of the line (you fill in)

      String dept =

      String name =

      double charge =

      // Calculate and save what is needed before going to next line








    } else { System.err.println("Line doesn't match RE:"+line); }
      line = in.readLine();
    }
  // print out total charges per department





  // print out maximum charge and individual with that charge

}
```

CONT'D ON PG 4

Question 2. [20 MARKS]

For this question you must complete a number of Java classes that will model an online ticket purchase. The class `Purchase` shown below models a single transaction with a client but that transaction may include many different tickets and different types of tickets. Interface `Ticket` is also shown.

```
public class Purchase {

  private String contactName;
  private List<Ticket> tickets = new ArrayList<Ticket>();

  public Purchase(String n) { contactName = n; }

  public void addTicket(Ticket t) { tickets.add(t); }

  public double totalPrice() {
    double totalPrice = 0.0;
    for (Ticket t: tickets) {
      totalPrice += t.getPrice();
    }
    return totalPrice;
  }

  public String toString() {
    String result = "Name: \t" + contactName;
    result +="\n Tickets Purchased\n";
    for (Ticket t: tickets) {
      result += t.toString() + "\n" + t.getPrice() + "\n";
    }
    result += "Total Price:\t" + totalPrice();
    return result;
  }
  public static void main(String[] args) {
    Purchase p = new Purchase("Purchaser Name");
    AirlineTicket t = new AirlineTicket("Passenger Name",50.99);
    t.addFlight("SEA","VAN",new GregorianCalendar(2008,11,15),189);
    t.addFlight("YYX","SEA",new GregorianCalendar(2008,11,16),189);
    t.addFlight("VAN","YYZ",new GregorianCalendar(2008,10,12),199);
    p.addTicket(t);

    Ticket hotel = new HotelStay("Delta Meadowvale",139.0,2);
    p.addTicket(hotel);
    System.out.println(p);
  }
}
```

```
public interface Ticket {
  public double getPrice();
}
```

Part (a) [5 MARKS]

On the next page, write the class `HotelStay` so that it can be used as one type of `Ticket` in the `Purchase` as shown in the example `main` method of `Purchase`. A `HotelStay` instance should have a name (of the hotel), a price per night and the number of nights for the visit.

CONT'D ON PG 5

Part (b) [10 MARKS]

On the next page write two classes that together model an airline ticket. The public outer class AirlineTicket implements Ticket and models an entire round-trip ticket which may visit any number of cities. Nested class Flight models a single flight between two cities. It has a departure city, a destination, a date and a cost. An AirlineTicket then has a list of Flights and its cost is the sum of the flight costs plus a surcharge. This surcharge plus the passenger name is provided to the constructor of AirlineTicket. Flights are added to an AirlineTicket with the following method.

```
/* Add Flight newFlight into the List of flights on this ticket
* inserting it based on date so that flights are ordered by date.  */
public void addFlight(String depCity, String dest, GregorianCalendar date, double price)
```

You must write the methods needed satisfy this description and to work with the code given in Purchase. There are lots of details from real world flights and tickets that this question ignores. Do not model tickets for multiple passengers. Do not worry about checking if the user attempts to add a second flight on a ticket at exactly the same time as an existing flight. You do need to use a GregorianCalendar object to represent the date and it is helpful that GregorianCalendar implements Comparable. Here is the relevant information from the JavaDoc for its compareTo method.

```
public int compareTo(Calendar anotherCalendar)
   Compares the time values (millisecond offsets from the Epoch) represented by two Calendar objects.
Returns:
the value 0 if the time represented by the argument is equal to the time represented by this Calendar;
a value less than 0 if the time of this Calendar is before the time represented by the argument;
and a value greater than 0 if the time of this Calendar is after the time represented by the argument.
```

CONT'D ON PG 6

Part (c) [1 MARK]

What will be the name of the source file that holds your answer for part b?

Part (d) [1 MARK]

Should the class `Flight` be static or not? YES ☐ NO ☐

Explain why or why not?

Part (e) [3 MARKS]

Write code that could be added to `Purchase.main()`. Your code must use an anonymous inner class to add a new Ticket to purchase p. This new ticket should have a `toString()` that returns `"Special Extra Ticket"` and should cost $49.23.

CONT'D ON PG 8

Question 3. [5 MARKS]

Write a static Java method called `LettersAvailable` that takes two `String` parameters named `available` and `need`. Your method should return true if and only if, you could completely make the string `need` from the letters in `available`. The table below shows a few example test cases that your method must pass.

need	available	method result
"cat"	"tacky"	true
"dog"	"dog"	true
"mouse"	"Mouse"	false
"tacky"	"cat"	false
"baby"	"bays"	false
"banana"	"ban"	false
""	""	true

CONT'D ON PG 9

Question 4. [14 MARKS]

Here is part of an XML document representing the nutritional content of a serving of many different food items.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="item_list.xsl" alternate=no ?>
<doc>
    <item name="Cheddar Popcorn">
        <cal>150</cal>
        <percent category="fat">12</percent>
        <percent category="sodium">10</percent>
    </item>
    <item name="Snickers">
        <cal>170</cal>
        <percent category="fat">12</percent>
        <percent category="sodium">4</percent>
        <percent category="carbs">7</percent>
    </item>
    <item name="Oreo Cookies">
        <cal>270</cal>
        <percent category="fat">17</percent>
        <percent category="carbs">14</percent>
    </item>
    ...
</doc>
```

Here is an XML file representing a menu of the items and servings that someone has consumed during the day.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="item_list.xsl" alternate=no ?>
<menu>
  <item name="Snickers" servings="3"/>
  <item name="Oreo Cookies" servings="1"/>
</menu>
```

Assume that the samples above demonstrate all of the tags and attribute names contained within the XML files. The attribute values for the names of the items and the nutrition categories (fat, carbs, etc.) are of course not all shown or even known to your program ahead of time. You may assume that there are no errors in either file, and that all numerical values are integers.

Part (a) [12 MARKS]

Write a Java application that requires three commandline arguments. The first parameter is the name of the XML nutritional information file (as shown in the first file above), the second is the name of an XML menu file, and the third is the nutritional category of interest.

Your program should print to standard output a report that lists for each menu item, the name of that item, how many servings were consumed, the relevant statistic for a single serving and the total for all the servings. At the end, it should print the total percentage of the daily recommended value of that category represented by that menu. Here is an example of the output for the menu and food files shown above with "fat" as the third commandline argument.

```
Item          Servings % per serving   Subtotal
Snickers         3          12            36
Oreo Cookies     1          17            17
Total Percentage of Daily Recommended Value 53
```

Include a header on your report but don't worry about matching the exact formatting of this example. Here are three additional requirements:

- If the user does not give three commandline arguments, print a helpful usage message to standard error and

CONT'D ON PG 10

immediately exit.

- If a food item appears in the menu and it isn't in the food xml file, immediatly throw an `UnknownMenuItemException` and don't finish processing the rest of the file. We will provide you space to write the `UnknownMenuItemException` class in a later subquestion.
- Not all foods in the food xml file give percentages for all the nutritional categories. If a food item occurs in the food xml file but there is no percent element for the nutritional category of interest, assume that that percentage of the daily recommended value for this food is 0.

CONT'D ON PG 11

Part (b) [2 MARKS] Write the class `UnknownMenuItemException` here.

CONT'D ON PG 12

Question 5. [12 MARKS]

Part (a) [4 MARKS]

Write a static method `findExceptions` that takes one String parameter `inputClass` that is the name of a Java class. Your code should return an object of type `Set<Class>` that contains all the classes for Exceptions that are declared to be thrown by any methods in `inputClass`. If the name of the parameter provided to `findExceptions` does not correspond to the valid class, your method should throw a `ClassNotFoundException`.

CONT'D ON PG 13

Part (b) [6 MARKS]

For this subquestion, you may use the `findExceptions` method that you wrote for part a. You may assume that it works according to the specification even if you were unable to fully answer that subquestion. Here, you must write a program that reads a class name `c` from the only command line argument and prints the names of all the declared exceptions that may be thrown by any methods in `c`. The output names should be in sorted order and an exception should only be listed once even if it is thrown by more than one method from class `c`. If the command line argument is not a valid class name, your program should print a helpful message to the standard error stream and exit gracefully. It should not print a stack trace.

Part (c) [2 MARKS]

Explain the relationship between exceptions that are declared in method signatures and checked exceptions.

CONT'D ON PG 14

Question 6. [5 MARKS]

Part (a) [4 MARKS]

You have a set of classes that all contain some similar methods with identical signatures. You are wondering about whether it makes sense to have the classes implement a common interface, extend a common concrete class, or extend a common abstract class. Outline the conditions under which each alternative is most appropriate.

- Extend a common abstract parent
- Extend a common concrete parent
- Implement a common interface
- Use neither a common interface nor a common parent class.

Part (b) [1 MARK]

In at most 2 sentences, explain the programming design philosophy of *fail early*.

CONT'D ON PG 15

Some API's that you might need.

```
==== java.util.HashSet<E> ====
 HashSet() // a new empty HashSet
 HashSet(Collection<? extends E> c) // a new Set with the same elements as c
 boolean add(E o) // If o not already in this set then add it and return true; else return false.
 Object clone() // Return a shallow copy of this HashSet.
 boolean contains(Object o) // Return true if this set contains o.
 boolean isEmpty() // Return true if this map contains no elements.
 Iterator<E> iterator() // Return an iterator over the elements of this set.
 boolean remove(Object o) // Remove o and return whether o was in the set.

==== java.util.ArrayList<E> ===
 ArrayList() // a new empty ArrayList
 boolean  add(E e) // Appends the specified element to the end of this list.
 void  add(int index, E element) // Inserts the specified element at the specified position
                                 // in this list.
 boolean  addAll(Collection<? extends E> c) // Appends all of the elements in the specified
                                            // collection to the end of this list
 void  clear() //Removes all of the elements from this list.
 Object clone() // Returns a shallow copy of this ArrayList instance.
 boolean  contains(Object o) // Returns true if this list contains the specified element.
 E  get(int index) // Returns the element at the specified position in this list.
 int  indexOf(Object o) // Returns the index of the first occurrence of the specified element
                        // in this list, or -1 if this list does not contain the element.
 boolean  isEmpty() //Returns true if this list contains no elements.
 E  remove(int index) // Removes the element at the specified position in this list.
 boolean  remove(Object o) // Removes the first occurrence of the specified element
                           // from this list, if it is present.
 E  set(int index, E element) // Replaces the element at the specified position in this list
                              // with the specified element.
 int  size() // Returns the number of elements in this list.

==== java.util.Collections ====
 static void sort(List<T> list)  // Sorts the specified list into ascending order,
                                 // according to the natural ordering of its elements.
==== java.lang.reflect.Method ====
Class<?> getDeclaringClass() // Returns the Class object representing the class
                             // or interface that declares the method
                             // represented by this Method object.
Class<?>[] getExceptionTypes() // Returns an array of Class objects that
                               // represent the types of the exceptions declared
                               // to be thrown by the underlying method
                               // represented by this Method object.
String getName() // Returns the name of the method represented by this Method
                 // object, as a String.
Class<?>[] getParameterTypes() // Returns an array of Class objects that
                               // represent the formal parameter types, in
                               // declaration order, of the method represented
                               // by this Method object.
Class<?> getReturnType() // Returns a Class object that represents the formal
                         // return type of the method represented by this Method
Object invoke(Object obj, Object... args) // Invokes the underlying method
                                 // represented by this Method object, on the
                                // specified object with the specified parameters.
```

CONT'D ON PG 18

```
==== java.lang.Class ====
static Class<?>forName(String className) // Returns the Class object associated
                                          // with the class or interface with the
                                          // given string name.

Class<?>[] getClasses()  // Returns an array containing Class objects representing
                         // all the public classes and interfaces that are members
                         // of the class represented by this Class object.

Annotation[] getDeclaredAnnotations() // Returns all annotations that are
                                      // directly present on this element.
Class<?>[] getDeclaredClasses()  // Returns an array of Class objects reflecting
                            // all the classes and interfaces declared as members
                            // of the class represented by this Class object.
Field getDeclaredField(String name) // Returns a Field object that reflects the
                            // specified declared field of the class or interface
Field[] getDeclaredFields() // Returns an array of Field objects reflecting all
                            // the fields declared by the class or interface
                            // represented by this Class object.
Method getDeclaredMethod(String name, Class<?>... parameterTypes)
      // Returns a Method object that reflects the specified declared method
      // of the class or interface represented by this Class object.
Method[] getDeclaredMethods() // Returns an array of Method objects reflecting
                              // all the methods declared by the class or
                              // interface represented by this Class object.
Class<?> getDeclaringClass() // If the class or interface represented by this
                             // Class object is a member of another class,
                             // returns the Class object representing the class
                             // in which it was declared.
Method getEnclosingMethod() // If this Class object represents a local or
                            // anonymous class within a method, returns a Method
                            // object representing the immediately enclosing
                            // method of the underlying class.
Field getField(String name) // Returns a Field object that reflects the
                            // specified public member field of the class or
                            // interface represented by this Class object.
Field[] getFields() // Returns an array containing Field objects reflecting all
                    // the accessible public fields of the class or interface
                    // represented by this Class object.
Class<?>[] getInterfaces() // Determines the interfaces implemented by the class
                           //or interface represented by this object.
Method getMethod(String name, Class<?>... parameterTypes)
      // Returns a Method object that reflects the specified public member
      // method of the class or interface represented by this Class object.
Method[] getMethods()// Returns an array containing Method objects reflecting
                     // all the public member methods of the class or interface
                     // represented by this Class object, including those
                     // declared by the class or interface and those inherited
String getName() // Returns the name of the entity (class, interface, array
                 // class, primitive type, or void) represented by this Class
                 // object, as a String.
Package getPackage() // Gets the package for this class.
Class<? super T> getSuperclass() //Returns the Class representing the superclass
boolean isAnonymousClass() //Returns true iff the underlying class is anonymous
```

Total Marks = 70