PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

AUGUST 2013 EXAMINATIONS

CSC 207 H1Y
Instructor: Petros Spachos

Duration — 3 hours

Examination Aids: None

Student Number: |_|_|_|_|_|_|_|_|_|_|

Family Name(s): _____

Given Name(s): _____

Do not turn this page until you have received the signal to start.
In the meantime, please fill in the identification section above and read the instructions below.

There is an API at the end of the exam for your reference.

In coding questions, you are always welcome to write helper methods.

You do not need to put `import` statements in your answers.

Javadoc and internal comments are not required except where indicated, although they may help us mark your answers.

There are two blank pages at the end for rough work. If you want any of it marked, indicate that clearly there, as well as in the question itself.

# 1:	_____/ 18
# 2:	_____/ 15
# 3:	_____/ 14
# 4:	_____/ 19
# 5:	_____/ 13
# 6:	_____/ 21
TOTAL:	_____/100

Good Luck!

CONT'D...

Question 1. [18 MARKS]

Part (a) [10 MARKS]

In the following table enter "T (true) or "F (false), as appropriate, for each statement.

Statement	*T/F*
svn checkout [url] - add new files to the repository.	
CRC -cards refer to Class Repository Collaboration cards.	
A List is a Collection that cannot contain duplicate elements.	
Validation checks that the product design satisfies or fits the specifications.	
In incremental development it is normal for the team to be working on several increments at the same time.	
All methods of an interface are abstract methods.	
In regular expressions spaces are not significant.	
Singleton pattern is a design pattern that restricts the instantiation of a class to one object.	
A Map is a Collection that cannot contain duplicate keys.	
JUnit is a unit testing framework for the Java.	

Part (b) [3 MARKS]

For your Assignment 2, you followed the Scrum process. Which are the three core roles of this process?

1.

2.

3.

Part (c) [2 MARKS]

Describe in one line one advantage and one disadvantage of waterfall model.

- Advantage:
-
- Disadvantage:
-

Part (d) [3 MARKS]

Write the three categories in which design patterns were originally grouped into.

1.

2.

3.

Question 2. [15 MARKS]

Part (a) [6 MARKS]

Consider the following code:

```
public class Test {
    public static void aMethod() throws Exception {
        try {
            throw new Exception();
        } finally {
            System.out.println("finally");
        }
    }

    public static void bMethod() throws Exception {
        System.out.println("entered bmethod");
        throw new Exception();
    }

    public static void main(String args[]) {
        try {
            aMethod();
            System.out.println("checkpoint 1");
            bMethod();
            System.out.println("checkpoint 2");
        } catch (Exception e) {
            System.out.println("exception");
        }
            System.out.println("finished");
    }
}
```

What is the output of the program?

Output:

Part (b) [9 MARKS]

Consider the following code:

```
class Animal {
    public void sayHi() {
        smile();
        System.out.println("Hi!");
    }

    public void smile() {
        ;
    }

    public void sayHiAnimal() {
        sayHi();
    }
}
```

```
class Dog extends Animal {
    public void sayHi() {
        System.out.print("Wrrrrrr.");
        super.sayHi();
    }

    public void smile() {
        System.out.print("Woof.");
        super.smile();
    }

    public void sayHiDog() {
        sayHi();
    }
}
```

[3 MARKS]For each line of code below, circle the right answer to indicate whether or not it compiles.

`an1.sayHiDog();`	compiles	does not compile
`an2.sayHiDog();`	compiles	does not compile
`dog.sayHiDog();`	compiles	does not compile

[6 MARKS]What is the output of the following program?

```
public class AnimalTest {
    public static void main(String args[]) {
        Animal an1 = new Animal();
        Animal an2 = new Dog();
        Dog dog = new Dog();
        an2.sayHiAnimal();
        an1.sayHi();
        an2.smile();
        dog.sayHi();
        dog.sayHiAnimal();
        dog.sayHiDog();
    }
}
```

Output:

Question 3. [14 MARKS]

Part (a) [6 MARKS]

Consider the following code:

```
public class Student {
    public String name;
    public double average;

    public Student(String Name, double avg){
        name=Name;
        average=avg;
    }
}
```

```
public class Course {

    List<Student> list = new ArrayList();

    public Iterator iterator() {
        return list.iterator();
    }
}
```

Complete the following method `maxAverage(Course c)` by using the function `iterator()` of class `Course`.

```
public class Test {
    /** This method must find the maximum average of any student in
     *  the course. It may assume that there is at least one student
     * in the course and that all averages are between 0 and 100.
     */
    public static double maxAverage(Course c) {
    // fill in the body of this method









    }

    public static void main(String[] args) {
        Student s1 = new Student("petros",5.7);
        Student s3 = new Student("john",4.9);

        Course CSC207 = new Course();
        CSC207.list.add(s1);
        CSC207.list.add(s3);
        double result=maxAverage(CSC207);
    }
}
```

Part (b) [8 MARKS]

The score during a soccer game might change frequently. These data needs to be updated to different systems when changed. We need a mechanism which allows us to notify the users whenever the score changes state. Assume you have a `LaptopGadjet.java` and a `MobileApp.java` which both needs to get notification whenever the score at `Score.java` changes. Part of the code for `LaptopGadjet.java` is given below, but it's incomplete. Complete the code for `LaptopGadjet.java` and write the code for `MobileApp.java` and `Score.java`. At the bottom of the next page, `ControlRoom.java` gives you an example of the main method along with the output.

```
   //This is LaptopGadget.java

public class LaptopGadget implements Observer {

    String name;

    public LaptopGadget(String Name){
        this.name=Name;
    }
//complete the necessary functionality









}
```

```
   //This is the MobileApp.java

public class                                          {









}
```

```
   //This is Score.java
public class                                    {









}

//This is  ControlRoom.java
public class ControlRoom {

    public static void main(String[] args) {
        MobileApp mpp = new MobileApp("MyMobile");
        LaptopGadget lp = new LaptopGadget("MyLaptop");
        Score startGame = new Score("0-0");

        startGame.addObserver(mpp);
        startGame.addObserver(lp);

        startGame.changeScore("0-1");

    }
}
```

Output:

```
!Update! for MyLaptop
LiveScore: 0-1
!Update! for MyMobile
LiveScore: 0-1
```

Question 4. [19 MARKS]

This question deals with regular expressions. Unless otherwise indicated, write plain regular expressions, with the mathematical sense, not as Java `Strings`.

Part (a) [3 MARKS]

Write all the strings that are matched by the following regular expression: `^ab?[cd]?$`

Part (b) [4 MARKS]

For each of the strings below, indicate whether or not it will match the following regular expression:

```
^a+(ab|cd?)(([^d])([\w]?))(.*?)
```

In columns **Group 1** and **Group 3**, write the contents of capturing group 1 and group 3, respectively, when the **String** is matched. Write "NO MATCH" in each table cell where a match is not found.

String	*Group 1*	*Group 3*
acaab		
aab4ca*		
abcefg		
ac5deee		

Part (c) [4 MARKS]

Some of the regular expressions below match this string:

```
This is the CSC 207 final examination
```

In column **Result**, write MATCH for the **Regular expression** which match the string or NO MATCH otherwise. There is a penalty for guessing. Every correct answer is 0.5 marks while every wrong answer you lose 0.5 marks. If there is no answer, there is no mark. The minimum mark on this question is 0.

Regular expression	*Result*
`^\w+.$`	
`^[\d|\w][.\s]*$`	
`^\w[a-zA-Z\d\s]*$`	
`([\w]+[\s]+[\w]+[\d])*`	
`([\w]+[\s]*)*`	
`[a-zA-Z ]*[0-9 ]*[\\w ]+$`	
`([a-zA-Z 0-9]*[\s]+)*[.]+$`	
`^[\w\s]*[^\d]*[\d]*[\w\s]*$`	

Part (d) [8 MARKS]

You have the following file which contains the Top 5 runners in 100 metres.

```
Usain Bolt          Jamaica         9.58
Taison Gay          USA             9.69
Yohan Blake         Jamaica         9.69
Asafa Powell        Jamaica         9.72
Nesta Carter        Jamaica         9.78
```

The name of the file is *IAAFRecord*. A quick visual scan over the first few lines of the file, reveals that in every line, there is the full name of an athlete, followed by the name of his country and finally his record in the distance. Complete the main method for the Java program below, so that it reads the *IAAFRecord* file and calculates the average time for the distance according to the Top 5 record file. Assume all the java imports have been correctly done.

```
public static void main(String[] args) {



        //define and compile your pattern


        //try to open the file


            FileInputStream fstream = new FileInputStream("IAAFRecord");
            BufferedReader br = new BufferedReader(new InputStreamReader(fstream));

            String line = br.readLine();

            while (line != null) {
            //declare your matcher


            //extract the relevant parts of the line









                }
                line = br.readLine();
            }
        } catch (Exception e) {
            System.out.print(e.getMessage());
        }
            //print out the average time

}
```

Question 5. [13 MARKS]

Part (a) [6 MARKS]

Complete the code below which should implement a program which reads user input as `String input`. Then it calls a boolean method `printClassInformation(String input)` which checks if there is any class with the name `input`. If there is a class with that name, it prints out the names of all the methods of the class. If any exceptions are thrown, have it print "Sorry, try again :)", and keep asking the user for an input until the user provides a valid input, which will be the name of an existing class. Make it produce output like this:

```
Class Name:WrongName
Sorry, try again :)
Class Name:Hockey
Method:0 public double Hockey.averageAge()
Method:1 public int Hockey.Year()
```

(but of course with whatever input from the user).

```
class Utilities {
    public boolean printClassInformation(String str) {









}
class Test {
    public static void main(String[] arg) {









    }
}
```

Part (b) [7 MARKS]

Write a method which has an input *String str* which is the name of a class to be analyzed. This method checks if there is a class with the name *str*. If there is a class, then it goes through all the fields of the class and checks the type of each field. The method returns an array of 3 elements which stores at index 0 the number of "double" data types, at index 1 the number of "int" data types and at index 2 the number of "long" data types. If any exceptions are thrown, have it print "Sorry :)", rather than letting the user see any exception.

```
public static int [] reportFieldTypes(String str)  {
```

```
}
```

Question 6. [21 MARKS]

Read the following program description and come up with a good program design.

A university start-up company wants to implement a system that will be used to book vacation during summer break. The system contains various packages: flights, hotels, cars and cruises. Each of these has a list of details: a unique Itinerary Number, a destination, a date and a price. All the users of the system should be able to search for vacation packages by destination or by dates and be able to see these information.

All the users of the system will be explicitly identified by a username and will require a password to login. Users will belong to two groups, customers and travel agents. The system needs also to have only one administrator. Each user has a favorite list of various packages which he or she is interested in. Customers have also a personal budget field (the money they would like to spend during vacation) and travel agents have a monthly package goal (the number of packages they should try to sell to customers per month) . Only travel agent can log into the system and change the price of a package. Whenever there is a change in the price of a package, the system administrator and the customer how is interested in that package should get a notification from the system.

By logging into the system, a customer should be able to:

- Checkout new vacation deals.
- View their iterations and packages they have already booked.
- Buy new vacation packages or add them to their favourite list.
- Get suggestions from the system for available packages according to personal budget.

While a travel agent should be able to:

- Add new packages to the system.
- Update the price of a package.
- Check the status of their monthly package goal.

Furthermore, for any user, the system should be able to print a list with all the favourite packages.

Part (a) [3 MARKS]

Which software development method would you use for this project idea and why?

Part (b) [3 MARKS]

What are some software design patterns which can be useful for this project and why?

Part (c) [15 MARKS]

Outline your design here by writing the Java classes, methods signatures and field declarations (static and instance variable and constants)

Here are some rules to keep you from getting hand cramps:

- **Dont implement any methods.** Just put empty curly braces unless a method is abstract.
- **Dont initialize any variable.** Just declare the type and the name.
- Follow the java naming conventions and choose good names for your classes, variables and methods.

Continue your design here:

Continue your design here:

[Use the space below for rough work. This page will **not** *be marked, unless you clearly indicate the part of your work that you want us to mark.]*

[Use the space below for rough work. This page will **not** *be marked, unless you clearly indicate the part of your work that you want us to mark.]*

Short Java APIs:

```
class Throwable:
    // the superclass of all Errors and Exceptions
    Throwable getCause() // the throwable that caused this throwable to get thrown
    String getMessage() // the detail message of this throwable
    StackTraceElement[] getStackTrace() // the stack trace info
class Exception extends Throwable:
    Exception(String m) // a new Exception with detail message m
    Exception(String m, Throwable c) // 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() // return a String representation.
    boolean equals(Object o) // = "this is o".
interface Comparable:
    // < 0 if this < o, = 0 if this is o, > 0 if this > o.
    int compareTo(Object 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.
    hasNext() // return true iff the iteration has more elements.
    next() // return the next element in the iteration.
    remove() // removes from the underlying collection the last element returned. (optional)
interface Collection extends Iterable:
    add(E e) // add e to the Collection
    clear() // remove all the items in this Collection
    contains(Object o) // return true iff this Collection contains o
    isEmpty() // return true iff this Set is empty
    iterator() // return an Iterator of the items in this Collection
    remove(E e) // remove e from this Collection
    removeAll(Collection<?> c) // remove items from this Collection that are also in c
    retainAll(Collection<?> c) // retain only items that are in this Collection and in c
    size() // return the number of items in this Collection
    Object[] toArray() // return an array containing all of the elements in this collection
interface List extends Collection, Iteratable:
    // A Collection that allows duplicate items.
    add(int i, E elem) // insert elem at index i
    get(int i) // return the item at index i
    remove(int i) // remove the item at index i
class ArrayList implements List
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:
    // An object that maps keys to values.
    containsKey(Object k) // return true iff this Map has k as a key
```

```
    containsValue(Object v) // return true iff this Map has v as a value
    get(Object k) // return the value associated with k, or null if k is not a key
    isEmpty() // return true iff this Map is empty
    Set keySet() // return the set of keys
    put(Object k, Object v) // add the mapping k -> v
    remove(Object k) // remove the key/value pair for key k
    size() // return the number of key/value pairs in this Map
    Collection values() // return the Collection of values
class HashMap implement Map
class Scanner:
    close() // close this Scanner
    hasNext() // return true iff this Scanner has another token in its input
    hasNextInt() // return true iff the next token in the input is can be interpreted as an int
    hasNextLine() // return true iff this Scanner has another line in its input
    next() // return the next complete token and advance the Scanner
    nextLine() // return the next line as a String and advance the Scanner
    nextInt() // return the next int and advance the Scanner
class String:
    char charAt(int i) // = the char at index i.
    compareTo(Object o) // < 0 if this < o, = 0 if this == o, > 0 otherwise.
    compareToIgnoreCase(String s) // Same as compareTo, but ignoring case.
    endsWith(String s) // = "this String ends with s"
    startsWith(String s) // = "this String begins with s"
    equals(String s) // = "this String contains the same chars as s"
    indexOf(String s) // = the index of s in this String, or -1 if s is not a substring.
    indexOf(char c) // = the index of c in this String, or -1 if c does not occur.
    substring(int b) // = s[b .. ]
    substring(int b, int e) // = s[b .. e)
    toLowerCase() // = a lowercase version of this String
    toUpperCase() // = an uppercase version of this String
    trim() // = 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) // print o without a newline
    println(Object o) // print o followed by a newline
class Pattern:
   static boolean matches(String regex, CharSequence input) // compiles regex and
       attempts to match input against it.
   static Pattern compile(String regex) // compiles regex into a pattern.
   Matcher matcher(CharSequence input) // creates a matcher that will match
       the given input against this pattern.
class Matcher:
   boolean find() // Attempts to find the next 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 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
class Observable:
    void addObserver(Observer o) // Add o to the set of observers if it isn't already there
    void clearChanged() // Indicate that this object has no longer changed
    boolean hasChanged() // Return true iff this object has changed.
    void notifyObservers(Object arg) // If this object has changed, as indicated by
        the hasChanged method, then notify all of its observers by calling update(arg)
        and then call the clearChanged method to indicate that this object has no longer changed.
    void setChanged() // Mark 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 tim

Total Marks = 100