UNIVERSITY OF TORONTO MISSISSAUGA
OCTOBER 2018 MIDTERM EXAMINATION
CSC 207H5F
Introduction to Software Design
Arnold Rosenbloom
Furkan Alaca
Larry Zhang
Duration — 110 minutes
Aids: *None*

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

Last Name: _____

First Name: _____

Signature: _____

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please fill out the identification section above. *Good Luck! Take a deep breath. You got this! :) This is your chance to show us what you've learned. Remember, a number does not define you.)*

This midterm examination consists of 5 questions on 18 pages (including this one), printed on both sides of the paper. **NOTE:** The end of the exam has an API and code for an earlier question. Feel free to rip these out to make it easier to refer to them.

When you receive the signal to start, please make sure that your copy of the examination is complete. Answer each question directly on the examination paper, in the space provided.

Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0) so write legibly.

# 1:	_____/ 8
# 2:	_____/ 4
# 3:	_____/ 7
# 4:	_____/ 5
# 5:	_____/16
TOTAL:	_____/40

Good Luck!

Question 1. [8 MARKS]

Short Answers

1.1. Which of the following access modifiers allow access from a child class (i.e., subclass)? Circle all that apply. [2 marks]

(a) public

(b) private

(c) protected

1.2. Which of the following statements about `static` are **true**? Circle all that apply. [2 marks]

(a) A static method is invoked only once.

(b) A static variable is initialized only once.

(c) The `main` method is a static method.

(d) Static variables reduce dependencies between objects.

1.3. What is the output of the following line of Java code? Write your answer in the space below. [2 marks]

```
System.out.println((float)5/2 + (double)(5/2) + (int)(5/2.0))
```

1.4. Consider a `Book` class with methods `open()`, `close()`, `getWeight()` and `getNumberOfPages()`. Below, explain **concisely** why the class `ElectronicBook` should NOT be a subclass of `Book`. [2 marks]

Question 2. [4 MARKS]

Consider the following program:

```
import java.util.ArrayList;

public class ArrayCobbler {

        public static void main(String args[]) {
                ArrayList<Integer> a1 = new ArrayList<Integer>();
                ArrayList<Integer> a2 = new ArrayList<Integer>();

                a1.add(new Integer(10));
                a1.add(new Integer(4));

                a2.add(new Integer(7));
                a2.add(new Integer(9));
                a2.add(new Integer(11));

                cobbleArrays(a1, a2);

                System.out.println("Contents of a1: ");
                for(int i = 0; i < a1.size(); i++)
                        System.out.println(a1.get(i));

                System.out.println("Contents of a2: ");
                for(int i = 0; i < a2.size(); i++)
                        System.out.println(a2.get(i));
        }

        public static void cobbleArrays(ArrayList<Integer> a1, ArrayList<Integer> a2) {
                a2 = a1;
                a2.add(new Integer(42));

                a1 = new ArrayList<Integer>();
                a1.add(new Integer(41));
        }
}
```

Print the output generated by running the above program:

Question 3. [7 MARKS]

Consider the following classes:

```
class A {
    public int num;

    public A() {
        num = 2;
    }

    public void zap(){
        this.print();
    }

    public void print() {
        System.out.println("A is " + num);
    }

    public void hello() {
        System.out.println("Hello, I am A.");
    }
}

class B extends A {

    public B() {
        super();
        num++;
    }

    public void print() {
        System.out.println("B is " + num);
    }

    public void hello() {
        System.out.println("Hello, I am B.");
    }
}

class C extends B {

    public C() {
        super();
        num++;
    }

    public void print() {
        System.out.println("C is " + num);
    }
}
```

(Continued on the next page ...)

Complete the comments in the main method below. If a line would produce an error, write "ERROR" and comment it out.

```
public class D {

    public static void main(String [] args){

        A var1 = new A();
        B var2 = new C();

        var1.print();       // THE OUTPUT IS: ____________________________


        var2.print();       // THE OUTPUT IS: ____________________________


        var1.hello();       // THE OUTPUT IS: ____________________________


        var2.hello();       // THE OUTPUT IS: ____________________________


        ((C) var2).hello(); // THE OUTPUT IS: ____________________________


        ((B) var1).print(); // THE OUTPUT IS: ____________________________


        var2.zap();         // THE OUTPUT IS: ____________________________

    }
}
```

Question 4. [5 MARKS]

JUnit Test

The following is the code of a class named `SuperArray`. This class keeps a list of integers and has a method called `doubleTheOdds()` which doubles the the odd numbers in the list. Please read the Javadoc of the method carefully.

Your task is to write a JUnit Tester for the `doubleTheOdds()` method (no need to test other methods). The first few lines of code are given to you. For full marks, you must write a **minimal** set of test cases that test the method **thoroughly**. Marks will be docked for both missing test cases and unnecessary test cases. You may **not** add any code to the `SuperArray` class.

```
public class SuperArray {

    private ArrayList<Integer> list;

    public SuperArray(ArrayList<Integer> list) {
        this.list = list;
    }

    public ArrayList<Integer> getList() {
        return this.list;
    }

    /**
     * Double the odd numbers in the list, i.e., each odd number is multiplied by 2
     * The even numbers stay unchanged.
     * This method should NEVER raise an error.
     */
    public void doubleTheOdds() {
        // implementation omitted
    }
}
```

Your tester code starts here. All your test cases should be written inside one method `testDoubleTheOdds`.

```
import static org.junit.Assert.*;
import static org.junit.*;

public class SuperArrayTester {

    @Test
    public void testDoubleTheOdds() {
        // YOUR TEST CASES HERE
```

(continued)

Question 5. [16 MARKS]

OO Java This question involves the **Jug** and **JugPuzzle** code in the appendix to this exam. Please scan it carefully, it is not exactly the same as the solution presented in Tutorial. You can use, but not modify Jug and JugPuzzle.

Part (a) [1 MARK]

Some methods and variables in JugPuzzle and Jug are protected, what does this mean?

Part (b) [5 MARKS]

Write class JugPuzzle2, a subclass of JugPuzzle, which can be used to play the following variation of the JugPuzzle game. A JugPuzzle2 is like a JugPuzzle except there are 4 jugs with capacities 8,5,3,2 respectively. Initially, the jugs have amounts 3,3,3,1 respectively. The target is 4,4,2,0 respectively.

Part (c) [10 MARKS]

Instead of extending class JugPuzzle for each variant of JugPuzzle, consider the following approach. There are two new classes, GJug and GJugPuzzle. A GJug is like a Jug, except it takes three inputs: its capacity, amount, and target amount. To play a particular variant of JugPuzzle, the user constructs a GJugPuzzle with the appropriate collection of GJugs. For example, to play the puzzle in Assignment 1...

```
GJug [] gJugs = new GJug[3];
gJugs[0]=new GJug(8,8,4);
gJugs[1]=new GJug(5,0,4);
gJugs[2]=new GJug(3,0,0);
GJugPuzzle gJugPuzzle=new GJugPuzzle(gJugs);
```

To play the variant in JugPuzzle2 above, you could instead ...

```
GJug [] gJugs2 = new GJug[4];
gJugs2[0]=new GJug(8,3,4);
gJugs2[1]=new GJug(5,3,4);
gJugs2[2]=new GJug(3,3,2);
gJugs2[3]=new GJug(2,1,0);
GJugPuzzle gJugPuzzle2=new GJugPuzzle(gJugs2);
```

On the next few pages, write classes GJug and GJugPuzzle. If you need to, you can use, but not modify Jug and JugPuzzle.

(continued...)

(continued...)

[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 Object:
    String toString() // return a String representation.
    boolean equals(Object o) // = "this is o".
class Assert:
    assertEquals(String message, Object expected, Object actual)
          Asserts that two objects are equal.
    assertArrayEquals(String message, Object[] expecteds, Object[] actuals)
          Asserts that two object arrays are equal
    assertTrue(boolean condition)
          Asserts that a condition is true.
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)
class Arrays:
    static sort(T[] list) // Sort list; T can be int, double, char, or Comparable
class Collections:
    static max(Collection coll) // the maximum item in coll
    static min(Collection coll) // the minimum item in coll
    static sort(List list) // sort list
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 Set extends Collection implements Iteratable:
    // A Collection that models a mathematical set; duplicates are ignored.
class HashSet implements Set
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 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 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
```

Appendix ...

```
package ca.utoronto.utm.assignment1.q1a;
public class Jug {
    private int capacity = 0; // Always 0<=capacity
    private int amount = 0; // Always 0<=amount<=capacity

    public Jug(int capacity, int amount) {
        if (0 <= amount && amount <= capacity) {
            this.capacity=capacity;
            this.amount=amount;
        }
    }

    public Jug(int capacity) { this(capacity, 0); }

    public void spillInto(Jug other) {
        if (this == other) return;
        else this.remove(other.add(this.amount));
    }

    public int add(int addAmount) {
        if (addAmount < 0) return 0;
        int freeSpace = this.capacity - this.amount;
        if (addAmount > freeSpace)addAmount = freeSpace;
        this.amount = this.amount + addAmount;
        return addAmount;
    }

    public int getAmount() { return amount; }

    public int getCapacity() { return capacity; }

    public int remove(int removeAmount) {
        if (removeAmount < 0) return 0;
        if (removeAmount > this.amount)removeAmount = this.amount;
        this.amount = this.amount - removeAmount;
        return removeAmount;
    }

    public String toString() { return "(" + amount + "/" + capacity + ")"; }
}
```

Appendix ...

```
package ca.utoronto.utm.assignment1.q1a;
public class JugPuzzle {
    protected Jug[] jugs;
    private int moves;

    public JugPuzzle() {
        this.jugs = new Jug[3];
        this.jugs[0] = new Jug(8, 8);
        this.jugs[1] = new Jug(5);
        this.jugs[2] = new Jug(3);
        this.moves = 0;
    }

    protected JugPuzzle(Jug [] jugs) {
        this.jugs = jugs;
        this.moves=0;
    }

    public int getMoves() { return moves; }

    protected Jug [] getJugs() { return this.jugs; }

    public boolean isPuzzleSolved() {
        return jugs[0].getAmount() == 4 && jugs[1].getAmount() == 4;
    }

    public void move(int from, int to) {
        if(0<=from && from<jugs.length && 0<=to && to<jugs.length){
            jugs[from].spillInto(jugs[to]);
            moves++;
        }
    }

    public String toString() {
        String s = moves + " ";
        for(int i=0;i<this.jugs.length;i++) {
            s+=" "+i+":"+jugs[i];
        }
        return s;
    }

    public int getNumJugs() { return this.jugs.length; }
}
```

Total Marks = 40