UNIVERSITY OF TORONTO MISSISSAUGA
October 2025 MIDTERM EXAMINATION
CSC 207H5F
Introduction to Software Design
Arnold Rosenbloom
Marc De Benedetti
Duration — 110 minutes
Aids: *None*

Signature: _____

*Do **not** turn this page until you have received the signal to start.*
This midterm examination consists of 5 question on 18 page (including this one), printed on both sides of the paper. **NOTE:** The end of the exam has some API reference sheets. **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: _____/ 6
2: _____/17
3: _____/ 7
4: _____/10
5: _____/17

TOTAL: _____/57

Good Luck!

Question 1. [6 MARKS]

Short Answers

1.1. [2 marks] Consider the following code in `FooBar.java`.

```
public class FooBar{
    public static String word = "Hello";
    public String str;

    public FooBar(String s){
        this.str = s;
    }

    public static void main(String[] args){
        FooBar fb = new FooBar("World");
        System.out.println(fb.word);
    }
}
```

What will be the result of running `java FooBar.java` in the console?

(a) `Hello`

(b) `World`

(c) It will throw an error because `word` is not an attribute of the `fb` object

(d) None of the above: something else will happen

1.2. [4 marks] Fill in the blanks for following Java class.

```
public class Student{
    private int number;
    private String name;

    //This is the constructor

    public  Student    (int num, String name){
        this.number = num;
        this.name = name;
    }

    //Getter method for student's name

    public String getName(){

        return this.name;
    }
}
```

+1 for each blank

Question 2. [17 MARKS]

This question uses a user-defined class `Card` that models a single playing card and a class `Hand` that models a set of cards. A card has a suit (either "clubs", "hearts", "diamonds" or "spades") and a face value. At any point in time, a card can be either face-up or face-down. The abbreviated JavaDoc for the `Card` class is provided here.

```
/** Generate a card at random independently with equal probability for all cards.
*   The card is initially face-down.
*/
public Card()

/** Return true iff the card is face-up. */
public boolean isFaceUp()

/** Flip the card over. If it was previously face-up it becomes face-down and
* if it was previously face-down, it becomes face-up. */
public void flip()

/** Return the suit of the card */
public String getSuit()

/** Return the integer representation of the face value of the card.
*   Jack, Queen, King and Ace are represented by 11, 12, 13 and 1 respectively. */
public int getValue()
```

Here is the partially-complete `Hand` class. In card-playing games, a *hand* is just a fancy name for a set of cards. Unlike in real card games, these `Hands` can contain two of the same `Card`. In each piece of this question, you will write another method to add to this class.

```
import java.util.ArrayList;

public class Hand {
    public ArrayList<Card> cards;

    /** Generate a Hand consisting of n cards all face-down.*/
    public Hand(int n) {
        cards = new ArrayList<Card>();
        for (int i = 0; i < n; i++) {
            Card c = new Card();
            cards.add(c);
        }
    }
}
```

(a) [3 marks]

Complete the `Hand`'s `getValue()` method.

```
/** Return the sum of the face values of all the cards in the Hand. */
public int getValue() {
    int total = 0;
    for (int i = 0; i < cards.size(); i++){      +1 for correctly looping through all the cards
        total += cards.get(i).getValue();        +1 for correctly retrieving card's value
    }
    return total                                 +1 for returning total

}
```

(b) [3 marks]
Write an overloaded `getValue` method that takes a single String argument representing a (valid) suit, and returns the sum of the face values of all the cards in the hand, from that suit.

```
public int getValue(String suit){                          +1 for correct method signature

    int total = 0;
    for (int i = 0; i< cards.size(); i++){
        if (cards.get(i).getSuit().equals(suit)){          +1 for correctly checking matching suit
            total += cards.get(i).getValue();
        }
    }
    return total;                                          +1 for everything else being correct
}
```

(c) [4 marks]
Write a method for class `Hand` called getFaceupCards() that returns an `ArrayList` of Cards that are face up.

```
                                                           -0.5 for missing generics.
public ArrayList<Card> getFaceupCards(){                   +1 for correct method signature.

    ArrayList<Card> result = new ArrayList<Card>();        +1 for correctly creating new ArrayList<Card>.
    for (int i = 0; i< cards.size(); i++){
        if (cards.get(i).isFaceUp()){                      +1 for correctly checking face up
            result.add(cards.get(i));                       and correctly adding to result.
        }
    }                                                      +1 for everything else being correct.
    return result;
}
```

(d) [7 marks]
Below is a skeleton `JUnit` testing suite for `Hand` (assume all imports are done). Set up and implement one test case for `getValue()` and one test for `getFaceupCards()`.

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.*;

public class HandTest {

    @Test
    public void testgetValue(){
        Hand h = new Hand(3);
        int ans;
        ans = h.cards.get(0).getValue() + h.cards.get(1).getValue() + h.cards.get(2).getValue();

        assertEquals(h.getValue(), ans, "getValue with 3 cards");
    }
     +3 marks
         +1 for correct signature (with "test" in the name") on BOTH tests +0.5 for one correct
         +1 for set up (creating a hand AND manually calculating the correct answer)
         +1 for correct assert

         0 marks if they use a return instead of assert
         -1 if they use a return and an assert
    @Test
    public void testgetFaceupCards(){
         Hand h = new Hand(5);
         ArrayList<Card> ans;

         //force all to be face down
         for (int i = 0; i< h.cards.size(); i++){
             if (h.cards.get(i).isFaceUp()){
                 h.cards.get(i).flip();
             }
         }
        //Now set a known amount to be face up
        h.cards.get(0).flip(); h.cards.get(3).flip(); h.cards.get(4).flip();

        //find the correct sum of known face up cards
        ans.add(h.cards.get(0)); ans.add(h.cards.get(3)); ans.add(h.cards.get(4));

        asserttrue(h.getFaceupCards().equals(ans), "getFaceupCards with 3 face up cards");

        +4 marks
            +1 set up: manually force a known cards to be face up
            +1 set up: creating your ans ArrayList correctly
            +1 correctly comparing equality between the two ArrayLists
            +1 for correct assert

            Note: more than one correct way to check equality of
            ArrayLists (e.g. element-wise checking in for-loop).
```

Question 3. [7 MARKS]

Consider the following classes:

```
class A {
    public int num;

    public A() { this(7); }

    public A(int num){ this.num=num; }

    public void f() {
        System.out.println("A f num=" + num);
    }

    public void g() {
        System.out.println("A g num=" + num);
    }

    public void h() { this.f(); }
}

class B extends A {
    public B() { super(); num++; }

    public B(int count){ super(count); }

    public void f() {
        System.out.println("B f num=" + num);
    }

    public void g() {
        System.out.println("B g num="+num);
    }
}
class C extends B {
    public C() { super(); num++; }

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

(continuted on next page...)

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

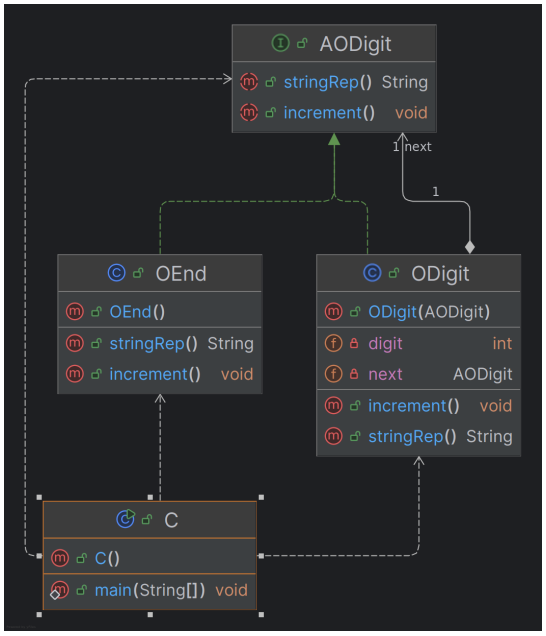
```
public class D {
    public static void main(String [] args){
        A a = new A();
        B b = new C();
        a.g();       // THE OUTPUT IS:  A g num=7

        a.f();       // THE OUTPUT IS:  A f num=7

        b.f();       // THE OUTPUT IS:  C f num=9

        b.g();       // THE OUTPUT IS:  B g num=9

        ((C) b).g(); // THE OUTPUT IS:  B g num=9

        //((B) a).g(); // THE OUTPUT IS:  ERROR

        b.h();       // THE OUTPUT IS:  C f num=9
    }
}
```

+1 mark for each correct blank

Question 4. [10 MARKS]

On the following page, draw the UML diagram for the following collection of classes.

```
public interface AODigit {
    public void increment();
    public String stringRep();
}
public class ODigit implements AODigit {
    private AODigit next; private int digit;

    public ODigit(AODigit next) { this.next=next; this.digit=0; }

    @Override
    public void increment() {
        this.digit=(this.digit+1)%10;
        if(this.digit==0)this.next.increment();
    }

    @Override
    public String stringRep() { return this.next.stringRep()+this.digit; }
}
public class OEnd implements AODigit {

    @Override
    public void increment() { return; }

    @Override
    public String stringRep() { return ""; }
}

public class C {
    public static void main(String[] args) {
        AODigit o = new OEnd();
        o = new ODigit(o);
        o = new ODigit(o);
        o = new ODigit(o);
        for(int i=0;i<25;i++) {
            System.out.println(o.stringRep());
            o.increment();
        }
    }
}
```



+1 For having all 4 classes/interfaces represented in some way.

+1 ODigit(AODigit) constructor
+1 ODigit digit and next including their types

+1 Encapsulation (public and private) everything public except private digit and next
+1 Methods: correct methods present for AODigit, OEnd, ODigit
+1 `AODigit` Connections: `OEnd` and `ODigit` connecting to `AODigit`
+1 Interface: `OEnd` and `ODigit`implements interface w/dotted line with solid arrow head

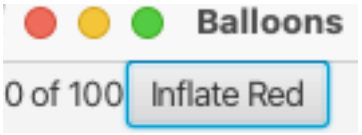
+1 `C` connects: to `AODigit`, `OEnd` and `ODigit`
+1 for dependency : `C` has dotted lines with open arrow

+1 Association: `ODigit` to `AODigit` solid line with open arrowhead.
-1 if AODigit is not labelled abstract in some way

(continued ...)

Question 5. [17 MARKS]

Below is a JavaFX application that produces the following GUI.




```
public class BadBalloon extends Application implements EventHandler<ActionEvent> {
    private String color = "Red";
    private int amount = 0;
    private int capacity = 100;
    private boolean isPopped = false;
    Label guiView1 = new Label(amount + " of " + capacity);
    GridPane grid = new GridPane();
    Button b1 = new Button("Inflate "+ getColor());

    public String getColor() {return this.color;}

    public void inflate(int amount) {
        if(amount<0 || isPopped){return;}
        this.amount = this.amount+amount;
        if(this.amount>this.capacity){this.pop();}
    }
    public void pop(){
        if(this.isPopped){return;}
        this.amount=0;
        this.capacity=0;
        this.isPopped=true;
    }
    public static void main(String[] args) {launch(args);}

    public void start(Stage stage) throws Exception {
        guiView1 = new Label(amount + " of " + capacity);
        grid = new GridPane();
        b1 = new Button("Inflate "+ getColor());

        grid.add(guiView1, 0, 0);
        grid.add(b1, 1, 0);
        b1.setOnAction(this);
        Scene scene = new Scene(grid, 200, 80);
        stage.setTitle("Balloons");
        stage.setScene(scene);
        stage.show();
    }

    @Override
    public void handle(ActionEvent e) {
        inflate(10);
        guiView1.setText(amount + " of " + capacity);
        if(isPopped){
            Button jb = (Button)e.getSource();
            jb.setDisable(true);
        }
    }
}
```

Re-implement this `BadBalloon` application as an MVC application. You do not have to worry about import statements. You should have clearly labelled Model, View and Controller classes, and a 4th `Application.java` class that creates instances of Model, View and Controller, connects them appropriately and then shows the GUI. You will still receive most of the marks if you can only split BadBalloon into MVC classes.

```
Model 3pts
public class Balloon extends Observable {  +1 for extending Observable

    private String color = "Red";
    private int amount = 0;
    private int capacity = 100;
    private boolean isPopped = false;

    public String getColor() {return this.color;}
    public int getAmount() {return this.amount;}
    public int getCapacity() {return this.capacity;}
    public boolean isPopped() {return this.isPopped;}

    public void inflate(int amount) {
        if(amount<0 || isPopped){return;}
        this.amount = this.amount+amount;
        if(this.amount>this.capacity){this.pop();}
        this.setChanged();                  +1 for setChanged in correct spots
        this.notifyObservers();             +1 for notifyObservers in correct spots
    }
    public void pop(){
        if(this.isPopped){return;}
        this.amount=0;
        this.capacity=0;
        this.isPopped=true;
        this.setChanged();
        this.notifyObservers();
    }
}


Note: max 2/3 if all of the above concepts are there, but not in correct places.
Max 1/3 if some notion of observable present
```

(continued ...)

```
View is worth 4 pts                                +1 Implements Observer
public class BalloonView extends Label implements Observer{
                        +1 extends Label

    public BalloonView(){
        super();
    }
    public BalloonView(String text) {
        super();
        this.setText(text);
    }

    @Override                              +1 Update method is present
    public void update(Observable o,  Object arg) {
        Balloon balloon = (Balloon)o;
                this.setText(balloon.getAmount() + " of " + balloon.getCapacity());
        }                                  +1 Update correctly implemented (get new model status and
    }                                         update text field)
}
```

Note, alternative valid solutions may include having View be the entire layout (GridPane). This would mean the view should have label and button attributes, and be responsible for disabling the button if needed.

```
public class BalloonView extends GridPane implements Observer{

    Label guiView1;
    Button b1;
    Balloon balloon;
    public BalloonView(Balloon b, BalloonController balloonController) {
        super();
        this.balloon=b;

        guiView1 = new Label(b.getAmount() + " of " + b.getCapacity());
        b1 = new Button("Inflate "+ b.getColor());

        this.add(guiView1, 0, 0);
        this.add(b1, 1, 0);

        b1.setOnAction(balloonController);
        balloon.addObserver(this);
    }

    @Override
    public void update(Observable o,  Object arg) {
        this.guiView1.setText(this.balloon.getAmount() + " of " + this.balloon.getCapacity());
        if(this.balloon.isPopped()){
            b1.setDisable(true);
        }
    }
}
```

(continued ...)

```
Controller 4pts                              +1 implements EventHandler<ActionEvent>
                                             Only 0.5 extends EventHandler<ActionEvent>)
public class BalloonController implements EventHandler<ActionEvent> {
   private Balloon balloon;                  +1 for Controller holding on to Model.

    public BalloonController(Balloon balloon){
        this.balloon=balloon;
    }

    @Override
    public void handle(ActionEvent e) {     +1 correct handle signature
        this.balloon.inflate(10);           +1 Correct handle implementation (inflate and disable)


//        if(balloon.isPopped()){
//            Button jb = (Button)e.getSource();
//            jb.setDisable(true);
//        }
    }
}
```

Note that teal-comment block above will NOT be in `View` if students had the alternate `View` approach

(continued ...)

```
Application worth 4 pts
public class BalloonApplication extends Application {

    public static void main(String[] args) {launch(args);}

    public void start(Stage stage) throws Exception {
                // Make a model
                Balloon b = new Balloon();

                // Make the view
                BalloonView v = new BalloonView(b.getAmount() + " of " + b.getCapacity());

                Button button = new Button("Inflate Red");

                // Make the controller                  +1 make instances of M, V, and C
                BalloonController c = new BalloonController(b);

                //Hook everything up
                b.addObserver(v);                       +1 for hooking up observer to model
                button.setOnAction(c);                  +1 for hooking up controller to button

                GridPane grid = new GridPane();
                grid.add(v, 0, 0);
                grid.add(button, 1, 0);

                Scene scene = new Scene(v, 200, 80);
                stage.setTitle("Balloons");
                stage.setScene(scene);
                stage.show();                           +1 for hooking everything else up
        }
}
```

Separating into 4 files worth +2pts
(and making a reasonable attempt at implementing each of the 4 components)
Alternate approach of having all of the GUI be the view: `Application` should resemble the following:

```
public class BalloonApplication extends Application {

    public static void main(String[] args) {launch(args);}

    public void start(Stage stage) throws Exception {
        Balloon b = new Balloon();
        BalloonController c = new BalloonController(b);
        BalloonView v = new BalloonView(b,c);

        Scene scene = new Scene(v, 200, 80);
        stage.setTitle("Balloons");
        stage.setScene(scene);
        stage.show();
    }
}
```

[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
    add(E e) //add e to the ArrayList
    size() // return the number of items in the ArrayList
    get(int i) // return the item at index i
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
```

Midterm— Solutions

Total Marks = 57