COMP 3021 Final Exam - Fall 2017 - HKUST

Date: Dec 15, 2017 (Friday)
Time Allowed: 3 hours, 12:30 – 3:30 pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **6** questions on **36** pages (including this cover page, appendices and 2 blank sheets at the end). You may detach the appendix pages and the blank sheets if you wish.
3. Write your name, student ID, email address, and seat number on this page.
4. Answer ALL questions and write your answers in the space provided.
5. All programming codes in your answers must be written in the Java version as taught in the class (i.e. Java 8).
6. For programming questions, unless otherwise stated, you are **NOT** allowed to define helper methods, additional structures, nor any library methods not mentioned in the questions.
7. Materials other than approved calculators are not allowed for this exam.

Student Name	
Student ID	
Email Address	
Seat Number	

For T.A. Use Only

Problem	Topic	Score
1	Multiple Choice Questions	/ 20
2	Interface	/ 7
3	Generic Class	/ 11
4	Lambda Expressions	/ 15
5	Multi-Threading	/ 20
6	GUI and Event-Driven Programming	/ 27
Total		/ 100

Problem 1 [20 points] Multiple Choice Questions

Write your answer of each question in the table below. Each question carries 2 points.

***** Note: Some questions may have more than one answers and no partial credit will be given for these questions. *****

Question	Answer	Question	Answer
(a)		(f)	
(b)		(g)	
(c)		(h)	
(d)		(i)	
(e)		(j)	

(a) Given

```
public abstract class AbstractClassTest {
  public int getValue() { return 1; }
  public abstract class InnerAbstractClass {
    public int getValue() { return 2; }
  }
  public static void main(String[] args) {
    AbstractClassTest test = new AbstractClassTest() {
      public int getValue() { return 3; }
    };
    AbstractClassTest.InnerAbstractClass inner = test.new InnerAbstractClass() {
      public int getValue() { return 4; }
    };
    System.out.println(inner.getValue() + " " + test.getValue());
  }
}
```

Which of the following best describes the given program?

A. It prints "`4 3`".

B. It prints "`1 2`".

C. It prints "`1 4`".

D. An exception occurs at runtime.

E. Compilation error occurs.

(b) Given

```
1  class SuperClass {
2    SuperClass() { }
3    SuperClass(String s) { }
4  }
5  class SubClass extends SuperClass {
6    SubClass() { }
7    SubClass(String s) { super(s); }
8    void method() {
9      // insert code here
10   }
11 }
```

Which of the following statements create an anonymous inner class within class `SubClass` at line 9? (Choose ALL that apply.)

A. `SuperClass superClass = new SubClass() { };`

B. `SuperClass superClass = new SuperClass() { String s; };`

C. `SubClass subClass = new SuperClass(String s) { };`

D. `SuperClass superClass = new SuperClass.SubClass(String s) { };`

(c) Given

```
class c1 { }
class c2 { }
interface i1 { }
interface i2 { }
class A extends c2 implements i1 { }
class B implements i1 implements i2 { }
class C implements c1 { }
class D extends c1, implements i2 { }
class E extends i1, i2 { }
class F implements i1, i2 { }
```

Which of the following is/are true? (Choose ALL that apply.)

A. Class A does not compile.

B. Class B does not compile.

C. Class C does not compile.

D. Class D does not compile.

E. Class E does not compile.

F. Class F does not compile.

G. Compilation succeeds for all of the classes.

(d) Given

```
import java.util.ArrayList;
class GenericTest1 {
  public static void main(String[] args) {
    ArrayList<int> intList = new ArrayList<>();
    intList.add(1);
    intList.add(2);
    System.out.println("The list is " + intList);
  }
}
```

Which of the following best describes the program?

A. It prints "`The list is [1, 2]`"

B. It prints "`The list is [2, 1]`"

C. It results in a compiler error.

D. It results in a runtime exception.

(e) Given

```
import java.util.ArrayList;
class GenericTest2 {
  public static void main(String[] args) {
    ArrayList<Integer> intList = new ArrayList<>();
    ArrayList<Double> doubleList = new ArrayList<>();
    System.out.println("First type: " + intList.getClass());
    System.out.println("Second type: " + doubleList.getClass());
  }
}
```

Note: `x.getClass()` returns the runtime class of object `x`.

Which of the following best describes the program?

A. It prints

```
First type: class java.util.ArrayList
Second type: class java.util.ArrayList
```

B. It prints

```
First type: class java.util.ArrayList<Integer>
Second type: class java.util.ArrayList<Double>
```

C. It results in a compiler error.

D. It results in a runtime exception.

(f) Given

```
1 class SeaCreature { }
2 class JellyFish extends SeaCreature { }
3 class Shark extends SeaCreature { }
4 public class MixAndMatch<T1 extends SeaCreature> {
5   public <T2 extends Shark> MixAndMatch<? super JellyFish> method(T1 t1, T2 t2) {
6     // Insert Code Here
7   }
8 }
```

Which, inserted independently at line 6, compile? (Choose ALL that apply.)

A. `return null;`

B. `return new MixAndMatch<JellyFish>();`

C. `return new MixAndMatch<SeaCreature>();`

D. `return new MixAndMatch<T1>();`

E. `return new MixAndMatch<T2>();`

F. `return new MixAndMatch<t1>();`

G. `return new MixAndMatch<t2>();`

(g) Which of the followings is / are true? (Choose ALL that apply.)

```
public class TestThread implements Runnable {
  public static void main(String[] args) {
    try {
      Thread t = new Thread(new TestThread());
      t.start();
      System.out.print("Haha ");
      t.join();
      System.out.print("Hehe ");
    }
    catch(InterruptedException e) {}
  }
  public void run() {
    System.out.print(":o ");
    System.out.print(":P ");
  }
}
```

A. Compilation error occurs.

B. The output could be `:o :P Haha Hehe`.

C. The output could be `Haha Hehe :o :P`.

D. The output could be `Haha :o :P Hehe`.

E. The output could be `Haha :o Hehe :P`.

F. An exception is thrown at runtime.

(h) Assume you have a class that holds two private `int` type instance variables: `a` and `b`. Which of the following code segment can prevent concurrent access problems in that class?
(Choose ALL that apply.)

A.
```
public int read() {
  return a+b;
}
public void set(int a, int b) {
  this.a = a; this.b = b;
}
```

B.
```
public synchronized int read() {
  return a+b;
}
public synchronized void set(int a, int b) {
  this.a = a; this.b = b;
}
```

C.
```
public int read(){
  synchronized(a) { return a+b; }
}
public void set(int a, int b) {
  synchronized(a) {
    this.a = a; this.b = b;
  }
}
```

D.
```
public int read() {
  synchronized(a) { return a+b; }
}
public void set(int a, int b) {
  synchronized(b) { this.a = a; this.b = b; }
}
```

E.
```
public synchronized(this) int read() {
  return a+b;
}
public synchronized(this) void set(int a, int b) {
  this.a = a; this.b = b;
}
```

F.
```
public int read() {
  synchronized(this) { return a+b; }
}
public void set(int a, int b) {
  synchronized(this) { this.a = a; this.b = b; }
}
```

(i) Which of the following is/are VALID lambda expression? (Select ALL that apply.)

A. `(int x) -> x * x`

B. `x -> x * x`

C. `-> 1`

D. `(x, int y) -> x * y`

(j) Given

```
1  class LambdaTest {
2    @FunctionalInterface
3    interface LambdaFunction {
4      int apply(int j);
5      boolean equals(Object o);
6    }
7    public static void main(String[] args) {
8      LambdaFunction lambdaFunction = i -> i * i;
9      System.out.println(lambdaFunction.apply(10));
10   }
11 }
```

Which of the following is correct?

A. This program results in a compilation error: interface cannot be defined inside class.

B. This program results in a compilation error: `@FunctionalInterface` used for `LambdaFunction` that defines two abstract methods.

C. This program results in a compilation error in code marked in line 8: syntax error.

D. This program compiles without errors, and when run, it prints 100 on screen.

Problem 2 [7 points] Interface

Given the file "`Student.java`" with incomplete definition and implementation class '`Student`', complete the program such that it produces the expected output given below when the program is compiled and executed.

```
import java.util.*;   /* File: Student.java */

public class Student implements /* TODO: Implement Comparable interface here  */ {
  private String name;
  private char gender;
  private int age;
  private double cgpa;

  public Student(String name, char gender, int age, double cgpa) {
    this.name = name;   this.gender = gender;
    this.age = age;     this.cgpa = cgpa;
  }

  public String getName() { return name; }
  public char getGender() { return gender; }
  public int getAge() { return age; }
  public double getCGPA() { return cgpa; }

  @Override
  public String toString() { return name + ", " + gender + ", " + age + ", " + cgpa; }

  public static void main(String[] args) {
    ArrayList<Student> arrayList = new ArrayList<>();
    arrayList.add(new Student("Peter", 'M', 20, 4.25));
    arrayList.add(new Student("Peter", 'M', 18, 3.2));
    arrayList.add(new Student("Peter", 'M', 18, 3.25));
    arrayList.add(new Student("Amy", 'F', 17, 2.5));

    Collections.sort(arrayList);
    for(Student s : arrayList)
      System.out.println(s);
  }

  // TODO: Override compareTo method here.
}
```

Expected output of the program:

```
Amy, F, 17, 2.5
Peter, M, 18, 3.2
Peter, M, 18, 3.25
Peter, M, 20, 4.25
```

(a) [2 points] Complete the following line such that `Student` class implements Comparable interface.

Answer:

```
public class Student implements _____ {
```

(b) [5 points] Override `compareTo` method that compares two `Student` objects, where the order is determined according to the followings:

1. First, by name
2. If name is equal, then by gender
3. If gender is equal, then by age (Assume male should be placed before female.)
4. If age is equal, then by cgpa

Requirement: Appropriate annotation should be added so that Java compiler will check whether the overriding is done properly.
Hint: `int compare(T o1, T o2)` method in numeric wrapper classes (e.g. `Character`, `Integer`, `Double`) may be useful. The method compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

Answer:

Problem 3 [11 points] Generic Class

Create a generic class, '`StatisticsCalculator`', with a type parameter T where T is a numeric object type (e.g., Integer, Double, or any class that extends java.lang.Number), with the following methods:

- `computeMean` that takes an `ArrayList` of type T and returns a double representing the mean (see formula below) of the values in the `ArrayList`.
- `computeStandardDeviation` that takes an `ArrayList` of type T and returns a double representing the standard deviation (see formula below) of the values in the `ArrayList`.

Useful formulas:

$$\text{List of data} = \{x_1, x_2, \ldots, x_N\}$$

$$\texttt{Mean} = \frac{1}{N}(x_1 + x_2 + \ldots + x_N)$$

$$\texttt{Standard Deviation} = \sqrt{\frac{1}{N-1}\left[(x_1 - \texttt{Mean})^2 + (x_2 - \texttt{Mean})^2 + \ldots + (x_N - \texttt{Mean})^2\right]}$$

Requirements:

- Your class should work with the following testing program to produce the given output.
- Also, your class should generate a **compilation error** if it is parameterized with **non-numeric elements**.

```
import java.util.*;
public class TestStatisticsCalculator {
  public static void main(String[] args) {
    ArrayList<Double> doubleArrayList = new ArrayList<>();
    Random random = new Random();
    for(int i=50; i<=100; i+=10) // Generate 6 double values
      doubleArrayList.add(new Double( (int)( (i+random.nextDouble()) * 100)/100.0 ));
    StatisticsCalculator<Double> doubleCalculator = new StatisticsCalculator<>();
    double doubleMean = doubleCalculator.computeMean(doubleArrayList);
    double doubleStd = doubleCalculator.computeStandardDeviation(doubleArrayList);
    System.out.println("Data: " + doubleArrayList);
    System.out.println("Mean: " + doubleMean + ", Standard Deviation: " + doubleStd);
    System.out.println();

    ArrayList<Integer> integerArrayList = new ArrayList<>();
    for(int i=30; i<=100; i+=12) // Generate 6 int values
      integerArrayList.add(new Integer(i));
    StatisticsCalculator<Integer> integerCalculator = new StatisticsCalculator<>();
    double intMean = integerCalculator.computeMean(integerArrayList);
    double intStd = integerCalculator.computeStandardDeviation(integerArrayList);
    System.out.println("Data: " + integerArrayList);
    System.out.println("Mean: " + intMean + ", Standard Deviation: " + intStd);
  }
}
```

Output of the program:

```
Data: [50.83, 60.66, 70.52, 80.43, 90.49, 100.8]
Mean: 75.62166666666667, Standard Deviation: 18.668910430624138

Data: [30, 42, 54, 66, 78, 90]
Mean: 60.0, Standard Deviation: 22.44994432064365
```

Hint: You may find the following methods useful:

- Math class static method `double pow(double a, double b)` returns the value of the `a` to the power of `b`.
- Math class static method `double sqrt(double a)` returns the positive square root of `a`.
- Numeric wrapper class method `double doubleValue()` returns the value of the specified number as a double, which may involve rounding.

Implement your class here.

Answer: `/* File StatisticsCalculator.java */`

```
import java.util.ArrayList;  // This is given.
```

Problem 4 [15 points] Lambda Expressions

(a) [6 points] Decide whether each of the following programs is syntactically INCORRECT - that is, it will produce compilation error(s). Circle "Yes" if it will give compilation error and "No" otherwise.

i.
```
/* Error:     Yes     /     No     */

interface I { int func(int n1, int n2); }
public class LambdaTest {
  public static void main(String[] args) {
    int i;
    I in = (n1, n2) -> { int i; return i; };
  }
}
```

ii.
```
/* Error:     Yes     /     No     */

interface I { int func(int n1, int n2); }
public class LambdaTest {
  public static void main(String[] args) {
    int i;
    I in = (n1, n2) -> { i = n1 + n2; return i; };
  }
}
```

iii.
```
/* Error:     Yes     /     No     */

interface I {
  int func(int n1, int n2);
  public String toString();
}
public class LambdaTest {
  public String toString() {
    return super.toString();
  }
  public void getResult() {
    I in = (n1, n2) -> {
      System.out.println(this.toString());
      return n1 + n2;
    };
    System.out.println(in.func(1,2));
  }
  public static void main(String[] args) {
    (new LambdaTest()).getResult();
  }
}
```

(b) [9 points] The following program contains 6 ERRORS (syntax errors, logical errors, etc.). Study the program carefully, identify all the errors by writing down the line number where an error occurs, and explain why is it an error.

```
1  import java.util.function.Function;
2
3  abstract class Food {
4    private String name;
5    private double calories;
6
7    public Food(String name, double calories) {
8      this.name = name;
9      this.calories = calories;
10   }
11
12   public void howToEat();
13
14   public String getName() { return name; }
15   public double getCalories() { return calories; }
16   @Override
17   public String toString() { return name + ": " + calories; }
18 }
19
20 @FunctionalInterface
21 interface Edible {
22   void eat(Food f);
23   void howToEat();
24 }
25
26 class Hamburger extends Food {
27   public Hamburger(String name, double calories) { }
28   @Override
29   public void howToEat() {
30     System.out.println("Hold it with both hands and take a bite.");
31   }
32 }
33
34 class Pancake extends Food {
35   public Pancake(String name, double calories) { super(name, calories); }
36   @Override
37   public void howToEat() {
38     System.out.println("Cut the pancakes with a knife and fork.");
39   }
40 }
41
42
```

```
43 class IceCream extends Food {
44   public IceCream(String name, double calories) { super(name, calories); }
45   @Override
46   public void howToEat() {
47     System.out.println("Eat an ice cream cone with a series of licks.");
48   }
49 }
50
51 public class LambdaExam {
52   static public void main(String[] args) {
53     Edible desmond = f -> { System.out.println(f); };
54     Edible bingyen = (Food f) ->
55       System.out.println(f);
56       System.out.println("I can finish it in 2 seconds! :)");
57     ;
58
59     Function<Food,Food> exchange = foodprovide -> {
60       new IceCream("Vanilla Ice Cream", 125.4);
61     };
62
63     Food f1 = new Hamburger("Cheeseburger with Bacon and Condiments", 608);
64     desmond.eat(f1);
65     f1.howToEat();
66     System.out.println();
67
68     Food f2 = new Pancake("Blueberrry Pancakes", 169);
69     f2.howToEat();
70     bingyen.eat(exchange.do(f2));
71   }
72 }
```

Assume the following is the expected output of the program:

```
Cheeseburger with Bacon and Condiments: 608.0
Hold it with both hands and take a bite.

Cut the pancakes with a knife and fork.
Vanilla Ice Cream: 125.4
I can finish it in 2 seconds! :)
```

Error#	Line#	Explanation
1		
2		
3		
4		
5		
6		

Problem 5 [20 points] Multi-Threading

(a) [9 points] Given the program "`Matrix.java`" with incomplete implementation of class '`Matrix`'. Complete the program so that it performs square matrix multiplication using multi-threading.

The following shows a sample output of the program.

```
Enter the size of n: 5

A =
0     7     -7    3     6
-6    -6    -2    4     0
2     5     -2    -2    2
3     -3    4     0     2
-5    -5    -5    7     4

B =
0     6     -8    8     9
-7    4     -3    4     0
-5    2     -3    6     -7
-2    -1    -2    -1    1
-4    6     -6    -1    7

C =
-44   47    -42   -23   94
44    -68   64    -88   -36
-29   42    -33   24    44
-7    26    -39   34    13
30    -43   32    -101  25
```

Formula for computing the matrix products of two square matrices A and B.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$C = AB = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Requirement: You are **NOT** allowed to add additional class for this question.

```java
/* File: Matrix.java */

import java.util.Scanner;
import java.util.Random;

/* A class representing an n x n matrix */
public class Matrix {
  private int n;                // Size of matrix
  private int[][] matrixA;      // Reference variables of 2D int array
  private int[][] matrixB
  private int[][] matrixC;
  private Thread[][] thread;    // Reference variable of 2D Thread array

  // Constructor
  public Matrix(int n) {
    this.n = n;                 // Assign n to instance variable n
    matrixA = new int[n][n];    // Allocate an n x n int array
    matrixB = new int[n][n];    // Allocate an n x n int array
    matrixC = new int[n][n];    // Allocate an n x n int array
    thread = new Thread[n][n];  // Allocate an n x n Thread array

    Random r = new Random();    // Instantiate object of Random type
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < n; j++) {
        matrixA[i][j] = r.nextInt() % 10; // Randomly generate a value between -9 to 9
        matrixB[i][j] = r.nextInt() % 10; // Randomly generate a value between -9 to 9
      }
    }
  }

  // Method for printing matrix values
  public void printMatrix(int[][] matrix) {
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < n; j++)
        System.out.print("\t" + matrix[i][j]);
      System.out.println();
    }
  }

  // Method for printing matrix multiplication result
  public void printMatrixMultiplicationResult() {
    System.out.println("\nA =");
    printMatrix(matrixA);
    System.out.println("\nB =");
    printMatrix(matrixB);
    System.out.println("\nC =");
    printMatrix(matrixC);
  }
```

```
  // Compute the matrix products of two square matrices AB
  // (i.e. matrixA x matrixB) using multi-threading
  public void doMatrixMultiplication() {
    /* ADD YOUR CODE HERE. */









  }

  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter the size of n: ");
    int n = sc.nextInt();
    Matrix m = new Matrix(n);
    m.doMatrixMultiplication();
    m.printMatrixMultiplicationResult();
  }
}
```

(b) [11 points] A timer is a facility in Java used to execute some predefined task periodically. A timer typically waits until a certain interval has elapsed and then executes the predefined task.

Suppose you are asked to design a timer that allows Java programmers to execute a task in every n milliseconds. To tackle this problem, you need to do the followings:

- Define an interface named "`Task`", which contains one abstract method called `execute()`. This abstract method accepts nothing and returns nothing.
- Create a class named "`MyTimer`" that implements `Runnable` interface, and has the following instance variables and methods.
  - An instance variable `task` of type "`Task`", which represents the task to be executed periodically.
  - An instance variable `milliSec` of type long, which represents the amount of time to wait for the next execution of task.
  - An instance variable `thread` of type Thread, which will reference to a thread object executing what is defined in the run method.
  - An instance variable `isPaused` of type boolean (with initial value `false`) describing whether the thread is paused or not.
  - An instance variable `running` of type boolean describing whether the thread is running.
  - Implement the constructor "`MyTimer(Task task, long milliSec)`", such that it
    (1) initializes instance variables, `task` and `milliSec`, according to the method parameters,
    (2) creates a thread executing what is defined in the run method.
  - Implement `void start()` method by making the thread to begin execution and set instance variable, `running` to true.
  - Implement `void stop()` method by setting `running` to false.
  - Implement `void pause()` method by setting `isPaused` to true.
  - Implement `void resume()` method by setting `isPaused` to false.
  - Implement run method in Runnable interface, such that it executes the task in every "`milliSec`" as specified by the instance variables, **if `running` is true and `isPaused` is false**.

Requirement: You need to make sure that **only one thread** can access `start()`, `stop()`, `pause()`, and `resume()` method at a time.

Your task is to complete the interface and the class such that the given testing program "`TestMyTimer.java`" will compile, run, and output "`I am working on COMP3021 exam questions ...`" every 1000 milliseconds, until 10000 milliseconds has elapsed.

```
public class TestMyTimer     /* File "TestMyTimer.java" */
{
  public static void main(String[] args)
  {
    Task task = ()
      -> System.out.println("I am working on COMP3021 exam questions ...");
    MyTimer timer = new MyTimer(task, 1000);
    timer.start();
    try {
      Thread.sleep(10000);
    }
    catch(InterruptedException e) {}
    timer.stop();
  }
}
```

(a) [2 points] Define the interface `Task` which will be saved in "`Task.java`" file.

Answer: `/* File "Task.java" */`

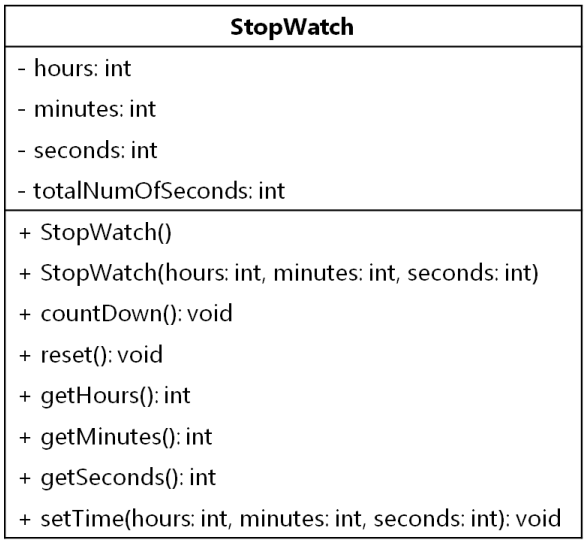
(b) [9 points] Implement the class `MyTimer` which will be saved in "`MyTimer.java`" file.

Answer: `/* File "MyTimer.java" */`

Problem 6 [27 points] GUI and Event-Driven Programming

You are asked to design a GUI-based countdown stopwatch for COMP3021 examination. To tackle this program, you need to implement two classes, namely '`StopWatch`' and '`CountDownStopWatchGUI`' (extends javafx.application.Application class), according to the details given below so that they will work as specified in the given output.

(a) [9 points] Implement the class '`StopWatch`' in the file "`StopWatch.java`" according to the class diagram given below.

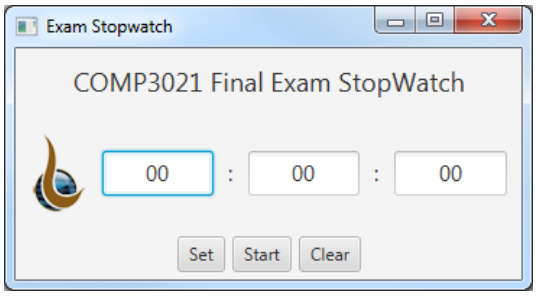


where

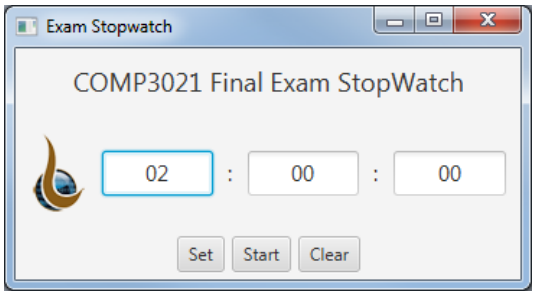
- The instance variables, `hours`, `minutes`, `seconds`, represent the hour, minute and second of the stopwatch.
- The instance variable, `totalNumOfSeconds`, represents the total number of seconds as specified by hours, minutes, and seconds.
- The default / no-arg constructor, `StopWatch()`, constructs an object that allocates storage space for instance variables.
- The constructor, `StopWatch(hours:int, minutes:int, seconds:int)` constructs an object that allocates storage space for instance variables and initializes them with the specified parameters.
- The method, `countDown()`, checks if `totalNumOfSeconds` is greater than 0. If so, decrease it by 1 and re-compute the `hours`, `minutes`, and `seconds`, accordingly.
  Note: An hour has 3600 seconds, and a minute has 60 seconds.
- The method, `reset()`, set all the instance variables to 0.
- The three accessor methods, each returns the corresponding instance variable value.
- The mutator method, `setTime(hours:int, minutes:int, seconds:int)`, assigns all the instance variables with the specified parameters, and compute `totalNumOfSeconds` according to the parameters.

Answer: `/* File "StopWatch.java" */`

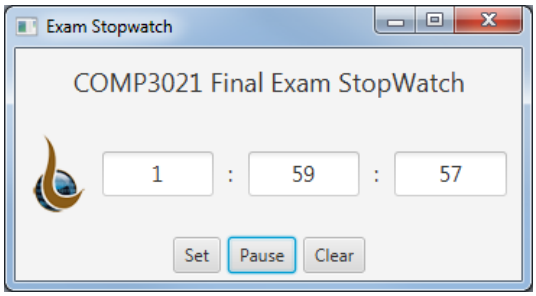
(b) [18 points] In this part, you are required to use the class `StopWatch` implemented in part (a) to implement a GUI stopwatch using JavaFX as shown in Figure (i).



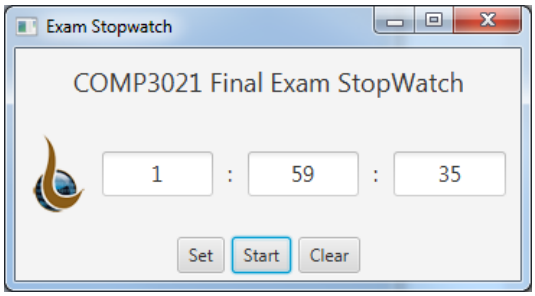
(i)



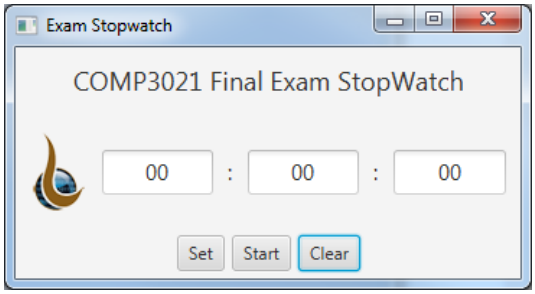
(ii)



(iii)



(iv)

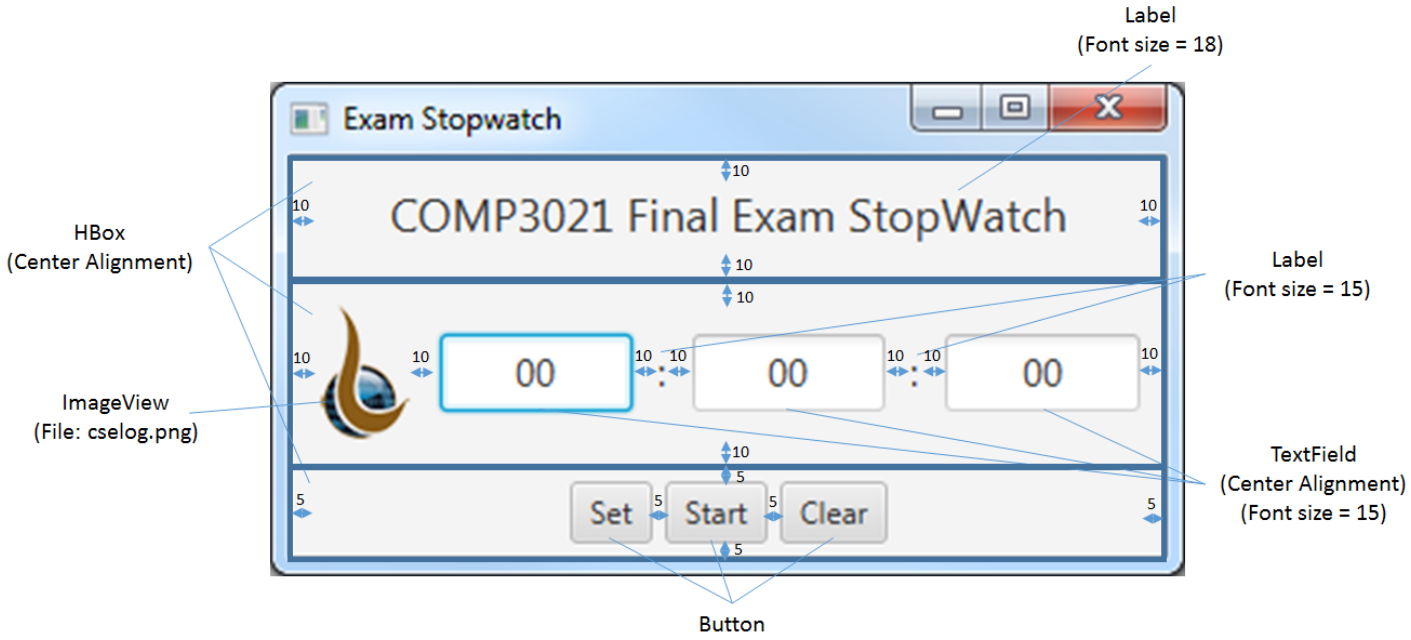


(v)

- When the user enters the time to the textfields indicating the hour, minute, and second, and clicks the `Set` button, as shown in Figure (ii), the program invokes the `setTime` method of StopWatch with the textfield data.
- When the user clicks the `Start` button, the program starts the timer so that it invokes the `countDown` method of StopWatch every second and updates the textfields according to the hour, minute, and second values of StopWatch. In addition, the program changes the button's text to `Pause`, as shown in Figure (iii).
- When the user clicks the `Pause` button, the program pauses the timer and changes the button's text to `Start`, as shown in Figure (iv).
- The `Clear` button resets the stopwatch, stop the timer, and changes the "`Start/Pause`" button's text to `Start`, and updates all the textfields to "00", as shown in Figure (v).

Your task is to complete the missing parts of "`CountDownStopWatchGUI.java`" in the space provided under Part(b)(i) - (b)(ii) "`ADD YOUR CODE HERE`".

```
import javafx.application.*;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.*;
import javafx.geometry.*;
import javafx.scene.text.Font;
import javafx.scene.control.*;
import javafx.scene.image.*;
import javafx.animation.*;
import javafx.util.Duration;

public class CountDownStopWatchGUI extends Application {
  // Textfields for hour, minute, and second of the stopwatch.
  TextField hourTextField, minuteTextField, secondTextField;

  // Reference variable that will reference to a StopWatch object.
  StopWatch stopWatch;

  // Timeline variable that will reference to a Timeline object
  // used to invoke countDown method of StopWatch in every second.
  Timeline timeline;

  // Default / no-arg constructor of CountDownStopWatchGUI
  // - Instantiates StopWatch object and makes it referenced by stopWatch
  // - Instantiates Timeline object that tiggers countDown method of StopWatch
  //   in every second. Also updates the textfields according to the
  //   hour, minute, and second values of StopWatch.
  //   (The object should be referenced by timeline.)
  // - Makes the number of timeline cycles to indefinite
  // ***** Note: Lambda expression should be used whenever is possible *****

  // Part (b)(i) - ADD YOUR CODE HERE
```




```
// Overrides the start method of Application by doing the followings:
// 1. Code a scene graph according to the layout shown above.
// 2. Make sure that when the set button is clicked, invokes setTime method of
//    StopWatch with the textfield data.
// 3. Make sure that when the start/pause button is clicked,
//    - Changes the button text to "Start" AND pauses the timeline
//      if the current text is "Pause"
//    - Changes the button text to "Pause" AND plays the timeline
//      if the current text is "Start".
// 4. Make sure that when the clear button is clicked, resets the StopWatch,
//    stops the timeline, and changes the "Start/Pause" button's text to "Start".
//    Also, updates all the textfields to "00".
// 5. Creates a scene of size (width = 350, height = 160) and binds it with the
//    root of the scene graph.
// 6. Sets the scene to the stage and sets the title of the stage to
//    "Exam Stopwatch".
// 7. Shows the stage.
//
// Note:
// ***** Appropriate annotation should be added so that Java compiler will check
//       whether your method overriding is done properly. *****
// ***** Lambda expression should be used whenever is possible. *****
// ***** Update of JavaFX UI elements should be done using JavaFX thread. *****
```

```
// Part (b)(ii) - ADD YOUR CODE HERE
```

/*** Continue Your Answer For Problem 6(b)(ii) On This Page ***/

---------- END OF PAPER ----------

Appendix I

(1) **java.util Class: ArrayList<E>**

Methods	**Description**
ArrayList( )	Constructs an empty list with an initial capacity of ten.
void add(E e)	Appends the specified element to the end of this list.
E get(int i)	Returns the element at index i.
int size()	Returns the number of elements in this list.
String toString()	Returns a string representation of the Arraylist. The string representation consists of a list of elements in the order they are in the container, enclosed in square brackets ("[ ]"). Adjacent elements are separated by the character ", " (comma and space).

(2) **java.lang Class: Collections**

Methods	**Description**
static <T> void sort(List<T> list)	Sorts the specified list into ascending order, according to the natural ordering of its elements. All elements in the list must implement the Comparable interface.

(3) **java.lang Interface: Comparable<T>**

Methods	**Description**
int compareTo(T o)	Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

(4) **java.util.function Interface: Function<T,R>**

Methods	**Description**
R apply(T t)	Applies this function to the given argument.

(5) **java.lang Class: Integer**

Methods	**Description**
static int parseInt(String s)	Parses the string argument as a signed decimal integer.

(6) **java.lang Interface: Runnable**

Methods	**Description**
void run()	When an object implementing interface Runnable is used to create a thread, starting the thread causes the object's run method to be called in that separately executing thread.

(7) **java.lang Class: String**

Methods	**Description**
static String valueOf(int i)	Returns the string representation of the int argument.

(8) **java.lang Class: Thread**

Methods	**Description**
Thread(Runnable target)	Allocates a new Thread object.
void start()	Causes this thread to begin execution; the JVM calls the run method of this thread.
void join()	Waits for this thread to die.
static void sleep(long millis)	Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.

Appendix II

(1) **javafx.scene.layout Class: BorderPane**

Method	**Description**
BorderPane()	Creates an BorderPane layout.
void setTop(Node value)	Sets the value of the property top.
void setCenter(Node value)	Sets the value of the property center.
void setBottom(Node value)	Sets the value of the property bottom.

(2) **javafx.scene.control Class: Button**

Method	**Description**
Button(String text)	Creates a button with the specified text as its label.
void setText(Text value)	Sets the value of the property text.
void setOnAction(EventHandler<ActionEvent> value)	Sets the value of the property onAction.

(3) **javafx.util Class: Duration**

Method	**Description**
static Duration millis(double ms)	Factory method that returns a Duration instance for a specified number of milliseconds.

(4) **javafx.scene.text Class: Font**

Method	**Description**
static Font font(double size)	Search for an appropriate font based on the default font family name and given font size.

(5) **javafx.scene.layout Class: HBox**

Method	**Description**
HBox(Node... children)	Creates an HBox layout with spacing = 0.
void setSpacing(double value)	Sets the value of the property spacing.
void setPadding(Insets value)	Sets the value of the property padding.
void setAlignment(Pos value)	Sets the value of the property alignment.

(6) **javafx.scene.image Class: ImageView**

Method	**Description**
ImageView(String url)	Allocates a new ImageView object with image loaded from the specified URL.

(7) **javafx.geometry Class: Insets**

Method	**Description**
Insets(double topRightBottomLeft)	Constructs a new Insets instance with same value for all four offsets.

(8) **javafx.animation Class: KeyFrame**

Method	**Description**
KeyFrame(Duration time, EventHandler<ActionEvent> onFinished, KeyValue... values)	Constructor of KeyFrame.

(9) **javafx.scene.control Class: Label**

Method	**Description**
Label(String text)	Creates Label with supplied text.
void setFont(Font value)	Sets the value of the property font.

(10) **javafx.application Class: Platform**

Method	Description
static void runLater(Runnable runnable)	Run the specified Runnable on the JavaFX Application Thread at some unspecified time in the future.

(11) **javafx.geometry Enum: Pos**

Enum constant	Description
CENTER	Represents positing on the center both vertically and horizontally.

(12) **javafx.scene Class: Scene**

Method	Description
Scene(Parent root, double width, double height)	Creates a Scene for a specific root Node with a specific size.

(13) **javafx.stage Class: Stage**

Method	Description
void setScene(Scene value)	Specify the scene to be used on this stage.
void setTitle(String value)	Sets the value of the property title.
void show()	Attempts to show this Window by setting visibility to true.

(14) **javafx.scene.control Class: TextField**

Method	**Description**
TextField(String text)	Creates a TextField with initial text content.
void setText(String value)	Sets the value of the property text.
void setFont(Font value)	Sets the value of the property font.
void setAlignment(Pos value)	Sets the value of the property alignment.
String getText()	Gets the value of the property text.

(15) **javafx.animation Class: Timeline**

Method	**Description**
Timeline(KeyFrame... keyFrames)	The constructor of Timeline.
void setCycleCount(int value)	Sets the value of the property cycleCount (Possible value: INDEFINITE).
void play()	Play animation from current position.
void pause()	Pauses the animation.

/* Rough work — You may detach this page */

/* Rough work — You may detach this page */