First Name (Print): _____ **Last Name (Print):** _____

Student Number: _____

The Irving K. Barber School of Arts and Sciences
COSC 111 Final Exam Winter Term II 2016

Instructor: Dr. Bowen Hui

Tuesday, April 19, 2016

Time: 6:00pm - 8:00pm

Instructions:

This is a closed book exam. No calculators are allowed. Turn off your cell phone. Have your student ID card out on your desk.

This exam has 16 pages (including this cover page).

Question	Score	Possible Marks
Part 1		6
Part 2 Question 1		6
Part 2 Question 2		11
Part 2 Question 3		8
Part 2 Question 4		8
Part 3		16
Total		52 (+ 3 bonus marks)

Part 1: Multiple Choice (6 points)

Circle one answer for each question below.

1. The ASCII value for character A is 65. What does the expression `System.out.println( "A" + 1 )` display?

   (a) A1
   (b) 66
   (c) B
   (d) Syntax error (illegal expression)

2. Suppose `int i = 5`, which of the following can NOT be used as an index for array `double[] t = new double[100]`?

   (a) `i`
   (b) `( int )( Math.random() * 10 )`
   (c) `( int )( Math.random() * -1 )`
   (d) `90 + i`

3. Assuming the loop body does not modify `hc` below, how many iterations will the following loop execute?

```
for (int hc = 0; hc <= n; hc++)
{
  // iteration
}
```

   (a) 2*n
   (b) n
   (c) n - 1
   (d) n + 1

4. A method that does not return any output should hae which return type in the method header?

   (a) `int`
   (b) `double`
   (c) `boolean`
   (d) `void`

5. To declare a constant `MAX_LENGTH` inside a class, you write:

   (a) `final static MAX_LENGTH = 99.98;`
   (b) `static double MAX_LENGTH = 99.98;`
   (c) `final double MAX_LENGTH = 99.98;`
   (d) `final int double MAX_LENGTH = 99.98;`

6. Which method header should be used if you want to provide an accessor method for a boolean attribute `finished`?

   (a) `public void getFinished()`
   (b) `public boolean getFinished()`
   (c) `public boolean isFinished()`
   (d) `public void isFinished()`

Part 2: Short Answers

Question 1. (6 points)

Write the Java code inside a `main` method that simulates a simple slot machine where three numbers between 0 and 9 (inclusive) are randomly selected and printed side by side. Based on these numbers, your program will display the output as follows:

- Print "Winner!" if all three numbers are the same
- Print "Almost there!" if any two numbers are the same
- Print "Better luck next time!" otherwise

The game will repeat unless the user enters "no". Sample output:

```
8, 6, 2
Better luck next time!
Play again?
yes
7, 1, 7
Almost there!
Play again?
no
```

Grading scheme:

- **[1 pt]** Generating random numbers
- **[1 pt]** All the print statements and proper output
- **[1 pt]** Comparison when all 3 numbers are the same
- **[1 pt]** Comparison when any 2 numbers are the same
- **[1 pt]** Repeating the game
- **[1 pt]** Checking user input for ending the game

This page is intentionally left blank.

Question 2. [**11 points**]

You are given the start of a Java program that helps a hospital analyze the flow of patients through the emergency room below:

```
import java.util.Scanner;
public class HospitalFlow
{
  public static void main( String[] args )
  {
    // constants
    final int NUMDAYS  = 20;    // data over 20 days
    final int NUMHOURS = 24;    // 24 hours in a day

    int[][] data = new int[ NUMHOURS ][ NUMDAYS ];

    Scanner input = new Scanner( System.in );
    int hour;
    int nump;

    // read input from user
    for( int hr=0; hr<NUMHOURS; hr++ )
    {
      System.out.println( "For hour " + hr + ": " );
      for( int dd=0; dd<NUMDAYS; dd++ )
      {
        System.out.print( "Enter number of patient on day " + (dd+1) + ": " );
        nump = input.nextInt();
        data[ hr ][ dd ] = nump;
      }
    }

    // analyze total number of patients per hour
    // ...

    // report busiest hour of day
    // ...
  }
}
```

This program asks the user to input the number of patients reporting in the emergency room over 20 days for each hour of the day, from midnight (hour 0), to 11pm (hour 23). This information is then stored as a 2D array called `data`. There are two steps missing from this program which you will have to complete in this question.

Continue on the next page...

Part a. Continuing with the above program, write the Java code that tallies up the total number of patients per hour. In particular:

- Create a new 1D array called `dataOverDays` that will store the total number of patients for each hour of the day
- Go through the `data` array, for each hour of the day, sum up the number of patients for that hour
- Record this sum into the appropriate hour slot of the `dataOverDays`

Grading scheme:

- **[1 pt]** Declare and create `dataOverDays`
- **[1 pt]** Loop through each hour of the array
- **[2 pts]** Compute the sum for that hour
- **[1 pt]** Store the sum into `dataOverDays`

Continue on the next page...

Part b. Continuing on from part (a), write the Java code analyzes the array `dataOverDays` and report the busiest hour in the day and how many patients there were in total. Sample output (some output omitted due to limited space):

```
For hour 0:
Enter number of patient on day 1: 3
Enter number of patient on day 2: 5
...
Enter number of patient on day 20: 3
For hour 1:
Enter number of patient on day 1: 3
Enter number of patient on day 2: 2
...
Enter number of patient on day 20: 1
...
For hour 23:
Enter number of patient on day 1: 0
Enter number of patient on day 2: 2
...
Enter number of patient on day 20: 1
Over the past 20 days, the busiest hour in the day is hour 14 which had 189 patients in total.
```

Grading scheme:

- **[1 pt]** Keep track of the busiest hour
- **[1 pt]** Keep track of the total number of patients at busiest hour
- **[2 pts]** Find the busiest hour in `dataOverDays`
- **[1 pt]** Update the proper variables
- **[1 pt]** Display analysis results as shown in the sample output above

Question 3. (8 points)

Given the following `Birthday` class, you will see that it has the day, month, and year of birth as attributes. This class also has various methods, such as a constructor that initializes these attributes, and an accessor method for each attribute.

```
public class Birthday
{
  // attributes
  private int day;
  private int month;
  private int year;

  // constructor
  public Birthday( int dd, int mm, int yy )
  {
    day   = dd;
    month = mm;
    year  = yy;
  }

  // accessors
  public int getDay()   { return day; }
  public int getMonth() { return month; }
  public int getYear()  { return year; }
}
```

For this question, you are asked to add a method called `compareToAnother` to this class that takes a `Birthday object` as input and returns an integer number as output. The purpose of this method is to compare itself to the input birthday object and see which birthday occurs earlier. In particular, the method will return an integer output as follows:

- Return 0 if this birthday and the input object occur on the same date
- Return 1 if this birthdate occurs earlier than the input object birthdate
- Return -1 if this birthdate occurs later than the input object birthdate

Grading scheme:

- [**2 pts**] Method header
- [**1 pt**] Logic works when both birthdays are the same
- [**1 pt**] Logic works when input birthday is earlier
- [**1 pt**] Logic works when input birthday is later
- [**1 pt**] Correct return value
- [**1 pt**] Use of accessor methods as needed
- [**1 pt**] Overall syntax

This page is intentionally left blank.

Question 4. (8 points)

You are given the following `Player` class:

```
public class Player
{
  private String name;
  private int    highScore;

  public Player( String alias, int score )
  {
    name      = alias;
    highScore = score;
  }

  public String getName()      { return name; }
  public int    getHighScore() { return highScore; }
}
```

For this question, define a `TestPlayer` class that has a `main` method. In the `main` method, create an array of `Player` objects called `scoreboard` and determine which `Player` objects have above average highscores as follows:

- In your array, create 4 `Player objects` such that:
  - The 1st player is called "tommy" and has a highscore of 342
  - The 2nd player is called "jonny" and has a highscore of 149
  - The 3rd player is called "cindy" and has a highscore of 241
  - The 4th player is called "sarah" and has a highscore of 329
- Compute the average highscore of the `Player` objects in your array and display it
- Go through each `Player` objects in your array and display the names of the ones whose highscore are above the computed average

Sample output:

```
The average high score is 265.25
Players with above average high score are:
tommy
sarah
```

Grading scheme:

- **[1 pt]** Class definition and structure
- **[1 pt]** Declare and create the `scoreboard` array
- **[2 pts]** Creating 4 `Player` objects into the array
- **[1 pt]** Computing the average highscore
- **[1 pt]** Finding the players with above average highscore
- **[1 pt]** Use of accessor methods
- **[1 pt]** Displaying output as required

This page is intentionally left blank.

Part 3 Long Answer [16 points]

In this question, write a Java program that records app reviews. To simplify the problem, the `Review` class has been provided to you as follows:

```
public class Review
{
  private String comment;
  private int    rating;

  public Review( String c, int r )
  {
    comment = c;
    rating  = r;
  }

  public String toString()
  {
    String str = "";
    str += "(" + rating  + "/5): ";
    str += comment + "\n";
    return str;
  }
}
```

You will need to use the above `Review` class in completing this question.
Create an `App` class that has a name, a maximum of 100 reviews, and an array of reviews. You may have additional attributes as needed to make your program work. Ensure your `App` class has the following methods:

- A constructor method that initializes the name of the app.
- A method that enables the submission of a review to be added to the app. This method should be called `submitReview`, and must take a `Review` object as input.
- The `toString` method that concatenates the name of the app, along with the reviews available for the app.

Lastly, create a `TestApp` class with a `main` method. In the `main` method, include the following:

- Create 3 reviews based on the class definition above
- Create an app (give it a meaningful name)
- Submit the reviews you created to this app
- Display the app info

Sample output:

```
App: The New Solitaire
    (5/5): awesome
    (4/5): pretty good
    (1/5): awesome: sucks
```

Grading scheme:

- **[1 pt]** Overall class definition and structure
- **[2 pts]** `App` class: attribute definitions
- **[2 pts]** `App` class: constructor method
- **[2 pts]** `App` class: `submitReview` method
- **[2 pts]** `App` class: `toString` method
- **[2 pts]** `App` class: functional and works
- **[1 pt]** `TestApp` class: define `Review` objects
- **[2 pts]** `TestApp` class: create app and call its `submitReview`
- **[2 pts]** `TestApp` class: call app's `toString` and display output

This page is intentionally left blank.

Common Methods and Definitions

- From the `Math` class:
  - `double random()`
    Returns a `double` value greater than or equal to 0.0 and less than 1.0
  - `int round( float a )`
    Returns the closest `int` value to `a`, with ties rounding up
  - `double pow( double a, double b )`
    Returns the value of `a` raised to the power of `b`
  - `double exp( double a )`
    Returns Euler's number e raised to the power of `a`
  - `double sqrt( double a )`
    Returns the correctly rounded positive square root of `a`
- From the `Character` class:
  - `boolean isDigit( char ch )`
    Returns true if `ch` is a digit, and false otherwise
  - `boolean isLetter( char ch )`
    Returns true if `ch` is a letter, and false otherwise
  - `boolean isLetterOrDigit( char ch )`
    Returns true if `ch` is a letter or a digit, and false otherwise
  - `boolean isLowerCase( char ch )`
    Returns true if `ch` is a lowercase letter, and false otherwise
  - `boolean isUpperCase( char ch )`
    Returns true if `ch` is an uppercase letter, and false otherwise
- From the `String` class:
  - `char charAt( int index )`
    Returns the character at position `index` of the string
  - `int indexOf( int ch )`
    Returns the index position within the string of the first occurrence of `ch`
  - `int lastIndexOf( int ch )`
    Returns the index position within the string of the last occurrence of `ch`
  - `int length()`
    Returns the number of characters in the string
  - `boolean startsWith( String prefix )`
    Returns true if the string starts with `prefix`, and false otherwise
  - `boolean endsWith( String suffix )`
    Returns true if the string ends with `suffix`, and false otherwise
  - `boolean contains( String str )`
    Returns true if the string contains `str`, and false otherwise
  - `int compareTo( String str )`
    Returns an integer result for comparing two strings lexicographically
  - `int compareToIgnoreCase( String str )`
    Returns an integer result for comparing two strings lexicographically while ignoring case differences
  - `boolean equals( Object stringObject )`
    Returns true if the string is the same as `stringObject`, and false otherwise
  - `boolean equalsIgnoreCase( String str )`
    Returns true if the string is the same as `str`, and false otherwise
  - `String substring( int beginIndex )`
    Returns a new string that is a substring starting at position `beginIndex` of this string
  - `String substring( int beginIndex, endIndex )`
    Returns a new string that is a substring starting at position `beginIndex` and up to `endIndex` of this string

  - `String toLowerCase()`
    Converts all the characters in this string to lower case
  - `String toUpperCase()`
    Converts all the characters in this string to upper case

- From the `Scanner` class:
  - `Scanner Scanner( InputStream source )`
    Creates a new `Scanner` object from the specified `source`
  - `int nextInt()`
    Scans the next token of the input as an `int`
  - `double nextDouble()`
    Scans the next token of the input as a `double`
  - `String next()`
    Returns the next complete token as a `String`
  - `String nextLine()`
    Returns everything in the current line as a `String`

- From the `Random` class:
  - `Random Random()`
    Creates a new `Random` object
  - `int nextInt()`
    Returns a randomly generated integer
  - `int nextInt( int n )`
    Returns a randomly generated integer between 0 and n-1 inclusive
  - `double nextDouble()`
    Returns a randomly generated decimal number between 0 and 1 inclusive