Duration: **50 minutes**
Aids Allowed: **The Java API: An Introduction for Students**

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

Last Name: ____________________________

First Name: ____________________________

Lecture Section: __________ **Instructor: Danny Heap**

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

MARKING GUIDE

# 1:	______/ 6
# 2:	______/ 3
# 3:	______/ 4
# 4:	______/ 6
# 5:	______/12
TOTAL:	______/31

This midterm test consists of 5 questions on 5 pages (including this one), printed on one side of the paper. *When you receive the signal to start, please make sure that your copy of the test is complete.* If you need more space, use the reverse side of the page and *indicate clearly the part of your work that should be marked.*

Good Luck!

Question 1. [6 MARKS]

The classes `Gaffe` and `BunchOfGaffes` both compile and `BunchOfGaffes` runs without error.

```
/** a class to illustrate some mistakes
 */
public class Gaffes {
  private int num;

  // * Insertion point A *

  public int incrNum(int increment) {

    // * Insertion point B *

    return num;
  }
}
```

```
/** a companion to Gaffes class
 */
public class BunchOfGaffes {
  private Gaffes m1;

  // * Insertion point C *

  public static void main(String[] args) {

    // * Insertion point D *

    Gaffes[] ms= new Gaffes[2];

    // * Insertion point E *

  }
}
```

Evaluate the insertions suggested below in numerical order. If you believe an insertion will cause a compilation error, write `compile-time error` underneath it. If you believe an insertion will compile properly, but will cause an error at run time, write `run-time error` underneath it. If you believe an insertion will cause no problem, write `okay` underneath it, and make the insertion. Notice that since you are considering the insertions in numerical order, an earlier insertion may affect your answer on a later insertion.

1. Insert above insertion point B: `int i;`

   `okay`

2. Insert below insertion point B: `num= increment + i;`

   `compile-time error`

3. Insert above insertion point C: `private static Gaffes m2= new Gaffes();`

   `okay`

4. Insert above insertion point D: `int j= m1.incrNum(2);`

   `compile-time error`

5. Insert below insertion point D. `int h= m2.num;`

   `compile-time error`

6. Insert above insertion point E: `int k= ms[0].incrNum(0);`

   `run-time error`

Question 2. [3 MARKS]

Part (a) [3 MARKS]

Read the implementation of the method below. Which values of `num` will cause `isYecleft(num)` to return `true`?

```
even numbers from 5 to 3586, inclusive
```

```
public boolean isYecleft(int num){
    if (num < 5) {
        return false;
    }
    if (num > 3586) {
        return false;
    }
    if ((num % 2) == 1) {
        return false;
    }
    return true;
}
```

Question 3. [4 MARKS]

Assume that `words` is an array of `String`, and that each element of `words` references a `String` of positive length. Write a code fragment (not a method) that uses a loop to store the first character of `words[0]` in `String` variable `initials`, then adds the first character of `words[1]` to the end of `initials`, and so on until it finally adds the first character of `words[words.length-1]` to the end of `initials`.

```
String initials= "";
for (int i=0; i != words.length; i++) {
    initials= initials + words[i].substring(0,1);
}
```

Question 4. [6 MARKS]

Read the implementation of the method below. What `String` is returned by `smush("abcdef")`?

```
fdbace is returned.
```

```
public static String smush(String orig) {
    String tmpString= "";
    for (int i= 0; i != orig.length(); i++) {
        if ((i % 2) == 0) {
            tmpString= tmpString + orig.charAt(i);
        }
        else {
            tmpString= orig.charAt(i) + tmpString;
        }
    }
    return tmpString;
}
```

Question 5. [12 MARKS]

Part (a) [7 MARKS]

Write the `constructor` and `equals` method for the class `Individual` below.

```
/** Individual models a person with a name and height
*/
public class Individual {
    // height, in centimeters
    private int height;
    // full name, as a single String
    private String name;

    /** [3 marks]
     *  constructor: create a new Individual with height h and
     *  name n.
     */
    public Individual (int h, String n) {
        height= h;
        name= n;
    }


    /**  [4 marks]
     *   equals: return true if this Individual's height differs from
     *   other's height by less than 2 centimeters, AND if this
     *   Individual's name has the same length as other's name AND
     *   they match, character-by-character, except in (at most) 2
     *   positions.  For example "Danny" and "Danyn" have the
     *   same length, and they differ in only 2 positions (the last 2).
     */
    public boolean equals(Individual other) {
        if (other == null || (other.name == null && name != null) ||
           (other.name != null && name == null) ||
           Math.abs(height - other.height) > 2) {
               return false;
        }
        if (other.name == null && name == null) { return true; }
        if (name.length() != other.name.length() ) { return false; }
        int diff=0;
        for (int i= 0; i != name.length() ; i++) {
            if (name.charAt(i) != other.name.charAt(i)) { diff++; }
        }
        return diff < 3;
    }
}
```

Part (b) [5 MARKS]

List five test cases for the `equals` method of class `Individual`. Beside each test case you suggest, explain how each test case covers an entire category of possible cases.

1. names and heights identical. This covers all cases where the values are identical.
2. names identical, heights differ by more than 2. This covers all cases where Individuals are different due to height.
3. names identical, heights differ by 2 or less. This covers all cases where heights are within the tolerance.
4. names differ in length, heights identical. This covers all cases where Individuals differ by the length of their names.
5. names differ in length, heights differ by more than 2. This covers all cases where Individuals differ in all respects.