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 A: `num= 0;`

   `compile-time error`

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

   `okay`

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

   `compile-time error`

4. Insert below insertion point C: `System.out.println("m1 declared.");`

   `compile-time error`

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

   `compile-time error`

6. Insert below insertion point E: `ms[2]= new Gaffes();`

   `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`?

```
all odd numbers from 75 to 3585, inclusive
```

```
public boolean isYecleft(int num){
    if (num < 75) {
        return false;
    }
    if (num > 3586) {
        return false;
    }
    if ((num % 2) == 0) {
        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 last character of `words[0]` in `String` variable `initials`, then adds the last character of `words[1]` to the end of `initials`, and so on until it finally adds the last 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(words[i].length() - 1);
}
```

## Question 4. [6 MARKS]

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

```
This returns: "bdfeca"
```

```
public static String smush(String orig) {
    String tmpString= "";
    for (int i= 0; i != orig.length(); i++) {
        if ((i % 2) == 1) {
            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 Person below.

```
/** models a person with a name and weight
*/
public class Person {
    // weight, in kilograms
    private double weight;
    // full name, as a single String
    private String name;

    /** [3 marks]
     * constructor: create a new Person with weight w and
     * name n.
     */
    public Person (double w, String n) {
        weight= w;
        name= n;
    }


    /**  [4 marks]
     *   equals: return true if this Person's weight differs from
     *   other's height by less than 0.5 kilograms, AND if this
     *   Person'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(Person other) {
        if (other == null ||
            Math.abs(weight - other.weight) >= 0.5 ||
            other.name == null && name != null ||
            other.name != null && name == null) {
                return false;
        }
        if (other.name == null && name == null) { return true; }
        if (other.name.length() != 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 Person. Beside each test case you suggest, explain how each test case covers an entire category of possible cases.

1. Two Persons with identical names and weights. Covers all cases with identical values.
2. Two Persons with identical names and weights that differ by at least 0.5 kilos. Covers all cases where Persons are distinguished by their weight.
3. Two Persons with identical weights and names with different lengths. Covers all cases where Persons are distinguished by the length of their names.
4. Two Persons with weights that differ by over 0.5 kilos, and different names. Covers cases where all values are different.
5. Two Persons with identical weights, names of the same length that differ in at least 3 places. Covers cases where Persons are distinguished by checking their names character-by-character.