

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**.)*

This midterm test consists of 2 questions on 4 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.*

MARKING GUIDE

1: _____/25

2: _____/ 7

TOTAL: _____/32

Good Luck!

Question 1. [25 MARKS]

Note: read parts (a) and (b) completely before answering.

Part (a) [9 MARKS]

Consider the following partially-implemented class, `Glass`. Write the missing methods `setBeverage`, `addBeverage`, and the constructor.

```
/* A Glass has a maximum capacity of a certain number of millilitres.
 * The fullness tells us how much the Glass currently contains.
 * All Glasses in a Dispensary contain the same beverage.
 */
public class Glass {

    private static String beverage; // our standard beverage
    private int capacity; // maximum millilitres this Glass can hold
    private int fullness; // how much is in the glass now

    // [3 marks]
    // setBeverage: Set the beverage to b for every Glass.
    public static void setBeverage(String b) {
        beverage= b;
    }

    // getVolumeRemaining: Return the number of millilitres that
    // can still be added
    public int getVolumeRemaining() {
        return capacity - fullness;
    }

    // [3 marks]
    // constructor: set the capacity of this glass to c.
    public Glass(int c) {
        capacity= c;
    }

    // [3 marks]
    // addBeverage: add n millilitres of beverage to this
    // Glass. You don't have to check whether it is
    // overflowing.
    public void addBeverage(int n) {
        fullness= fullness + n;
    }
}
```

Part (b) [6 MARKS]

Follow the instructions given in the comments for the class `Dispensary` below. This class uses `Glass` from part (a).

```
/** The class Dispensary uses the class Glass.
 */
public class Dispensary {
    public static void main(String[] args) {
        // [1 mark] Write a line of Java to set the beverage
        // to "cider" for every Glass
        Glass.setBeverage("cider");

        Glass mug= new Glass(300);

        // [1 mark] Write a line of Java that adds 150
        // millilitres of beverage to mug
        mug.addBeverage(150);

        int moreFluid= 170;

        // [4 marks]
        // Write a code fragment (several lines of code) that
        // checks whether there is enough room left in mug to
        // add moreFluid millilitres of beverage. If there is
        // enough room, then add moreFluid millilitres of beverage
        // and print "Millilitres remaining: X", where X is how many
        // more millilitres can be added after adding moreFluid.
        // If there is not enough room, print "Not enough room"
        // and do not add moreFluid millilitres of beverage.
        if (mug.getVolumeRemaining() >= moreFluid) {
            mug.addBeverage(moreFluid);
            System.out.println("Millilitres remaining: " +
                               mug.getVolumeRemaining());
        }
        else {
            System.out.println("Not enough room");
        }
    }
}
```

Part (c) [1 MARK]

Write the output of `Dispensary`.

Not enough room

Part (d) [2 MARKS]

Write the name of each **instance** variable in `Glass` and `Dispensary`.

`capacity`, `fullness`

Part (e) [2 MARKS]

Write the name of each **local** variable in `Glass` and `Dispensary`.

`mug`, `moreFluid`

Part (f) [3 MARKS]

Write the name of each **parameter** in `Glass` and `Dispensary`.

`b`, `c`, `n`, `args`

Part (g) [1 MARK]

Write the name of each **static** variable in `Glass` and `Dispensary`.

`beverage`

Question 2. [7 MARKS]**Part (a)** [2 MARKS]

The class `String` is a standard part of Java, and a `String` is immutable (unchangeable). In light of this, explain why (or why not) the following code fragment is possible in Java:

```
String s= "hi";  
s= s + s;
```

The `String` variable `s` is initialized to contain the address of `String` "hi". On the next line `s` is assigned the address of the `String` "hihi". No `String` is changed in this process.

Part (b) [5 MARKS]

Complete the method `chopString` below.

```
/** chopString removes the first s.length()/2 characters from the  
 * beginning of s, adds them to the end (in the same order), and  
 * and returns the result. You may assume that s is not null.  
 */  
public static String chopString(String s) {  
    int mid= s.length()/2;  
    return s.substring(mid) + s.substring(0,mid);  
}
```