To Students: This file contains the solutions to the midterm test, together with the marking scheme and comments for each question. Please read the solutions and the marking schemes and comments carefully. Make sure that you understand why the solutions given here are correct, that you understand the mistakes that you made (if any), and that you understand *why* your mistakes were mistakes.

Also, remember that although you may not agree completely with the marking scheme given here, it was followed the same way for all students. We will remark your test only if you can clearly demonstrate that the marking scheme was not followed correctly. We will make *no* exception to the marking scheme, except if you can clearly demonstrate that it is somehow incorrect.

For all remarking requests, please follow the directions given in our "Policy on Remarking" (available on the "Policies and Forms" section of the Main Webpage). For all other questions, please don't hesitate to ask your instructor during office hours or by e-mail.

To Marker: In general, having the right idea should count for approximately half of the marks, even if it isn't expressed perfectly. Start from the assumption that students know what they are doing, unless they give you evidence to the contrary. In particular, a correct answer followed or preceded by an incorrect justification or explanation deserves fewer marks than the correct answer alone. In contrast, an incorrect answer accompanied by a correct explanation or followed by an acknowledgement that the answer is incorrect deserves more marks than the incorrect answer alone. In general, try not to be too picky on the details, except when they are critical to the solution.

Remember that students should get 20% of the marks for a question (don't round) if they write "I do not know how to answer" or something similar, as long as they write nothing else (or that they cross off anything else they wrote to make it clear it should not be marked). If a student writes "I do now know how to answer" before or after an attempted solution, mark only the attempted solution and ignore the sentence stating that they do not know. Please do not give half marks: in general, give students the benefit of the doubt and give them a full mark, or don't give them the mark at all if it seems unreasonable. The important goal is consistency: even if you feel your choice is a little harsh, or a little lenient, stick with it for everyone.

While you mark, please keep track of common student errors and how they were marked, as well as any interpretations of or minor modifications to the marking scheme (including any further breakdown of the marks that you decide to use), so that it is easy to figure out later how a question was marked. Also please make note of how well each question or each part of a question was answered in general, and of any serious misconceptions or apparent gaps in student's knowledge that you noticed. (These comments will be typeset with the solutions and posted on the course website so that students can find out where and why they lost marks.)

Finally, please remember to give students enough feedback on their copy of the test so that they can easily figure out what they did wrong (if anything) from the solutions, marking schemes, and comments, together with your feedback. In particular, you may find it convenient to use codes (like "$E1$") to report common errors on a student's paper, and list the codes and their meaning in your marking comments.

Question 1. [16 marks]

Part (a) [2 marks]

Complete the identification section at the top of page 1, then write your student number **legibly** at the bottom of every page of this test except page 1 (where indicated).
(Hint: The questions on this test are in no particular order; start with the easier questions first!)

Marking Scheme:

- -1 for each page where the student number is missing, or for writing the student number at the bottom of page 1, or for each line of information missing from the identification section on page 1.

Marker's Comments:

- These were essentially "free" marks (you did not need to know any material), but you had to follow the directions precisely in order to get them!

Question 1. (CONTINUED)

Part (b) [14 MARKS]

Below, write code for methods `deleteAfter` and `contains` so that they meet their specification.

SOLUTION: (See the code below.)

```
public class SomeNode {
    public Comparable data;    public SomeNode link;
    public SomeNode(Comparable data, SomeNode link)
      { this.data = data;  this.link = link; }
}

public class SomeList {
    private SomeNode head; // the first node in this list; ‘null’ if this list is empty
    // Constructor and other methods go here...

    // Remove the node immediately following ‘n’ from this list.  If n == null, remove
    // the first node in this list.  (Do nothing if there is no node following ‘n’.)
    private void deleteAfter(SomeNode n) {

        //// SOLUTION:

        if (n == null) {
            if (head != null)  { head = head.link; }
        } else {
            if (n.link != null)  { n.link = n.link.link; }
        }

    } // deleteAfter()

    // Return true if this list contains ‘item’; false otherwise.
    // Requires: item != null  (Use the ‘equals’ method to compare elements.)
    public boolean contains(Object item) {

        //// SOLUTION:

        SomeNode curr = head;
        while (curr != null && !item.equals(curr.data))
          { curr = curr.link; }

        return (curr != null);

    } // contains()
} // SomeList
```

Question 1. (continued)

Part (b) (continued)

Marking Scheme:

- 4 marks for general Java syntax: −1 per error (each error penalized only the first time) ignore simple "typos" like missing semicolons or parentheses
- 5 marks for the body of method `deleteAfter`:
  - B. 1 mark for attempting to remove the first node in the list if `n == null`
  - C. 1 mark for attempting to remove the node immediately following `n` otherwise
  - D. 1 mark for testing `head` (or `n.link`) before attempting to remove a node
  - E. 1 mark for correctly updating `head` to remove the first node
  - F. 1 mark for correctly updating `n.link` to remove the node following `n`
- 5 marks for the body of method `contains`:
  - G. 1 mark for using a local `SomeNode` reference to traverse the list
  - H. 1 mark for correct loop structure (initialization, loop condition, increment), even if syntax is incorrect
  - J. 1 mark for stopping the loop when `item` is found
  - K. 1 mark for returning a `boolean` value (even if it is incorrect)
  - L. 1 mark for returning the correct value

Marker's Comments:

- For method `deleteAfter`, many students misunderstood the question and tried to remove a node that comes before a node that stores `n` as data.
- For method `deleteAfter`, many students forgot to check `head != null` before writing `head = head.link`.
- For method `deleteAfter`, many students searched for `n` in the list unnecessarily; this was not penalized.
- There was some confusion about the types of the parameter for method `deleteAfter`: `n` was a node, not data (and certainly not an `int`)!

Instructor's Comment:

After working on Assignment 2 and completing the lab for week number 5 (writing class `LinkedQueue`), there was no excuse for having trouble with very simple linked list manipulations like the ones in this question. If you have trouble with the course material, ask questions!

Question 2. [17 MARKS]

Consider the following Java program.

```
public class A {                                public class B extends A {
    public Integer i;                               public Integer i;
    public A(Integer i)  { this.i = i; }            public B(Integer i)  { super(i);  this.i = i; }
    public boolean equals(Object o)                 public boolean equals(Object o) { // DRAW HERE
      { return this.i == ((A) o).i; }                   return this.i.equals(((A) o).i);
}                                                   }
                                                }
public class Driver {
    public static void main(String[] args) {
        A a = new A(new Integer(73));    B b = new B(new Integer(73));    sioc(a, b);
        System.out.println(a.i+", "+b.i+", "+a.equals(b));         // OUTPUT: 73, 73, false
    }
    public static void sioc(A a, A b) {
        b.i = new Integer(35);
        System.out.println(((B) b).i+", "+b.i+", "+b.equals(a));   // OUTPUT: 73, 35, true
        b = new A(new Integer(99));
    }
}
```

Part (a) [5 MARKS]

Write the output of the program, next to each call to `System.out.println` (where indicated by the comments).
(This is *not* a "trick question": the program compiles and runs without error.)

SOLUTION:

(See the answers in the code above.)

MARKING SCHEME:

- −1 mark for each error—if one of the values is incorrect (*e.g.*, "99" instead of "73") but correctly stays the same for each line of output, only take off 1 mark

MARKER'S COMMENTS:

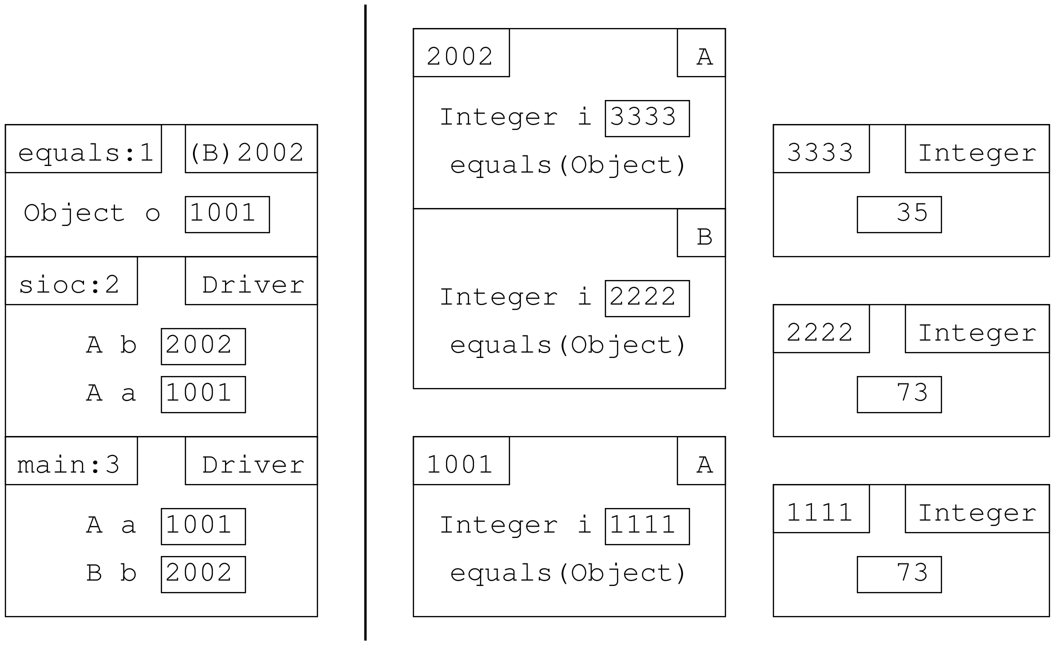
- Mostly done poorly.

Question 2. (continued)

Part (b) [12 marks]

Below, draw a picture of the memory model when execution of the program reaches the line labelled "`DRAW HERE`". Include only the Call Stack and the Object Space in your picture (*i.e.*, do not draw the Static Space).

Solution:



Marking Scheme: Note that the marks below add up to more than 12, but each student's total mark cannot exceed 12: this simply means that students can get perfect even if they have a few mistakes.

- Call Stack:

  A. 1 mark for the general format of method frames: each frame contains method name and line number, scope, local variables and parameters (even if their values are incorrect)

  B. 1 mark for having a frame for method `main`

  C. 1 mark for having a frame for method `sioc`

  D. 1 mark for having a frame for method `equals`

  E. 1 mark for having no other method frame on the Call Stack

  F. 1 mark for having the correct scope for each method frame: `Driver` for the `main` and `sioc` frames; `2002` (the value of parameter `b` in the `sioc` frame) for the `equals` frame (0 if any scope is incorrect)

  G. 1 mark for having the correct local variables or parameters in each method frame, with or without the types indicated: `a`, `b` in the `main` frame; `a`, `b` in the `sioc` frame; `o` in the `equals` frame (0 if any variable or parameter is missing)

  H. 1 mark for having correct values for the variables in the `main` frame (0 if any value is incorrect)

  J. 1 mark for having correct values for the variables in the `sioc` frame: `a` and `b`'s values should be the same as in `main` (0 if any value is incorrect)

  K. 1 mark for having the correct value for parameter `o` in the `set` frame: should be the same as `a`'s value in `sioc`

Question 2. (CONTINUED)

Part (b) (CONTINUED)

MARKING SCHEME (CONTINUED):

- Object Space:

  L. 1 mark for the general format of objects: every object has a unique address, and clearly labelled boxes for each part of the object (even if the values are incorrect)

  M. 1 mark for having one instance of class `A`

  N. 1 mark for having one instance of class `B`

  P. 1 mark for having three separate instances of class `Integer` (0 if any instance is missing)

  Q. 1 mark for having the correct values in the instances of class `Integer` (0 if any value is incorrect)

  R. 1 mark for having variable `i` and method `equals()` in the instance of class `A`

  S. 1 mark for having the correct value for variable `i` in the instance of class `A`

  T. 1 mark for having separate boxes for the `B` part and the `A` part of the instance of class `B`

  U. 1 mark for having variable `i` and method `equals()` in each part of the instance of class `B`

  V. 2 marks for having the correct values for both variables `i` in the instance of class `B` (1 mark each)

MARKER'S COMMENTS:

- Most students missed points "F" to "K" of the marking scheme: missing frames and incorrect values for local variables.
- Students did better in the Object Space, but many missed points "P" and "Q".

INSTRUCTOR'S COMMENT:

The marking scheme for this question was incredibly generous: you had the possibility of getting 21 marks, so you needed to make at least 10 mistakes in order to get less than perfect. **Don't let this fool you!** Even if you received 12 marks for this question, you may still have serious misconceptions about the Java memory model, and understanding how Java programs run. Make sure that you understand exactly what the program in this question does, and why it produces the output given in the solutions: if you have trouble, ask questions!

Question 3. [17 marks]

Suppose that you have a class `SomeQueue` that `implements` the following `Queue` interface.

```
public interface Queue {
    int size(); // Return the number of elements I contain.
    void enqueue(Object o); // Add ‘o’ to the back of my elements.
    Object dequeue(); // Remove and return my front element.  Requires: I am not empty.
}
```

Below, write the body of method `removeFirst` so that it meets its specification.

```
// Remove the first occurrence of ‘o’ from ‘q’ ("first" means "closest to the front").
// Requires: q != null, o != null.  Ensures: ‘q’ contains the same elements as before,
// in the same order, except for the first occurrence of ‘o’ that has been removed.
// Throws: NoSuchElementException if ‘o’ does not appear in ‘q’.
public static void removeFirst(Queue q, Object o) {
    // HINT: Create a new object and add it to the queue as a "sentinel", so that you
    // know when to stop examining elements (this is only part of what you must do).

    //// SOLUTION:

    // Add "sentinel" object at the end of the queue.  We create a new object
    // to guarantee that it cannot be in ‘q’ already.
    String s = new String("END");
    q.enqueue(s);

    boolean found = false; // have we found ‘o’ yet?

    // Go through elements of ‘q’ until ‘s’ is reached.
    Object p = q.dequeue();
    while (p != s) {
        if (!found && o.equals(p))
          { found = true; } // don’t put the first ‘o’ back
        else
          { q.enqueue(p); } // put ‘p’ back into ‘q’
        p = q.dequeue();
    }

    // Throw NoSuchElementException if ‘o’ was not found.
    if (!found)  { throw new NoSuchElementException(); }

} // removeFirst()
```

Marking Scheme:

A. 4 marks for code that correctly removes the first occurrence of `o` from `q`

B. 4 marks for code that correctly leaves the contents of `q` unchanged except for the first occurrence of `o`

C. 3 marks for correctly throwing the `NoSuchElementException`

D. 6 marks for Java syntax

Question 3. (continued)

Marker's Comments:

- −4 marks (points A, B) if syntax is completely wrong but I can still tell what you are trying to do
- −3 marks (point D) if code accesses non-existent members of `q` (*e.g.*, "`q[i]`", "`q.length`", "`contents[i]`", etc.) Remember that `q` is a `Queue`, *not* an array or a linked list: you have no knowledge of or access to the way that `q` is implemented!
- −2 marks (point D) for any of the following syntax errors:
  - writing `new Queue()` (`Queue` is an `interface`)
  - calling `compareTo` to compare objects (this only works if references are of type `Comparable`, and that was not the case in this question; you should have used `equals` instead)
  - `return`ing something (the method has type `void`: it does not return anything)
  - expecting `q.dequeue()` to return `null` when the queue is empty (there is nothing in the description of the `dequeue` method that says this; in fact, it is simply not true in general)
  - trying to copy elements from one stack to another by writing something like "`s = t`" (this only copies the reference from one local variable to another: it does absolutely nothing to change the contents of those stacks)
- −2 marks (point A) for removing every occurrence of `o` (or only the last one), instead of only the first one
- −2 marks (point B) for not putting elements back into `q` after finding `o`, or for trying to transfer elements back by writing only something like "`q = t`"
- −1 mark (point C) for not throwing the exception under the correct conditions
- −1 mark (point B) for writing a loop that tests "`i < q.size()`" (when an element is removed from `q`, its size will change and the loop will stop too soon)
- −1 mark (point D) for doing something like "`q.enqueue("...")`" and then "`while (!t.equals("..."))`" (this relies on an assumption that the String `"..."` does not already appear in `q`, and that is not a good assumption to make)

Instructor's Comments:

The logic of the code you had to write was not obvious, and I expected people to make many small mistakes (this was supposed to be the tricky question on the test). But everyone *should* have known how to use a `Queue`, especially after the work that we've done in the closed labs. Also, there were a number of more serious errors (like misunderstandings about references). If you are having trouble with these basic concepts, *ask questions*! I am here to help, but I cannot do that unless you come to me with your questions.
Also, many students apparently forgot about the "20% rule". Remember that if you have no idea how to answer a question, you will almost certainly do better by writing "I do not know how to answer" and automatically getting 20% of the marks for the question. (This rule applies only for the test and exam and was clearly written on the cover page, as well as being announced on the `Midterm Test` page of the course website since the start of the term).