

Sample Midterm

This sample midterm contains the kinds of questions that will be on the actual midterm examinations. It is much longer than the total size of the all midterm questions will be. At least one question on the midterm will be all but identical to one here.

We will not post solutions to the sample midterm problems because, in our experience, people will be tempted to look at the solution before completely doing the problem. If anyone wishes to check their answers, they may give us a paper copy for review.

We will also answer questions by email, generally by giving hints or asking leading questions back.

1 Definitions and Motivation

- What does the acronym “ADT” stand for? What is an ADT? What is the connection, if any, between an ADT and a data structure? Use an example to illustrate the answer to your last question.
- Why do we distinguish data structure from operational behavior? How are they each represented in Java programs? Give two reasons why we separate the operational behavior from data structure when defining abstract data types.
- What does it mean to throw an exception? How does one express throwing an exception in Java? Give two different ways program handle errors other than throw exceptions. Give one reason to use exceptions instead of these other ways.
- How are exceptions useful for implementing libraries or generic ADT’s used by different applications?
- In Homework # 2, we changed the size of the array we were using. Since arrays are fixed size, how did we accomplish this feat?
- Compare and contrast internal versus external iterators. What are the advantages of each? Give examples.
- What requirement must be established if you are going to dereference a variable? Explain how one can recognize a dereference operation in the code, and then explain what the requirement for a safe dereference is and why.
- Distinguish exogenous from endogenous linked lists.
- Why might bugs only be found in the invariant checker rather than by the driver? Why is it *preferable* that they are found in the invariant checker rather than the driver?
- What is an *interface*? Why do we use them?
- What is an abstract class? Why do we have them?
- Explain `AbstractCollection`. How is it able to implement almost all the required methods? What can you say about the implementation?
- Give three reasons why one would override a method in an abstract superclass.

- Why is a nested class used to implement iterators not declared static?
- Explain the different flavors of linked lists that we have used in homework.
- What are *generic classes*? Why do we use them? Clearly explain the advantage(s). Give an example generic class from the standard Java collection library.
- Suppose that you convert a `Bag` class to be generic. Name at least one place where the word `Bag` remains just `Bag` instead of changing to `Bag<Item>`.
- A generic type parameter (such as `T`) can be used most places where a class name can be used. Where can it *not* be used? Why not?
- Looping through a structure erasing elements as you go is very dangerous. Could you explain with an example what may happen?

2 Code

- Write a version of a new public member function `Bag::contains`, assuming `Bag` has the following fields:

```
private int _count;
private Coin contents[CAPACITY];
```

Fill the code in the following skeleton:

```
/** Return when a matching coin is found in the Bag
 * Return false otherwise.
 * @param item coin to match against (must not be null)
 * @throw NullPointerException if item is null
 */
public boolean contains(Coin item)
{

    // Your code here!

}
```

- Do the same code assuming the following data structure:

```
private class Node {
    public Coin data;
    public Node next;
    public Node(Coin c, Node n) { data = c; next = n; }
}
private Node head;
public Bag() { head = null; }
```

- Implement the following function (using the array data structure)

```

/** Remove and discard all coins matching the argument.
 * @param item coin to match against (must not be null)
 * @throw NullPointerException if item is null
 * @return number of coins discarded
 */
public int removeAll(Coin item)
{

    // Your code here!

}

```

- Do the previous problem, but with a linked list data structure.
- Write the object invariant for a singly-linked list with a head pointer, a tail pointer and a count of items.

3 Debugging

- In an array-based Seq implementation, a friend wrote the following code for an insert method:

```

116 public void insert(CarControl element)
117 {
118     ensureCapacity(2*contents.length+1);
119
120     ++manyItems;
121
122     if (currentIndex == manyItems) {
123         contents[currentIndex] = element;
124     } else if (currentIndex == 0) {
125         for (int i=manyItems; i > 0; --i) {
126             contents[i] = contents[i-1];
127         }
128     } else {
129         for (int i=currentIndex; i < manyItems; ++i) {
130             contents[i+1] = contents[i];
131         }
132     }
133
134     contents[currentIndex] = element;
135
136 }

```

1. As soon as you see the code, without even looking at the actual details of the logic, you can see that it is likely to have bugs. Why? Explain!
2. When you run the code with the driver, the result is:

```
java.lang.ArrayIndexOutOfBoundsException: 1
```

```

at edu.uwm.cs351.DiskSeq.insert(DiskSeq.java:126)
at Driver.testDiskSeq(Driver.java:182)
at Driver.main(Driver.java:27)
Initial tests
Passed 112 tests.
Failed 1 test.

```

Your friend says “Great! I failed only 1 out of 113 tests!” You gently tell them ... [what?]

3. Why would “1” be a bad array index in the given code? Surely the array has at least one element?
 4. You fix a problem on line 125, and now the driver passes all tests. But there are still two errors. The first was not found by the driver because it never tested inserting an element before the second element of a sequence with at least three elements. (This is fixed now!) What is the bug?
 5. Why does the driver not detect the problem even though it inserts at all locations in a two element sequence (at the start, in the middle and at the end?)
 6. There is another problem which happens when one inserts 28 elements in the sequence. It gets slower and slower and then there is an `OutOfMemoryError`. What is this error? Why did the driver not catch it?
- Consider the following code to insert an item in a sorted linked list without a dummy node.

```

61 public void insert(T t) {
62     for (Node p = head; head != null; p = p.next)
63         if (p.data.compareTo(t) > 0) break;
64     if (p == head) p = new Node(t, head);
65     else p.next = new Node(t,p);
66 }

```

1. If we insert the value 5 into an empty list, nothing happens. The list stays empty. Why did we get this problem? Fix the bug.
 2. Now assume we have successfully inserted 5 into an empty sequence. If we then try to insert the value 7, it crashes with a “null pointer exception” on line 63. In the debugger it says that `p` is null. There’s a simple thing wrong with the loop header on line 62. What? Fix it. (This doesn’t fix the full problem, of course.)
 3. Once we fix that problem, it still crashes when we attempt to insert 7 into the list, this time on the ‘else’ line (line 65). Again the debugger says that `p` is null. What happened? Fix the bug. This will require a bigger change.
 4. Depending on how you fixed the previous bug, there may still be nasty bugs.
- Consider the following buggy definition for a doubly-linked list:

```

1 class DoublyLinkedIntSeq {
2     private class Node {
3         int data;
4         Node prev, next;
5         Node(int d) { data = d; next = prev = null; }

```

```
6     };
7     private Node head, tail;
8     private Node current;

15    private boolean _wellFormed() { ... }

35    public DoublyLinkedIntSeq() {
36        head = tail = null;
37        assert _wellFormed() : "Invariant false after constructor";
38    }

41    public void insert(int v) {
42        assert _wellFormed() : "Invariant false before insert";
43        if (current == null) {
44            if (tail != null) {
45                Node n = new Node(v);
46                n.next = tail;
47                tail.next = n;
48                tail = tail.next;
49            } else {
50                head = tail = new Node(v);
51            }
52            current = tail;
53        } else {
54            Node p = new Node(v);
55            p.prev = current.prev;
56            p.next = current;
57            current.prev.next = p;
58            current.prev = p;
59            current = p;
60        }
61        assert _wellFormed() : "Invariant false after insert";
62    }

63
64    public void advance() { // no bugs here!
65        if (!hasCurrent()) throw new IllegalStateException("at end already");
66        current = current.next;
67    }

68
69    public boolean hasCurrent() { // no bugs here!
70        return current != null;
71    }

72
73    public void print() { // no bugs here!
74        for (Node p = head; p != null; p = p.next) {
75            if (p != head) System.out.print(",");
76            System.out.print(p.data);
```

```
77     }
78     System.out.println();
79 }
```

To test the code, we try the following test:

```
82     DoublyLinkedIntSeq a = new DoublyLinkedIntSeq();
83     a.insert(112);
84     a.insert(66);
85     a.print();
```

This code crashes on line 57:

```
Exception in thread "main" java.lang.NullPointerException
  at DoublyLinkedIntSeq.insert(DoublyLinkedIntSeq.java:57)
  at DoublyLinkedIntSeq.main(DoublyLinkedIntSeq.java:75)
```

- In class we talked about rules for dereferencing. How was this rule violated on line 57?
- If we do the simplest thing to follow the rule and avoid the crash, the print call on rule 76 will only print 112. What should be on line 57? (Fix the whole bug)
- In an independent test, someone does:

```
87     DoublyLinkedIntSeq b = new DoublyLinkedIntSeq();
88     b.insert(42);
89     b.advance();
90     b.insert(88);
91     b.print();
```

Whether or not the first bug (b) is fixed, this will print 42,8842,8842,8842,... ad infinitum. Why? Explain!

- This problem would have been detected *before* the infinite loop by `_wellFormed()` but it didn't get a chance. What is the job of `wellFormed()` and What problem could it have detected? Why doesn't it always run? Explain the reasoning behind this convention!
- Fix the code.