Your five-digit number: _____

Problem 0 (1 point)

Put your secret number on each page. Also make sure you have provided the information requested on the first page.

Problem 1 (7 points)

Part a

Using inheritance, define a class TrackedQueue that behaves just like Queue except for an extra method:

```
public int maxSizeSoFar ( );
```

which returns the maximum number of elements in this queue at any time since the queue was constructed. (Our version of the Queue class is described in another document handed out at the exam.) Take advantage of the superclass's methods and variables as much as possible rather than reinventing the wheel.

Your five-digit number: _____

Problem 1, continued

Part b

Identify the result of attempting to compile and run each of the program segments below. For those that produce an error, explain whether it is a compile-time or a run-time error. For those segments that execute without error, indicate what they do, drawing your answers from the following list.

- prints 0;
- prints 1;
- calls Queue's enqueue method
- calls TrackedQueue's enqueue method
- calls Queue's maxSizeSoFar method
- calls TrackedQueue's maxSizeSoFar method

To earn credit for this part, you must have an essentially correct answer for part a.

program segment	*effect(s)*
`Queue myQ = new TrackedQueue ( );` `System.out.println (myQ.maxSizeSoFar ( ));`	
`TrackedQueue myQ = new Queue ( );` `System.out.println (myQ.maxSizeSoFar ( ));`	
`Queue myQ = new TrackedQueue ( );` `myQ.enqueue ("MIKE");`	

Your five-digit number: _____

Problem 2 (8 points)

Consider the following implementations for a queue.

A. a java.util.ArrayList with the front of the queue at the end of the array (the N–1st element in a queue of N elements) and the back of the queue at the start of the array (element 0). Assume that there is room in the array to enqueue an element.

B. a singly linked list with an additional reference to the last node in the list (the tail), with the front of the queue last in the list and the back of the queue at the head of the list.

For each implementation, give estimates for the number of operations necessary to enqueue an element and to dequeue an element. Your estimate may use "big-Oh" notation, or "time proportional to ___". Briefly explain your answers.

Implementation	*Operation*	*Estimate of running time in terms of N, the number of elements currently in the queue, along with a brief explanation.*
A	Enqueue	
A	Dequeue	
B	Enqueue	
B	Dequeue	

Your five-digit number: _____

Problem 3 (4 points)

Recent lab activities involved constructing a HashMap of Person objects for use as a phone book.

Part a

What change in the HashMap behavior would result from replacing the Person.hashCode method by the following?

```
public int hashCode ( ) {
   return 10;
}
```

Briefly explain your answer.

Part b

What change in the HashMap behavior would result from replacing the Person.equals method by the following?

```
public boolean equals (Object obj) {
   return true;
}
```

Briefly explain your answer.

Your five-digit number: _____

Problem 4 (8 points)

Part a

Amoebas in an AmoebaFamily are accessed by name. Thus it's important that amoebas in a family have unique names. Write a void isOK method for the AmoebaFamily class that iterates through the amoebas in the family and throws IllegalStateException if it encounters two amoebas with the same name. You may assume that a correct version of the AmoebaIterator class from a recent lab is available. However, don't add any code to the Amoeba or AmoebaIterator classes.

Your isOK method should run as fast as possible on the average. Hint: use a hash table.

```
public void isOK ( ) throws IllegalStateException {
```

Your five-digit number: _____

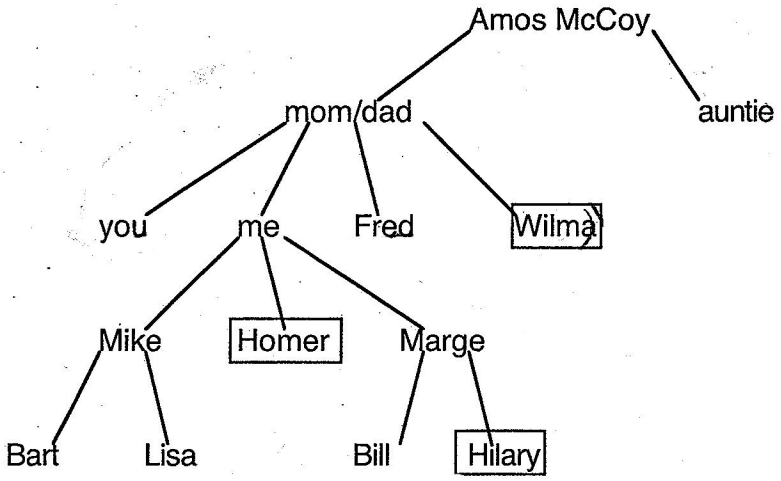
Problem 4, continued

Part b

Suppose that the iterator used a stack as a "fringe" structure, and that the next method updated the stack as follows:

```
public Amoeba next ( ) {
   Amoeba x = (Amoeba) fringe.pop ( );
   for (int k=0; k<x.myChildren.size( ); k++) {
      fringe.push (x.myChildren.get (k));
   }
   return x;
}
```

Indicate which of the three amoebas boxed in the diagram below is visited first, which is visited second, and which is visited third in the iteration.



Of the amoebas named Homer, Hilary, and Wilma, which is visited first? _____

Of the amoebas named Homer, Hilary, and Wilma, which is visited second? _____

Of the amoebas named Homer, Hilary, and Wilma, which is visited third? _____

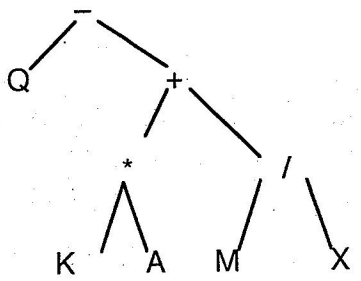
Your five-digit number: _____

Problem 5 (8 points)

In lab, you worked with a binary tree representation of an arithmetic expression tree. The BinaryTree class represented the tree as a whole; TreeNode objects stored operators and operands.

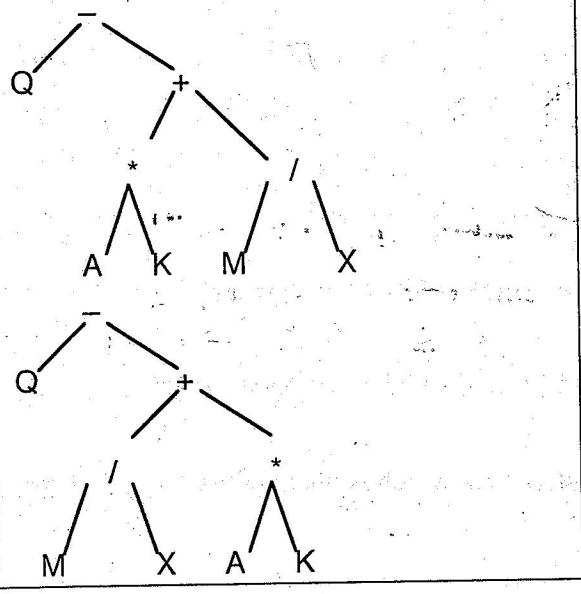
We are simplifying the tree somewhat for this problem. In particular, each myItem is a character ('+', '–', '*', '/', or a one-letter variable name). Declarations are given in an accompanying document.

On the next page, define a boolean BinaryTree method named sameComputation. This method, given another binary expression tree as argument, checks if the two trees represent the same computation, that is, are the same except that the operands to the commutative operators "+" and "*" may be exchanged.

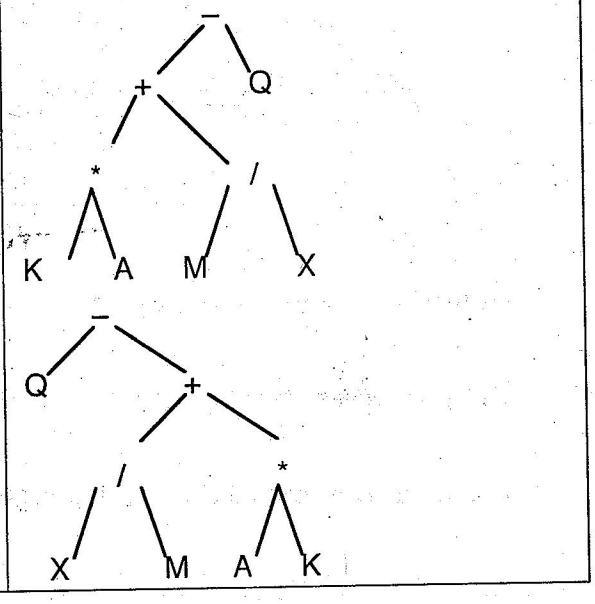


Suppose that this tree represents the expression tree on the right. Two examples that do the same computation appear below on the left; two that don't appear below on the right.

Same computation



Different computations



Trees with different variables, or those with completely different structure (e.g. having a different number of nodes), also represent different computations. You may assume both trees represent legal nonempty expression trees.

Problem 5, continued

```
public boolean sameComputation (BinaryTree expr) {
```

Your five-digit number: _____

Problem 6 (4 points)

Add exactly three assignment statements (one per blank) to the destructive reverseHelper method to complete the List reverse method. (Declarations for the List and ListNode classes appears in a separate document.) Don't change any of the existing code, and don't create any new list structure.

```
public void reverse ( ) {
   if (myHead != null) {
      myHead = reverseHelper (myHead);
   }
}

private static ListNode reverseHelper (ListNode head) {
   ListNode soFar = null;
   for (ListNode p=head; p!=null; /* no increment */ ) {
      ListNode temp = p.myRest;

      _____ ;

      _____ ;

      _____ ;

   }
   return soFar;
}
```