Computer Science 136 Exam 1

Sample exam

Possible answers

1. True/false statements (2 points each). Justify each answer with a sentence or two.

   a. *Free question from 2/28.* Two instances of `class Association` in the `structure` package are equal if and only if their `key`s are equal, regardless of their `value`s.

   True, because the `equals()` method for an `Association` behaves this way.

   b. An instance variable declared as `protected` can be accessed by any method of the class in which it is declared.

   True. All instance variables can be accessed by any method of the class in which it is declared.

   c. A binary search can locate a value in a sorted `Vector` in $O(\log n)$ time.

   True. We can apply the "divide and conquer" approach to halve the search size at each step.

   d. A binary search can locate a value in a sorted `SinglyLinkedList` in $O(\log n)$ time.

   False. Since there is no direct access to the middle of the list, we cannot apply a binary search.

   e. A method with no precondition should return with its postcondition true every time it is called.

   True. No precondition implies that the method should work correctly for all inputs and terminate with its postcondition true.

   f. Elements stored in an instance of `class Vector` can be of any Java data type.

   False. Items must be derived from `Object`, so primitive types cannot be stored directly in a `Vector`.

   g. Instance variables are specified in an interface file.

   False. Instance variables are specified in the class definition, not in the interface.

2. Consider the following Java program, which should look very familiar. (10 points)

```
class Container {
    protected int count;
    protected static int staticCount;
```

```
    public Container(int initial) {
        count = initial;
        staticCount = initial;
    }

    public void setValue(int value) {
        count = value;
        staticCount = value;
    }

    public int getCount() {
        return count;
    }

    public int getStaticCount() {
        return staticCount;
    }
}

class WhatsStatic {

    public static void main(String[] args) {
        Container c1 = new Container(17);
        System.out.println("c1 count=" +c1.getCount()+
                           ", staticCount=" + c1.getStaticCount());

        Container c2 = new Container(23);
        System.out.println("c1 count=" +c1.getCount()+
                           ", staticCount=" + c1.getStaticCount());
        System.out.println("c2 count=" +c2.getCount()+
                           ", staticCount=" + c2.getStaticCount());

        c1.setValue(99);
        System.out.println("c1 count=" +c1.getCount()+
                           ", staticCount=" + c1.getStaticCount());
        System.out.println("c2 count=" +c2.getCount()+
                           ", staticCount=" + c2.getStaticCount());

        c2.setValue(77);
        System.out.println("c1 count=" +c1.getCount()+
                           ", staticCount=" + c1.getStaticCount());
        System.out.println("c2 count=" +c2.getCount()+
                           ", staticCount=" + c2.getStaticCount());
    }
}
```

a. What will the output be when the program is run (`java WhatsStatic`)? Assume no exceptions occur. (4 points)

```
c1 count=17, staticCount=17
c1 count=17, staticCount=23
c2 count=23, staticCount=23
c1 count=99, staticCount=99
c2 count=23, staticCount=99
c1 count=99, staticCount=77
c2 count=77, staticCount=77
```

b. What memory is allocated for `Containers` `c1` and `c2` at the time the line `c1.setValue(99)` is executed? Show any existing local variables and instance variables. (6 points)

The references to our two objects are stored in `c1` and `c2`, which are local variables to the `main` method. The two objects themselves exist, each with its own copy of instance variable `count`. They share one copy of `staticCount`.

3. (26 points) In this problem you are to design a Java interface and class for a data structure which represents sets of characters. As usual for sets, no repeated elements are allowed. Thus, the collection `'a'`, `'e'`, `'i'`, `'o'`, `'u'` is a legal set, but `'a'`, `'e'`, `'a'` is not. This data structure will have two methods:

- `insert(char newChar)` adds `newChar` to the set.
- `contains(char findChar)` returns a boolean value indicating if `findChar` is an element of the set.

a. Write a legal Java `interface` called `CharSetInterface` for this data structure. Include preconditions and postconditions for the methods. (6 points)

```
public interface CharSetInterface {

  public void insert(char newChar);
  // Pre: none.
  // OR Pre: newChar is not in the set
  // Post: newChar is added to the set

  public boolean contains(char findChar);
  // Pre: none.
  // Post: return value indicates if findChar was found in the set
}
```

It is up to you if you want to make insertion of a char already in the set an error condition or just have it return.

b. Suppose we decide to implement `CharSetInterface` by a class in which a singly-linked list holds all of the elements. Write the definition of this class. This should be a full and legal Java class definition *with all method bodies filled in.* Don't forget to declare instance variables, include a constructor, and use qualifiers such as `public` and `protected` when appropriate. You need not repeat your pre- and post- conditions from part a. Please call your class `CharSet`. (10 points)

```
public class CharSet implements CharSetInterface {

  protected List charList;

  public CharSet() {
    charList = new SinglyLinkedList();
  }

  public void insert(char newChar) {
    // this corresponds to the no precondition answer above
    if (!contains(newChar)) {
       charList.add(new Character(newChar));
    }
  }

  public boolean contains(char findChar) {

    return charList.contains(new Character(findChar));
  }
}
```

c. If `CharSet` is implemented as in part b, what would the worst-case time complexity be for the insert operation when the set has n elements? (Use "Big O" notation.) (4 points)

$O(n)$**, as the contains operation is** $O(n)$ **which is more significant than the** $O(1)$ **list insertion.**

d. Suppose we design an alternative implementation in which the set is represented by an array of `boolean`s called `rep` with subscripts ranging from 0 to 65535 (these represent the codes for all of the characters representable in Unicode). For example, the Unicode for `'a'` is 97, so `'a'` is in the set if and only if `rep[97]` is `true`. What is the worse-case complexity of insert with this representation? (You may assume there is a constant-time function available which computes the Unicode value of a given character.) (6 points)

$O(1)$**, as we can determine if the letter is in the set in constant time, and can**

do the insertion in constant time.

4. (20 points) Consider the following class, `ReversibleList`, that extends the `SinglyLinkedList` class by adding a method for reversing the list.

```
public class ReversibleList extends SinglyLinkedList {

  public ReversibleList() {
    super();
  }

  public void reverse() {
    // Pre: list is not empty.
    // Post: list is reversed.
    Assert.pre(!isEmpty(), "Cannot reverse an empty list");
    head = recReverse(head);
  }

  private static SinglyLinkedListElement recReverse(SinglyLinkedListElement current) {
    // Pre: current is not null.
    // Post: list headed by current is reversed;
    //       head elemet of reversed list is returned.
    if (current.next() == null)
      return current;
    else {
      SinglyLinkedListElement newHead = recReverse(current.next());
      current.next().setNext(current);
      current.setNext(null);
      return newHead;
    }
  }
}
```

a. Prove by induction that `reverse()` behaves correctly. (Hint: focus on `recReverse(current)`) (8 points).

We proceed by induction on the length of the list passed to `recReverse()`. Base: `recReverse(current)` when `current.next() == null` corresponds to the one-element case, and clearly works. Inductive hypothesis: Suppose the method works for lists up to size $n-1$. Inductive step: We first make the recursive call `recReverse(current.next())` which, by the inductive hypothesis, correctly reverses items 2 through n. It remains to place `current` in its correct position, at the end of the reversed list. `current.next()` is a reference to the former second element of the list, now the last element of the reversed list. So we set that element's `next` reference to be `current`, and set `current`'s `next` reference to `null`, and the list of size n is now reversed.

b. What is the running time of `reverse()` (2 points)?

$O(n)$.

c. Prove using mathematical induction that your answer to part b is correct. (10 points)

We proceed by counting calls to the `setNext()` method, and claim that there are $2(n-1)$ calls needed to reverse a list of size n. We proceed by mathematical induction on n. Base: For a list of size 1, there are $2(1-1)=0$ calls. Inductive hypothesis: Assume it takes $2(k-1)$ calls to `setNext()`, where $k<n$. Inductive step: For a list of size n, we know by the inductive hypothesis that the recursive call to `recReverse()` makes $2(n-2)$ calls to `setNext()`. We make two additional calls, giving a total of $2(n-2)+2=2(n-1)$, which is $O(n)$.

5. Growth of functions. Using "Big O" notation, give the rate of growth for each of these functions. Justify your answers. (3 points each, 12 total)

a. $f(x)=x^2+17x+2001$

$O(x^2)$

b. $f(x)=\cos(x^4+\log x)$

$O(1)$**, since** cos **always remains in** $[-1,1]$**.**

c. $f(x)=7x$ when x is odd, $f(x)=\frac{x}{7}$ when x is even.

$O(x)$**, since** $\frac{x}{7}$ **is always less than** $7x$**. and** $7x$ **is** $O(x)$**.**

d. $f(x)=5x^3$ for $x<23$, $f(x)=37$ otherwise.

$O(1)$**, since it only matters what happens as** x **gets large.**