Com S 362
Fall 2002

Name: _____

Object-Oriented Analysis and Design

Exam 1 on Design Heuristics and Java

This test has 4 questions and pages numbered 1 through 9.

Reminders

This test is open book and notes. However, it is to be done individually and you are not to exchange or share materials with other students during the test.

If you need more space, use the back of a page. Note when you do that on the front.

This test is timed. We will not grade your test if you try to take more than the time allowed. Therefore, before you begin, please take a moment to look over the entire test so that you can budget your time.

For diagrams and programs, clarity is important; if your diagrams or programs are sloppy and hard to read, you will lose points. Correct syntax also makes some difference.

1. (5 points) Suppose your company has written 10 programs that do accounting for other companies, but these programs are all written in COBOL, without using object-oriented features. Would it be a good idea to adopt iterative design and object-oriented practices in your company to write another, similar, accounting program? Answer either "yes" or "no", and give a brief explanation.

2. (40 points) This problem relates to the readings from Arthur J. Reil's book, chapter 2. Consider the code in class `Interval`, which implements the interface `IntList`. The code for these types, which appears on the following pages, all works correctly.

   In the space below on this page, list (a) all the heuristics violated by the code for `Interval`, which starts on page 4, and for each (b) briefly explain how the heuristic is violated by giving a specific example from the code of `Interval`. (Assume that `IntList` expresses what clients want.) If the code does not violate any heuristics, write that.

```
package lists;

/** Lists of integers. */
public interface IntList {
    /** Tell if this list is empty. */
    boolean isEmpty();

    /** Return the number of elements in this list. */
    int length();

    /** Return the first element in this list. */
    int head();

    /** Return the list of elements in this list following the first. */
    IntList tail();

    /** Return an IntList that contains i
     *  followed by the elements of this list. */
    IntList addFirst(int i);

    /** Return an IntList that contains the elements of this list
     *  followed by i. */
    IntList addLast(int i);

    /** Return true if o is not null and has the same class
     * and value as this list. */
    boolean equals(Object o);

    /** Return a string representation of this list. */
    String toString();
}
```

```
package lists;

/** Lists whose elements are drawn from a consecutive interval of integers.
 * For example, [1, 2, 3, 4]. */
public class Interval implements IntList {

    /** The first int in this interval. */
    public int first;
    /** The last int in this interval. */
    int last;

    /** Initialize this interval to be [first, last]. */
    public Interval(int first, int last) {
        this.first = first;
        this.last = last;
    }

    public boolean isEmpty() {
        return last < first;
    }

    public int length() {
        if (last >= first) {
            return last - first + 1;
        } else {
            return 0;
        }
    }

    public int size() {
        return this.length();
    }

    public int head() {
        return first;
    }

    public IntList tail() {
        return new Interval(first + 1, last);
    }

    public IntList removeFirst() {
        return new Interval(first + 1, last);
    }

    public IntList addFirst(int i) {
        if (i == first - 1) {
            return new Interval(i, last);
```

```
        } else {
            throw new IllegalArgumentException();
        }
    }

    public IntList addLast(int i) {
        if (i == last + 1) {
            return new Interval(first, i);
        } else {
            throw new IllegalArgumentException();
        }
    }

    public boolean equals(Object o) {
        if (o == null || !(o instanceof Interval)) {
            return false;
        }
        Interval il = (Interval) o;
        return this.first == il.first && this.last == il.last;
    }

    public String toString() {
        return leftBracket + first + ", " + last + rightBracket;
    }

    public String leftBracket = "[";
    public String rightBracket = "]";
}
```

3. (40 points) This problem relates to the readings from Arthur J. Reil's book, chapter 5. Consider the code in classes `Cons` and `Empty`. These both implement the interface `IntList` from page 3. The code for these types, which appears on the following pages, all works correctly.

   In the space below on this page, list (a) all the heuristics violated by the code for `Cons` and `Empty`, and for each (b) briefly explain how the heuristic is violated by giving a specific example from the code of these two classes. (Again, assume that `IntList` expresses what clients want.) If the code does not violate any heuristics, write that.

```
package lists;

/** NonEmpty lists of integers. */
public class Cons implements IntList {

    /** The first element in this list. */
    protected int first;
    /** The rest of this list. */
    protected IntList rest;

    /** Initialize this list to contain i followed by the elements of tail. */
    public Cons(int i, IntList il) {
        first = i;
        rest = il;
    }

    public boolean isEmpty() {
        return false;
    }

    public int length() {
        return 1 + rest.length();
    }

    public int head() {
        return first;
    }

    public IntList tail() {
        return rest;
    }

    public IntList addFirst(int i) {
        return new Cons(i, this);
    }

    public IntList addLast(int i) {
        return new Cons(first, rest.addLast(i));
    }

    public boolean equals(Object o) {
        if (! (o != null && o instanceof Cons) ) {
            return false;
        }
        Cons c = (Cons) o;
        return first == c.first && rest.equals(c.rest);
    }
```

```
    public String toString() {
        return "[" + first + ". " + rest.toString() + "]";
    }

}
```

```
package lists;

/** Empty lists. */
public class Empty extends Cons {
    public Empty() {
        super(0, null);
    }

    public boolean isEmpty() {
        return true;
    }

    public int length() {
        return 0;
    }

    public int head() {
        throw new IllegalStateException("Head of an empty list");
    }

    public IntList tail() {
        throw new IllegalStateException("Tail of an empty list");
    }

    public IntList addFirst(int i) {
        return new Cons(i, this);
    }

    public IntList addLast(int i) {
        return new Cons(i, this);
    }

    public boolean equals(Object o) {
        return o != null && o instanceof Empty;
    }

    public String toString() {
        return "[]";
    }
}
```

4. (15 points) In the space below, indicate the changes that would be needed to make the code for `Cons` (on pages 7-8) and the code `Empty` (on page 8) follow the heuristics from Reil's chapter 5. Your answer should consist of, for each class, (a) a short explanation and (b) the the changed parts of the code for that class, with an indication that the rest of code is unchanged. If fixing some of the problems requires a change to `IntList`, just leave them unfixed, as we are taking `IntList` as a given. If the code does not violate any heuristics, just write that.