Name:
UVa Email ID:

Directions

Work alone. You may not discuss these problems or anything related to the material covered by this exam with anyone except for the course staff between when this exam is posted and after class Tuesday. You may not obtain any help from other humans except for the course staff (who will only answer clarification questions).

Open resources. You may use any books you want, lecture notes, slides, your notes, and problem sets. You may use Eclipse and Java any other language interpreter you want. You may also use external non-human sources including books and web sites. If you use anything other than the course book, problem sets, slides, and notes, you should cite what you used.

Answer well. Answer as many of questions 1-9 as you can as well as the optional, ungraded questions on the last page. A “full credit” answer for each question is worth 10 points (but it is possible to get more than 10 points for especially elegant and insightful answers). You also get 10 points for filling in the cover page correctly (including following the course pledge). You may either: (1) write your answers on this exam or (2) type and write your answers into the provided Word template:


Whichever you choose, you must turn in your answers printed on paper and they must be clear enough for us to read and understand.

The marked boxes are designed to be large enough to fit a full-credit answer, but if you want more space you may use the backs or attach extra sheets. If you do, make sure the answers are clearly marked. The questions are not necessarily in order of increasing difficulty. There is no time limit on this exam, but it should not take a well-prepared student more than three hours to complete. It may take you longer, though, so please do not delay starting the exam.

Full credit depends on the clarity and elegance of your answer, not just correctness. Your answers should be as short and simple as possible, but not simpler. Your programs will be judged for correctness, clarity and elegance, but you will not lose points for minor syntactic errors.
1. We have argued that language designers often have to trade-off between various desirable goals in designing a language.

(a) Give one concrete example where the Java programming language designers sacrificed expressiveness for truthiness (that is, the likelihood code means what the developer intends). An ideal answer would illustrate your example with code snippets showing something that is difficult to express concisely because of the Java language's emphasis on truthiness.

(b) Give one concrete example where the Java programming language designers sacrificed simplicity for performance.
2. Write a clear, complete, declarative specification for the Java procedure defined below.

```java
public static void reverse (int [] a) {
    for (int i = 0; i < a.length / 2; i++) {
        int tmp = a[i];
        a[i] = a[a.length - 1 - i];
        a[a.length - 1 - i] = tmp;
    }
}
```

For example,
```java
int [] a = { 1, 2, 3, 4};
reverse(a);
System.out.println(Arrays.toString(a));
```
outputs:
```java
[4, 3, 2, 1]
```
3. (a) Describe a good set of black-box test cases for the procedure you specified in question 1.

(b) Are there any additional glass-box test cases if the implementation shown in question 1 is used?
Consider the MultiSet abstract datatype partially specified below:

```
public class MultiSet {
    // OVERVIEW: A MultiSet is a mutable datatype that represents a multiset of integers.
    // A MultiSet is an unordered collection of integers, except unlike a regular set each
    // element may occur more than once. A typical MultiSet is:
    // { x_1, x_2, ..., x_n }
    // where the set braces mean the collection is ordered, but the same value may appear
    // in the set more than once (so x_i may equal x_j for some i != j).

    public MultiSet()
        // EFFECTS: Initializes this as the empty MultiSet: {} 
    }
```

Pat Programmer proposes to implement the MultiSet type using this rep and abstraction function:

```
private HashMap<Integer,Integer> map;
// Abstraction function:
// AF(c) = { x_1, ..., x_n } where
// forall key: rep.map()
// the value of key appears in x map.get(key) number of times.
```

4. What would be a reasonable rep invariant for this implementation?
Consider the insert method specified and implemented below:

```java
public void insert(int v)
    // REQUIRES: v is a key in map
    // EFFECTS: Looks up the entry for v in the map, and adds one to the associated value.
{
    rep.put(v, rep.get(v) + 1);
}
```

Note: here is the specification for the HashMap get operation (I have “specialized” this specification for a HashMap<Integer, Integer> instead of the generic HashMap as provided by the Java API to make things simpler):

```java
public int get(int key)
    EFFECTS: Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
```

5. Does the implementation of insert satisfy its specification? (Include a brief justification supporting your answer.)
6. There are many flaws in Pat’s specification (repeated below). Describe at least three major problems with it.

```java
public void insert(int v)
    // REQUIRES: v is a key in map
    // EFFECTS: Looks up the entry for v in the map, and adds one to the associated value.
```

(1)

(2)

(3)
7. Provide a better specification for insert.
Taylor Tweaker suggests the following alternative rep and abstraction function for the MultiSet datatype:

```java
private int[] countpos;
private int[] countneg;
// Abstraction function:
//     AF(c) = (x_1, ..., x_n)
//     where forall 0 <= i < countpos.length,
//          if countpos[i] > 0, the value of i appears countpos[i] times, and
//          forall 0 <= j < countneg.length,
//          if countneg[j] > 0, the value of -j appears countneg[j] times.
// Rep Invariant(c):
//     countpos != null and countneg != null
//     forall 0 <= i < countpos.length: countpos[i] >= 0
//     forall 0 <= i < countneg.length: countneg[i] >= 0
```

8. Which representation (Pat’s or Taylor’s) is a better choice? Provide a convincing argument supporting your answer.
9. Using either Pat's or Taylor's representation, write a correct implementation of the
insert method, satisfying your specification from question 6.
(optional, ungraded) Do you feel your performance on this exam will fairly reflect your understanding of the course material so far? If not, explain why.

(optional, ungraded) How do you hope the rest of the class will be different from what we have done so far?

End of Exam