CSE 331 18su Final Exam

Name ____

Welcome to the 331 Final!

Please wait to turn the page until everybody is told to begin.

Friendly reminders:

1. **Attempt every problem.** You can receive partial credit.
2. **Write clearly!** We can't give you credit for answers that we can't read.
3. **If you make a mistake** and need to correct it, be sure that your final answer is very clearly indicated.
4. **If you run out of room** for any problem, there is extra room on page 5. At the original location, be sure to indicate where your answer continues.

This is an opportunity to show off what you have learned. Good luck and have fun!

Part 1: Postfix Calculator

Please turn to the last page of the exam and rip off the page containing the source code for PostfixCalculator. Refer to the PostfixCalculator code to answer the following questions.

1.1. (6 points) Your friend proposes that instead of having a stack as a private instance field, PostfixCalculator should extend the Stack class. Would this be proper use of subclassing? Explain why or why not.

1.2. (6 points) Your company will implement an amazing and advanced new technology, five-function calculator. Your manager gives you part of the specification, including the class overview and the method signatures.

```
/**
 * Makes integer arithmetic computations, given a sequence of tokens
 *   in postfix notation
 * Accepts two kinds of tokens, integer and operator. Here are their formats:
 *     Integer is a sequence of decimal characters, e.g. 5367
 *     Operator is one of the following five characters
 *       +     (add)
 *       -     (subtract)
 *       *     (multiply)
 *       /     (divide)
 *       ^     (power)
 */
public class FiveFunctionCalculator {
    public FiveFunctionCalculator();
    public void accept(String token);
    public boolean hasResult();
    public int getResult() throws IllegalStateException;
    public void clear();
}
```

Your manager tells you to write the implementation and Javadocs. You notice that the given spec is similar to the spec for PostfixCalculator. Can FiveFunctionCalculator be a subclass of PostfixCalculator? Why or why not? If not, explain how you would (or would not) use PostfixCalculator in your implementation. (You do not have to write the implementation and Javadocs in your response, but you may write part of it if you wish to illustrate a point.)

1.3. (6 points) According to the specification of java.util.Stack, the pop method throws an EmptyStackException if there is nothing to pop. The current implementation of accept in PostfixCalculator propagates any EmptyStackException up to the client. Explain why propagating the EmptyStackException is bad design. Rewrite the accept method to correct this issue. Include the implementation of any helper methods or internal classes that you may need.

1.4. (4 points) Your teammate heard that lazy evaluation is all the rage, and proposes to use lazy evaluation in the PostfixCalculator. In the proposed implementation, the accept method would simply accept operands and operators and unconditionally push them all onto the stack. It would not do any computation until the client calls getResult or hasResult. Then it would compute the result all at once. Explain why this design might make it difficult for a client to debug their code that uses PostfixCalculator.

1.5. (6 points) Adapter Pattern. Consider the Adder interface.

```
interface Adder {
    public int add(int a, int b);
}
```

Suppose that a client relies on the Adder interface, but decides that they want to use PostfixCalculator because they could not find a better implementation. The problem is that PostfixCalculator does not have the interface that the client needs. Write an adapter that enables the client to use PostfixCalculator without changing their code that relies on Adder. You do not need to document your code.

1.6. (6 points) Generics. Find and correct improper use of raw types in the code for PostfixCalculator.
a) Write the line number(s) in which improper use of raw types occurs.
b) Rewrite each of those lines to be correct.
c) Explain why raw types in Java are not type safe.

1.7. (6 points) Time Management. Your teammate is working on the implementation of the intrepid TrigonometricCalculator. Your teammate sets the following goal: "Implement and test 90% of the code for TrigonometricCalculator by Friday, September 13 at 5pm." Describe why this might not be a good goal, and write a better goal.

You made it through part 1,
You're more than halfway done!

[Extra room for answers]

Part 2: MapReduce

Let us define MapReduce as an operation that takes in a collection of numeric values and performs two operations: map and reduce, which we will define as follows:

- A map operation takes in a collection, performs the same operation on every element, and returns the resulting collection.
- A reduce operation ∆ is a binary operation with the following properties:
  - commutative: a ∆ b = b ∆ a
  - associative: (a ∆ b) ∆ c = a ∆ (b ∆ c)

  Any operation that is commutative and associative can be used to reduce a collection to a single scalar value. The arithmetic operations of addition and multiplication are two examples of reduce operations.

The following code performs a MapReduce operation, where the map operation is squaring and the reduce operation is addition.

```
public static int squareSum(Collection<Integer> input) {
    Collection<Integer> squares = new ArrayList<>();
    // first square...
    for(int element : input) {
        squares.add(element * element);
    }
    // then sum
    int sum = 0;
    for(int element : squares) {
        sum += element;
    }
    return sum;
}
```

2.1. (12 points) Write a static method called mapReduce that can perform a MapReduce operation for **any** map and reduce operations provided by the client, assuming that the provided operations operate over java Integers. (Hint: It may be helpful to think of the map and reduce operations as callbacks that are passed in to the mapReduce method. You may specify the way in which the client must provide the operations.)

Part 3 (16 points) You are helping the UW Waterfront Activities Center to inventory their equipment. Consider the following class hierarchy:

```
Sailboat extends Watercraft;
Sunfish extends Sailboat;
Motorboat extends Watercraft;
```

Consider the following variable declarations

```
Object o;
Watercraft w;
Sailboat s;
Sunfish f;
Motorboat m;
List<? extends Watercraft> lew;
List<? super Sailboat> lss;
List<? extends Sailboat> les;
```

For each of the following statements, write "ok" if it compiles with no type errors and "error" if there is a type error.

`lew = new ArrayList<Watercraft>();` ____________

`lew = new ArrayList<Sunfish>();` ____________

`lss = new ArrayList<Sunfish>();` ____________

`les = new ArrayList<Sunfish>();` ____________

`les = new ArrayList<Motorboat>();` ____________

`les.add(o);` ____________

`lss.add(f);` ____________

`les.add(f);` ____________

`lss.add(m);` ____________

`les.add(m);` ____________

`lew.add(null);` ____________

`w = lss.remove(0);` ____________

`s = les.remove(0);` ____________

`f = les.remove(0);` ____________

`m = lss.remove(0);` ____________

`m = les.remove(0);` ____________

Part 4 (6 points)

Please choose one of the following prompts and write a brief paragraph on that topic below:

- How could applying the ideas you learned in 331 help to make the world a better place?
- What 331 topic do you wish you had learned earlier? When would it have helped?
- What 331 topic do you think will be most useful in the future? Why?

You made it! Congrats!

Thanks for taking CSE 331! It's been my pleasure to have you in class :) Enjoy your break!

PostfixCalculator Code - Do not write answers on this page.

```
/**
 * Makes integer arithmetic computations, given a sequence of tokens
 *   in postfix notation
 * Accepts two kinds of tokens, integer and operator. Here are the formats:
 *     Integer is a sequence of decimal characters, e.g. 5367
 *     Operator is one of the following four characters
 *       +     (add)
 *       -     (subtract)
 *       *     (multiply)
 *       /     (divide)
 */
 public class PostfixCalculator {
     private Stack<Integer> stack;

     /**
      * Construct a new postfix calculator with clean state
      * @effects constructs a new PostfixCalculator
      */
     public PostfixCalculator() {
         stack = new Stack();
     }

     /**
      * Take in an operator or operand.
      * @requires the given token must be a legal token
      * @requires the sequence of tokens passed in so far must form a legal
      *    (so far) postfix notation sequence representing a legal
      *    arithmetic operation
      * @param token - the token to accept
      * @modifies this
      * @effects adds the token to this calculator's internal state
      */
     public void accept(String token) {
         compute(token);
     }

     // Take in an operator or operand
     // If operand, push it onto stack
     // If operator, pop its operands and push the result onto the stack
     private void compute(String token) {
             if (token.matches("\\d+")) { // non-empty sequence of digits
             stack.push(Integer.parseInt(token));
         } else if ("+".equals(token)) {
             int addend2 = stack.pop();
             int addend1 = stack.pop();
             stack.push(addend2 + addend1);
         } else if ("-".equals(token)) {
             int subtrahend = stack.pop();
             int minuend = stack.pop();
```

```
            stack.push(minuend - subtrahend);
        } else if ("*".equals(token)) {
            int factor2 = stack.pop();
            int factor1 = stack.pop();
            stack.push(factor1 * factor2);
        } else if ("/".equals(token)) {
            int divisor = stack.pop();
            int dividend = stack.pop();
            stack.push(dividend / divisor);
        } else {
            throw new IllegalArgumentException(
                                "Unrecognized token " + token);
        }
    }

    /**
     * Say whether the calculator has a result.
     * @return true if the calculator has consumed all accepted tokens and
     *     has a result available, false otherwise.
     */
    public boolean hasResult() {
        return stack.size() == 1;
    }

    /**
     * Return the calculator's result. If there are unconsumed tokens
     *     other than the result, throw an exception.
     * Client should call hasResult to make sure it is safe
     *     to call this method.
     * @return the calculator's result
     * @throws IllegalStateException if there are unconsumed tokens
     *     other than the result
     */
    public int getResult() throws IllegalStateException {
        if (stack.size() != 1) {
            throw new IllegalStateException("No result at this time");
        }
        return stack.pop();
    }

    /**
     * Reset the calculator to its initial state
     * @modifies this
     * @effects clears the internal state of this
     */
    public void clear() {
        while(!(stack.empty())) {
            stack.pop();
        }
    }
}
```