10.62

```
/**
 * Person class, implements Comparable to be used with TreeSet.
 */

public class Person implements Comparable<Person>
{
        private int age;

        /**
         * Person constructor.
         * @param age Person's age.
         */
        public Person(int age) {
                this.age = age;
        }

        /**
         * Compares the Person's age to another's.
         * @param p Other operand.
         * @return Difference between this person's age and another's.
         */
        public int compareTo(Person p){
                return age - p.getAge();
        }

        /**
         * Get age.
         * @return The person's age.
         */
        public int getAge() {
                return age;
        }
}
```

RPN

Stack

```
/**
 * Stack interface
 * @author lemming
 * @version shoop'
 */
```

```
public interface Stack<T> {
        public void push(T o);
        public T pop();
        public T top();
        public int size();
        public boolean isEmpty();
}

/**
 * A singly linked stack.
 *
 * @author slemming
 * @version 0x2A? OR IS IT!?
 */
public class MyStack<T> implements Stack<T> {
    private StackElement<T> first;    // Top element in stack.
    private int size;                 // Number of elements in stack.

    /**
     * A stack element.
     */
    private static class StackElement<T>
    {
        public T data;
        public StackElement<T> next;

        public StackElement(T data) {
            this.data = data;
            this.next = null;
        }
    }

    /**
     * Creates an empty stack.
     */
    public MyStack() {
    }

    /**
     * Inserts the given element at the top of the stack.
     */
    public void push(T element) {
        StackElement<T> foo = new StackElement<T>(element);
        foo.next = first;

        first = foo;
        size++;
}

/**
 * Removes and returns the first element on this stack.
 */
public T pop() {
        if(first == null)
                throw new java.util.EmptyStackException();

        T value = first.data;
        first = first.next;
        size––;

        return value;
}

/**
 * Gives top element of the stack without removing it.
 */
public T top() {
        if(first == null)
                        throw new java.util.EmptyStackException();
        return first.data;
}

/**
 * Removes all of the elements from this stack.
 */
public void clear() {
        first = null;
        size = 0;
}

/**
 * Returns the number of elements in this list.
 */
public int size() {
        return size;
}

/**
 * Returns <code>true</code> if this list contains no elements.
 */
public boolean isEmpty() {
        return size == 0;
        }
    }

import junit.framework.TestCase;

/**
 * Test of MyStack
 * @author lemming
 */
public class MyStackTest extends TestCase {
        Stack<Integer> stack;
        /**
         * Stack constructor
         * @param name
         */
        public MyStackTest(String name) {
                super(name);
        }

        /* (non–Javadoc)
         * @see junit.framework.TestCase#setUp()
         */
        protected void setUp() throws Exception {
                stack = new MyStack<Integer>();
                super.setUp();
        }

        /**
         * Tests pushing of elements on to the stack.
         */
        public void testPush() {
                stack.push(5);
                assertEquals(stack.top(), new Integer(5));
                stack.push(10);
                assertEquals(stack.top(), new Integer(10));
        }
        /**
         * Pops element off top of the stack. Also tests for the exception.
         */
        public void testPop() {
                stack.push(5);
                stack.push(10);
                assertEquals(stack.pop(), new Integer(10));
                assertEquals(stack.pop(), new Integer(5));
                try {
                        stack.pop();
                }
                catch (java.util.EmptyStackException e) {
                        return;
                }
                fail("Expected EmptyStackException");
        }

        /**
         * Tests if its giving valid size
         */
        public void testSize() {
                assertEquals(stack.size(), 0);
                stack.push(5);
                stack.push(10);
                assertEquals(stack.size(), 2);
                stack.pop();
                assertEquals(stack.size(), 1);
        }

        /**
         * Tests if its isEmpty method is valid..
         */
        public void testIsEmpty() {
                assertEquals(stack.isEmpty(), true);
                stack.push(5);
                stack.push(10);
                assertEquals(stack.isEmpty(), false);
                stack.pop();
                assertEquals(stack.isEmpty(), false);
                stack.pop();
                assertEquals(stack.isEmpty(), true);
        }

        /* (non-Javadoc)
         * @see junit.framework.TestCase#tearDown()
         */
        protected void tearDown() throws Exception {
                super.tearDown();
        }

}
```

PostFixCalculator

```
/**
 * Postfix calculator. Used to evaluate postfix strings.
 *
 * @author le ming
 * @version ”0 FF x”
 */
public class PostFixCalculator
{
        /**
         * Evalutes a postfix expression.
         * @param expression Postfix expression to be evaluated.
         * @return Evaluated postfix expression as an int.
         * @throws EmptyStackException Thrown if less than two operands are available upon o
         * @throws NumberFormatException Occurs when something besides an int and valid op
         * @throws IllegalArgumentException Occurs when there is more than one operand left o
         */
        public static int EvaluatePostFix(String expression)
        {
                Stack<Integer> stack = new MyStack<Integer>();
                if(expression.length() != 0) {
                        for(String token : expression.split(" "))
                        {
                                if(isInt(token))
                                {
                                        stack.push(Integer.valueOf(token));
                                        continue;
                                }

                                if(isOperator(token.charAt(0)))
                                {
                                        char operator = token.charAt(0);

                                        int right = stack.pop(), left = stack.pop();

                                        int value = 0;

                                        switch(operator)
                                        {
                                                case '+':
                                                        value=left+right;
                                                        break;
                                                case '−':
                                                        value=left−right;
                                                        break;
                                                case '*':
                                                        value=left*right;
                                                        break;
                                                case '/':
```

```
                            value=left/right;
                            break;
                        default:
                            break;
                    }
                    stack.push(value);

                    continue;
                }
                throw new NumberFormatException(); // Each token sh
            }
        }

        if(stack.size() != 1)
            throw new IllegalArgumentException(); // Still operands left, inval

        return stack.pop();
    }

    /**
     * Determines a string's validity as an integer.
     * @param s Input string.
     * @return Returns true if the string is a valid int.
     */
    private static boolean isInt(String s)
    {
        try{
            Integer.parseInt(s);
            return true;
        }
        catch (NumberFormatException e){
            return false;
        }
    }

    /**
     * Determines if a character is a valid operator.
     * @param c Input character.
     * @return Returns true if the character is a valid operator.
     */
    private static boolean isOperator(char c)
    {
        return c == '+' || c == '-' || c == '*' || c == '/';
    }
}
```

```java
import junit.framework.TestCase;

/**
 * Tests PostFixCalculator
 * @author lemming
 */
public class PostFixCalculatorTest extends TestCase {

        /**
         * @param name
         */
        public PostFixCalculatorTest(String name) {
                super(name);
        }

        /* (non-Javadoc)
         * @see junit.framework.TestCase#setUp()
         */
        protected void setUp() throws Exception {
                super.setUp();
        }

        /* (non-Javadoc)
         * @see junit.framework.TestCase#tearDown()
         */
        protected void tearDown() throws Exception {
                super.tearDown();
        }

        public void testValidPostFix(){
                assertEquals(PostFixCalculator.EvaluatePostFix("-2 2 +"), 0);
                assertEquals(PostFixCalculator.EvaluatePostFix("2 5 +"), 7);
                assertEquals(PostFixCalculator.EvaluatePostFix("1 2 + 3 *"), 9);
                assertEquals(PostFixCalculator.EvaluatePostFix("1 2 - 3 4 + *"), -7);
                assertEquals(PostFixCalculator.EvaluatePostFix("1 2 + 3 * 4 - 5 /"), 1);
                assertEquals(PostFixCalculator.EvaluatePostFix("2 3 4 5 + - *"), -12);
        }

        /**
         * Tests error handling for missing operands.
         */
        public void testEmptyStack(){
                try { // Stack underflows
                        PostFixCalculator.EvaluatePostFix("2 4 5 + - *");
                }
        catch (java.util.EmptyStackException e) {
            return;
        }

        fail("Missing EmptyStackException");
    }

    /**
     * Tests for handling of empty expression and excess operands.
     */
    public void testIllegalArgument(){
        try { // Invalid arguments
            PostFixCalculator.EvaluatePostFix("2 2 2 +");
        }

        catch (IllegalArgumentException e) {
            return;
        }

        fail("Missing IllegalArgumentException");
    }

    /**
     * Tests error handling of non-operand and non-operators.
     */
    public void testNumberFormat(){
        try { // Invalid arguments
            PostFixCalculator.EvaluatePostFix("2 2 + z");
        }

        catch (NumberFormatException e) {
            return;
        }

        fail("Missing NumberFormatException");
    }
}
```