UC Berkeley – Computer Science **BETA MIDTERM SOLUTIONS**
CS61B: Data Structures (Produced by TAs delirious from grading)

Midterm #1, Spring 2015

This test has 9 questions worth a total of 35 points. The exam is closed book, except that you are allowed to use a one page written cheat sheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. Write the statement out below, and sign once you're done with the exam.

"I have neither given nor received any assistance in the taking of this exam."

I have neither giving nor received any assassins in the talking of this exam.

Amundo Cow

	Score		Score
0	/0.5	5	/3
1	/1.5	6	/0
2	/3	7	/8
3	/5	8	/5
4	/7	9	/2
Sub 1	/17	Sub 2	/18

Total	36 /35

Name: Amanda Chow

Three-letter Login ID: ftw
ID of Person to Left: omg
ID of Person to Right: lul

Exam Room (circle):
Wheeler Pimentel

Tips:

- Happy Lunar New Year!
- There are a lot of problems on this exam. Work through the ones with which you are comfortable first. Do not get overly captivated by interesting design issues or complex corner cases you're not sure about.
- Not all information provided in a problem may be useful.
- All given code on this exam should compile. All code has been compiled and executed before printing, but in the unlikely event that we do happen to catch any bugs in the exam, we'll announce a fix. The correct answer is not 'does not compile.'
- Don't panic! Recall that we shoot for around a 60% median. You should not expect to be able to do all of the problems on this exam.

Optional. Mark along the line to show your feelings on the spectrum between ☹ and ☺.

Before exam: [☹____hungry_____☺].
After exam: [☹_____hungry_____☺].

0. So it begins. (0.5 points). Write your name and ID on the front page. Circle the exam room. Write the IDs of your neighbors. Write the given statement. Sign when you're done with the exam. Write your login in the corner of every page. Enjoy your free half point.

1. **IntLists (1.5 points).**

```
public static void main(String[] args) {
    IntList a = new IntList(5, null);
    System.out.println(a.head);                          _5__
    IntList b = new IntList(9, null);
    IntList c = new IntList(1, new IntList(7, b));
    a.tail = c.tail;
    a.tail.tail = b;
    b.tail = c.tail;
    IntList d = new IntList(9001, b.tail.tail);

    System.out.println(d.tail.tail.tail.head);           _9__
    System.out.println(a.tail.head);                     _7__

    c.tail.tail = c.tail;

    System.out.println(a.tail.tail.tail.tail.head);      _7__
}
```

In the four blanks beside the print statements above, write the result of the print statement. The answer to the first one is already provided for you. **Show your work at the bottom of this page (it may be considered for partial credit).** For your reference, the definition of the `IntList` class is given below:

```
public class IntList {
    private int head;
    private IntList tail;

    public IntList (int i, IntList n){
        head = i;
        tail = n;
    }
}
```

2. **The Ole Flitcharoo (3 points).**

```
public class Foo {
        public int x, y;
        public Foo (int x, int y) {
            this.x = x;
            this.y = y;
        }

    public static void switcheroo (Foo a, Foo b) {
        Foo temp = a;
        a = b;
        b = temp;
    }

    public static void fliperoo (Foo a, Foo b) {
        Foo temp = new Foo(a.x, a.y);
        a.x = b.x;
        a.y = b.y;
        b.x = temp.x;
        b.y = temp.y;
    }

    public static void swaperoo (Foo a, Foo b) {
        Foo temp = a;
        a.x = b.x;
        a.y = b.y;
        b.x = temp.x;
        b.y = temp.y;
    }

    public static void main(String[] args) {
        Foo foobar = new Foo(10, 20);
        Foo baz = new Foo(30, 40);

        switcheroo(foobar, baz);

        fliperoo(foobar, baz);

        swaperoo(foobar, baz);
     }
}
```

foobar		baz	
x	y	x	y
10	20	30	40
10	20	30	40
30	40	10	20
10	20	10	20

Fill in the contents of the boxes above with the contents of the foobar and baz variables at the indicated points in time. The first row has been completed for you.

3. **IntList Manipulation (5 points).**

You are given the following class implementation for an `IntList`:

```
public class IntList {
    public int head;
    public IntList tail;

    public IntList(int v, IntList s) {
        head = v;
        tail = s;
    }

    /* Non-destructive. Assume skip > 0. */
    public static IntList skipBy(int skip, IntList s) {
        if (s == null) {
            return null;
        }
        else {
            IntList p = s;
            int count = skip;
            while (p != null && count > 0) {
                p = p.tail;
                count--;
            }
            return new IntList(s.head, skipBy(skip, p));
        }
    }
}
```

Fill in the method `skipBy` to return a new `IntList` that contains the elements of the list `s` if we skipped by `skip` number of nodes. For example, if `s` was the following: [1 2 3 4 5 6 7], then calling `IntList.skipBy(3, s)` returns [1 4 7] and `IntList.skipBy(9, s)` returns [1]. Your implementation should be non-destructive (none of the original IntList objects should change). You may assume that `skip` > 0, and that the `IntList` has no cycles.

4. Debugging (7 points).

(a) 61B student Bilbo Gargomeal runs the following code while studying for the midterm and finds that the first print statement outputs "Chocolate". He fears the presence of evil spirits in his code.

```
public class IceCream {
    public static String flavor;

    public IceCream(String f) {
        flavor = f;
    }

    public void melt() {
        flavor = "melted " + flavor;
    }

    public static void main(String[] args) {
        IceCream vanilla = new IceCream("Vanilla");
        IceCream chocolate = new IceCream("Chocolate");
        System.out.println(vanilla.flavor);
        //chocolate.melt();
        //System.out.println(vanilla.flavor);
    }
}
```

Why is the print statement outputting "Chocolate" and not "Vanilla"? Give your answer in 10 words or less:

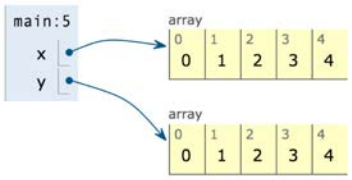
`flavor` is static, so instantiating `chocolate` changes it.

If we uncomment the two lines of code above, will the code compile? If so, what is the output of the print statement?

Yes. Prints `"melted Chocolate"`.

(b) Our hero Bilbo tries to execute the code below, and finds to his surprise that the two arrays are not considered equal. Consulting Head First Java, he reads that == returns true "if two references refer to the same object". He remembers from lecture that an array consists of a length and N simple containers (where N = length), and reasons (correctly) that this means that the underlying objects pointed to by x and y must be different. He then runs it through the visualizer and observes the figure to the right.

```
public static void main(String[] args) {
    int[] x = new int[]{0, 1, 2, 3, 4};
    int[] y = new int[]{0, 1, 2, 3, 4};
    System.out.println(x == y);
}
```



Given what Bilbo has learned while debugging his code, for each of the following, answer whether the code will print **true**, print **false**, or state that there is **not enough information**. Please use the three blanks provided to the right of each print statement. **Assume that the code compiles and executes without error.**

```
public static void main(String[] args) {
    int[] x = new int[]{0, 1, 2, 3, 4};
    int[] y = new int[]{0, 1, 2, 3, 4};
    y = someUnknownFunction(x, y);
    System.out.println(x == y);              not enough information
}
```

```
public static void main(String[] args) {
    int[] x = new int[]{0, 1, 2, 3, 4};
    int[] y = new int[]{0, 1, 2, 3, 4};
    anotherUnknownFunction(x, y);
    System.out.println(x == y);              _______false_______
}
```

```
public static void main(String[] args) {
    int[] x = new int[]{0, 1, 2, 3, 4};
    int[] y = new int[]{0, 1, 2, 3, 4};
    System.arraycopy(x, 0, y, 0, 5);
    System.out.println(x == y);              _______false_______
}
```

Explanation:

Recall that == tests for reference equality – that is, whether x and y references to the same object in memory, rather than whether their contents are the same. The first part could work with the following definition:

```
public static int[] someUnknownFunction(int[] x, int[] y) {return x;}
```

Since x and y are not reassigned in the latter two problems, x and y will always be different objects. Thus, neither part will print true.

(c) Bilbo tries the hardMode exercise for lecture 6 and comes up with the following, where a StringNode is defined exactly like an IntNode, but the item field is of type String:

```
/** SentinelSSList: Similar to hardMode exercise but with Strings.
 *  @author Bilbo Gargomeal
 */

public class SentinelSSList {
    private StringNode sent;

    public SentinelSSList() {
        sent = new StringNode(null, null);
    }

    public SentinelSSList(String x) {
        sent = new StringNode(x, null);
    }

    public void insertFront(String x) {
        sent.next = new StringNode(x, sent.next);
    }

    public String getFront() {
        if (sent.next == null) return null;
        return sent.next.item;
    }

    public void insertBack(String x) {
        StringNode p = sent;
        while (p.next != null) {
            p = p.next;
        }

        p.next = new StringNode(x, null);
    }
}
```

The code compiles fine but the autograder is giving unhelpful messages about why it isn't working. There is exactly one bug in this code. You want to help Bilbo but don't want to give away the answer. Provide a simple JUnit test below that will fail on Bilbo's code (but would pass on a correct list). For possible partial credit, also explain the bug.

```
@Test
public void testBilboList() {
    SentinelSList lst = new SentinelSSList("Dennis smells good");
    assertEquals("Dennis smells good", lst.getFront());
}
```

Explanation:

The error was in the second constructor. It takes in a `String x` and sets it for the sentinel node. However, correct behavior would be to create a sentinel node, and then another node following sentinel which contains that `String x`.

Recall that the front value in a `SentinelList` is not the one in the sentinel, but rather the one in the node afterwards. Thus, `insertFront` and `getFront` are correct. There are also no problems with the empty constructor and `insertBack`.

5. **Bits (3 points).** What does the following function do? Give a simple description (ten words or less) of the return value in terms of the argument.

```
public static int mystery(int a) {
    int b = 0;
    for (int i = 0; i < 32; i++) {
        b = b << 1;
        b = b | (a & 1);
        a = a >>> 1; // This is a logical shift.
    }
    return b;
}
```

Returns the int that has the reversed binary representation of a.

An example of a reverse:

```
a: 00101....01
        |
        v
b: 10....10100
```

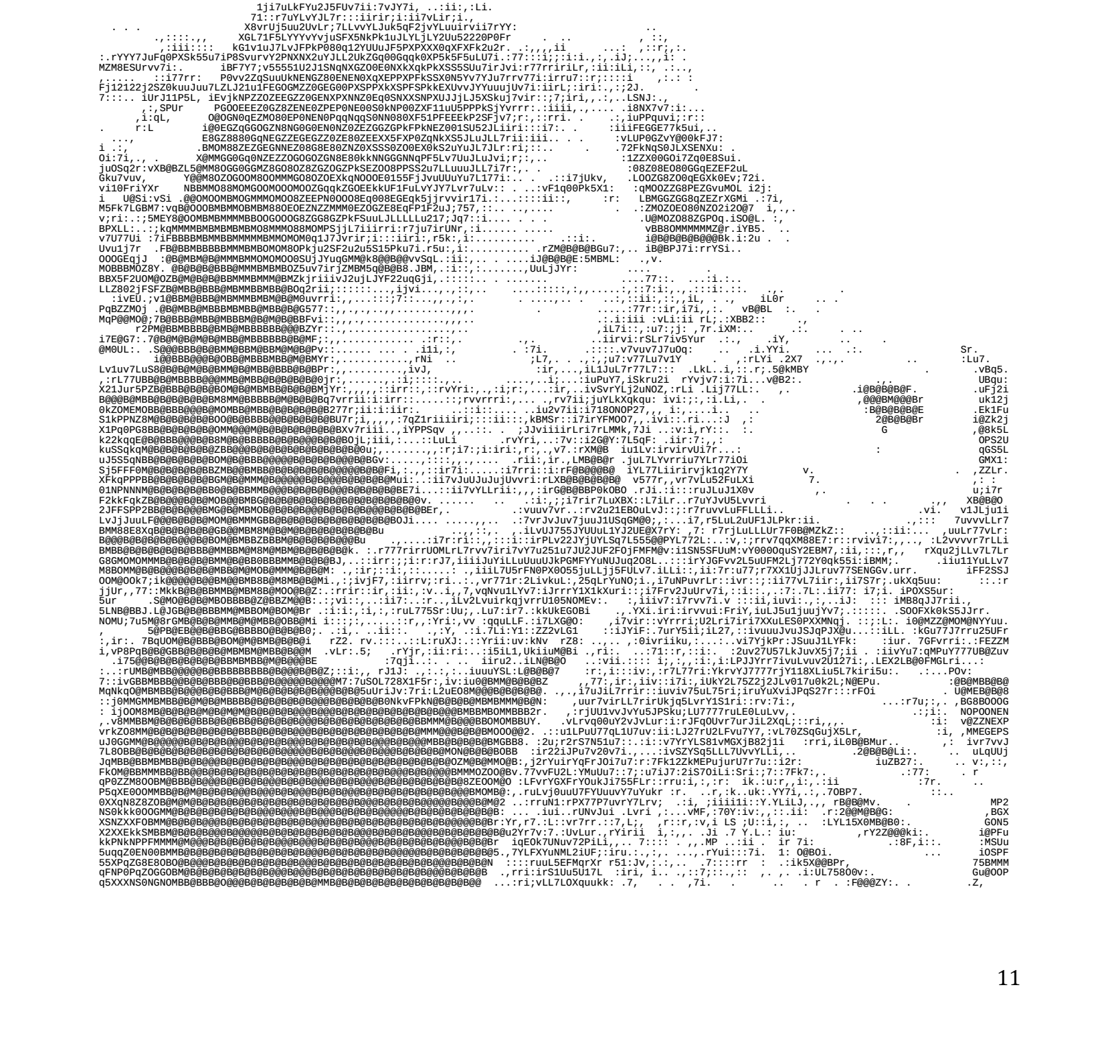
Some answers that received partial credit: "flipping" the bits (since this could be interpreted as changing a 0-bit to a 1-bit and vice versa), "inverting" the bits (for similar misinterpretations), "rotating" the bits (since this could be interpreted as a circular shift of the bits), and "copying the bits from right to left" (this was not specific enough, as it could be interpreted as merely copying the rightmost bit of a into the rightmost position of b, etc.). In most cases, if we saw some scratch work which showed an example which demonstrated that the student meant "reverse" while using one of the above descriptions, full marks were awarded for the problem.

6. **PNH (0 points).** This game's hardest difficulty was unlocked with the code LLLRRRLR.

Dance Dance Revolution 3rd Mix (must specify mix number)

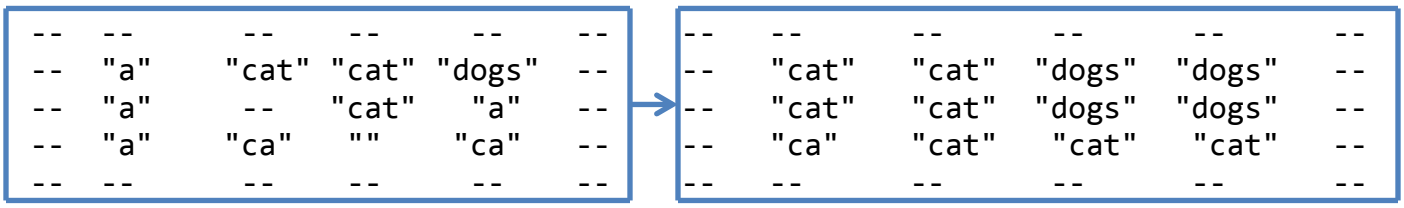
This area is a designated fun zone. Perhaps you would like to draw something in the space? One possibility is some sort of mythical creature and a volcano.

#Amanda-volcano



7. **Higher Order Functions and Arrays (8 Points).**

For this problem, you'll develop a method `step()` that takes a 2D array of strings and replaces every string with its longest neighbor, EXCEPT the edges, which you will assume are always null and should never change. The neighbors of a string are the eight strings surrounding it (including diagonals). For example, if given the 2D array on the left, `step` will return the array on the right. We use two dashes `--` to represent a null pointer. The `""` in row 4, column 3 is an empty string.



For the sake of allowing easy customization, your program will compare strings using an object that implements the `NullSafeStringComparator` interface defined below.

```
public interface NullSafeStringComparator {
    /** Returns a negative number if s1 is 'less than' s2, 0 if 'equal',
      * and a positive number otherwise. Null is considered less than
      * any String. If both inputs are null, return 0. */
    public int compare(String s1, String s2);
}
```

(a) Write a new class `LengthComparator` that implements `NullSafeStringComparator`. The `LengthComparator` should compare strings based on their lengths. The length of a string `s` can be retrieved using `s.length()`. Do not provide a constructor, as it is unnecessary.

```
public class LengthComparator implements NullSafeStringComparator {
    public int compare(String s1, String s2) {
        if ((s1 == null) && (s2 == null)) return 0;
        if (s1 == null) return -1;
        if (s2 == null) return 1;
        return s1.length() - s2.length();
    }
}
```

(b) Complete the helper function `max`, which returns the maximum string of a 1D array using the `StringComparator sc` to judge the string. **You can do this part even if you skipped part a!**

```
public static String max(String[] a, NullSafeStringComparator sc) {
    String maxStr = a[0];
    for(int i = 0; i < a.length; i += 1) {
        if (sc.compare(a[i], maxStr) > 0){
            maxStr = a[i];
        }
    }
    return maxStr;
}
```

Valid modifications include: initializing `i` to 1, swapping the order of `a[i]` and `maxStr` in the comparator then checking that it was less than 0, and using `>=` or `<=` instead of `>` or `<`.

(c) Complete the `step` function so that it completes the task described on the previous page. You may assume that every row has the same number of columns, that the size is at least 3x3, and that the edges are all null (as in the figure on the previous page). You may **not** assume that the number of rows is equal to the number of columns. **You may not add any semicolons. You may not use the ++ or -- operators. Use only the blanks provided. You can complete this part even if you skipped parts a and b.**

```
public static String[][] step(String[][] arr) {
    String[][] stepped = new String[arr.length][arr[0].length];
    for (int i = 1; i < arr.length - 1; i += 1) {
        for (int j = 1; j < arr[0].length - 1; j += 1) {
            String[] temp = new String[9];
           // temp holds all the neighbors + itself: there will
            //  be exactly 8 neighbors + self
            int count = 0;
            for (int k = -1; k <= 1; k += 1) {
                for (int m = -1; m <= 1; m += 1) {
                    temp[count] = arr[i+k][j+m];
                     // Store the all the neighbors
                    count += 1;
                }
            }
            stepped[i][j] = max(temp, new LengthComparator());
           // Here we need to construct a new LengthComparator
            // every time, since it's not static.
        }
    }
    return stepped;
}
```

Alternate accepted solutions involved assigning values in temp using a polynomial involving k and m that counted from 0 to 8 (e.g. 3k + m + 4).

8. A Robot Renegade Cop (5 Points).

```
public class Robot {
  public int energy = 0;
  public String className = "Robot";
  public void enervate(Robot r) {r.energy -= 5;}
  public void reportEnergy() {System.out.println(energy);}
  public void reportName() {System.out.println(className);}
}
public interface Police {
  public void detain();
}
public class Robocop extends Robot implements Police {
  public String className = "Robocop";
  @Override public void detain() { System.out.println("halt. citizen."); }
  @Override public void enervate(Robot r) {r.energy -= 20;}
  @Override public void reportEnergy() {System.out.println(energy);}
  @Override public void reportName() {System.out.println(className);}
}
```

For each line in the RobotLauncher class below, fill in the blanks. For blanks (right hand side of page), you should write out the results of the print statement on that line. If a print statement on a line will not compile, write INVALID in the blank. For the two assignment statements (lines 6 and 9), write a valid cast if required. If a cast is not required (i.e. the line will compile just fine), leave the cast blank.

Points were deducted for any type casts because none were required.

```
1: public class RobotLauncher {
2:   public static void main(String[] args) {
3:   Robocop rCop = new Robocop();
4:   rCop.reportEnergy();                                          4:_____0__
5:   rCop.detain();                                               5:halt. citizen.
6:   Robot r =  (_______) rCop;
7:   r.reportEnergy();                                           7:_____0__
8:   r.detain();                                                  8: INVALID
9:   Police p = (_______) rCop;
10:  p.reportEnergy();                                          10: INVALID
11:  p.detain();                                                11: halt. citizen.
12:  r.enervate(r); rCop.enervate(r); rCop.reportEnergy();      12:____-40___
13:  r.energy = 100;
14:  r.enervate(rCop); rCop.enervate(rCop); r.reportEnergy();14: 60
15:  r.reportName();                                            15: Robocop
16:  rCop.reportName();                                         16: Robocop
17:  rCop.className = "ketchup friend";
18:  r.reportName();                                            18: ketchup friend
19:  }
20:} // Other than (maybe) print statements and casts, code compiles.
```

9. Static Shock (2 Points).

```
public class Shock {
    public static int bang;
    public static Shock baby;
    public Shock() {
       this.bang = 100;
    }
    public Shock (int num) {
        this.bang = num;
        baby = starter();
        this.bang += num;
    }
    public static Shock starter() {
        Shock gear = new Shock();
        return gear;
    }
    public static void shrink(Shock statik) {
        statik.bang -= 1;
    }
    public static void main(String[] args) {
        Shock gear = new Shock(200);
        System.out.println(gear.bang);           300
        shrink(gear);
        shrink(starter());
        System.out.println(gear.bang);           99
    }
}
```

Observe everything is `static`. Then, there will only ever be one `bang` variable per execution of the program. Then we only need to track changes to `bang`.