CSE 142, Spring 2010
Final Exam Key

1. Array Mystery

Expression	Final Contents of Array
`int[] a1 = {42, 99, 42};` `arrayMystery(a1);`	`{42, 84, 42}`
`int[] a2 = {6, 8, 4, 2};` `arrayMystery(a2);`	`{6, 10, 12, 2}`
`int[] a3 = {7, 7, 20, 8, 1};` `arrayMystery(a3);`	`{7, 7, 15, 16, 1}`
`int[] a4 = {4, 5, 3, 2, 1, 0};` `arrayMystery(a4);`	`{4, 7, 9, 10, 10, 0}`
`int[] a5 = {6, 0, -1, 80, 5, 0, -3};` `arrayMystery(a5);`	`{6, 5, -1, 4, 4, 1, -3}`

3. Reference Semantics Mystery

```
11,22 [33, 44] 55
44,22 [44, 77] 0
44,22 [44, 77] 55
55,22 [88, 77] 0
55,22 [88, 77] 55
```

3. Inheritance Mystery

```
Leela2 Fry2 Bender2 Leela1
We're doomed!
Leela1

Bender2 Bender1
We're doomed!
Bender1

Bender2 Farnsworth1
Good news everyone!
Farnsworth1

Fry2 Bender2 Bender1
We're doomed!
Bender1
```

4. File Processing

```
// fencepost solution (a bit long)
public static int mostCommonNames(Scanner input) {
    int totalCount = 0;
    while (input.hasNextLine()) {
        String line = input.nextLine();

        Scanner words = new Scanner(line);

        String mostCommon = words.next();
        totalCount++;

        int mostCount = 1;
        String current = mostCommon;
        int currentCount = 1;

        while (words.hasNext()) {
            String next = words.next();

            if (next.equals(current)) {
                currentCount++;
                if (currentCount > mostCount) {
                    mostCount = currentCount;
                    mostCommon = current;
                }
            } else {
                current = next;
                currentCount = 1;
                totalCount++;
            }
        }
        System.out.println("Most common: " + mostCommon);
    }
    return totalCount;
}


// non-fencepost solution
public static int mostCommonNames(Scanner input) {
    int total = 0;
    while (input.hasNextLine()) {
        Scanner words = new Scanner(input.nextLine());
        String most = "";
        String prev = "";
        int max = 0;
        int count = 0;
        while (words.hasNext()) {
            String next = words.next();
            if (prev.length() == 0 || next.equals(prev)) {
                count++;
            } else {
                count = 1;
                total++;
            }
            if (count > max) {
                max = count;
                most = next;
            }
            prev = next;
        }
        System.out.println("Most common: " + most);
    }
    return total;
}
```

5. Array Programming

```
// "increase i by 2 each time" solution
public static void swapPairs(String[] a) {
    for (int i = 0; i < a.length - 1; i += 2) {
        String temp = a[i];
        a[i] = a[i + 1];
        a[i + 1] = temp;
    }
}


// "i % 2" solution
// (doesn't work if you use i%2==1 and [i-1]
public static void swapPairs(String[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        if (i % 2 == 0) {
            String temp = a[i];
            a[i] = a[i + 1];
            a[i + 1] = temp;
        }
    }
}


// "multiply indexes by 2" solution
public static void swapPairs(String[] a) {
    for (int i = 0; i < a.length / 2; i++) {
        String temp = a[i*2];
        a[i*2] = a[i*2 + 1];
        a[i*2 + 1] = temp;
    }
}
```

6. Array Programming

```
// "two nested loops, boolean found flag" solution
public static void banish(int[] a1, int[] a2) {
    for (int i = 0; i < a1.length; i++) {
        // see whether a1[i] is contained in a2
        boolean found = false;
        for (int j = 0; j < a2.length && !found; j++) {
            if (a1[i] == a2[j]) {
                found = true;
            }
        }
        if (found) {
            // shift all elements of a1 left by 1
            for (int j = i + 1; j < a1.length; j++) {
                a1[j - 1] = a1[j];
            }
            a1[a1.length - 1] = 0;
            i--;
        }
    }
}

// "triple nested loops to remove" solution
public static void banish(int[] a1, int[] a2) {
    for (int i = 0; i < a1.length; i++) {

        int found = 0;
        for (int j = 0; j < a2.length; j++) {
            if (a1[i] == a2[j]) {
                found++;

                for (int k = i + 1; k < a1.length; k++) {
                    a1[k - 1] = a1[k];
                }
                a1[a1.length - 1] = 0;
            }
        }

        if (found > 0) {
            i--;
        }
    }
}


// a2-based solution
public static void banish(int[] a1, int[] a2) {
   int count = 0;
   for (int j = 0; j < a2.length; j++) {
      for (int i = 0; i < a1.length; i++) {
         if (a1[i] == a2[j]) {
            for (int k = i; k < a1.length - 1; k++) {
               a1[k] = a1[k+1];
            }
            count++;
            a1[a1.length - count] = 0;
            i--;
         }
      }
   }
}
```

7. Critters

```
public class Raptor extends Critter {  // "stompCount drops from 20 to 0" solution
    private boolean east;
    private int stompCount;

    public Raptor(boolean startEast) {
        east = startEast;
        stompCount = 0;
    }

    public boolean eat() {
        stompCount = 20;
        east = !east;
        return true;
    }

    public Direction getMove() {
        if (stompCount > 0) {
            stompCount--;
            if (stompCount % 2 == 1) {
                return Direction.NORTH;
            } else {
                return Direction.SOUTH;
            }
        } else {
            if (east) {
                return Direction.EAST;
            } else {
                return Direction.WEST;
            }
        }
    }
}

public class Raptor extends Critter {  // "stompCount goes up from 0 to 20" solution
    private boolean walkEast;
    private int stompCount = 0;
    private boolean stomping = false;

    public Raptor(boolean walkEast) {
        this.walkEast = walkEast;
    }

    public boolean eat() {
        stompCount = 0;
        stomping = true;
        return true;
    }

    public Direction getMove() {
        if (stompCount >= 20) {
            stompCount = 0;
            stomping = false;
            if (walkEast == true) {
                walkEast = false;
            } else {
                walkEast = true;
            }
        }
        if (stomping) {
            if (stompCount % 2 == 0) {
                stompCount++;
                return Direction.NORTH;
            } else {
                stompCount++;
                return Direction.SOUTH;
            }
        } else {
            if (walkEast) {
                return Direction.EAST;
            } else {
                return Direction.WEST;
            }
        }
    }
}
```

8. Objects

```
public void transfer(BankAccount other, double amount) {
    if (amount > 0.00) {
        if (amount + 5.00 <= balance) {
            withdraw(amount + 5.00);
            other.deposit(amount);
        } else if (balance > 5.00) {
            other.deposit(balance - 5.00);
            withdraw(balance);
        }
    }
}
```

```
public void transfer(BankAccount other, double amount) {
    if (balance > 5 && amount > 0) {
        balance -= 5;
        if (balance < amount) {
            amount = balance;
        }

        other.deposit(amount);
        withdraw(amount);
    }
}
```

```
public void transfer(BankAccount other, double amount) {
    if (balance > 5 && amount > 0) {
        transactions++;
        other.transactions++;
        balance -= 5;
        if (balance >= amount) {
            other.balance += amount;
            balance -= amount;
        } else {
            other.balance += balance;
            balance = 0.0;
        }
    }
}
```