CSE 142, Autumn 2009
Final Exam Key

1. Array Mystery

Expression	Final Contents of Array
`int[] a1 = {42, 42};` `arrayMystery(a1);`	`{42, 42}`
`int[] a2 = {6, 2, 4};` `arrayMystery(a2);`	`{6, 8, 4}`
`int[] a3 = {7, 7, 3, 8, 2};` `arrayMystery(a3);`	`{7, 3, 8, 2, 2}`
`int[] a4 = {4, 2, 3, 1, 2, 5};` `arrayMystery(a4);`	`{4, 5, 3, 4, 7, 5}`
`int[] a5 = {6, 0, -1, 3, 5, 0, -3};` `arrayMystery(a5);`	`{6, 5, 9, 11, 6, 3, -3}`

3. Reference Semantics Mystery

```
11 [21] 30,41
10 [21] 30,41
11 [23] 31,42
10 [23] 31,42
```

3. Inheritance Mystery

```
Biggie a   JayZ a   Tupac b
Tupac b
Biggie

Tupac a
Tupac b
Tupac

JayZ a   Tupac b
Tupac b
Tupac

Biggie a   JayZ a   FiftyCent b
FiftyCent b
Biggie
```

4. File Processing (3 solutions shown)

```
// count up money as a double of dollars/cents; use printf at end
public static void countCoins(Scanner input) {
    double total = 0.0;
    while (input.hasNext()) {
        int count = input.nextInt();
        String coin = input.next().toLowerCase();
        if (coin.equals("nickels")) {
            count = count * 5;
        } else if (coin.equals("dimes")) {
            count = count * 10;
        } else if (coin.equals("quarters")) {
            count = count * 25;
        }
        total = total + (double) count / 100;
    }
    System.out.printf("Total money: $%.2f\n", total);
}

// count up money as a total number of cents; use / and % at end
public static void countCoins(Scanner input) {
    int totalCents = 0;
    while (input.hasNext()) {
        int count = input.nextInt();
        String coin = input.next().toLowerCase();
        if (coin.equals("nickels")) {
            totalCents += count * 5;
        } else if (coin.equals("dimes")) {
            totalCents += count * 10;
        } else if (coin.equals("quarters")) {
            totalCents += count * 25;
        }
    }

    int dollars = totalCents / 100;
    int cents = totalCents % 100;

    System.out.print("Total money: $" + dollars + ".");
    if (cents < 10) {
        System.out.print("0");
    }
    System.out.println(cents);
}

// implicit /100 in the multipliers
public static void countCoins(Scanner input) {
    double total = 0.0;
    while (input.hasNext()) {
        double count = input.nextInt();
        String coin = input.next().toLowerCase();

        if (coin.equals("pennies")) {
            count *= 0.01;
        } else if (coin.equals("nickels")) {
            count *= 0.05;
        } else if (coin.equals("dimes")) {
            count *= 0.10;
        } else {
            count *= 0.25;
        }
        total += count;
    }
    System.out.printf("Total money: $%.2f\n", total);
}
```

5. File Processing

```
public static void matchIndex(Scanner input) {
    int lines = 0;
    while (input.hasNextLine()) {
        String line1 = input.nextLine();
        String line2 = input.nextLine();
        lines += 2;

        System.out.print("lines " + (lines - 1) + " and " + lines + ":");

        // print any matches found
        boolean matchedAny = false;
        int length = Math.min(line1.length(), line2.length());
        for (int i = 0; i < length; i++) {
            if (line1.charAt(i) == line2.charAt(i)) {
                matchedAny = true;
                System.out.print(" " + i);
            }
        }

        if (!matchedAny) {
            System.out.print(" none");
        }
        System.out.println();
    }
}
```

6. Array Programming (3 solutions shown)

```
// concise solution w/Math.max
public static String[] longer(String[] a1, String[] a2) {
    String[] a3 = new String[Math.max(a1.length, a2.length)];
    for (int i = 0; i < a3.length; i++) {
        if (i >= a1.length || i >= a2.length) {
            // out of bounds case; must check this before accessing a1[i] or a2[i]
            a3[i] = "oops";
        } else if (a1[i].length() >= a2[i].length()) {
            a3[i] = a1[i];
        } else {
            a3[i] = a2[i];
        }
    }

    return a3;
}

// longer solution with if/else, altered test order
public static String[] longer(String[] a1, String[] a2) {
    int len = 0;
    if (a1.length >= a2.length) {
        len = a1.length;
    } else {
        len = a2.length;
    }

    String[] a3 = new String[len];
    for (int i = 0; i < len; i++) {
        if (i < a1.length && i < a2.length) {
            if (a1[i].length() >= a2[i].length()) {
                a3[i] = a1[i];
            } else {
                a3[i] = a2[i];
            }
        } else {
            a3[i] = "oops";
        }
    }

    return a3;
}

// "fill with oops" solution
public static String[] longer(String[] a1, String[] a2) {
    String[] a3 = new String[Math.max(a1.length, a2.length)];
    Arrays.fill(a3, "oops");
    for (int i = 0; i < Math.min(a1.length, a2.length); i++) {
        if (a1[i].length() >= a2[i].length()) {
            a3[i] = a1[i];
        } else {
            a3[i] = a2[i];
        }
    }
    return a3;
}
```

7. Array Programming (9 solutions shown)

```
// swap evens to beginning
public static void evenBeforeOdd(int[] a) {
    int fixed = 0;
    for (int i = 0; i < a.length; i++) {
        if (a[i] % 2 == 0) {
            int temp = a[fixed];  // swap
            a[fixed] = a[i];
            a[i] = temp;
            fixed++;
        }
    }
}

// walk j counter from back to i
public static void evenBeforeOdd(int[] a) {
    int j = a.length - 1;
    for (int i = 0; i < j; i++) {
        while (i < j && a[j] % 2 != 0) {
            j--;
        }

        int temp = a[j];  // swap
        a[j] = a[i];
        a[i] = temp;
    }
}

// invent bubble sort
public static void evenBeforeOdd(int[] a) {
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a.length - 1; j++) {
            if (a[j] % 2 != 0 && a[j + 1] % 2 == 0) {
                int temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
}

// modified selection sort
public static void evenBeforeOdd(int[] a) {
    for (int i = 0; i < a.length; i++) {
        for (int j = i + 1; j < a.length; j++) {
            if (a[i] % 2 != 0 && a[j] % 2 == 0) {
                int temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
}

// selection sort #2
public static void evenBeforeOdd(int[] a) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] % 2 != 0) {
            // then i is out of place;  look for another index
            // k that represents the 2nd index to swap i with
            int k = -1;

            for (int j = i + 1; j < a.length; j++) {
                if (a[j] % 2 == 0) {
                    k = j;
                }
            }

            if (k >= 0) {
                int temp = a[i];
                a[i] = a[k];
                a[k] = temp;
            }
        }
    }
}
```

```
// partition (invent partial quicksort)
public static void evenBeforeOdd(int[] a) {
    int even = 0;
    int odd = a.length - 1;
    while (even < odd) {
        while (even < a.length && a[even] % 2 == 0) {
            even++;
        }
        while (odd >= 0 && a[odd] % 2 != 0) {
            odd--;
        }

        if (even < odd) {
            int temp = a[even];  // swap
            a[even] = a[odd];
            a[odd] = temp;
        }
    }
}

// shift evens to front
public static void evenBeforeOdd(int[] a) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] % 2 == 0) {
            int even = a[i];

            // found an even; shift others over and put at index 0
            for (int j = i; j > 0; j--) {
                a[j] = a[j - 1];
            }
            a[0] = even;
        }
    }
}

// walk from ends and swap
public static void evenBeforeOdd(int[] a) {
    for (int i = a.length - 1; i >= 0; i--) {
        if (a[i] % 2 == 0) {
            // found an even; look for an odd and swap
            for (int j = 0; j < i; j++) {
                if (a[j] % 2 != 0) {
                    int temp = a[i];  // swap
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }
    }
}

// evil scanner/string solution (should have been banned)
public static void evenBeforeOdd(int[] a) {
    String evens = "";
    String odds = "";
    for (int i = a.length - 1; i >= 0; i--) {
        if (a[i] % 2 == 0) {
            evens += a[i] + " ";
        } else {
            odds += a[i] + " ";
        }
    }

    int i = 0;
    Scanner evenScan = new Scanner(evens);
    while (evenScan.hasNextInt()) {
        a[i] = evenScan.nextInt();
        i++;
    }
    Scanner oddScan = new Scanner(odds);
    while (oddScan.hasNextInt()) {
        a[i] = oddScan.nextInt();
        i++;
    }
}
```

8. Critters (4 solutions shown)

```
public class Tigger extends Critter {    // all-ints solution with one move counter
    private int moves;
    private int max;
    private int initialMax;

    public Tigger(int initialMax) {
        this.initialMax = initialMax;
        max = initialMax;
        moves = 0;
    }

    public boolean eat() {
        max = initialMax;             // reset movement
        moves = 0;
        return true;
    }

    public Direction getMove() {
        moves++;
        if (moves > 2 * max) {        // reached bottom of bounce; start next bounce
            moves = 1;
            max++;
        }

        if (moves <= max) {
            return Direction.NORTH;   // on the way up
        } else {
            return Direction.SOUTH;   // on the way down
        }
    }
}

// boolean flag for movement solution
public class Tigger extends Critter {
    private int moves;
    private int max;
    private int initialMax;
    private boolean goingUp;

    public Tigger(int x) {
        this.initialMax = x;
        eat();   // to initialize the other fields
    }

    public boolean eat() {
        max = initialMax;
        moves = 0;
        goingUp = true;
        return true;
    }

    public Direction getMove() {
        moves++;
        if (moves > max) {
            moves = 1;
            if (!goingUp) {
                max++;
            }
            goingUp = !goingUp;
        }

        if (goingUp) {
            return Direction.NORTH;
        } else {
            return Direction.SOUTH;
        }
    }
}
```

```
// all-ints solution with two move counters
public class Tigger extends Critter {
    private int northMoves;   // initialized to 0
    private int southMoves;   // initialized to 0
    private int max;
    private int initialMax;

    public Tigger(int initialMax) {
        this.initialMax = initialMax;
        max = initialMax;
    }

    public boolean eat() {
        max = initialMax;             // reset movement
        northMoves = 0;
        southMoves = 0;
        return true;
    }

    public Direction getMove() {
        if (northMoves < max) {
            northMoves++;
            return Direction.NORTH;   // on the way up
        } else if (southMoves < max) {
            southMoves++;
            return Direction.SOUTH;   // on the way down
        } else {
            northMoves = 1;
            southMoves = 0;
            max++;
            return Direction.NORTH;   // start the next bounce
        }
    }
}

// boolean flag for eating solution
public class Tigger extends Critter {
    private int moves;
    private int max;
    private int initialMax;
    private boolean ate;

    public Tigger(int initialMax) {
        this.initialMax = initialMax;
        max = initialMax;
        moves = 0;
        ate = false;
    }

    public boolean eat() {
        ate = true;
        return true;
    }

    public Direction getMove() {
        if (ate == true) {
            max = initialMax;         // reset movement
            moves = 0;
        }

        moves++;
        if (moves > 2 * max) {        // reached bottom of bounce; start next bounce
            moves = 1;
            max++;
        }

        if (moves <= max) {
            return Direction.NORTH;   // on the way up
        } else {
            return Direction.SOUTH;   // on the way down
        }
    }
}
```

9. Objects (8 solutions shown)

```
// AM/PM based zen-like solution
public boolean isWorkTime() {
    if (amPm.equals("AM")) {
        return hour >= 9 && hour <= 11;
    } else {  // PM
        return (1 <= hour && hour <= 4) || hour == 12 || (hour == 5 && minute == 0);
    }
}

// separate out the false cases first
public boolean isWorkTime() {
    if ((1 <= hour && hour <= 8 || hour == 12) && amPm.equals("AM")) {
        return false;   // too early in the morning
    } else if ((hour == 5 && minute > 0 || 6 <= hour && hour <= 11) && amPm.equals("PM")) {
        return false;   // too late in the evening
    } else {
        return true;
    }
}

// separate out the true cases
public boolean isWorkTime() {
    if (amPm.equals("AM") && hour >= 9 && hour <= 11) {
        return true;    //  9:00 AM - 11:59 AM
    } else if (amPm.equals("PM") && hour == 12) {
        return true;    // 12:00 PM - 12:59 PM
    } else if (amPm.equals("PM") && hour <= 4) {
        return true;    //  1:00 PM -  4:59 PM
    } else if (amPm.equals("PM") && hour == 5 && minute == 0) {
        return true;    //  5:00 PM
    } else {
        return false;
    }
}

// ugly style (not very "zen") but it works
public boolean isWorkTime() {
    if (hour == 1 || hour == 2 || hour == 3 || hour == 4) {
        if (amPm.equals("PM")) {
            return true;
        } else {
            return false;
        }
    } else if (hour == 5) {
        if (minute == 0 && amPm.equals("PM")) {
            return true;
        } else {
            return false;
        }
    } else if (hour == 6 || hour == 7 || hour == 8) {
        return false;
    } else if (hour == 9 || hour == 10 || hour == 11) {
        if (amPm.equals("AM")) {
            return true;
        } else {
            return false;
        }
    } else {  // hour == 12
        if (amPm.equals("PM")) {
            return true;
        } else {
            return false;
        }
    }
}

// more "zen-like" version of previous solution
public boolean isWorkTime() {
    if (hour <= 4) {
        return amPm.equals("PM");
    } else if (hour == 5) {
        return minute == 0 && amPm.equals("PM");
    } else if (hour <= 8) {
        return false;
    } else if (hour <= 11) {
        return amPm.equals("AM");
    } else {  // hour == 12
        return amPm.equals("PM");
    }
}
```

```
// "absolute time" solution #1
public boolean isWorkTime() {
    int totalMins = hour % 12 * 60 + minute;
    if (amPm.equals("PM")) {
        totalMins += 12 * 60;
    }
    return 9 * 60 <= totalMins && totalMins <= 17 * 60;
}

// absolute minutes solution #2
public boolean isWorkTime() {
    int absTime = getMinute();
    if (getHour() < 12) {
        absTime += getHour() * 60;
    }

    if (getAmPm().equals("AM")) {
        return absTime >= 540;   // 9:00 == 9 * 60 == 540
    } else {
        return absTime <= 300;   // 5:00 == 5 * 60 == 300
    }
}

// take advantage of a temporary ClockTime object
public boolean isWorkTime() {
    ClockTime temp = new ClockTime(9, 00, "AM");
    int mins = 0;
    while (hour != temp.hour || minute != temp.minute ||
           !amPm.equals(temp.getAmPm())) {
        temp.advance(1);
        mins++;
    }
    return mins <= 60 * 8;    // 9-5 is an 8-hour day
}
```