CSE 142, Summer 2014
Midterm Exam Answer Key

Name: ______________________________

Section: ____________ **TA:** ____________

Student ID #: ____________

- You have **60 minutes** to complete this exam.
  You may receive a deduction if you keep your test booklet open after time is called.
- This exam is closed-book and closed-notes. Students with additional paper will receive a deduction.
- You may not use any computing devices including calculators.
- Code will be graded on proper behavior/output and not on style, unless otherwise indicated.
- Do not abbreviate code, such as "ditto" marks or dot-dot-dot ... marks.
  The only abbreviations that *are* allowed for this exam are:
  - `S.o.p` for `System.out.print`
  - `S.o.pln` for `System.out.println`
  - `S.o.pf` for `System.out.printf`
- You do not need to write `import` statements in your code.
- If you enter the room, you must turn in an exam before leaving the room.

Good luck!

Score summary: (for grader only)

Problem	Description	Earned	Max
1	Expressions		10
2	Parameter Mystery		12
3	If/Else Simulation		9
4	While Loop Simulation		12
5	Assertions		15
6	Programming		18
7	Programming		15
8	Programming		9
TOTAL	**Total Points**		**100**

1. Expressions (10 points)

For each expression at left, indicate its value in the right column. List a value of appropriate type and capitalization. e.g., `7` for an `int`, `7.0` for a `double`, `"hello"` for a `String`, `true` or `false` for a `boolean`.

Expression	Value
`3 * -1 + 7 - 5 / 2`	**`2`**
`2 + 2 + "(2 + 2)" + 2 + (2 + 2)`	**`"4(2 + 2)24"`**
`13 / 3 - 27 / 5 * 0.5 + (7.5 - 6)`	**`3.0`**
`2 % 11 % 2 + 11 % 2 + 2`	**`3`**
`(5 / 3 == 1 && 10 < 4 + 5) != false`	**`false`**

2. Parameter Mystery (12 points)

At the bottom of the page, write the output produced by the following program, as it would appear on the console.

```
public class ParameterMystery {
   public static void main(String[] args) {
      String eggman = "googoo";
      String walrus = "eggman";
      String googoo = "me";
      String me = "he";
      String he = "walrus";
      String lucy = "in the sky";

      magicalMystery(he, me, lucy);
      magicalMystery(walrus, eggman, googoo);
      magicalMystery("eggman", eggman, walrus);
      magicalMystery(lucy, "we " + he, "googoo");
   }

   public static void magicalMystery(String we, String he, String me) {
      System.out.println("I am " + he + " as you are " + me + " as you are " + we);
   }
}
```

```
I am he as you are in the sky as you are walrus
I am googoo as you are me as you are eggman
I am googoo as you are eggman as you eggman
I am we walrus as you are googoo as you are in the sky
```

3. If/Else Simulation (9 points)

For each call below to the following method, write the output that is produced, as it would appear on the console:

```
public static void ifElseMystery(int a, int b) {
    int c = 2;
    if (a + c < b) {
        c = c + 8;
    } else {
        b = b + 10;
    }
    if (a + c < b) {
        c = c + 8;
    } else {
        b = b + 10;
    }
    System.out.println(b + " " + c);
}
```

Method Call	Output
`ifElseMystery(7, 17);`	`27 10`
`ifElseMystery(12, 5);`	`15 10`
`ifElseMystery(16, 8);`	`28 2`

4. While Loop Simulation (12 points)

For each call below to the following method, write the output that is produced, as it would appear on the console:

```
public static void whileMystery(int n) {
    int x = 1;
    int y = 1;
    while (n > y) {
        x++;
        y = 10 * y + x;
    }
    System.out.println(x + " " + y);
}
```

Method Call	Output
`whileMystery(0);`	`1 1`
`whileMystery(7);`	`2 12`
`whileMystery(32);`	`3 123`
`whileMystery(256);`	`4 1234`

5. Assertions (15 points)

For each of the five points labeled by comments, identify each of the assertions in the table below as either being *always* true, *never* true, or *sometimes* true / sometimes false. (You may abbreviate them as A, N, or S.)

```
public static void mystery(int x, int y) {
    int z = 0;

    // Point A
    while (x < y) {
        // Point B
        z++;
        if (z % 2 == 0) {
            x = x * 2;
            // Point C
        } else {
            y--;
            // Point D
        }
    }

    // Point E
    System.out.println(z);
}
```

	x < y	z == 0	z % 2 == 0
Point A	Sometimes	Always	Always
Point B	Always	Sometimes	Sometimes
Point C	Sometimes	Never	Always
Point D	Sometimes	Never	Never
Point E	Never	Sometimes	Sometimes

6. Programming (18 points)

Write a static method named `yardSale` that accepts two parameters, a console Scanner and an initial amount of money, and returns the amount of money the user ends up with after a series of purchases. Your method should prompt the user for a series of potential purchases specified by a unit price per single item and the quantity of that item until the user's available amount of money is less than $5. The user buys everything that doesn't exceed their current amount of money and that is considered a "good deal" (unit price is under $10). A message should be printed when such a purchase is made. In addition, your method should print the total quantity of items purchased and the total cost of the most expensive item purchased. Results do not need to be rounded after the decimal place. You may assume the user will always type valid input.

For example, the following call

```
Scanner console = new Scanner(System.in);
double remainingBudget = yardSale(console, 50.75);
```

would generate an interaction like this (user input underlined and bold)

```
Price? 8.75
Quantity? 2
What a deal! I'll buy it.
Remaining money: $33.25

Price? 11.50
Quantity? 3
Remaining money: $33.25

Price? 5.10
Quantity? 5
What a deal! I'll buy it.
Remaining money: $7.75

Price? 6.95
Quantity? 2
Remaining money: $7.75

Price? 0.75
Quantity? 8
What a deal! I'll buy it.
Remaining money: $1.75

Total quantities purchased: 15
Most expensive purchase: $25.5
```

You must exactly reproduce the format of this sample execution. Notice that the second item is not purchased because it is not a good deal (unit price is not under $10) and the fourth item is not purchased because the total cost of $13.90 exceeds the amount of money the user has left ($7.75). The most expensive purchase is based on the total cost (5 items at $5.10) rather than the unit cost (2 items at $8.75).

This sample call would return 1.75 as the remaining money because the user spent a total of $49.00 (17.50 + 25.50 + 6.00) and started with $50.75.

One possible solution

```
public static double yardSale(Scanner console, double money) {
    double mostExpensive = 0;
    int totalCount = 0;
    while (money >= 5) {
        System.out.print("Price? ");
        double price = console.nextDouble();
        System.out.print("Quantity? ");
        int quantity = console.nextInt();
        double nextPrice = quantity * price;
        if (nextPrice <= money && price < 10) {
            System.out.println("What a deal! I'll buy it.");
            money = money - nextPrice;
            totalCount = totalCount + quantity;
            if (nextPrice > mostExpensive) {
                mostExpensive = nextPrice;
            }
        }
        System.out.println("Remaining money: $" + money);
        System.out.println();
    }
    System.out.println("Total quantities purchased: " + totalCount);
    System.out.println("Most expensive purchase: $" + mostExpensive);
    return money;
}
```

7. Programming (15 points)

Write a static method named `randomPattern` that takes an integer `size` as a parameter and that prints a square where each line has a series of 1 or more "\" characters followed by 0 or more "/" characters, with each slash separated by a "-" character. The square should contain `size` lines and `size` total backslash/slash characters on each line. Your method should construct a `Random` object that it uses to choose the number of "\"s to print on each line (a number between 1 and size inclusive). You may assume that the parameter passed to your method is at least 2.

The following table lists some calls to your method and one of their possible expected outputs. Keep in mind that since the number of "\" characters on each line is random, your output may not match exactly.

Call	`randomPattern(5);`	`randomPattern(3);`	`randomPattern(2);`	`randomPattern(10);`
Example Output	`\-\-\-\-/`	`\-/-/`	`\-/`	`\-\-\-/-/-/-/-/-/-/`
	`\-\-\-\-\`	`\-/-/`	`\-\`	`\-/-/-/-/-/-/-/-/-/`
	`\-\-/-/-/`	`\-\-/`		`\-\-\-\-/-/-/-/-/-/`
	`\-\-/-/-/`			`\-\-\-\-\-/-/-/-/-/`
	`\-\-\-/-/`			`\-\-\-\-\-\-/-/-/-/`
				`\-\-\-/-/-/-/-/-/-/`
				`\-\-\-/-/-/-/-/-/-/`
				`\-\-\-\-\-\-\-/-/-/`
				`\-\-\-\-\-/-/-/-/-/`
				`\-\-\-\-\-\-\-/-/-/`

One possible solution

```java
public static void randomPattern(int size) {
    Random r = new Random();
    for (int i = 1; i <= size; i++) {
        int backslashes = r.nextInt(size) + 1;
        System.out.print("\\");
        for (int j = 1; j <= backslashes - 1; j++) {
            System.out.print("-\\");
        }
        for (int j = 1; j <= size - backslashes; j++) {
            System.out.print("-/");
        }
        System.out.println();
    }
}
```

8. Programming (9 points)

Write a static method named `collapseDigits` that accepts an integer parameter and that returns a new number obtained by replacing every pair of repeated adjacent digits with one of that digit. For example, the integer 558834226 has three repeated adjacent digits: 55, 88 and 22. This means that `collapseDigits(558834226)` should return the integer 583426. You may assume that no digit is repeated more than twice.

Sample calls:

Call	Value Returned
`collapseDigits(44223553)`	`42353`
`collapseDigits(346623)`	`34623`
`collapseDigits(14436344)`	`143634`
`collapseDigits(121212)`	`121212`
`collapseDigits(0)`	`0`
`collapseDigits(8)`	`8`
`collapseDigits(44)`	`4`
`collapseDigits(22005)`	`205`

You may assume that the integer passed as a parameter to your method is greater than or equal to 0. You may not use Strings to solve this problem.

Two possible solutions

```
public static int collapseDigits(int n) {
   int result = 0;
   int prevDigit = -1;
   int multiplier = 1;
   while (n > 0) {
      int currDigit = n % 10;
      if (prevDigit != currDigit) {
          result = result + currDigit * multiplier;
          multiplier = multiplier * 10;
      }
      prevDigit = currDigit;
      n = n / 10;
   }
   return result;
}

public static int collapseDigits(int n) {
   int result = 0;
   int places = 0;
   while (n > 0) {
      int rightDigit = n % 10;
      int leftDigit = n / 10 % 10;
      result = result + rightDigit * (int) Math.pow(10, places);
      places++;
      if (leftDigit == rightDigit) {
         n = n / 100;
      } else {
         n = n / 10;
      }
   }
   return result;
}
```