1. Expressions (5 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 * 4 + 5 * 6`	`42`
`23 % 5 + - 17 % (16 % 10)`	`-2`
`"1" + 2 + 3 * 4 + (5 + 6)`	`"121211"`
`1.5 * 2 + 20 / 3 / 4.0 + 6 / 4`	`5.5`
`345 / 10 / 3  + 10 / (5 / 2.0)`	`15.0`

2. Array Mystery (10 points)

Consider the following method:

```
public static void arrayMystery(int[] a) {
    for (int i = 1; i < a.length - 1; i++) {
        a[i] = a[i - 1] - a[i] + a[i + 1];
    }
}
```

Indicate in the right-hand column what values would be stored in the array after the method `arrayMystery` executes if the array in the left-hand column is passed as its parameter.

Original Contents of Array	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 (9 points)

The following program produces 4 lines of output. Write the output below, as it would appear on the console.

```
import java.util.*;   // for Arrays class

public class Rectangle {
    int w;
    int h;

    public Rectangle(int width, int height) {
        w = width;
        h = height;
    }

    public String toString() {
        return "w: " + w + ", h: " + h;
    }
}

public class ReferenceMystery {
    public static void main(String[] args) {
        int n = 20;
        int[] a = {40};  // an array with just one element
        Rectangle r = new Rectangle(50, 10);

        mystery(n, a, r);
        System.out.println(n + " " + Arrays.toString(a) + " " + r);

        a[0]++;
        r.w++;
        mystery(n, a, r);
        System.out.println(n + " " + Arrays.toString(a) + " " + r);
    }

    public static int mystery(int n, int[] a, Rectangle r) {
        n++;
        a[0]++;
        r.h++;
        System.out.println(n + " " + Arrays.toString(a) + " " + r);
        return n;
    }
}
```

Solution:

```
21 [41] w: 50, h: 11
20 [41] w: 50, h: 11
21 [43] w: 51, h: 12
20 [43] w: 51, h: 12
```

4. Inheritance Mystery (12 points)

Assume that the following four classes have been defined:

```
public class Gala extends Apple {
    public void method1() {
        System.out.print("gala 1  ");
    }

    public String toString() {
        return "gala " +
               super.toString();
    }
}

public class Fruit {
    public void method1() {
        System.out.print("fruit 1  ");
    }

    public void method2() {
        System.out.print("fruit 2  ");
    }

    public String toString() {
        return "fruit";
    }
}
```

```
public class Fuji extends Apple {
    public void method1() {
        System.out.print("fuji 1 ");
    }
}

public class Apple extends Fruit {
    public void method2() {
        method1();
        System.out.print("apple 2  ");
    }

    public String toString() {
        return "apple";
    }
}
```

Given the classes above, what output is produced by the following code?

```
Fruit[] elements = {new Gala(), new Fruit(), new Fuji(), new Apple()};
for (int i = 0; i < elements.length; i++) {
    elements[i].method1();
    System.out.println();
    elements[i].method2();
    System.out.println();
    System.out.println(elements[i]);
    System.out.println();
}
```

Solution:

```
gala 1
gala 1  apple 2
gala apple

fruit 1
fruit 2
fruit

fuji 1
fuji 1 apple 2
apple

fruit 1
fruit 1  apple 2
apple
```

5. File Processing (12 points)

Write a static method named **`computeGrade`** that accepts as its parameter a `Scanner` for an input file whose data represents a student's grades for tests and assignments. Your method should compute the student's overall grade percentage from the total points earned for all tests and assignments versus the total points possible for all tests and assignments. The input consists a series of one or more score records. Each score record consists of three tokens, where the first is the name of the assignment or test, the second is the number of points the student earned and the third is the number of points possible.

For example, if the input file contains the following text:

```
homework1 40 57 test1 78 100 test2 67 80
```

Your method would produce the following output. Notice that the grade percentage is truncated to an integer (truncated not rounded).

```
homework1: 40/57
test1: 78/100
test2: 67/80

grade: 78%
```

The 78% comes from adding (40 + 78 + 67) and dividing by (57 + 100 + 80). Unlike in your assignment 4, you should not cap the percentage.

Here is a second example. Suppose the input file contains the following text. Notice the capitalization and spacing:

```
HomeWork1   10      20   HOMEWORK2      30 30
HoMeWoRk19

  10    10 exam 100
                              100
```

Then your method would produce the following output. All text output should be in lowercase.

```
homework1: 10/20
homework2: 30/30
homework19: 10/10
exam: 100/100

grade: 93%
honor roll
```

"`honor roll`" should be output if the student has earned 90% or higher. You may assume that the file contains at least one score record (a set of 3 tokens). You may also assume that the input is valid; that the input has sets of 3 tokens and that the second two are always integers.

One possible solution:

```
public static void computeGrade(Scanner input) {
      int totalPoints = 0;
      int totalEarned = 0;
      while(input.hasNext()) {
            String name = input.next();
            int score = input.nextInt();
            int total = input.nextInt();
            System.out.println(name.toLowerCase() + ": " + score + "/" + total);
            totalEarned += score;
            totalPoints += total;
      }
      System.out.println();
      double grade = totalEarned * 100.0 / totalPoints;
      System.out.println("grade: " + (int)grade + "%");
      if(grade >= 90) {
            System.out.println("honor roll");
      }
}
```

6. File Processing (13 points)

Write a static method named **goodBooks** that accepts as its parameters a `String` containing an input file name and a real number `d`. Your method should output information about books that have a rating of `d` or higher.

The input file is comprised of a series of lines each containing information about a different book. The lines are in the format `<publication date> <rating> <title>`.

For example, suppose the input file contains the following text:

```
1900 6.4 The     Wizard                of Oz
1969 7.2 The Hungry                                  Caterpillar
1981 8.0           The         Hitchhiker's Guide to      the Galaxy
1947 4.1 Goodnight            Moon
1595 6.1 The Comedy    of Errors
2013 9.9 Building Java               Programs
2012 6.2 Collected    Poems
```

When passed the above file and a `d` of `6.2`, your method would produce the following output. Notice that your method must place just one space between each word in the title.

```
The Wizard of Oz, 113 years old
The Hungry Caterpillar, 44 years old
The Hitchhiker's Guide to the Galaxy, 32 years old
Building Java Programs, 0 years old
Collected Poems, 1 years old
```

Notice that the output contains the age of the book, not its publication date. You should calculate the date by determining the difference between the publication date and the current year, 2013.

If your method is passed a `d` higher than the ratings of all books in the passed in file it should output the following:

```
No books found.
```

One possible solution:

```
public static void goodBooks(String fileName, double rating) throws
            FileNotFoundException {
      Scanner input = new Scanner(new File(fileName));
      boolean results = false;
      while(input.hasNextLine()) {
            String line = input.nextLine();
            Scanner lineScan = new Scanner(line);
            int year = lineScan.nextInt();
            double currentRating = lineScan.nextDouble();
            String name = "";
            while (lineScan.hasNext()) {
                  name += " " + lineScan.next();
            }

            if(currentRating >= rating) {
                  results = true;
                  System.out.println(name.toUpperCase() + ", " + (2013 - year) +
                                     " years old");
            }
      }
      if (!results) {
            System.out.println("No books found");
      }
}
```

7. Array Programming (10 points)

Write a static method named `isConsecutive` that accepts an array of `int`s as a parameter and returns `true` if the list of integers contains a sequence of increasing consecutive integers and returns `false` otherwise. Consecutive integers are integers that come one after the other, as in `5, 6, 7, 8, 9,` etc.

For example, if a variable called `list1` stores the following values:

```
[16, 17, 18, 19]
```

A call on `isConsecutive(list1)` should return `true`.

If instead `list1` stored the following sequence of values:

```
[16, 17, 18, 19, 20, 19]
```

A call on `isConsecutive(list1)` should return `false`.

An array of fewer than two elements is considered to be consecutive.

One possible solution:

```
public static boolean isConsecutive(int[] a) {
    for(int i = 0; i < a.length - 1; i++) {
        if(a[i] != a[i + 1] - 1) {
            return false;
        }
    }
    return true;
}
```

8. Critters (14 points)

Write a critter class **`HoneyBadger`** along with its fight, movement and eating behavior. All unspecified aspects of `HoneyBadger` use the default behavior. Write the complete class with any fields, constructors, etc. necessary to implement the behavior.

`HoneyBadgers` just don't care. They are always hungry and so eat whenever they come across food. In a fight they are vicious and attack by choosing randomly with equal probability between pouncing and scratching on each turn.

When a `HoneyBadger` is created it gets passed an initial amount of steps. It always starts moving by going North for 1 step and then moves (steps – 1) times West. A `HoneyBadger` restarts its movement pattern every time it eats. `HoneyBadger(4)` would move in this pattern:

- N, W, W, W, N, W, *(eats food)*, **N, W, W, W**, N, W, W, W, N, W, ...

As in assignment 8, all fields must be declared private and fields that need to be initialized to a non-default value must be set in a constructor.

One possible solution:

```
public class HoneyBadger extends Critter {
        private int moves;
        private int steps;

        public HoneyBadger(int steps) {
                moves = -1;
                this.steps = steps;
        }

        public boolean eat() {
                moves = -1;
                return true;
        }

        public Attack fight(String opponent) {
                Random r = new Random();
                if(r.nextInt(2) == 0) {
                        return Attack.Pounce;
                } else {
                        return Attack.Scratch;
                }
        }

        public Direction getMove() {
                moves++;

                if (moves % steps == 0) {
                        return Direction.North;
                } else {
                        return Direction.West;
                }
        }
}
```

9. Array Programming (15 points)

Write a static method named **`distributeCount`** that accepts an array of integers and an integer `n` as parameters. Each array element stores a number of tokens for a game. The tokens at index `n` are to be distributed to the other array positions one at a time starting with the index `n + 1`.

For example, if a variable called `list` stores the following values:

```
[1, 2, 3, 4, 5]
```

Then after the call `distributeCount(list, 1)`, `list` should store:

```
[1, 0, 4, 5, 5]
```

Notice that the 2 tokens that were at index 1 have been distributed to indexes 2 and 3.

In distributing tokens, you may reach the end of the array. In this case, you should move to the beginning of the array (index 0) and continue distributing tokens. For example, if `list` instead stored this sequence of values:

```
[1, 7, 3, 4, 5]
```

Then after the call `distributeCount(list, 1)`, `list` should store:

```
[2, 1, 5, 6, 6]
```

The 7 tokens that were at index 1 have been distributed to indices 2, 3, 4, 0, 1, 2, 3.

You may assume that `n` is a valid index of the array. Do not make any assumptions about the length of the array or the size of values stored in it.

You may not use any temporary arrays to help you solve this problem. (But you may declare as many simple variables as you like, such as `int`s.) You also may not use any other data structures or complex types such as `String`s, or other data structures that were not taught in CSE 142 such as the `ArrayList` class from Chapter 10.

One possible solution:

```
public static void distributeCount(int[] a, int n) {
      int count = a[n];
      a[n] = 0;
      for(int i = 0; i < count; i++) {
            a[(n + 1 + i) % a.length]++;
      }
}
```