Please answer the questions on a seperate sheet and return the exam when you are done.

Name: _____

1. Explain the difference between:
   (a) (1 point) `if (true) {}` and `while(true) {}`

   Solution: `if` executes the statement between brackets once, if the condition is true. `while` executes the statement between brackets until the condition becomes `false`. If it the condition is `false` to begin with, the statement between brackets is skipped.

   (b) (1 point) `float a = 3/4` and `float a = 3.0/4.0`

   Solution: `3/4` is an integer division, which is rounded down to 0. Then it is converted to float, yielding 0.0. `3.0/4.0` is a float division and yields `0.75`.

   (c) (1 point) `new MyClass(5)` and `new MyClass[5]`

   Solution: `new MyClass(5)` creates a new class, where the constructor is called with argument `5`. `new MyClass[5]` creates an array of type `MyClass` with `5` elements.

2. (2 points) What does the following code print on the screen? Provide a full explanation.

```
void setup() {
    int x = 4;
    multiplyByTwo(x);
    println(x);
}

int x = 3;

void multiplyByTwo(int x) {
  x = x * 2;
}
```

   Solution: The global variable `x` is shadowed by the local variable `x`, therefore, its value will never be used. The function `multiplyByTwo` modifies the variable `x` local to the function (a copy of the argument). Therefore, it does not change the function argument. So, the `x` in setup retains its value `4`, and that is what is printed on the screen.

3. Insertion sort is an algorithm to sort an array. In this case we call an array sorted if the numbers descend. Insertion sort works like this: for each element in the array, you check how far you can move it to the left without encountering a value that is larger. Next, you make room for the number by moving all the smaller values to the right, and you place the number in its new position.

(a) (1 point) Below you see an example implementation for insertion sort. Fill in the blanks on your answer sheet.

```
void InsertionSort(int[] num)
{
    int key;
    int i;

    for (int j = 1; j < num.length; j++) {
        key = num[j];
        for(i = j - 1; i >= 0 && num[i] < key; i--) {
            num[_____] = num[_____];
        }
        num[i + 1] = key;
    }
}
```

Solution: `num[i + 1] = num[i]`, since each element must be copied to the right.

(b) (1 point) Why can `j` start at 1 instead of 0?

Solution: `num[j]` is compared to values on its left. Since it is the first, there are no values on its left.

(c) (1 point) Why does `i` have to be defined outside of the second for-loop?

Solution: It is used in `num[i + 1]`, and identifies the position where the key should be placed (minus 1).

(d) (1 point) Why do you have to store `num[j]` in `key`? Why not just use `num[j]` instead?

Solution: The second loop that shifts all the values to the right overwrites `num[j]`.

4. A biologist has come up with a new model that predicts how many birds will migrate through The Netherlands in the winter. Her model is based on migration patterns of previous years. Her prediction is that the number of birds in the current year is 1.1 times the number of birds last year plus 0.1 times the number of birds two years ago. This model is valid from 2012 and onward. In 2010, there were 25 million migrating birds and in 2011 there were 26 million birds.

(a) (3 points) Write a recursive implementation to find the number of birds for any year `y >= 2010`.

Solution: We call $N(y)$ the number of birds (counted as millions) at year y. We write the recursive formula:

$$N(2010) = 25 \tag{1}$$
$$N(2011) = 26 \tag{2}$$
$$N(y) = 1.1 \cdot N(y-1) + 0.1 \cdot N(y-2) \tag{3}$$

We convert this to the following code:

```
float N(int y) {
    if (y == 2010) return 25;
    if (y == 2011) return 26;

    return 1.1 * N(y - 1) + 0.1 * N(y - 2);
}
```

(b) (1 point) Write out how the recursion is done for $y = 2014$.

Solution:

```
N(2014) = 1.1 * N(2013) + 0.1 * N(2012)
        = 1.1 * (1.1 * N(2012) + 0.1 * N(2011))
              + 0.1 * (1.1 * N(2011) + 0.1 * N(2010))
        = 1.1 * (1.1 * (1.1 * N(2011) + 0.1 * N(2010))
            + 0.1 * N(2011))
              + 0.1 * (1.1 * N(2011) + 0.1 * N(2010))
```

5. (a) (2 points) Give the conditions for booleans a, b, and c for when the following expression returns `true`:
`(a || ((b != a) && c))`

Solution:

a	b	c
true	true	true
true	true	false
true	false	true
true	false	false
false	true	true

(b) (2 points) Simplify (rewrite the expression using fewer operators) the following expression, where `a,b,c` are integers:
`(a < b || (a >= b && (b < c || b > c)))`

Solution:
`(a < b || b != c)`

6. (2 points) Write a function called *sum* that calculates the sum of all the integers in a range. For example: `sum(2,3)` $= 2 + 3 = 5$, and `sum(1,4)` $= 1 + 2 + 3 + 4 = 10$.

Solution:

```
int sum(int a, int b) {
    int s = 0;
    for (int i = a; i <= b; i++)
        s += i;
    return s;
}
```

7. (5 points) Write a piece of code that counts how often each digit appears in a global array called `a`. It should print the result to the screen. For example, when given the following array:

```
int a[] = new int[]{1,2,3,8,4,3,6,7,6,9};
```

your code should print:

```
Digit count:
0 : 0
1 : 1
2 : 1
3 : 2
4 : 1
5 : 0
6 : 2
7 : 1
8 : 1
9 : 1
```

Solution:

```
void count() {
  int[] counter = new int[10];

  for (int i = 0; i < a.length; i++) {
    counter[a[i]]++;
  }

  for (int i = 0; i < counter.length; i++) {
    println(i + " " + counter[i]);
  }
}
```