1. (Total = 12 marks)

(a) Assume that in a certain country, tax is payable at the following rates:

15% on your first $50000 income
25% on any amount over $50000

Write a method that takes in an annual income value and returns the amount of tax payable. The method should have the following signature:

```
public float tax(int income)
```

(5 marks)

(b) Given below is a Java method:

```
public void mystery(int[] number, int N) {
    int i;

    for (i = 0; i <= N; i++) {
        if (number[i] < 0 || number[i] > 15)
            i++;
        else
            System.out.println("number[" + i + "] = " + number[i]);
    }
}
```

Suppose that we have an `int` array defined as follows:

```
int[] a = new int[] {10, 15, -1, 0, 2, 7, 20, 30};
```

What will be printed if `mystery(a, 6)` is called?

(3 marks)

(c) Rewrite the method in (b) using a `while` loop.

(4 marks)

---

2. (Total = 12 marks)

(a) Write a method

```
public int reverse(int num)
```

which reverses the digits in a number. For example, if the number is 4395, then its reverse is 5934. Your method need not make any allowance for numbers which end with zero (for example, the reverse of 450 will just come out as 54).

(6 marks)

(b) Given below is the `drawLine` method of the `Canvas` class:

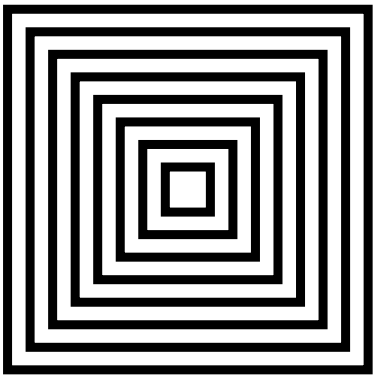
```
public void drawLine(int x1, int y1, int x2, int y2)
```

The method allows you to draw a line connecting two given points, `(x1,y1)` and `(x2,y2)`, on the current canvas.

Write the following method for the `Canvas` class:

```
public void drawSquares(int x, int y, int side)
```

The method should take in `(x,y)` as the coordinates of the top-left corner and `side` as the length of the sides of the outer square to be drawn and produce nested squares as shown below:



In the diagram, the *edges* of each square are 1 pixel wide and adjacent squares have a 1 pixel gap. Thus, if the sides of the largest square are 10 pixels, then the sides of the next smaller square should be 6 pixels, and so on. For the smallest (i.e., the most inner) square, its sides must be at least 2 pixels but at most 5 pixels long.

(6 marks)

3. (Total = 12 marks)

Suppose that we need to develop a simple class called `Country` for use in a geographical database system. The class will need variables to hold (a) the `name` of a country, which contains mainly alphabetic characters, (b) its `capital` city as alphabetic characters, (c) `area` in square km as a double, and (d) `population` as an integer. The only constructor for this class requires four arguments, the `name`, `capital`, `population`, and `area`. The methods needed are `getName()` which returns the name of the country, `getCapital()` which returns the name of the capital city, `getArea()` which returns the area in square km, `getPopulation()` which returns the population size, and `setPopulation(popSize)` which changes the size of population to the value `popSize`.

(a) Write out a skeleton of the `Country` class, clearly showing all variable declarations and method definitions. Their respective types, signatures, and arguments must be included. You are NOT required to include any code in the methods.

(2 marks)

(b) In an existing `Geography` class the following method is required:

```
public static String largestCountry(Country[] countryList)
```

It should return the name of the country that has the largest land area. Write out the method in detail.

(5 marks)

(c) Suppose that we are also interested in finding the country that is most densely populated (i.e., having the most number of people per square km). Write a method

```
public static String mostDenselyPopulatedCountry(Country[] countryList)
```

that returns the name of the most densely populated country.

(5 marks)

---

4. (Total = 12 marks)

(a) The value $\ln 2$ (that is, the natural log of 2) can be approximated by an infinite series as follows:

$$\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \cdots$$

Write a method

```
public double myLn2(int n)
```

that approximates $\ln 2$ using the first n terms of this series.

(4 marks)

(b) Suppose that we have written a `Complex` class for representing complex numbers:

```
public class Complex {
    double real;        // real part
    double imag;        // imaginary part

    public Complex(double r, double i) {
        real = r;
        imag = i;
    }
}
```

Write a method

```
public Complex[] constructComplexArray(double[] r, double[] i)
```

that takes in two arrays of `double` and returns an array of `Complex` objects. For the two input arrays of float point numbers, the first array stores the real parts and the second array stores the corresponding imaginary parts of the complex numbers. You may assume that the two arrays are of the same length.

(4 marks)

(c) Write a method

```
public boolean isPalindrome(String s)
```

that determines whether the argument string `s` is a palindrome. (A *palindrome* is a string that reads the same backwards and forwards, such as MADAM or ATOYOTA.)

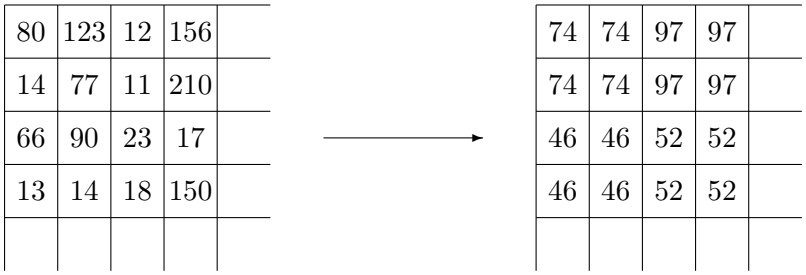
(4 marks)

5. (Total = 12 marks)

Consider a class `Picture` that represents the data for a grey-scale image. Assume that this class has an array `int[][] pixels` as a private instance variable, storing the pixel values, in such a way that `pixels[i][j]` is the grey-scale value of the (i, j)-pixel (in Java graphics co-ordinates).

The operation of *pixellation* produces a "blocky" effect on a picture by dividing the picture into $k \times k$ blocks of pixels, and replacing all the pixels in each block with the average value of the pixels in that block.

Here is an example where each block has size 2×2 pixels:



Write a method

```
public void pixellate(int k)
```

that will pixellate the `Picture` using blocks of size k.

Your method need only work if k is a positive integer, and the number of rows and columns of the `Picture` are exactly divisible by k. You should check that this precondition is met, and take appropriate action otherwise.

Remember that your method should *only* affect the underlying `Picture` data — no attempt should be made to *display* the `Picture`.

(12 marks)

---

6. (Total = 12 marks)

Suppose we start with a positive integer x, and repeatedly apply the rules

- If x is even, then divide it by 2.
- If x is odd, then multiply it by 3 and add 1.

stopping when x reaches 1.

For example, if we start with 17, then we get the sequence of numbers:

$$17 \rightarrow 52 \rightarrow 26 \rightarrow 13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1.$$

(a) Write a method

```
public int numSteps(int x)
```

that calculates the sequence starting at `x` and returns the number of steps it takes to reach 1. For example, `numSteps(17)` should return the value 12, because it takes 12 steps for the sequence starting at 17 to reach 1.

(6 marks)

(b) Write a method that will allow the user to determine which positive integer less than or equal to 10000 gives the *longest* sequence. (You must specify both the method signature and its implementation.)

(6 marks)
