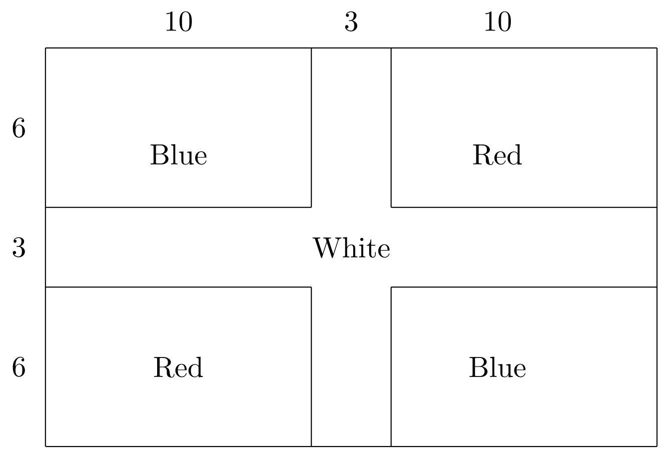
1.

The Dominican flag is a red, white and blue flag with the following design (where the numbers show the relative sizes of the different coloured components).



Write a method

```
public void drawFlag()
```

that will cause the flag to be drawn on the screen with a height of 150 pixels.

Your method should *create* and use a `SimpleCanvas` (as used in lectures and laboratories) to draw on. The method should also create the necessary `Color` objects. The method should not rely on any instance variables.

(12 marks)

2.

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)

3.

(a) Write a method

```
public int indexOfMax(int[] a)
```

that will return the index of the largest value in the one-dimensional array `a`.

(6 marks)

(b) 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 approximateLn2(int n)
```

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

(6 marks)

4.

Consider a class *MyDictionary* that can represent a subset of the English language. The argument of the single *constructor* of this class is an array `String[] words` that represent the words to store in the dictionary. Note that the words are case-sensitive and not ordered in the array.

The constructor requires a helper method from the *Sort* class to sort the array in normal alphabetical order.

Write a method for the *Sort* class using the *InsertionSort* algorithm, with the following signature

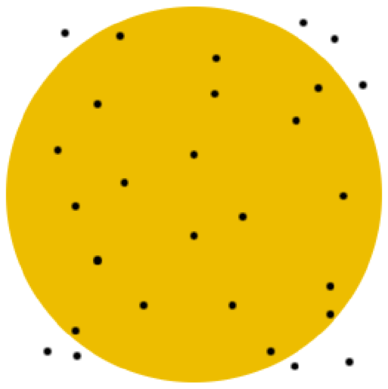
```
public static void alphaInsertionSort(String[] words)
```

(12 marks)

5.

Imagine a pizzeria in which the pizza chef is careless when tossing olives onto a pizza base. In fact he throws them completely randomly in the square area containing the pizza.

One can actually get a simple approximation to π by counting the proportion of the olives that hit the pizza.



A pizza of diameter 1 has an area of $\pi/4$, while the area of the entire square containing the pizza is 1. Therefore the proportion of olives hitting the pizza multiplied by 4 is an approximation to π. In the example, 20 out of 28 olives have hit the pizza and so this gives the (rather poor) approximate value of 4 * 20 / 28 = 2.857

Write a method,

```
public static double pi(int olives)
```

that simulates the random throwing of olives on to a pizza base in order to approximate the value of π. Your method should use an object from the library class `java.util.Random` to generate a sequence of random olive positions inside a suitable square, and keep track of how many land on the pizza. The client will specify the number of olives required, and you should return the corresponding approximation. Your method may <u>not</u> refer to `Math.PI`

You may wish to use the following method from the class `java.util.Random`:

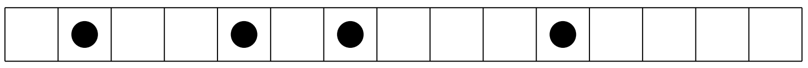
```
public double nextDouble()

    Returns the next pseudorandom, uniformly distributed double
    value between 0.0 and 1.0 from this random number generator's
    sequence.
```

(12 marks)

6.

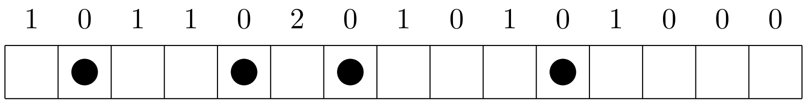
The Game of Life is a 2-dimensional cellular automaton. However there are also 1-dimensional cellular automata, where the world is simply a 1-dimensional strip of cells, each of which is either occupied or vacant. A 15 cell world is shown below with marks signifying occupied cells.

Each cell has just two neighbours which are the cells one step to the right or left of it. We assume that the strip "wraps around" so that the left-hand edge is joined to the right-hand edge (forming a circular strip).

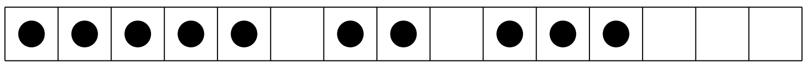


At each time-step the world evolves according to the following rules: any cell with 0 or 2 occupied neighbours remains in its current state, while any cell with a single occupied neighbour changes state (that is, from occupied to unoccupied or vice versa).

For example, for the configuration shown above, each cell has the following number of occupied neighbours.



and so applying the rule to all cells *simultaneously* the world evolves to its next state:



Write a method

```
public boolean[] nextGeneration(boolean[] map)
```

that takes a boolean array representing the current state of the cellular automaton and computes and returns a boolean array representing the next generation.

(12 marks)