Computer Project

(2017-2019)

Satvik Saha

Class: XII B

Roll number: 34

"Writing code a computer can understand is science. Writing code other programmers can understand is an art."

— **Jason Gorman**

"If Java had true garbage collection, most programs would delete themselves upon execution."

— **Robert Sewell**

Problem 17 The classical *Möbius function* $\mu(n)$ is an important function in number theory and combinatorics. For positive integers n, $\mu(n)$ is defined as the sum of the primitive n^{th} roots of unity. It attains the following values.

$\mu(1) = +1$
$\mu(n) = -1$ if n is a square-free positive integer with an odd number of prime factors.
$\mu(n) = \ \ 0$ if n has a squared prime factor.
$\mu(n) = +1$ if n is a square-free positive integer with an even number of prime factors.

Compute the $\mu(n)$ for positive integers n within a specified range.

Solution For any given $n \in \mathbb{N}$, all we have to do is search for factors by trial-division, and find their multiplicity. If this is greater than 1, we can stop here since we have found squared prime factors. Otherwise, we can reduce the problem by dividing out these factors from n and repeating. By trying factors in ascending order and then discarding them from n, we are guaranteed to hit only prime factors, and can thus skip primality checks.

`main (lo:Integer, hi:Integer)`

1. Assert that the integers in the range `[lo, hi)` are all positive.
2. For each `i` $\in \{$`lo`$, $`lo`$ + 1, \ldots,$ `hi`$ - 1\}$:
   (a) Call and display `mobius(i)`.
3. **Exit**

`mobius (n:Integer)`

1. If `n` is one, **return** `1`.
2. Initialize an integer variable `mob` to one.
3. For `i` $\in \{2, 3, \ldots, n\}$:
   (a) Initialize an integer `multiplicity` to zero.
   (b) While `i` divides `n`, assign `n / i` to `n` and increment `multiplicity`.
   (c) If `multiplicity` is one, flip the sign of `mob`.
   (d) If `multiplicity` is greater than one, **return** `0`.
4. **Return** `mob`

Source Code

```
public class Mobius {
	/* Elements of a basic graph */
	public static final String[] graph =
		{"*       ",
		 "   *    ",
		 "       *"};
	public static void main (String[] args) {
		try {
			/* Parse the first command line argument as the lower limit */
			int lo = Integer.parseInt(args[0]);
			/* Parse the second commmand line argument as the upper limit
			    */
			int hi = Integer.parseInt(args[1]);
			/* Incorrct input */
			if (lo < 1 || hi <= lo)
				throw new NumberFormatException();
			for (int i = lo; i < hi; i++) {
				int m = mobius(i);
				System.out.printf(" (%d)\t\t = %2d%24s\n", i, m, graph[m
				    + 1]);
			}
		} catch (NumberFormatException | IndexOutOfBoundsException e) {
			/* Handle missing or incorrectly formatted arguments */
			System.out.println("Enter 2 arguments (lower_limit[integer,
			    >0], upper_limit[integer, >lower_limit])!");
		}
	}

	public static int mobius (int n) {
		/* Ignore negative numbers */
		if (n < 1)
			return 0;
		/* Trivial case */
		if (n == 1)
			return 1;
		/* Start with +1 */
		int mob = 1;
		for (int i = 2; i <= n; i++) {
			int multiplicity = 0;
			/* Count the number of times (i) appears */
			while ((n % i) == 0) {
				/* Reduce 'n' */
				n /= i;
				multiplicity++;
```

```
42                      }
43                      if (multiplicity == 1) {
44                              /* Flip the sign */
45                              mob = -mob;
46                      } else if (multiplicity > 1) {
47                              /* Squared factor found */
48                              return 0;
49                      }
50              }
51              return mob;
52      }
53  }
```

Variable Description

Mobius::main(String[])		
int	lo	Lower bound of integers to evalute
int	hi	Upper bound of integers to evalute
int	i	Counter variable, stores the integer to be evaluated

Mobius::mobius(int)		
int	n	The number where the mobius function is to be evaluated
int	mob	Sign of the value of the mobius function
int	i	Counter variable, stores the current factor to be tested
int	multiplicity	The power of i in the factorisation of n

"The mathematics is not there till we put it there."

— **Arthur Eddington**

Problem 18 A *set* is a collection of distinct objects. Implement a simple model of *sets*, capable of holding *integers*.

Solution This implementation uses *arrays* as the framework for storing elements. The set is sorted during insertion of elements, allowing for fast *binary searching*.

`Set (maxSize:Integer)`

1. Copy `maxSize` into the object data.
2. Initialize an array of integers `elements`, with length `maxSize`.
3. Initialize an integer `top` to `-1`.
4. **Define** the following functions:
   (a) `Set::updateMaxSize(newMaxSize)`
   (b) `Set::contains(n)`
   (c) `Set::add(n)`
   (d) `Set::remove(n)`
   (e) `Set::indexOfEqualOrGreater(n)`
5. **Return** the resultant object.

`Set::updateMaxSize (newMaxSize:Integer)`

1. Initialize an array of integers `temp`, with length `newMaxSize`.
2. Set `maxSize` to `newMaxSize`.
3. If the new size cannot accomodate the present elements of the set, discard them by setting `top` to `maxSize - 1`.
4. Copy all integers from indices `0` to `top` from `elements` to `temp`.
5. Set `elements` to `temp`.

`Set::contains (n:Integer)`

1. Call `this->indexOfEqualOrGreater(n)`. Call the returned value `i`.
2. If `i` is a valid index within the set, and the element at that index is equal to `n`, **return** `true`, otherwise **return** `false`.

`Set::add (n:Integer)`

1. Assert that the set is large enough to hold the new element.
2. If the set already contains `n`, **return** `false`.

3. Call `this->indexOfEqualOrGreater(n)`. Call the returned value `i`.
4. Shift all integers in `elements` from indices `i` to `top` one place to the right.
5. Set `elements[i]` to `n`.
6. **Return** `true`

`Set::remove (n:Integer)`

1. Assert that the set is not empty.
2. If the set does not already contain `n`, **return** `false`.
3. Call `this->indexOfEqualOrGreater(n)`. Call the returned value `i`.
4. Shift all integers in `elemets` from indices `i + 1` to `top` one place to the left.
5. **Return** `true`

`Set::indexOfEqualOrGreater (n:Integer)`

1. Initialize an integer `hi` to `top + 1`.
2. Initialize an integer `lo` to `0`;
3. While `lo < hi`:
   (a) Set a temporary integer `mid` to `(lo + hi) / 2`.
   (b) If `n` is less than the element at mid, set `hi` to `mid`.
   (c) If `n` is greater than the element at mid, set `lo` to `mid + 1`.
   (d) If `n` is equal to the element at mid, **return** `mid`.
4. **Return** `hi`

`union (a:Set, b:Set)`

1. Create a new `Set`, capable of holding the combined elements of `a` and `b`. Call it `r`.
2. For each element `n` in `a`, call `r->add(n)`.
3. For each element `n` in `b`, call `r->add(n)`.
4. **Return** `r`.

`intersection (a:Set, b:Set)`

1. Create a new `Set`, with its `maxSize` equal to either of the sizes of `a` or `b`. Call it `r`.
2. For each element `n` in `a`, also contained in `n`, call `r->add(n)`.
3. **Return** `r`

`difference (a:Set, b:Set)`

1. Create a new `Set`, with its `maxSize` equal to either of the sizes of `a` or `b`. Call it `r`.

2. For each element `n` in `a`, not contained in `n`, call `r->add(n)`.
3. **Return** `r`

Source Code

```
1  import java.util.Iterator;
2
3  public class Set implements Iterable<Integer> {
4         protected int maxSize;
5
6         /* Simple list setup */
7         protected int[] elements;
8         protected int top;
9
10        /* Let the maximum capacity be specified during instantiation */
11        public Set (int maxSize) {
12               this.maxSize = maxSize;
13               this.elements = new int[maxSize];
14               this.top = -1;
15        }
16
17        /* Returns the number of elements in the set */
18        public int getSize () {
19               return top + 1;
20        }
21
22        /* Returns the maximum capacity of the set */
23        public int getMaxSize () {
24               return maxSize;
25        }
26
27        /* Expands or contracts the set as necessary, discards elements if
28           they cannot be accomodated */
29        public void updateMaxSize (int newMaxSize) {
30               int[] temp = new int[newMaxSize];
31               this.maxSize = newMaxSize;
32               /* Make sure that the top index isn't out of bounds */
33               this.top = Math.min(top, newMaxSize - 1);
34               /* Copy data to the new list */
35               for (int i = 0; i <= top; i++)
36                      temp[i] = elements[i];
37               this.elements = temp;
38        }
39
40        /* Checks whether an element is present in the set */
```

```
41          public boolean contains (int n) {
42                  int i = indexOfEqualOrGreater(n);
43                  return ((i >= 0) && (i <= top) && (elements[i] == n));
44          }
45
46          /* Checks whether the set is empty */
47          public boolean isEmpty () {
48                  return top < 0;
49          }
50
51          /* Clears all elements from the set */
52          public void clear () {
53                  /* Only the top index has to be updated, since values byond it
54                     cannot be accessed */
55                  this.top = -1;
56          }
57
58          /* Adds an element to the set. Returns 'false' if it is already
59             present, or there isn't enough space. */
60          public boolean add (int n) {
61                  if (getSize() >= getMaxSize())
62                          return false;
63                  /* Find the breakpoint to shift elements */
64                  int i = indexOfEqualOrGreater(n);
65                  if ((i >= 0) && (i <= top) && (elements[i] == n))
66                          return false;
67                  /* Shift elements greater than 'n' to make room for it */
68                  for (int j = top; j >= i; j--)
69                          elements[j + 1] = elements[j];
70                  elements[i] = n;
71                  top++;
72                  return true;
73          }
74
75          /* Removes an element from the set. Returns 'false' if it isn't
76             already present. */
77          public boolean remove (int n) {
78                  if (isEmpty())
79                          return false;
80                  /* Find the location of the element */
81                  int i = indexOfEqualOrGreater(n);
82                  if ((i < 0) || (i > top) || (elements[i] != n))
83                          return false;
84                  /* Shift elements into the desired element, erasing it */
85                  for (int j = i; j < top; j++)
86                          elements[j] = elements[j + 1];
```

```
87              top--;
88              return true;
89      }
90
91      /* Returns the union of two sets */
92      public static Set union (Set a, Set b) {
93              Set r = new Set(a.getSize() + b.getSize());
94              /* The 'add' methods take care of duplicates */
95              for (int n : a)
96                      r.add(n);
97              for (int n : b)
98                      r.add(n);
99              return r;
100     }
101
102     /* Returns the intersection of two sets */
103     public static Set intersection (Set a, Set b) {
104             Set r = new Set(a.getSize());
105             for (int n : a)
106                     if (b.contains(n))
107                             r.add(n);
108             return r;
109     }
110
111     /* Returns the difference of two sets */
112     public static Set difference (Set a, Set b) {
113             Set r = new Set(a.getSize());
114             for (int n : a)
115                     if (!b.contains(n))
116                             r.add(n);
117             return r;
118     }
119
120     /* Finds the index of the element equal to or greater than
121        the desired element via binary search */
122     private int indexOfEqualOrGreater (int n) {
123             int hi = top + 1;
124             int lo = 0;
125             while (lo < hi) {
126                     int mid = (lo + hi) / 2;
127                     if (n < elements[mid])
128                             hi = mid;
129                     else if (n > elements[mid])
130                             lo = mid + 1;
131                     else
132                             return mid;
```

```
                }
                return hi;
        }

        /* Format the set elements as a list */
        @Override
        public String toString () {
                if (getSize() == 0)
                        return "[]";
                String s = "";
                for (Integer n : this)
                        s += n + " ";
                return "[" + String.join(", ", s.split("\\s+")) + "]";
        }

        /* Allow 'Set' to be iterable, providing easy access to elements
           without indexing */
        @Override
        public Iterator<Integer> iterator () {
                return new Iterator<Integer>() {
                        private int currentIndex = 0;

                        @Override
                        public boolean hasNext () {
                                return currentIndex <= top;
                        }

                        @Override
                        public Integer next () {
                                return elements[currentIndex++];
                        }

                        @Override
                        public void remove () {
                                throw new UnsupportedOperationException();
                        }
                };
        }
}
```

```
public class SetDemo {
        public static void main (String[] args) {
                /* Create 3 sets with random elements */
                Set a = new Set(10);
                Set b = new Set(10);
```

```
6               Set c = new Set(10);
7               for (int i = 0; i < 10; i++)
8                   a.add((int) (Math.random() * 10));
9               for (int i = 0; i < 10; i++)
10                  b.add((int) (Math.random() * 10));
11              for (int i = 0; i < 10; i++)
12                  c.add((int) (Math.random() * 10));
13
14              /* Demonstrate simple output formatting */
15              System.out.printf("A [%2d] = %s\n", a.getSize(), a);
16              System.out.printf("B [%2d] = %s\n", b.getSize(), b);
17              System.out.printf("C [%2d] = %s\n", c.getSize(), c);
18              System.out.println();
19
20              /* Demonstrate set operations */
21              System.out.printf("A union B [%2d] = %s\n",
22                                Set.union(a, b).getSize(),
23                                Set.union(a, b));
24              System.out.printf("B union C [%2d] = %s\n",
25                                Set.union(b, c).getSize(),
26                                Set.union(b, c));
27              System.out.printf("C union A [%2d] = %s\n",
28                                Set.union(c, a).getSize(),
29                                Set.union(c, a));
30              System.out.printf("A union B union C [%2d] = %s\n",
31                                Set.union(Set.union(a, b), c).getSize(),
32                                Set.union(Set.union(a, b), c));
33              System.out.println();
34              System.out.printf("A intersection B [%2d] = %s\n",
35                                Set.intersection(a, b).getSize(),
36                                Set.intersection(a, b));
37              System.out.printf("B intersection C [%2d] = %s\n",
38                                Set.intersection(b, c).getSize(),
39                                Set.intersection(b, c));
40              System.out.printf("C intersection A [%2d] = %s\n",
41                                Set.intersection(c, a).getSize(),
42                                Set.intersection(c, a));
43              System.out.printf("A intersection B intersection C [%2d] = %s\n",
44                                Set.intersection(Set.intersection(a, b),
                                      c).getSize(),
45                                Set.intersection(Set.intersection(a, b), c));
46              System.out.println();
47              System.out.printf("A - B [%2d] = %s\n",
48                                Set.difference(a, b).getSize(),
49                                Set.difference(a, b));
50              System.out.printf("B - C [%2d] = %s\n",
```

```
51                                      Set.difference(b, c).getSize(),
52                                      Set.difference(b, c));
53                  System.out.printf("C - A [%2d] = %s\n",
54                                      Set.difference(c, a).getSize(),
55                                      Set.difference(c, a));
56          }
57  }
```

Variable Description

Set		
int	maxSize	The maximum number of elements the set can hold
int[]	elements	The collection of elements contained in the set
int	top	The index of the topmost element in `elements`

Set::Set(int)		
int	maxSize	The maximum number of elements the set can hold

Set::updateMaxSize(int)		
int	newMaxSize	The maximum number of elements the set is to hold
int[]	temp	The new copy of `elements` with the updated size

Set::add(int)		
int	n	The element to be added to the set
int	i	The index of the breakpoint from which elements have to be shifted

Set::remove(int)		
int	n	The element to be removed from the set
int	i	The index of the breakpoint from which elements have to be shifted

Set::indexOfEqualOrGreater(int)		
int	n	The element to be searched for
int	hi	The upper index where `n` can be
int	lo	The lower index where `n` can be
int	mid	The midpoint of `hi` and `lo`

"Mathematics is the art of giving the same name to different things."

— **Henri Poincaré**

Problem 19 A *vector space* is a collection of objects called *vectors*, which may be added together and multiplied (scaled) by *scalars*. One way of implementing a *vector* is to describe the space $\mathbb{R}^n$, i.e. all possible ordered tuples of n real numbers. For example, the vector $(1, 7, 0, 1)$ belongs to the vector space $\mathbb{R}^4$ – it is a four-dimensional vector.

Addition, scalar multiplication, the dot product and the magnitude of vectors is defined as follows. $(a_i, b_i, k \in \mathbb{R})$

$$\begin{aligned}
(a_1, a_2, \ldots, a_n) + (b_1, b_2, \ldots, b_n) &= (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n) && \text{(Addition)} \\
k\,(a_1, a_2, \ldots, a_n) &= (ka_1, ka_2, \ldots, ka_n) && \text{(Scalar Multiplication)} \\
(a_1, a_2, \ldots, a_n) \cdot (b_1, b_2, \ldots, b_n) &= a_1b_1 + a_2b_2 + \cdots + a_nb_n && \text{(Dot Product)} \\
\|(a_1, a_2, \ldots, a_n)\| &= \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2} && \text{(Magnitude)}
\end{aligned}$$

Implement a simple model of *vectors* as defined above.

Solution

`Vector (components:FloatingPoint[])`

1. Set a constant integer `dimension` to the length of `components`.
2. Copy `components` into the object data as a constant.
3. **Define** the functions:
   (a) `Vector::getComponent(index)`
   (b) `Vector::getAbsoluteValue()`
4. **Return** the resultant object.

`Vector::getComponent (index:Integer)`

1. **Return** `components[index - 1]`

`Vector::getAbsoluteValue ()`

1. Initialize a floating point `abs` to zero.
2. For each `component` in `components`, add `component * component` to `abs`.
3. **Return** the square root of `abs`.

`add (a:Vector, b:Vector)`

1. Assert that `a` and `b` have the same `dimension`.

2. Create an array of floating points `sum`, with length equal to their common `dimension`.
3. For each `i` $\in \{1, 2, \ldots,$ `dimension`$\}$:
   (a) Set `sum[i-1]` to `a->getComponent(i) + b->getComponent(i)`.
4. Create a new `Vector`, pass it `sum` and **return** the resultant object.

`multiplyByScalar (v:Vector, k:FloatingPoint)`

1. Create an array of floating points `t`, with length equal to the `dimension` of `v`.
2. For each `i` $\in \{1, 2, \ldots,$ `dimension`$\}$:
   (a) Set `t[i-1]` to `v->getComponent(i) * k`.
3. Create a new `Vector`, pass it `t` and **return** the resultant object.

`dotProduct (a:Vector, b:Vector)`

1. Assert that `a` and `b` have the same `dimension`.
2. Initialize a floating point `dotProduct` to zero.
3. For each `i` $\in \{1, 2, \ldots,$ `dimension`$\}$:
   (a) Add `a->getComponent(i) * b->getComponent(i)` to `dotProduct` .
4. **Return** `dotProduct`

Source Code

```
1  public class Vector {
2        /* Vector data is immutable */
3        protected final int dimension;
4        protected final double[] components;
5
6        /* Use varargs to create an arbitrary dimensional vector */
7        public Vector (double ... components) {
8              this.dimension = components.length;
9              this.components = new double[dimension];
10             for (int i = 0; i < dimension; i++)
11                   this.components[i] = components[i];
12       }
13
14       /* Returns the dimensionality of the vector */
15       public int getDimension () {
16             return this.dimension;
17       }
18
19       /* Returns the component at the specified index.
20          This uses indexing starting at '1' per mathematical convention */
21       public double getComponent (int index) {
22             return this.components[index - 1];
```

```
23          }
24
25          /* Returns the absolute value/magnitude of the vector */
26          public double getAbsoluteValue () {
27                  double abs = 0.0;
28                  for (int i = 0; i < dimension; i++)
29                          abs += (components[i] * components[i]);
30                  return Math.sqrt(abs);
31          }
32
33          /* Wrapper methods which call static ones */
34
35          public Vector multiplyByScalar (double k) {
36                  return Vector.multiplyByScalar(this, k);
37          }
38
39          public Vector add (Vector v) {
40                  return Vector.add(this, v);
41          }
42
43          public double dotProduct (Vector v) {
44                  return Vector.dotProduct(this, v);
45          }
46
47          public double angleBetween (Vector v) {
48                  return Vector.angleBetween(this, v);
49          }
50
51          public boolean equals (Vector v) {
52                  return Vector.equals(this, v);
53          }
54
55          /* Format vector components neatly */
56          @Override
57          public String toString () {
58                  String s = "(";
59                  for (double component : components)
60                          s += component + ", ";
61                  return s.replaceAll(", $", ")");
62          }
63
64          /* Checks for equality between two vectors */
65          public static boolean equals (Vector a, Vector b) {
66                  /* Dimensionalities must be equal */
67                  if (a.getDimension() != b.getDimension())
68                          return false;
```

```
69              /* Corresponding components must be equal */
70              for (int i = 1; i <= a.getDimension(); i++)
71                      if (a.getComponent(i) != b.getComponent(i))
72                              return false;
73              return true;
74      }
75
76      /* Multiplies a vector by a scalar to return a vector */
77      public static Vector multiplyByScalar (Vector v, double k) {
78              double[] t = new double[v.getDimension()];
79              for (int i = 0; i < t.length; i++)
80                      t[i] = v.getComponent(i+1) * k;
81              return new Vector(t);
82      }
83
84      /* Adds two vectors to return a vector */
85      public static Vector add (Vector a, Vector b) {
86              double[] sum = new double[a.getDimension()];
87              /* Add corresponding components */
88              for (int i = 0; i < sum.length; i++)
89                      sum[i] = a.getComponent(i+1) + b.getComponent(i+1);
90              return new Vector(sum);
91      }
92
93      /* Adds multiple vectors to return a vector */
94      public static Vector add (Vector ... vectors) {
95              Vector v = vectors[0];
96              /* Repeatedly use the binary addition method */
97              for (int i = 1; i < vectors.length; i++)
98                      v = Vector.add(v, vectors[i]);
99              return v;
100     }
101
102     /* Returns the dot product of two vectors */
103     public static double dotProduct (Vector a, Vector b) {
104             double dotProduct = 0.0;
105             /* Multiply corresponding components */
106             for (int i = 1; i <= a.getDimension(); i++)
107                     dotProduct += a.getComponent(i) * b.getComponent(i);
108             return dotProduct;
109     }
110
111     /* Returns the angle between two vectors in radians.
112        If 'u' and 'v' are vectors, with an angle 'A' between them,
113                     u.v = |u||v| cos(A)  */
114     public static double angleBetween (Vector a, Vector b) {
```

```
115                 return Math.acos(Vector.dotProduct(a, b) / (a.getAbsoluteValue() *
                        b.getAbsoluteValue()));
116         }
117 }
```

```
1  public class VectorDemo {
2         public static void main (String[] args) {
3                 /* Simple 2D vector with magnitude sqrt(2) */
4                 Vector a = new Vector(1, 1);
5                 System.out.printf("Magnitude of %s is %f\n", a, a.getAbsoluteValue());
6
7                 /* Create 3 random 3D vectors */
8                 Vector b = new Vector(random(-10, 10), random(-10, 10), random(-10,
                      10));
9                 Vector c = new Vector(random(-10, 10), random(-10, 10), random(-10,
                      10));
10                Vector d = new Vector(random(-10, 10), random(-10, 10), random(-10,
                      10));
11
12                /* Demonstrate addition, dot products, angle measurement */
13                System.out.printf("Sum of vectors %s, %s, %s is %s\n", b, c, d,
                      Vector.add(b, c, d));
14                System.out.printf("Dot product of %s and %s is %d\n", b, c, (int)
                      Vector.dotProduct(b, c));
15                System.out.printf("The angle between %s and %s is %f degrees\n", b, c,
16                                                  Math.toDegrees(Vector.angleBetween(b,
                                                        c)));
17        }
18
19        /* Returns random integers in a specified range */
20        public static int random (int lo, int hi) {
21                return (int) (lo + ((hi - lo) * Math.random()));
22        }
23 }
```

Variable Description

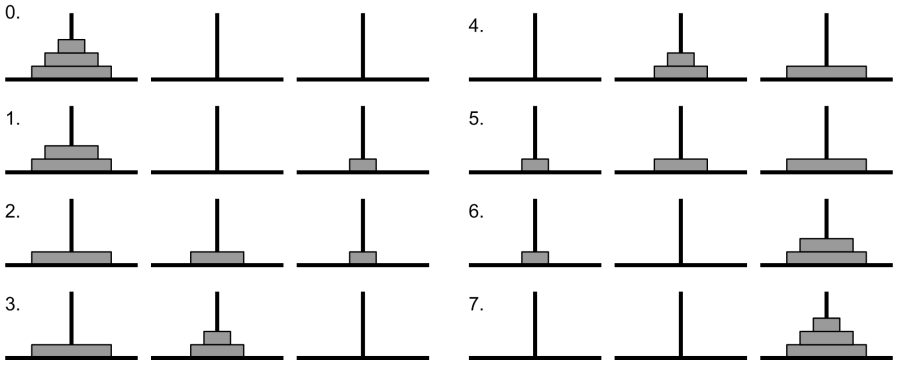
Vector		
int	dimension	The dimension of the vector
double[]	components	The ordered list of components of the vector
Vector::Vector(double[])		
double[]	components	The ordered list of components of the vector
Vector::getComponent(int)		
int	index	The index of the component to be retrieved
Vector::getAbsoluteValue()		
double	abs	Stores the square of the magnitude of the vector
int	i	Counter variable, counts through components of the vector
Vector::multiplyByScalar(double)		
double	k	The scalar to multiply the vector by

"In order to understand recursion, one must first understand recursion."

— **Anonymous**

Problem 20 The *Tower of Hanoi* is a mathematical puzzle, consisting of three rods and a number of disks of different sizes which can slide onto any rod. The puzzle starts with all disks, in ascending order of size, on one rod. The objective of the puzzle is to move the entire stack to another rod, obeying the following rules.

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one stack and placing it on the top of another stack or empty rod.
3. No disk can be placed on a smaller disk.



Solution to the Towers of Hanoi with 3 disks.

Solve the *Tower of Hanoi* puzzle for an arbitrary number of disks, enumerating the required moves.

Solution The main insight here is that the problem involving n disks can be reduced to one with $n-1$ disks. Labelling the rods A, B and C, and the disks with numerals 1 through n (smallest to largest), our aim is to move the entire stack from A to C. If we can solve the problem with $n-1$ disks, all we have to do is to move the topmost $n-1$ disks from A to B, move the remaining disk on A to C, and again move the $n-1$ disks on B to C. The base case for this recursive solution is moving 1 disk, which is trivial.

Clearly, if the problem with n disks takes k_n number of moves, the problem with $n+1$ moves will take $k_n + 1 + k_n = 2k_n + 1$ moves. For the base case with one disk,

$k_1 = 1$. With this infromation, we see that the *Tower of Hanoi* with n disks can be solved in exactly $2^n - 1$ moves.

`main (disks:Integer)`

1. Call `solveHanoi(disks, "A", "C", "B")`.
2. **Exit**

`solveHanoi (disk:Integer, source:String, destination:String, spare:String)`

1. If `disk` is zero, **return**.
2. Call `solveHanoi(disk - 1, source, spare, destination)`.
3. Move disk number `disk` has to be moved from `source` to `destination`.
4. Call `solveHanoi(disk - 1, spare, destination, source)`.
5. **Return**

Source Code

```
1  public class TowersOfHanoi {
2        public static void main (String[] args) {
3              try {
4                     /* Parse the first command line argument as the number of
                          disks */
5                     int disks = Integer.parseInt(args[0]);
6                     /* Make sure there is at least one disk */
7                     if (disks < 1)
8                           throw new NumberFormatException();
9                     /* Initiate the recursive steps */
10                    solveHanoi(disks, "A", "C", "B");
11             } catch (NumberFormatException | IndexOutOfBoundsException e) {
12                    /* Handle missing or incorrectly formatted arguments */
13                    System.out.println("Enter 1 argument (number_of_disks[integer,
                          >0])!");
14             }
15       }
16
17       /* Displays moves to solve the Towers of Hanoi problem with 3 pegs */
18       public static void solveHanoi (int disk, String source, String destination,
             String spare) {
19             /* Base case - nothing to do */
20             if (disk == 0)
21                    return;
22             /* Move the stack of (n-1) disks to the spare peg */
23             solveHanoi(disk - 1, source, spare, destination);
24             /* Move the largest disk to the destination */
```

```
25                System.out.printf("(%d) : %s -> %s%n", disk, source, destination);
26                /* Move the stack of (n-1) disks back on top of the largest
27                   disk, on the destination peg */
28                solveHanoi(disk - 1, spare, destination, source);
29        }
30  }
```

Variable Description

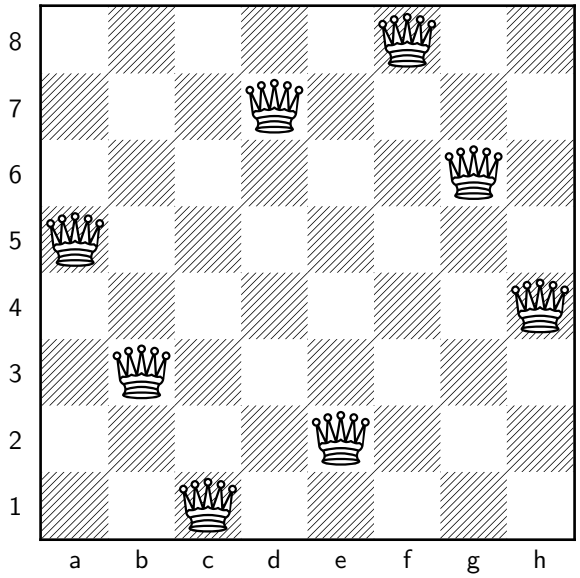
TowersOfHanoi::main(String[])		
int	disks	The number of disks in the problem

TowersOfHanoi::solveHanoi(int, String, String, String)		
int	disk	The current disk to be moved
String	source	The rod from which the stack is to be moved
String	destination	The rod to which the stack is to be moved
String	spare	The additional rod, where the remaining n-1 disks are temporarily moved

"Chess is the gymnasium of the mind."

— Blaise Pascal

Problem 21 The *8 queens puzzle* involves placing 8 queens on an 8×8 chessboard such that no two queens threaten each other, i.e. no two queens share the same rank, file or diagonal. It was first published by the chess composer *Max Bezzel* in 1848. This puzzle has 92 solutions, including reflections and rotations. Below is one of them.



The *n queens puzzle* is an extension of this puzzle, involving n queens on an $n \times n$ chessboard. Count the total number of solutions for the *n queens puzzle*, including reflections and rotations.

Solution This problem can be solved with *recursion* and *backtracking*. Starting from the topmost row of the chessboard, we can place a queen and for each available choice, place a queen on the next row, and so on, recursively shrinking the chessboard to solve. Invalid solutions can thus be discarded as they are formed without brute-forcing every possible permutation of queens on the board.

Finally, by noting that exactly one queen must occupy each row, we can optimize the board by storing only the column numbers of queens on each row in an array, instead of simulating a full 2D board.

`main (size:Integer, drawSolutions:Boolean)`

1. Create an `NQueens` object by passing it `size` and `drawSolutions`. Call it `q`.
2. Call `q->countSolutions()` and display the result.
3. **Exit**

`NQueens (size:Integer, drawSolutions:Boolean)`

1. Copy `size` and `drawSolutions` into the object data.
2. Initialize an integer `numberOfSolutions` to zero.
3. Initialize an array of integers with length `size`. Call it `board`.
4. **Define** the functions:
   (a) `NQueens::countSolutions()`
   (b) `NQueens::solveNQueens(row)`
   (c) `NQueens::isThreatened(row)`
5. **Return** the resultant object.

`NQueens::countSolutions ()`

1. Call `this->solveNQueens(0)`.
2. **Return**

`NQueens::solveNQueens (row:Integer)`

1. If `row` is equal to `size`:
   (a) Increment `numberOfSolutions`.
   (b) If `drawSolutions` is set to `true`, display the current state of `board`.
   (c) **Return**
2. For each $\texttt{i} \in \{0, 1, \ldots, \texttt{size} - 1\}$:
   (a) Place a queen at row `row`, column `i`, i.e. set `board[row]` to `i`.
   (b) Call `this->isThreatened(row)`. If this returns `false`, call `this->solveNQueens(row + 1)`.
3. **Return**

`NQueens::isThreatened (row:Integer)`

1. For each $\texttt{i} \in \{0, 1, \ldots, \texttt{size} - 1\}$:
   (a) If there are two queens on the same column in rows `row` and `i`, or the columns in which those two queens are on are on the same diagonal, **return** `true`.
2. **Return** `false`

Source Code

```
1  public class NQueens {
2  	private final int size;
3  	private int[] board;
4  	private int numberOfSolutions;
5  	private final boolean drawSolutions;
6
7  	/* Sets the size of the board and the number of queens */
8  	public NQueens (int size, boolean drawSolutions) {
9  		this.size = size;
10 		this.drawSolutions = drawSolutions;
11 		this.initBoard();
12 	}
13
14 	/* Returns the number of solutions to a board of given size */
15 	public int countSolutions () {
16 		solveNQueens(0);
17 		return numberOfSolutions;
18 	}
19
20 	/* Initializes the board */
21 	private void initBoard () {
22 		this.board = new int[size];
23 		this.numberOfSolutions = 0;
24 		for (int i = 0; i < size; i++)
25 			board[i] = -1;
26 	}
27
28 	/* Determines whether the queen on a specified row is threatened
29 	   by a queen on a previous row */
30 	private boolean isThreatened (int row) {
31 		for (int i = 0; i < row; i++) {
32 			if ((board[row] == board[i])
33 			    || ((board[row] - board[i]) == (row - i))
34 			    || ((board[row] - board[i]) == (i - row))) {
35 				return true;
36 			}
37 		}
38 		return false;
39 	}
40
41 	/* Recursively solves the n-queens problem */
42 	private void solveNQueens (int row) {
43 		if (row == size) {
44 			/* Reached maximum recursion depth - found a solution */
```

```
45                      numberOfSolutions++;
46                      if (drawSolutions) {
47                              drawBoard();
48                              System.out.println();
49                      }
50                      return;
51              }
52              /* Place queens on all possible columns on the row */
53              for (board[row] = 0; board[row] < size; board[row]++) {
54                      if (!isThreatened(row)) {
55                              /* Recurse if the board is valid so far */
56                              solveNQueens(row + 1);
57                      }
58              }
59      }
60
61      /* Displays the current configuration of the board */
62      public void drawBoard () {
63              for (int i = 0; i < size; i++) {
64                      for (int j = 0; j < size; j++) {
65                              System.out.print(((board[i] == j)? "Q" : "-") + " ");
66                      }
67                      System.out.println();
68              }
69      }
70
71      public static void main (String[] args) {
72              try {
73                      /* Parse the first command line argument as the size of the
                            board */
74                      int size = Integer.parseInt(args[0]);
75                      /* Parse the second command line argument as a boolean,
76                         indicating whether to draw the solved boards.
77                         Defaults to not showing the solutions */
78                      boolean drawSolutions = (args.length > 1)?
                            Boolean.parseBoolean(args[1]) : false;
79                      /* Make sure the board exists */
80                      if (size < 1)
81                              throw new NumberFormatException();
82                      /* Create a 'NQueens' object */
83                      NQueens q = new NQueens(size, drawSolutions);
84                      /* Display the number of solutions */
85                      System.out.println(q.countSolutions());
86              } catch (NumberFormatException | IndexOutOfBoundsException e) {
87                      /* Handle missing or incorrectly formatted arguments */
```

```
88                      System.out.println("Enter at least 1 argument
                            (size_of_board[integer], <show_solutions>[true/false])!");
89                      System.out.println("(show_solutions defaults to false)");
90              }
91      }
92  }
```

Variable Description

NQueens		
int	size	The number of rows and columns in the chessboard
int[]	board	The list of positions of queens in columns, with their rows corresponding to their index.
int	numberOfSolutions	Counts the number of solutions found
boolean	drawSolutions	Stores whether to display solved boards or not

NQueens::isThreatened(int)		
int	row	The row of the queen to test
int	i	Counter variable, stores the row of the queen to test against

NQueens::solveNQueens(int)		
int	row	The current row on which a queen is to be placed

NQueens::drawBoard()		
int	i, j	Counter variables, store the row and column to be currently displayed

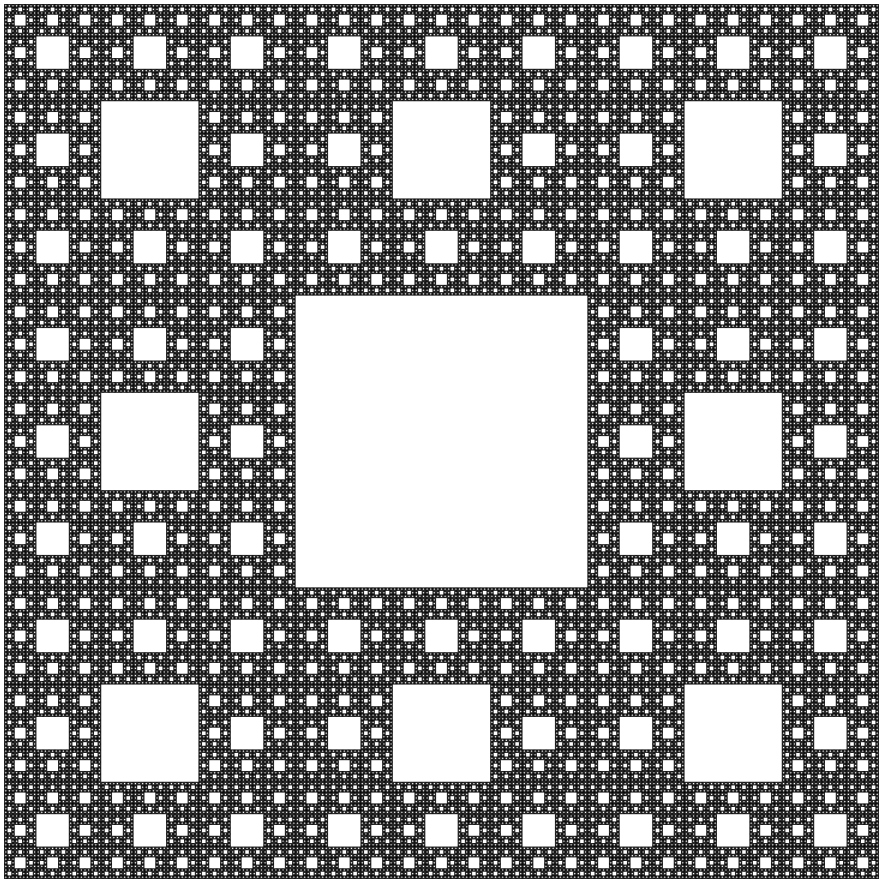
NQueens::main(String[])		
int	size	The number of rows and columns in the chessboard
boolean	drawSolutions	Stores whether to display solved boards or not
NQueens	q	Object capable of solving the *n queens* problem

"In the mind's eye, a fractal is a way of seeing infinity."

— **James Gleick**

Problem 22 The *Sierpinski Carpet* is a plane fractal. It can be produced iteratively by taking a solid square, dividing it into 9 congruent squares in a 3-by-3 grid, removing the centre square, and recursively applying the same procedure on each of the remaining squares *ad infinitum*.

Display the *Sierpinski Carpet* to a specified number of iterations.



The Sierpinski Carpet

Solution In an ASCII terminal, we can only display a rough representation of the *Sierpinski Carpet*, a few levels deep. A level n carpet will have a width and height of 3^n. Within this grid, every character lies either in the centre of a 3-by-3 square, in which case it is not in the carpet, or it lies on the edge, in which case it is in the carpet. If neither can be determined, we can scale up the search square to the next level, and repeat recursively.

Here, points in the carpet are drawn, while points not in the the carpet are left as whitespace.

`main (level:Integer)`

1. For each pair (`i` , `j`) $\in \{0, 1, \ldots, 3^n - 1\} \times \{0, 1, \ldots, 3^n - 1\}$:
   (a) Call `isInSierpinskiCarpet(i, j)`. If it returns `true`, display a solid block at (`i`, `j`), otherwise, leave a blank space there.
2. **Exit**

`isInSierpinskiCarpet (x:Integer, y:Integer)`

1. If either of `x` or `y` is zero, the point (`x`, `y`) is on the edge of a square of some level. **Return** `true`.
2. If both `x` and `y` leave a remainder of one on division by 3, the point (`x`, `y`) is at the centre of a square of some level. **Return** `false`.
3. Call `isInSierpinskiCarpet(x / 3, y / 3)`, and **return** the returned value.

Source Code

```
1  public class SierpinskiCarpet {
2        public static void main (String[] args) {
3              try {
4                    /* Parse the first command line argument as the level of
                        detail of the carpet */
5                    int level = Integer.parseInt(args[0]);
6                    /* Make sure that the level is positive */
7                    if (level < 0)
8                          throw new NumberFormatException();
9                    /* Iterate over every 'point' in the carpet */
10                   for (int i = 0; i < Math.pow(3, level); i++) {
11                         for (int j = 0; j < Math.pow(3, level); j++) {
12                               /* Display a full block for points 'in' the
                                    carpet */
13                               System.out.print(isInSierpinskiCarpet(i, j)?
                                    "\u2588\u2588" : " ");
14                         }
```

```
15                              System.out.println();
16                      }
17              } catch (NumberFormatException | IndexOutOfBoundsException e) {
18                      /* Handle missing or incorrectly formatted arguments */
19                      System.out.println("Enter 1 argument
                            (order_of_carpet[integer])!");
20              }
21      }
22
23      /* Determines whether a point is in the carpet */
24      public static boolean isInSierpinskiCarpet (int x, int y) {
25              /* Blocks are in the carpet if they are on the edge */
26              if (x == 0 || y == 0)
27                      return true;
28              /* Blocks at the centres of 3-by-3 squares on any level are
29                 not in the carpet */
30              if (((x % 3) == 1) && ((y % 3) == 1))
31                      return false;
32              /* Recurse to the next, larger level */
33              return isInSierpinskiCarpet(x / 3, y / 3);
34      }
35  }
```

Variable Description

SierpinskiCarpet::main(String[])		
int	level	The depth to which to render the carpet
int	i, j	Counter variables, represent a point on the screen to be displayed

SierpinskiCarpet::isInSierpinskiCarpet(int, int)		
int	x, y	Counter variables, represent the point in question

"Computers are useless. They can only give you answers."

— Pablo Picasso

Problem 23 *Reverse Polish Notation (RPN)* or *postfix notation* is a mathematical notation for writing arithmetic expresssions in which operators follow their operands. Thus, as long as each operator has a fixed number of operands, the use of parentheses or rules of precedence are no longer required to write unambiguous expressions. For example, the expression `2 3 * 3 2 ^ 2 - *` evaluates to `42`.

Create a program capable of evaluating *RPN* expressions which use the following operators.

`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`^`	Exponentiation

Solution The nature of *RPN* lends itself to a very simple implementation with a stack for pushing operands into as they appear in an expression. When an operator is encountered, the required number of operands are popped from the stack, the operation is carried out, and the result is popped back into the stack. This continued until the entire expression has been parsed, leaving only the evaluated result in the stack.

`main (expression:String)`

1. Call `evaluateRPNExpression(expression)` and display the returned value.
2. **Exit**

`evaluateRPNExpression (expression:String)`

1. Split `expression` along whitespace into an array of tokens. Call it `tokens`.
2. Create a stack of floating points large enough to hold all elements in `tokens`. Call it `operandStack`.
3. For each string `token` $\in$ `tokens`:
   (a) If `token` is a floating point:
      i. Push `token` onto `operandStack`.
      ii. Get the next `token` from `tokens`.
      iii. Jump back to (3a).
   (b) Pop an operand from `operandStack` and call it `rightOperand`.
   (c) Pop another operand from `operandStack` and call it `leftOperand`.

(d) Depending on which operator `token` represents, evaluate the operation with `token` as the operator and `leftOperand` and `rightOperand` as the respective operands. Call it `result`.

(e) Push `result` onto `operandStack`.

4. Pop and operand from `operandStack` and **return** it.

Source Code

```
1  import java.util.Scanner;
2
3  public class RPNCalculator {
4         /* Simple stack setup */
5         private static double[] operandStack;
6         private static int top;
7
8         public static void main (String[] args) {
9                /* Prompt an RPN expression from the terminal */
10               System.out.printf("Reverse Polish Expression : ");
11               String expression = (new Scanner(System.in)).nextLine();
12               /* Evaluate the expression and display the result */
13               double result = evaluateRPNExpression(expression);
14               System.out.printf("Evaluated Expression  :   %s %n",
                     Double.toString(result));
15        }
16
17        /* Evaluates expression in RPN */
18        public static double evaluateRPNExpression (String expression) {
19               /* Split the expression into tokens */
20               String[] tokens = expression.split("\\s+");
21               /* Initialize the stack with an appropriately large capacity */
22               top = -1;
23               operandStack = new double[tokens.length];
24
25               /* Iterate through all tokens in the expression */
26               for (String token : tokens) {
27                      /* Push operands into the stack and continue */
28                      if (isDouble(token)) {
29                             pushOperand(Double.parseDouble(token));
30                             continue;
31                      }
32
33                      /* Pop operands from the stack */
34                      double rightOperand = popOperand();
35                      double leftOperand = popOperand();
```

```
36                      /* Determine the operator encountered and calculate the
                            appropriate result */
37                      double result = 0.0;
38                      switch (token.charAt(0)) {
39                              case '+' :    result = leftOperand + rightOperand;
40                                            break;
41                              case '-' :    result = leftOperand - rightOperand;
42                                            break;
43                              case '*' :    result = leftOperand * rightOperand;
44                                            break;
45                              case '/' :    result = leftOperand / rightOperand;
46                                            break;
47                              case '^' :    result = Math.pow(leftOperand,
                                    rightOperand);
48                                            break;
49                              default :     System.out.printf("Unknown operator
                                    (%s)!\n", token);
50                                            System.exit(0);
51                      }
52                      /* Push the result onto the stack */
53                      pushOperand(result);
54              }
55              /* Return the last item in the stack */
56              return popOperand();
57      }
58
59      /* Pushes an operand onto the stack */
60      private static void pushOperand (double n) {
61              operandStack[++top] = n;
62      }
63
64      /* Pops an operand from the stack. Exits on failure. */
65      private static double popOperand () {
66              if (top < 0) {
67                      System.out.println("Insufficient operands!");
68                      System.exit(0);
69              }
70              return operandStack[top--];
71      }
72
73      /* Determines whether a token is a number */
74      private static boolean isDouble (String n) {
75              try {
76                       Double.parseDouble(n);
77                       return true;
78              } catch (NumberFormatException e) {}
```

```
79                  return false;
80          }
81  }
```

Variable Description

RPNCalculator		
double[]	operandStack	The stack of operands in order of appearance.
int	top	The index of the topmost element of operandStack

RPNCalculator::main(String[])		
String	expression	The expression in RPN to be evaluated
double	result	The evaluated form of expression

RPNCalculator::evaluateRPNExpression(String)		
String	expression	The expression in RPN to be evaluated
String[]	tokens	The individual tokens in expression, separated by whitespace
String	token	An individual token from tokens
double	rightOperand	The operand to be taken on the right side of the operator
double	leftOperand	The operand to be taken on the left side of the operator
double	result	The result on evaluating the operator token on rightOperand and leftOperand

RPNCalculator::pushOperand(double)		
double	n	The operand to be pushed into operandStack

RPNCalculator::isDouble(String)		
String	n	The string to be tested on whether it is a floating point or not

"Computer Science is no more about computers than astronomy is about telescopes."

— **Edsger W. Dijkstra**

Problem 24 A *queue* is a linear data structure which allows storage and retrieval of elements in accordance with the *First In First Out (FIFO)* principle. Thus, elements exit a *queue* in the same order they entered it.

Implement a *queue* capable of holding an arbitrary number of elements of a specified type.

Solution The use of *linked lists*[12] is apropriate here. *Generics* ensure that once a queue is declared with a data type, only elements of that data type can be added to it, as opposed to merely storing `Objects`.

`Node<T> (item:T)`

1. Copy `item` as an object variable.
2. Declare two variables `left` and `right`, both of type `Node<T>`.
3. **Return** the resultant object.

`link (left:Node<T>, right:Node<T>)`

1. Set `left->right` to `right`.
2. Set `right->left` to `left`.

`LinkedQueue<T> ()`

1. Declare two constants `HEAD` and `TAIL`, both of type `Node<T>` with arbitrary data items.
2. Link `TAIL` and `HEAD`.
3. **Define** the functions:
   (a) `LinkedQueue<T>::enqueue(item)`
   (b) `LinkedQueue<T>::dequeue()`
   (c) `LinkedQueue<T>::peek()`
   (d) `LinkedQueue<T>::clear()`
   (e) `LinkedQueue<T>::isEmpty()`
   (f) `LinkedQueue<T>::size()`

[12] A linked list is a linear data structure where each element is a separate object, or *node*. Each *node* contains both *data* and *addresses* of the surrounding nodes.

4. **Return** the resultant object.

`LinkedQueue<T>::enqueue (item:T)`

1. Create a new `Node<T>`, pass it `item`, and call it `newNode`.
2. Link `HEAD->left` and `newNode`.
3. Link `newNode` and `HEAD`.

`LinkedQueue<T>::dequeue ()`

1. If the queue is empty, return `null`.
2. Temporarily store the node `TAIL->right` as `lastNode`.
3. Link `TAIL` and `lastNode->right`.
4. **Return** the item contained in `lastNode`.

`LinkedQueue<T>::peek ()`

1. **Return** the item in the node `TAIL->right`.

`LinkedQueue<T>::clear ()`

1. Link `TAIL` and `HEAD`.

`LinkedQueue<T>::isEmpty ()`

1. If the `TAIL->right` is `HEAD`, **return** `true`, otherwise **return** `false`.

`LinkedQueue<T>::size ()`

1. Initialize an integer `n` to zero.
2. Set a variable `current` to `TAIL`.
3. While `current->right` is not `HEAD`, set `current` to `current->right` and increment `n`.
4. **Return** `n`.

Source Code

```
1  public class Node<T> {
2        /* Item data is immutable */
3        protected final T item;
4
5        /* References to other nodes */
6        protected Node<T> left;
7        protected Node<T> right;
8
9        /* Set the data item */
```

```
10          public Node (T item) {
11                  this.item = item;
12          }
13
14          /* Get the data item */
15          public T getItem () {
16                  return item;
17          }
18
19          /* Use the data item's 'toString()' method */
20          @Override
21          public String toString () {
22                  return item.toString();
23          }
24
25          /* Doubly link two nodes */
26          public static <T> void link (Node<T> left, Node<T> right) {
27                  left.right = right;
28                  right.left = left;
29          }
30  }
```

```
1   import java.util.Iterator;
2
3   /* Use generics to allow arbitrary data typed queues, with type checking
4      enforced at compile-time */
5   public class LinkedQueue<T> implements Iterable<T> {
6          /* Special nodes surrounding data nodes */
7          protected final Node<T> HEAD = new Node<T>(null);
8          protected final Node<T> TAIL = new Node<T>(null);
9
10         public LinkedQueue () {
11                 Node.<T>link(TAIL, HEAD);
12         }
13
14         /* Enqueues a data item of generic type into the head */
15         public void enqueue (T item) {
16                 Node<T> newNode = new Node<T>(item);
17                 Node.<T>link(HEAD.left, newNode);
18                 Node.<T>link(newNode, HEAD);
19         }
20
21         /* Dequeues a data item from the tail */
22         public T dequeue () {
23                 if (this.isEmpty())
```

```
24                      return null;
25              Node<T> lastNode = TAIL.right;
26              Node.<T>link(TAIL, lastNode.right);
27              return lastNode.getItem();
28      }
29
30      /* Returns the data item at the tail without removing it */
31      public T peek () {
32              return TAIL.right.getItem();
33      }
34
35      /* Clears the queue */
36      public void clear () {
37              /* Garbage collection takes care of orphaned nodes */
38              Node.<T>link(TAIL, HEAD);
39      }
40
41      /* Checks if the queue is empty */
42      public boolean isEmpty () {
43              return TAIL.right == HEAD;
44      }
45
46      /* Returns the size of the queue */
47      public int size () {
48              int n = 0;
49              /* Start at the tail */
50              Node<T> current = TAIL;
51              /* Iterate through all nodes until the head */
52              while ((current = current.right) != HEAD)
53                      n++;
54              return n;
55      }
56
57      /* Formats the elements of the queue neatly */
58      @Override
59      public String toString () {
60              String[] elements = new String[this.size()];
61              Node<T> current = TAIL;
62              int n = 0;
63              while ((current = current.right) != HEAD)
64                      elements[n++] = current.toString();
65              return "[" + String.join(", ", elements) + "]";
66      }
67
68      /* Allow the elements of the queue to be iterated over simply */
69      @Override
```

```
70          public Iterator<T> iterator () {
71                  return new Iterator<T>() {
72                          private Node<T> current = TAIL.right;
73
74                          @Override
75                          public boolean hasNext () {
76                                  return current != HEAD;
77                          }
78
79                          @Override
80                          public T next () {
81                                  T item = current.getItem();
82                                  current = current.right;
83                                  return item;
84                          }
85
86                          @Override
87                          public void remove () {
88                                  throw new UnsupportedOperationException();
89                          }
90                  };
91          }
92  }
```

```
1   public class QueueDemo {
2           public static void main (String[] args) {
3                   /* Create an integer queue */
4                   LinkedQueue<Integer> q = new LinkedQueue<Integer>();
5
6                   /* Enqueue random numbers to the queue */
7                   for (int i = 0; i < (10 + (int) (10 * Math.random())); i++) {
8                           Integer n = (int) (100 * Math.random());
9                           System.out.printf("Enqueuing : %s\n", n);
10                          q.enqueue(n);
11                  }
12                  /* Demonstrate simple output formatting */
13                  System.out.printf("Queue[%2d] : %s\n", q.size(), q);
14
15                  /* Demonstrate peeking */
16                  System.out.printf("Number about to be dequeued : %s\n", q.peek());
17
18                  /* Demonstrate the FIFO principle in effect */
19                  System.out.println("(Dequeuing 10 numbers)");
20                  for (int i = 0; i < 10; i++)
21                          System.out.printf("Dequeuing : %s\n", q.dequeue());
```

```
22                System.out.printf("Queue[%2d] : %s\n", q.size(), q);
23
24                /* Demonstrate iteration until empty */
25                System.out.println("(Dequeueing until empty)");
26                while (!q.isEmpty())
27                       System.out.printf("Dequeuing : %s\n", q.dequeue());
28                System.out.printf("Queue[%2d] : %s\n", q.size(), q);
29        }
30  }
```

Variable Description

Node<T>		
T	item	The data stored in the node
Node<T>	left	Reference to the node to the left of this
Node<T>	right	Reference to the node to the right of this
LinkedQueue<T>		
Node<T>	HEAD	Special node, marks the point of entry of new data
Node<T>	TAIL	Special node, marks the point of exit of data
LinkedQueue<T>::enqueue(T)		
T	item	The data to be enqueued
Node<T>	newNode	The node containing the data to be enqueued
LinkedQueue<T>::dequeue()		
Node<T>	lastNode	The node containing the data to be dequeued
LinkedQueue<T>::size()		
int	n	Stores the number of elements in the queue
LinkedQueue<T>::toString()		
String[]	elements	Temporary array, stores the string representations of the data items in the queue
int	n	Counter variable

"A good way to have good ideas is by being unoriginal."

— Bram Cohen

Problem 25 A *double ended queue*, or *DEqueue* is a linear data structure which allows the insertion and deletion of data items from both the front and rear.

Implement a *double ended queue* capable of holding an arbitrary number of elements of a specified type.

Solution This problem can be solved by extending the funcitonality of the *queue* defined in the previous problem. The algorithms for insertion and deletion at one end mirror those for the other.

`LinkedDEQueue<T> ()`

1. Call the constructor of the superclass `LinkedQueue`.
2. **Define** the functions:
   (a) `LinkedDEQueue<T>::enqueueRear(item)`
   (b) `LinkedDEQueue<T>::dequeueFront()`
3. **Return** the resultant object.

`LinkedDEQueue<T>::enqueueRear (item:T)`

1. Create a new `Node<T>`, pass it `item`, and call it `newNode`.
2. Link `newNode` and `TAIL->right`.
3. Link `TAIL` and `newNode`.

`LinkedDEQueue<T>::dequeueFront ()`

1. If the queue is empty, return `null`.
2. Temporarily store the node `HEAD->left` as `firstNode`.
3. Link `firstNode->left` and `HEAD` .
4. **Return** the item contained in `firstNode`.

Source Code

```
1  import java.util.Iterator;
2
3  /* Extend LinkedQueue<T> to build on existing functionality */
4  public class LinkedDEQueue<T> extends LinkedQueue<T> {
5
6         /* Enqueues a data item of generic type into the tail */
7         public void enqueueRear (T item) {
8                Node<T> newNode = new Node<T>(item);
9                Node.<T>link(newNode, TAIL.right);
10               Node.<T>link(TAIL, newNode);
11        }
12
13        /* Dequeues a data item from the head */
14        public T dequeueFront () {
15               if (this.isEmpty())
16                      return null;
17               Node<T> firstNode = HEAD.left;
18               Node.<T>link(firstNode.left, HEAD);
19               return firstNode.getItem();
20        }
21
22        /* Descending iterator */
23        public Iterator<T> descendingIterator () {
24               return new Iterator<T>() {
25                      private Node<T> current = HEAD.left;
26
27                      @Override
28                      public boolean hasNext () {
29                             return current != TAIL;
30                      }
31
32                      @Override
33                      public T next () {
34                             T item = current.getItem();
35                             current = current.left;
36                             return item;
37                      }
38
39                      @Override
40                      public void remove () {
41                             throw new UnsupportedOperationException();
42                      }
43               };
44        }
```

```
45 }
```

```
1  public class DEQueueDemo {
2         public static void main (String[] args) {
3                 /* Create an integer DEqueue */
4                 LinkedDEQueue<Integer> dq = new LinkedDEQueue<Integer>();
5
6                 /* Enqueue random numbers to the front of the DEqueue */
7                 for (int i = 0; i < (7 + (int) (5 * Math.random())); i++) {
8                         Integer n = (int) (100 * Math.random());
9                         System.out.printf("Enqueuing (Front) : %s\n", n);
10                        dq.enqueue(n);
11                }
12                /* Enqueue random numbers to the rear of the DEqueue */
13                for (int i = 0; i < (7 + (int) (5 * Math.random())); i++) {
14                        Integer n = (int) (100 * Math.random());
15                        System.out.printf("Enqueuing ( Rear) : %s\n", n);
16                        dq.enqueueRear(n);
17                }
18                /* Demonstrate simple output formatting */
19                System.out.printf("DEQueue[%2d] : %s\n", dq.size(), dq);
20
21                /* Dequeue items from the front of the DEQueue */
22                System.out.println("(Dequeuing 10 numbers (Front))");
23                for (int i = 0; i < 10; i++)
24                        System.out.printf("Dequeuing : %s\n", dq.dequeueFront());
25                System.out.printf("Queue[%2d] : %s\n", dq.size(), dq);
26
27                /* Dequeue items from the rear of the DEQueue until empty */
28                System.out.println("(Dequeueing until empty (Rear))");
29                while (!dq.isEmpty())
30                        System.out.printf("Dequeuing : %s\n", dq.dequeue());
31                System.out.printf("DEQueue[%2d] : %s\n", dq.size(), dq);
32        }
33 }
```

Variable Description

LinkedDEQueue<T>::enqueueRear(T)		
T	item	The data to be enqueued

LinkedDEQueue<T>::dequeueFront()		
Node<T>	firstNode	The node containing the data to be dequeued

"You can't trust code that you did not totally create yourself."

— **Ken Thompson**

Problem 26 Arrange the words in a given sentence of input in alphabetical order. *(Ignore case, duplicated words.)*

Solution This problem can be solved using a data structure called a *binary tree*.

A *binary tree* consists of multiples *nodes*, each of which holds a data item. Ideally, these items can be *ordered*, i.e., there is a way to compare them, using a value called a *key*. Each node is connected to two nodes below it — the *left child* and the *right child*. The left child has lower *key*, while the right child has a higher *key* than the parent node. The node at the top of a given binary tree is called its *root*.

Binary trees have a nice recursive form, in that the left and right children of the root can be regarded as roots of individual binary trees — the *left* and *right subtrees* of the root. This makes it easy to write recursive algorithms for searching, inserting, and deleting nodes from a binary tree.

Searching and insertion in a binary tree containing n nodes have an average time complexity $O(\log n)$.

`TreeNode<T> (item:T)`

1. Copy `item` as an object variable.
2. Declare two variables `left` and `right`, both of type `Node<T>`.
3. **Return** the resultant object.

`BinaryTree<T> (root:TreeNode<T>)`

1. Copy `root` as an object variable.
2. **Define** the functions:
   (a) `BinaryTree<T>::contains(item)`
   (b) `BinaryTree<T>::search(item)`
   (c) `BinaryTree<T>::add(item)`
3. **Return** the resultant object.

`BinaryTree<T>::contains (item:T)`

1. If `this->search(item)` returns a non-null object, **return** `true`, otherwise **return** `false`.

`BinaryTree<T>::search (item:T)`

1. **Return** `search(this->root, item)`

`BinaryTree<T>::add (item:T)`

1. Set `this->root` to the `TreeNode` returned by `add(this->root, item)`.

`search (root:TreeNode<T>, item:T)`

1. If `item` < `root->item`, **return** `search(root->left, item)`
2. If `item` > `root->item`, **return** `search(root->right, item)`
3. **Return** `root`

`add (root:TreeNode<T>, item:T)`

1. If `root` is `null`, set it to a new `TreeNode<T>` containing `item` and **return** `root`.
2. If `item` < `root->item`, set `root->left` to `add(root->left, item)`.
3. If `item` > `root->item`, set `root->right` to `add(root->right, item)`.
4. **Return** `root`

`traverseInOrder (node:TreeNode<T>)`

1. If `node` is `null`, **return** an empty string.
2. **Return** `traverseInOrder(node->left)` + `node` + `traverseInOrder(node->right)` *(with spacing as necessary)*.

Source Code

```
1  public class TreeNode<T extends Comparable<T>> {
2        /* Item data is immutable */
3        protected final T item;
4
5        /* References to child nodes */
6        public TreeNode<T> left;
7        public TreeNode<T> right;
8
9        /* Set the data item */
10       public TreeNode (T item) {
11             this.item = item;
12             this.left = null;
13             this.right = null;
14       }
15
16       /* Get the data item */
17       public T getItem () {
18             return item;
```

```
        }

        /* Use the data item's 'toString()' method */
        @Override
        public String toString () {
                return item.toString();
        }
}
```

```
public class BinaryTree<T extends Comparable<T>> {
        /* The root node is at the top of all other nodes */
        protected TreeNode<T> root;

        public BinaryTree (TreeNode<T> root) {
                this.root = root;
        }

        /* Default to a 'null' root node */
        public BinaryTree () {
                this(null);
        }

        /* Checks whether the tree contains a given item */
        public boolean contains (T item) {
                return this.search(item) != null;
        }

        /* Returns the node containing a given item. If not found, returns 'null' */
        public TreeNode<T> search (T item) {
                return BinaryTree.<T>search(root, item);
        }

        /* Adds an item to the tree in order, if not already present */
        public void add (T item) {
                root = BinaryTree.<T>add(root, item);
        }

        /* Formats the items in the tree neatly, in order */
        @Override
        public String toString () {
                return BinaryTree.<T>traverseInOrder(this.root).trim();
        }

        /* Recursive binary search */
        public static <T extends Comparable<T>> TreeNode<T> search (TreeNode<T>
```

```
            root, T item) {
37              if (item.compareTo(root.item) < 0)
38                      return BinaryTree.<T>search(root.left, item);
39              if (item.compareTo(root.item) > 0)
40                      return BinaryTree.<T>search(root.right, item);
41              return root;
42      }
43
44      /* Recursive insertion of a node in a binary tree */
45      public static <T extends Comparable<T>> TreeNode<T> add (TreeNode<T> root, T
            item) {
46              if (root == null)
47                      root = new TreeNode<T>(item);
48              else if (item.compareTo(root.item) < 0)
49                      root.left = BinaryTree.<T>add(root.left, item);
50              else if (item.compareTo(root.item) > 0)
51                      root.right = BinaryTree.<T>add(root.right, item);
52              return root;
53      }
54
55      /* Recursive in order traversal of a binary tree */
56      public static <T extends Comparable<T>> String traverseInOrder (TreeNode<T>
            node) {
57              if (node == null)
58                      return "";
59              return traverseInOrder(node.left) + " "
60                      + node + " "
61                      + traverseInOrder(node.right);
62      }
63  }
```

```
1   import java.util.Scanner;
2
3   public class BinaryTreeDemo {
4       public static void main (String[] args) {
5               /* Create a binary tree which holds strings */
6               BinaryTree<String> bTree = new BinaryTree<String>();
7
8               /* Get a line of input */
9               System.out.print("Enter a sentence : ");
10              String sentence = (new Scanner(System.in)).nextLine();
11
12              /* Only retain letters */
13              sentence = sentence.toUpperCase().replaceAll("[^A-Z]", " ");
14
```

```
15              /* Insert each word into the tree. This implicitly sorts them. */
16              for (String word : sentence.split("\\s+"))
17                      bTree.add(word);
18
19              /* In order traversal of the tree */
20              System.out.print("Sorted words : ");
21              System.out.println(bTree);
22      }
23  }
```

Variable Description

TreeNode<T>		
T	item	The data stored in the node
TreeNode<T>	left	Reference to the left child of this
TreeNode<T>	right	Reference to the right child of this
BinaryTree<T>		
TreeNode<T>	root	The root node of the binary tree
BinaryTree<T>::contains(T)		
T	item	The item to check for
BinaryTree<T>::search(T)		
T	item	The item to search for
BinaryTree<T>::add(T)		
T	item	The item to be added
BinaryTree<T>::search(TreeNode<T>, T)		
TreeNode<T>	root	The current node being checked
T	item	The item to search for
BinaryTree<T>::add(TreeNode<T>, T)		
TreeNode	root	The current node being compared
T	item	The item to be added

"One should always play fairly when one has the winning cards."

— Oscar Wilde

Problem 27 Simulate a deck of playing cards.

Solution A deck of cards can be simulated by a list of 'Card' objects. A playing card is wholly defined by its *suit*, of which there are 4, and its *rank*, of which there are 12. A standard deck contains 52 cards, such that every permutation of suit and rank is present. Cards can only be dealt from a deck, or shuffled in the deck.

There are many algorithms for shuffling a list, but the simplest is the *Knuth shuffle*, also known as the *Fisher-Yates shuffle*. It involves choosing a random card from the list, putting it aside, then repeating until the list is exhausted. This generates an *unbiased permutation* of the list.

`Card (suit:Suit, rank::Rank)`

1. Copy `suit` and `rank` as constants into the object.
2. **Return** the resultant object.

`Deck ()`

1. Create a stack of `Card` objects of capacity 52.
2. For each ordered pair (`s`,`r`) ∈`Suit->values()`×`Rank->values()`:
   (a) Create a new `Card`, pass it `s` and `r`, and add it to the card stack.
3. **Define** the functions:
   (a) `Deck::deal()`
   (b) `Deck::shuffle()`
4. **Return** the resultant object.

`Deck::deal ()`

1. If there are no cards in the stack, **return** a `null` object.
2. Pop a card from the stack and **return** it.

`Deck::shuffle ()`

1. Let there be n cards in the stack.
2. For each `i` $\in \{n-1, n-2, \ldots, 1\}$:
   (a) Let `j` be a random integer such that $0 \leq$`j`$\leq$`i`.
   (b) Swap the cards at indices `i` and `j` in the stack.

Source Code

```
/* List all possible suits */
public enum Suit {
	SPADES,
	HEARTS,
	DIAMONDS,
	CLUBS;
}
```

```
/* List all possible ranks, along with their equivalent numeric values */
public enum Rank {
	ACE    (1),
	TWO    (2),
	THREE  (3),
	FOUR   (4),
	FIVE   (5),
	SIX    (6),
	SEVEN  (7),
	EIGHT  (8),
	NINE   (9),
	TEN    (10),
	JACK   (11),
	QUEEN  (12),
	KING   (13);

	protected int value;

	Rank (int value) {
		this.value = value;
	}

	public int getValue () {
		return this.value;
	}
}
```

```
/* Abstraction of a standard playing card */
public class Card {
	/* Each card has an immutable suit and rank */
	public final Suit suit;
	public final Rank rank;

	/* Short names of cards */
	public static final String rankShort = " A 2 3 4 5 6 7 8 910 J Q K";
```

```
9
10          public Card (Suit suit, Rank rank) {
11                  this.suit = suit;
12                  this.rank = rank;
13          }
14
15          /* Formats the card details neatly */
16          @Override
17          public String toString () {
18                  return rank + " of " + suit;
19          }
20
21          /* Formats the card as a 2-character string */
22          public String toStringShort () {
23                  int r = rank.getValue();
24                  String rs = rankShort.substring(2 * r, 2 * (r + 1)).trim();
25                  char ss = suit.toString().charAt(0);
26                  return rs + ss;
27          }
28  }
```

```
1   /* Abstraction of a deck of cards */
2   public class Deck {
3           /* Setup a simple stack */
4           protected Card[] cards;
5           protected int top;
6
7           public Deck () {
8                   cards = new Card[52];
9                   top = -1;
10                  /* Initialize a full deck */
11                  for (Suit suit : Suit.values())
12                          for (Rank rank : Rank.values())
13                                  cards[++top] = new Card(suit, rank);
14          }
15
16          /* Checks if the deck is empty */
17          public boolean isEmpty () {
18                  return top < 0;
19          }
20
21          /* Returns the number of cards in the deck */
22          public int size () {
23                  return top + 1;
24          }
```

```
25
26          /* Pops the topmost card from the deck */
27          public Card deal () {
28                  if (this.isEmpty())
29                          return null;
30                  return cards[top--];
31          }
32
33          /* Shuffles the deck using the Fisher-Yates, or Knuth shuffle */
34          public void shuffle () {
35                  for (int i = top; i > 0; i--) {
36                          int j = random(0, i + 1);
37                          swap(i, j);
38                  }
39          }
40
41          /* Utility method for swapping cards in the deck */
42          private void swap (int i, int j) {
43                  Card t = cards[i];
44                  cards[i] = cards[j];
45                  cards[j] = t;
46          }
47
48          /* Format the cards in the deck neatly */
49          @Override
50          public String toString () {
51                  if (this.isEmpty())
52                          return "[]";
53                  String s = "[";
54                  for (int i = top; i >= 0; i--)
55                          s += cards[i].toStringShort() + ", ";
56                  return s.substring(0, s.length() - 2) + "]";
57
58          }
59
60          /* Utility method for generating random integers in a given range */
61          private static int random (int lo, int hi) {
62                  return (int) (lo + (Math.random() * (hi - lo)));
63          }
64  }
```

```
1   public class DeckDemo {
2           public static void main (String[] args) {
3                   /* Create a new deck of cards in standard order */
4                   Deck d = new Deck();
```

```
5               System.out.println(d);
6
7               /* Shuffle the deck */
8               d.shuffle();
9               System.out.println(d);
10
11              /* Deal out 26 cards */
12              for (int i = 0; i < 26; i++)
13                      System.out.println(d.deal());
14
15              /* Show the deck */
16              System.out.println(d);
17      }
18  }
```

Variable Description

Card		
Suit	suit	The suit of the playing card
Rank	rank	The rank of the playing card

Deck		
Card[]	cards	The stack of cards making up the deck
int	top	The index of the card at the top of the stack
Suit	suit	The suit of the playing card being added
Rank	rank	The rank of the playing card being added

Deck::shuffle()		
int	i, j	The indices of the cards to be swapped

Deck::swap(int, int)		
int	i, j	The indices of the cards to be swapped

"Code never lies, comments sometimes do."

— **Ron Jeffries**

Problem 28 Remove all comments from given source code.

Solution Java comments can be classified into two broad types — single line comments beginning with the sequence '`//`' and ending with a newline, and multiple line comment beginning with the sequence '`/*`' and ennding with the sequence '`*/`'. Care must be taken to ignore such sequences within quotes *both single and double*, as well as within other comments. Escape sequences also have to be dealt with.

While parsing the given source code character by character, it becomes necessary to keep track of a *state variable*. This will store information about what is currently being parsed, and different sets of checks are executed accordingly. Java *enums*, or *enumerated lists*, are ideal for this purpose.

`main (filename:String)`

1. Create a `ReadSourceFile` object called `s`, and pass it `filename` and a buffer size of 10.
2. Declare a state variable called `currentState`, and set it to `SOURCE`.
3. Declare a character called `matchingQuotes`, and set it to a black space.
4. **While** `s->hasNextChar()`:
   (a) Store the character returned by `s->getChar()` as `c`.
   (b) If `c` is a backslash, display it, get another character from `s->getChar()`, display that, and jump to (4).
   (c) If `currentState` is `SOURCE`:
      i. If `c` is a quotation mark, set `currentState` to `QUOTES`, set `matchingQuotes` to `c`. Display `c` and jump to (4).
      ii. If `c` is a forward slash, get another character called `n` from `s->getChar()`.
         A. If `n` is an asterisk, set `currentState` to `MULTIPLE_LINE_COMMENT`.
         B. If `n` ia another forward slash, set `currentState` to `SIMGLE_LINE_COMMENT`.
         C. If none of the above, call `s->putChar(n)`, display `c` and jump to (4).
      iii. If none of the above, display `c` and jump to (4).
   (d) If `currentState` is `SIMGLE_LINE_COMMENT` and `c` is a newline, set `currentState` to `SOURCE`, display `c` and jump to (4).
   (e) If `currentState` is `MULTIPLE_LINE_COMMENT` and `c` is an asterisk:
      i. Get another character called `n` from `s->getChar()`.

ii. If `n` is a forward slash, set `currentState` to `SOURCE`.
iii. Jump to (4).

(f) If `currentState` is `QUOTES` and `c` is equal to `matchingQuotes`, set `currentState` to `SOURCE`, `matchingQuotes` to an blank space. Display `c` and jump to (4).

(g) If none of the above, display `c`.

`ReadSourceFile (filename:String, bufferSize:integer)`

1. Initialize a new `FileReader` *unbuffered* called `fileReader` and pass it `filename`.
2. Create a simple *buffer* of integers, implemented using a stack. *This will store characters, but the* ***`char`*** *data type cannot store special characters, such as the character which indicates the end of a file.*
3. **Define** the functions:
   (a) `ReadSourceFile::hasNextChar()`
   (b) `ReadSourceFile::getChar()`
   (c) `ReadSourceFile::putChar(c)`
4. **Return** the resultant object.

`ReadSourceFile::hasNextChar ()`

1. Read a new character from `fileReader`, and call it `c`.
2. If `c` is equal to -1, **return** `false`.
3. Call `this->putChar(c)`
4. **Return** `true`

`ReadSourceFile::getChar ()`

1. If the buffer has some characters, pop one off and **return** it.
2. Read a character from `fileReader` and **return** it.

`ReadSourceFile::putChar (c:Integer)`

1. If the buffer has space, push `c` onto it and **return** `true`. Otherwise, **return** `false`.

Source Code

```
/* List of possible states */
public enum State {
	SOURCE, SINGLE_LINE_COMMENT, MULTIPLE_LINE_COMMENT, QUOTES;
}
```

```
import java.io.IOException;
import java.io.FileReader;

public class ReadSourceFile {
	protected String filename;

	/* Setup a simple stack as a buffer */
	protected int[] buffer;
	protected int top;

	/* Use a FileReader to collect input */
	protected FileReader fileReader;

	/* Sets the filename and the buffer size */
	public ReadSourceFile (String filename, int bufferSize) throws IOException {
		this.filename = filename;
		this.buffer = new int[bufferSize];
		this.top = -1;
		this.fileReader = new FileReader(filename);
	}

	/* Checks whether there are more characters */
	public boolean hasNextChar () throws IOException {
		/* Read a character */
		int c = fileReader.read();
		if (c == -1)
			return false;
		/* Push the character onto the buffer */
		putChar(c);
		return true;
	}

	/* Returns the next character in the file */
	public int getChar () throws IOException {
		/* Pop from the buffer */
		if (top >= 0)
			return buffer[top--];
		/* Read directly from file */
		return fileReader.read();
```

```
        }

        /* Pushes a character onto the buffer */
        public boolean putChar (int c) {
                /* Check for stackoverflow */
                if (top == (buffer.length - 1))
                        return false;
                buffer[++top] = c;
                return true;
        }

        /* Close all resources */
        public void close () throws IOException {
                fileReader.close();
        }
}
```

```
import java.io.IOException;

public class RemoveComments {
        public static void main (String[] args) throws IOException {
                /* Parse first command line argument as the file to read from.
                   Allocate a small buffer */
                ReadSourceFile s = new ReadSourceFile(args[0], 10);

                /* Initialize the current state to plain source */
                State currentState = State.SOURCE;
                /* Initialize the current matching quote to empty */
                char matchingQuotes = ' ';

                /* Loop through all characters */
                while (s.hasNextChar()) {
                        /* Get a character from the file */
                        char c = (char) s.getChar();

                        /* Escaped characters - display the backslash and the
                            following character */
                        if (c == '\\') {
                                System.out.print(c + "" + ((char) s.getChar()));
                                continue;
                        }
                        switch (currentState) {
                                case SOURCE:
                                        switch (c) {
                                                /* Single and double opening quotes */
```

```
28                          case '\"':
29                          case '\'':
30                                 /* Set the new state */
31                                 currentState = State.QUOTES;
32                                 /* Set the matching closing quote */
33                                 matchingQuotes = c;
34                                 System.out.print(c);
35                                 break;
36                          /* Possible comment */
37                          case '/':
38                                 char n = (char) s.getChar();
39                                 if (n == '*')
40                                        currentState =
                                           State.MULTIPLE_LINE_COMMENT;
41                                 else if (n == '/')
42                                        currentState =
                                           State.SINGLE_LINE_COMMENT;
43                                 else {
44                                        s.putChar(n);
45                                        System.out.print(c);
46                                 }
47                                 break;
48                          default:
49                                 System.out.print(c);
50                   }
51                   break;
52            case SINGLE_LINE_COMMENT:
53                   /* Exit state to plain source on newline */
54                   if (c == '\n') {
55                          currentState = State.SOURCE;
56                          System.out.print(c);
57                   }
58                   break;
59            case MULTIPLE_LINE_COMMENT:
60                   /* Exit state to plain source on closing
                        characters */
61                   if (c == '*') {
62                          char n = (char) s.getChar();
63                          if (n == '/')
64                                 currentState = State.SOURCE;
65                   }
66                   break;
67            case QUOTES:
68                   /* Exit state on encountering closing quote */
69                   if (c == matchingQuotes) {
70                          currentState = State.SOURCE;
```

```
71                                          matchingQuotes = ' ';
72                                      }
73                                      /* Display anything in quotes verbatim */
74                                      System.out.print(c);
75                                      break;
76                                  default:
77                                      System.out.print(c);
78                              }
79                          }
80                          s.close();
81                  }
82          }
```

Variable Description

ReadSourceFile		
String	filename	The file containing the source code to be read
int[]	buffer	The stack of characters read from the file
int	top	The index of the character at the top of the buffer

RemoveComments::main(String[])		
ReadSource File	s	The source file reader
State	currentState	Indicates the type of code currently being parsed
char	matchingQuotes	Indicates the type of ending quote which pairs with the opening quote, if currently inside a string in the source code
char	c, n	Stores the current and next characters in the source code being parsed

"A program that produces incorrect results twice as fast is infinitely slower."

— John Ousterhout

Problem 29 Compare the runtimes of the following sorting algorithms — *bubble sort*, *insertion sort* and *quicksort*.

Solution *Bubble sort* is a sorting algorithm which repeatedly steps through an unsorted list, compares adjacent elements and swaps them if they are in the wrong order. It has an average time complexity of $O(n^2)$.

Insertion sort is a sorting algorithm which builds a sorted list one element at a time by repeatedly selecting an unsorted element and inserting it into the correct position in the sorted portion. It too has an average time complexity of $O(n^2)$

Quicksort is a *divide and conquer* sorting algorithm which splits an unsorted list along a pivot, with elements less than it shifted before and elements greater than it shifted after. The two halves are then sorted recursively. This algorithm has an average time complexity of $O(n \log n)$.

Each of these algorithms have different strengths and weaknesses. *Insertion sort* and *bubble sort* perform progressively slower than *quicksort* on long lists with a large spread of randomly shuffled numbers. On the other hand, *insertion sort* performs faster than *bubble sort*, which in turn performs faster than quicksort on shorter lists with randomly shuffled numbers. Again, *bubble sort* performs faster than *insertion sort*, which performs significantly faster on long lists with a small spread of numbers, i.e., almost sorted lists.

`BubbleSorter::sort (a:Integer[])`

1. Initialize an integer `right` to the length of `a`.
2. Initialize a boolean `swapped` to `true`.
3. **While** `swapped`:
   (a) Set `swapped` to `false`
   (b) For `i` $\in \{1, 2, \ldots, right - 1\}$:
      i. If `a[i - 1]` > `a[i]`:
         A. Swap the elements in `a` at indices `i-1` and `i`.
         B. Set `swapped` to `true`.
   (c) Decrement `right`.

`InsertionSorter::sort (a:Integer[])`

1. Let n be thenumber of elements in `a`.
2. For `i` $\in \{1, 2, \ldots, n-1\}$:
   (a) Set an integer `k` to `a[i]`.
   (b) Set an integer `j` to `i - 1`.
   (c) **While** (`j` ≥ 0) and (`a[j]` $>$ `k`):
      i. Set `a[j + 1]` to `a[j]`.
      ii. Decrement `j`.
   (d) Set `a[j + 1]` to `k`.

`QuickSorter::sort (a:Integer[])`

1. Let `l` be the number of elements in `a`.
2. Call `this->sort(a, 0, l - 1)`

`QuickSorter::sort (a:Integer[], lo:Integer, hi:Integer)`

1. If `hi` $\leq$ `lo`, **return**.
2. Call `this->partition(a, lo, hi)`, and store the returned integer as `pivot`.
3. Call `this->sort(a, lo, pivot - 1)`
4. Call `this->sort(a, pivot + 1, hi)`

`QuickSorter::partition (a:Integer[], lo:Integer, hi:Integer)`

1. Set an integer `pivotValue` to `a[hi]`.
2. Set an integer `pivot` to `lo - 1`.
3. For `i` $\in \{$`lo`,`lo + 1`,$\ldots$,`hi - 1` $\}$:
   (a) If `a[i]` $\leq$ `pivotValue`:
      i. Increment `pivot`.
      ii. Swap the elements in `a` at indices `i` and `pivot`.
4. Increment `pivot`.
5. Swap the elements in `a` at indices `i` and `pivot`.
6. **Return** `pivot`

Source Code

```
/* Abstract integer array sorter */
public abstract class IntegerArraySorter {
	/* Each sorter has a common sort method */
	public abstract void sort (int[] a);

	/* Utility method for swapping elements in an array */
	public static void swap (int[] a, int i, int j) {
		int t = a[i];
		a[i] = a[j];
		a[j] = t;
	}
}
```

```
public class BubbleSorter extends IntegerArraySorter {
	@Override
	public void sort (int[] a) {
		int right = a.length;
		boolean swapped = true;
		while (swapped) {
			swapped = false;
			for (int i = 1; i < right; i++) {
				if (a[i - 1] > a[i]) {
					swap(a, i - 1, i);
					swapped = true;
				}
			}
			right -= 1;
		}
	}

	@Override
	public String toString () {
		return "BubbleSort";
	}
}
```

```
public class InsertionSorter extends IntegerArraySorter {
	@Override
	public void sort (int[] a) {
		for (int i = 1; i < a.length; i++) {
			int k = a[i];
			int j = i - 1;
			while ((j >= 0) && (a[j] > k)) {
```

```
                        a[j + 1] = a[j];
                        j -= 1;
                    }
                    a[j + 1] = k;
                }
        }

        @Override
        public String toString () {
                return "InsertionSort";
        }
}
```

```
public class QuickSorter extends IntegerArraySorter {
        @Override
        public void sort (int[] a) {
                sort(a, 0, a.length - 1);
        }

        /* Recursive quicksort */
        private void sort (int[] a, int lo, int hi) {
                if (hi <= lo)
                        return;
                int pivot = partition(a, lo, hi);
                sort(a, lo, pivot - 1);
                sort(a, pivot + 1, hi);
        }

        /* Lomuto partition scheme */
        private int partition (int[] a, int lo, int hi) {
                int pivotValue = a[hi];
                int pivot = lo - 1;
                for (int i = lo; i < hi; i++)
                        if (a[i] <= pivotValue)
                                swap(a, i, ++pivot);
                swap(a, hi, ++pivot);
                return pivot;
        }

        @Override
        public String toString () {
                return "QuickSort";
        }
}
```

```java
public class SortCompare {
	public static void main (String[] args) {
		/* Initialize the sorters */
		IntegerArraySorter[] sorters = {
			new BubbleSorter(),
			new InsertionSorter(),
			new QuickSorter()
		};

		/* Parse the first command line argument as the length of the list */
		int length = Integer.parseInt(args[0]);

		/* Parse the second command line argument as the upper bound of
		    integers in the list */
		int range = Integer.parseInt(args[1]);

		/* Create a random list */
		int[] a = randomArray(length, range);

		long t0 = 0, t1 = 0;
		for (IntegerArraySorter s : sorters) {
			/* Clone the list */
			int[] b = a.clone();

			/* Start the timer */
			t0 = System.nanoTime();

			/* Sort */
			s.sort(b);

			/* Stop the timer */
			t1 = System.nanoTime();

			System.out.printf("%16s : %16d ns\n", s, t1 - t0);
		}
	}

	/* Generates a random array of given length and given upper limit of random
	    numbers */
	public static int[] randomArray (int length, int hi) {
		int[] a = new int[length];
		for (int i = 0; i < length; i++)
			a[i] = (int) (Math.random() * hi);
		return a;
	}
}
```

Variable Description

`IntegerArraySorter::sort(int[])`		
`int[]`	`a`	The array whose elements are to be sorted

`IntegerArraySorter::swap(int[], int, int)`		
`int[]`	`a`	The array whose elements are to be swapped
`int`	`i, j`	The indices of the elements to be swapped

`BubbleSorter::sort(int[])`		
`int[]`	`a`	The array whose elements are to be sorted
`int`	`right, i`	Counter variables
`boolean`	`swapped`	Keeps track of whether any swaps were performed in the current iteration

`InsertionSorter::sort(int[])`		
`int[]`	`a`	The array whose elements are to be sorted
`int`	`i, j`	Counter variables
`int`	`k`	The element to be inserted

`QuickSorter::sort(int[])`		
`int[]`	`a`	The array whose elements are to be sorted

`QuickSorter::sort(int[], int, int)`		
`int[]`	`a`	The array whose elements are to be sorted
`int`	`lo, hi`	The lower and upper indices of the unsorted list
`int`	`pivot`	The index of the value about which the list is partitioned

`QuickSorter::partition(int[], int, int)`		
`int[]`	`a`	The array whose elements are to be sorted
`int`	`lo, hi`	The lower and upper indices of the unsorted list
`int`	`pivotValue`	The value about which the list is partitioned
`int`	`pivot`	The index of the value about which the list is partitioned
`int`	`i`	Counter variable

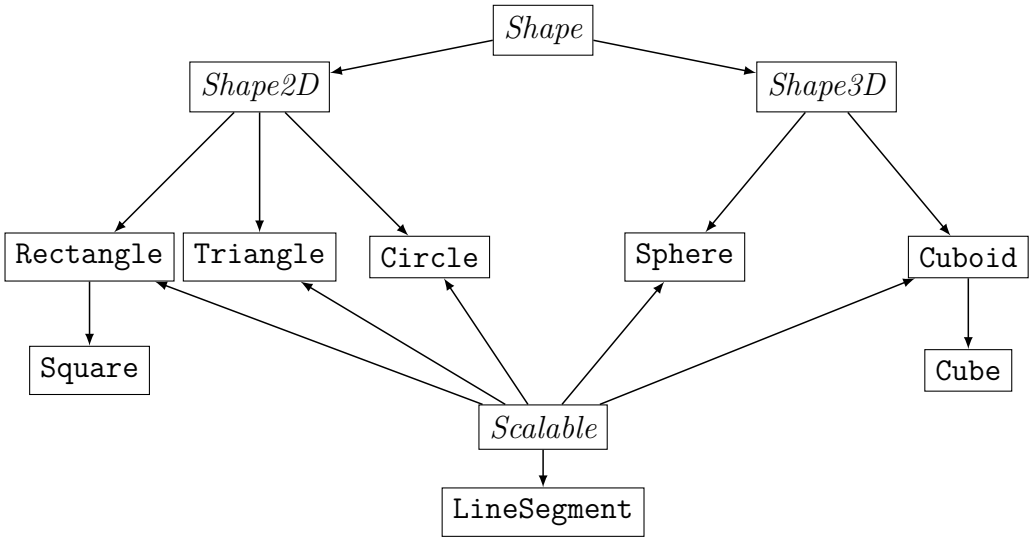
"Design is choosing how you will fail."

— **Ron Fein**

Problem 30 Showcase *class inheritance* in *Java*, by designing a hierarchy of geometric shapes.

Solution Here, we use an interface `Shape` as the superclass of the interfaces `Shape2D` and `Shape3D`, each of which has subclasses sharing common behaviour. For example, all $2D$ shapes have computable areas and perimeters, while all $3D$ shapes have computable volumes and surface areas. This structure illustrates *multilevel inheritance*.

All these shapes can be *scaled*, i.e., their dimensions can be changed by some factor. This behaviour is defined by the interface `Scalable`, which the above classes all implement. This structure illustrates *multiple inheritance*. The class `LineSegment` is also `Scalable`, despite not being a `Shape`.



Following is a general implementation of a shape '`MyShape`', which has a computable property '`myProperty`' and is `Scalable`.

`MyShape (parameters... :Number)`

1. Copy each `parameter` as an immutable constant into the object data.
2. **Define** the functions:
   (a) `MyShape::getMyProperty()`
   (b) `MyShape::scale(scaleFactor())`
3. **Return** the resultant object.

`MyShape::getMyProperty ()`

1. Compute `myProperty` using the `parameters`, and **return** the result.

`MyShape::scale (scaleFactor:FloatingPoint)`

1. Create a new `MyShape`, whose `parameters` are the parameters of `this` multiplied by `scaleFactor`, and `return` it.

Source Code

```
1  public interface Scalable<T> {
2        public T scale (double scaleFactor);
3  }
```

```
1  public class LineSegment implements Scalable<LineSegment> {
2        protected final double length;
3
4        public LineSegment (double length) {
5              this.length = length;
6        }
7
8        @Override
9        public LineSegment scale (double scaleFactor) {
10             return new LineSegment(length * scaleFactor);
11       }
12
13       @Override
14       public String toString () {
15             return String.format("LineSegment (length = %f)", length);
16       }
17 }
```

```
1  public interface Shape {}
```

```
1  public interface Shape2D extends Shape {
```

```
        public double getArea ();
        public double getPerimeter ();
}
```

```
public class Circle implements Shape2D, Scalable<Circle> {
        protected final double radius;

        public Circle (double radius) {
                this.radius = radius;
        }

        @Override
        public double getArea () {
                return Math.PI * radius * radius;
        }

        @Override
        public double getPerimeter () {
                return 2 * Math.PI * radius;
        }

        @Override
        public Circle scale (double scaleFactor) {
                return new Circle(radius * scaleFactor);
        }

        @Override
        public String toString () {
                return String.format("Circle (radius = %f)", radius);
        }
}
```

```
public class Triangle implements Shape2D, Scalable<Triangle> {
        protected final double a;
        protected final double b;
        protected final double c;

        public Triangle (double a, double b, double c) {
                this.a = a;
                this.b = b;
                this.c = c;
        }

        @Override
        public double getArea () {
```

```
14              double s = (a + b + c) / 2.0;
15              return Math.sqrt(s * (s - a) * (s - b) * (s - c));
16      }
17
18      @Override
19      public double getPerimeter () {
20              return a + b + c;
21      }
22
23      @Override
24      public Triangle scale (double scaleFactor) {
25              return new Triangle(a * scaleFactor, b * scaleFactor, c *
                    scaleFactor);
26      }
27
28      @Override
29      public String toString () {
30              return String.format("Triangle (sides = {%f, %f, %f})", a, b, c);
31      }
32  }
```

```
1   public class Rectangle implements Shape2D, Scalable<Rectangle> {
2       protected final double length;
3       protected final double breadth;
4
5       public Rectangle (double length, double breadth) {
6               this.length = length;
7               this.breadth = breadth;
8       }
9
10      @Override
11      public double getArea () {
12              return length * breadth;
13      }
14
15      @Override
16      public double getPerimeter () {
17              return 2 * (length + breadth);
18      }
19
20      @Override
21      public Rectangle scale (double scaleFactor) {
22              return new Rectangle(length * scaleFactor, breadth * scaleFactor);
23      }
24
```

```
25          @Override
26          public String toString () {
27                  return String.format("Rectangle (length = %f, breadth = %f)", length,
                        breadth);
28          }
29  }
```

```
1   public class Square extends Rectangle {
2           public Square (double side) {
3                   super(side, side);
4           }
5
6           @Override
7           public Square scale (double scaleFactor) {
8                   return new Square(length * scaleFactor);
9           }
10
11          @Override
12          public String toString () {
13                  return String.format("Square (side = %f)", length);
14          }
15  }
```

```
1   public interface Shape3D extends Shape {
2           public double getVolume ();
3           public double getSurfaceArea ();
4   }
```

```
1   public class Sphere implements Shape3D, Scalable<Sphere> {
2           protected final double radius;
3
4           public Sphere (double radius) {
5                   this.radius = radius;
6           }
7
8           @Override
9           public double getVolume () {
10                  return 4.0 * Math.PI * radius * radius * radius / 3.0;
11          }
12
13          @Override
14          public double getSurfaceArea () {
15                  return 4 * Math.PI * radius * radius;
16          }
17
```

```
        @Override
        public Sphere scale (double scaleFactor) {
                return new Sphere(radius * scaleFactor);
        }

        @Override
        public String toString () {
                return String.format("Sphere (radius = %f)", radius);
        }
}
```

```
public class Cuboid implements Shape3D, Scalable<Cuboid> {
        protected final double length;
        protected final double breadth;
        protected final double height;

        public Cuboid (double length, double breadth, double height) {
                this.length = length;
                this.breadth = breadth;
                this.height = height;
        }

        @Override
        public double getVolume () {
                return length * breadth * height;
        }

        @Override
        public double getSurfaceArea () {
                return 2.0 * ((length * breadth) + (breadth * height) + (height *
                    length));
        }

        @Override
        public Cuboid scale (double scaleFactor) {
                return new Cuboid(length * scaleFactor, breadth * scaleFactor, height
                    * scaleFactor);
        }

        @Override
        public String toString () {
                return String.format("Cuboid (length = %f, breadth = %f, height =
                    %f)", length, breadth, height);
        }
}
```

```java
public class Cube extends Cuboid {
	public Cube (double side) {
		super(side, side, side);
	}

	@Override
	public Cube scale (double scaleFactor) {
		return new Cube(length * scaleFactor);
	}

	@Override
	public String toString () {
		return String.format("Cube (side = %f)", length);
	}
}
```

```java
public class ShapeDemo {
	public static void main (String[] args) {
		/* Shapes of all kinds can be stored under the same type - Shape */
		Shape[] shapes = {
			new Circle(1.0),
			new Cube(2.0),
			new Triangle(3.0, 4.0, 5.0)
		};

		/* Each shape overrides the toString() method */
		System.out.println("Shapes :");
		for (Shape s : shapes)
			System.out.println(s);
		System.out.println();

		/* 2D shapes can be stored under the same type - Shape2D */
		Shape2D[] flatShapes = {
			new Circle(1.0),
			new Triangle(1.0, 1.0, 1.0),
			new Square(1.0)
		};

		/* Each shape overrides the getArea() and getPerimeter() methods */
		System.out.println("2D Shapes :");
		for (Shape2D s2D : flatShapes)
			System.out.printf("%-66s area = %4f perimeter = %8f\n",
					s2D,
					s2D.getArea(),
					s2D.getPerimeter());
```

```
30              System.out.println();
31
32              /* Scalable shapes can be stored under the same type - Scalable */
33              Scalable[] scalable = {
34                     new LineSegment(1.0),
35                     new Sphere(1.0),
36                     new Cuboid(1.0, 2.0, 3.0)
37              };
38
39              /* Each scalable shape overrides the scale() method */
40              System.out.println("Scalable :");
41              for (Scalable sc : scalable)
42                     System.out.printf("%-66s scaled by 3 is %-66s\n", sc,
                           sc.scale(3));
43              System.out.println();
44      }
45  }
```

Variable Description

LineSegment		
double	length	The length of the line segment

Circle		
double	radius	The radius of the circle

Triangle		
double	a, b, c	The lengths of the sides of the triangle

Rectangle		
double	length, breadth	The dimensions of the rectangle

Sphere		
double	radius	The radius of the sphere

Cuboid		
double	length, breadth, height	The dimensions of the cuboid

"If brute force doesn't solve your problems, then you aren't using enough."

— **Anonymous**

Problem 31 Spell out a given number in words.

Solution In English, digits are grouped in sets of 3, with the first digit representing the number of 'hundreds', the second representing the number of 'tens', and the third representing the number of 'ones'. Each set is given a suffix such as 'thousand', 'million', 'billion', and so on. A special case exists for the two digit numbers 'eleven' to 'nineteen'.

Digits following a decimal point are simply spelt out in succession.

`main (number:String)`

1. Assert that `number` can be parsed as a floating point number.
2. Call and display `numberToWords(number)`.
3. **Exit**

`numberToWords (number:String)`

1. Split `number` into an `integerPart` and a `decimalPart` along the decimal point (`.`).
2. Replace `integerPart` with `stringToWords(integerPart)`.
3. If there is a decimal part, replace `decimalPart` with `stringToDigits(decimalPart)`. Otherwise, **return** `integerPart`.
4. **Return** `integerPart + "point" + "decimalPart"`

`stringToDigits (number:String)`

1. Initialize an empty string `s`.
2. For each character `c` in `number`:
   (a) Convert `c` to its corresponding digit `d`.
   (b) Append the English word for `d` to `s`.
3. **Return** `s`

`stringToWords (number: String)`

1. If `number` starts with a minus sign (`-`), remove it and **return** `"minus" + stringToWords(number)`.
2. Initialize an empty string `s`.
3. Initialize a counter `blockNumber` to zero.

4. **While** `number` is non-empty:
   (a) Remove a block of three characters from `number`, and store them as an integer `temp`.
   (b) If `temp` is non-zero, add `threeDigitsToWords(temp)` and the English word for the power of thousand correspinding to `blockNumber` to the beginning of `s`.
   (c) Increment `blockNumber`.
5. If `s` is empty, **return** `"zero"`.
6. **Return** `s`

`threeDigitsToWords (n:Integer)`

1. Store the first, second, and third digits of `n` as integers `h`, `t`, and `o` respectively.
2. Initialize an empty string `s`.
3. If `h` is non-zero, append its corresponding English word and the word `"hundred"` to `s`.
4. If `t` is `1`, append the corresponding English word for the last two digits of `n` *(which are in the 'teens')* to `s` and **return** it.
5. Append the English word for the multiple of ten corresponding to `t` to `s`.
6. If `o` is non-zero, append its corresponding English word to `s`.
7. **Return** `s`

Source Code

```
1  public class NumberToWords {
2        /* Map of single digits to words */
3        public static final String[] singleDigits = {
4              " zero",
5              " one",
6              " two",
7              " three",
8              " four",
9              " five",
10             " six",
11             " seven",
12             " eight",
13             " nine"
14       };
15
16       /* Map of numbers in the 'teens' to words */
17       public static final String[] twoDigits = {
18             " ten",
19             " eleven",
```

```
20              " twelve",
21              " thirteen",
22              " fourteen",
23              " fifteen",
24              " sixteen",
25              " seventeen",
26              " eighteen",
27              " nineteen"
28      };
29
30      /* Map of multiples of tens into words */
31      public static final String[] tenMultiples = {
32              "",
33              "",
34              " twenty",
35              " thirty",
36              " forty",
37              " fifty",
38              " sixty",
39              " seventy",
40              " eighty",
41              " ninety"
42      };
43
44      /* Map of suffixes of powers of thousand into words */
45      public static final String[] thousandPowerGroups = {
46              "",
47              " thousand",
48              " million",
49              " billion",
50              " trillion",
51              " quadrillion",
52              " quintillion",
53              " sextillion",
54              " septillion",
55              " octillion",
56              " nonillion",
57              " decillion"
58      };
59
60      public static void main (String[] args) {
61              try {
62                      /* Parse the first command line argument as the number
63                         to be spelt out */
64                      Double.parseDouble(args[0]);
65                      System.out.println(numberToWords(args[0]));
```

```
66              } catch (IndexOutOfBoundsException e) {
67                      System.out.println("Enter 1 argument! ([number])");
68              } catch (NumberFormatException e) {
69                      System.out.println("Invalid number!");
70              }
71      }
72
73      /* Convert a string of digits into words */
74      public static String numberToWords (String n) {
75              /* Deal with the integral and fractional parts separately */
76              String parts[] = n.split("\\.");
77              String integerPart = stringToWords(parts[0]);
78              /* Check for the fractional part */
79              if (parts.length == 1)
80                      return integerPart.trim();
81              String decimalPart = stringToDigits(parts[1]);
82              return (integerPart + " point" + decimalPart).trim();
83      }
84
85      /* Convert the digits of the fractional part into words */
86      public static String stringToDigits (String digits) {
87              String s = "";
88              for (int i = 0; i < digits.length(); i++) {
89                      /* Map digits to their corresponding words */
90                      int d = digits.charAt(i) - '0';
91                      s += singleDigits[d];
92              }
93              return s;
94      }
95
96      /* Convert the digits of the integral part into words */
97      public static String stringToWords (String n) {
98              /* Negative sign is simply read off as 'minus' */
99              if (n.charAt(0) == '-')
100                     return "minus" + stringToWords(n.substring(1));
101             String s = "";
102             int left = Math.max(0, n.length() - 3);
103             int blockNumber = 0;
104             /* Loop through blocks of three */
105             while (n.length() > 0) {
106                     String temp = n.substring(left);
107                     int blockOfThree = Integer.parseInt(temp);
108                     if (blockOfThree != 0) {
109                             s = threeDigitsToWords(blockOfThree)
110                                     + thousandPowerGroups[blockNumber]
111                                     + "," + s;
```

```
                        }
                        blockNumber++;
                        /* Cut off evaluated part */
                        n = n.substring(0, left);
                        left = Math.max(0, left - 3);
                }
                /* Special case */
                if (s.equals(""))
                        return "zero";
                return s.substring(0, s.length() - 1);
        }

        /* Convert a block of three digits into words */
        public static String threeDigitsToWords (int n) {
                /* Extract each digit */
                int h = n / 100;
                int t = (n / 10) % 10;
                int o = n % 10;
                String s = "";
                /* Only convert the 'hundreds' if it is non-zero */
                if (h > 0) {
                        s += singleDigits[h] + " hundred";
                }
                /* Special case of 'teens' */
                if (t == 1) {
                        s += twoDigits[o];
                        return s;
                }
                s += tenMultiples[t];
                /* Only convert 'ones' if it is non-zero */
                if (o > 0) {
                        s += singleDigits[o];
                }
                return s;
        }
}
```

Variable Description

NumberToWords		
String[]	singleDigits	Map of English words corresponding to single digits
String[]	twoDigits	Map of English words corresponding to two digit numbers in the 'teens'
String[]	tenMultiples	Map of English words corresponding to multiples of ten
String[]	thousand PowerGroups	Map of English words corresponding to powers of thousand

NumberToWords::numberToWords(String)		
String	n	The number to be spelt out
String[]	parts	Stores the integer and fractional parts of number
String	integerPart	The integer part in words
String	decimalPart	The digits after the decimal in words

NumberToWords::stringToDigits(String)		
String	digits	The string of digits to be spelt out
String	s	digits in words
int	i	Counter variable
int	d	The current digit to be spelt out

NumberToWords::stringToWords(String)		
String	n	The integer to be spelt out
int	left	The left index of the current block of three
int	blockNumber	Counter variable, stores the current block number
String	temp	Stores the current block of three
int	blockOfThree	Stores the current block of three as an integer

NumberToWords::threeDigitsToWords(int)		
int	n	The integer to be spelt out
int	h	The first digit of n
int	t	The second digit of n
int	o	The third digit of n

"Please, Oh please, publish me in your collection of self-referential sentences!"

— **Douglas Hofstadter**

Problem 32 A *quine* is a non-empty computer program which takes no input and produces a copy of its own source code as its only output.

Write a *quine* in *Java*.

(Note that a program which finds its source code file and displays it is not considered a quine, since it takes a file as input.)

Hofstadter (`xkcd.com/917`)

Solution The name *quine* was coined by *Douglas Hofstadter* in his brilliant book *Gödel, Escher, Bach: An Eternal Golden Braid*, in honour of the philosopher *Willard Van Orman Quine*, who extensively studied indirect self reference, in particular the following statement known as *Quine's paradox*.

> "Yields falsehood when preceded by its quotation" yields falsehood when preceded by its quotation.

Although writing a *quine* in *Java* seems impossible at first glance, it can be shown that *quines* exist in any *Turing complete* programming language.

We might start off by writing the following code.

```
1 public class Quine {
2       public static void main (String[] args) {
3             System.out.println("public class Quine {\n\t ... System.out.println(
                  ??? )}");
4       }
5 }
```

A problem arises — what can we write in place of `???` ? This part of the string must contain the entire string itself. Is this possible without the string being infinitely long?

The problem is that the string we seek must contain the characters to be printed, and also be able to be used to print itself. The following code snippet illustrates this.

```
1             String s = "???";
2             System.out.println(???);
```

What can replace `???` so that the entirety of line 1 is displayed?

A solution is as follows.

```
1             String s = "String s = ";
2             System.out.println(s + '"' + s + '"' + ';');
```

We can now use this template to move the entirety of the code into the string, including the print statement itself. This leads to another problem — double quotes are now inside double quotes, and must be escaped (`\"`). However, the backslashes themselves will not appear in the output. This can be solved by using the ASCII value for an double quote, which is `34`, in place of an escaped double quote. Discarding newlines and delcaring the string `s` as a global variable at the very end of the program minimizes the amount of code considerably.

The result is the following *quine*.

```
1 public class Quine { public static void main (String[] args) { char q = 34;
    System.out.println(s + q + s + q + ';' + '}'); } public static String s =
    "public class Quine { public static void main (String[] args) { char q = 34;
    System.out.println(s + q + s + q + ';' + '}'); } public static String s = ";}
```

Variable Description

Quine		
String	s	Stores the entire source code of the program
Quine::main()		
char	q	Stores a double quote

This project was compiled with XeLaTeX.

All files involved in the making of this project can be found at
https://github.com/sahasatvik/Computer-Project/tree/master/ISC

Satvik Saha
sahasatvik@gmail.com
https://sahasatvik.github.io