

```
1 public class Computer {
2
3     /** This constant represents the size of the memory unit of this Computer
4      * (number of memory registers). */
5     public final static int MEM_SIZE = 100;
6
7     /** This constant represents the memory address at which the constant 0 is
8     stored. */
9     public final static int LOCATION_OF_ZERO = MEM_SIZE - 2;
10
11    /** This constant represents the memory address at which the number 1 is
12    stored. */
13    public final static int LOCATION_OF_ONE = MEM_SIZE - 1;
14
15    // Op-code definitions:
16    private final static int WRITE = 9;
17    private final static int READ = 8;
18    private final static int GOTOP = 7;
19    private final static int GOTOZ = 6;
20    private final static int GOTO = 5;
21    private final static int STORE = 4;
22    private final static int LOAD = 3;
23    private final static int SUB = 2;
24    private final static int ADD = 1;
25    private final static int STOP = 0;
26
27    // Put the rest of the op-code definitions here.
28
29    /** The Computer consists of a Memory unit, and two registers, as follows: */
30    private Memory m;
31    private Register dReg;
32    private Register pc;
33
34    public Computer() {
35        m = new Memory(MEM_SIZE);
36        dReg = new Register();
37        pc = new Register();
38        reset();
39    }
40
41    public void reset() {
42        m.reset();
43        m.setValue(LOCATION_OF_ZERO, 0);
44        m.setValue(LOCATION_OF_ONE, 1);
45        pc.setValue(0);
46        dReg.setValue(0);
47    }
48
49    public void run() {
50        int stop = 0;
51        while (stop != -1) {
52            int value = m.getValue(pc.getValue());
53
54            if (value / 100 == ADD) {
55                execAdd(value - 100);
56            } else if (value / 100 == SUB) {
57                execSub(value - 200);
58            } else if (value / 100 == LOAD) {
59                execLoad(value - 300);
60            } else if (value / 100 == STORE) {
61                execStore(value - 400);
62            }
63        }
64    }
65}
```

```
60         } else if(value / 100 == GOTO) {
61             execGoto(value - 500);
62         } else if(value / 100 == GOTOZ) {
63             execGotoz(value - 600);
64         } else if(value / 100 == GOTOPT) {
65             execGotopt(value - 700);
66         } else if(value / 100 == READ) {
67             execRead();
68         } else if(value / 100 == WRITE) {
69             execWrite(dReg.getValue());
70         } else {
71             stop = execStop();
72         }
73     }
74 }
75
76 // Private execution routines, one for each Vic command
77 private void execLoad (int addr) {
78     dReg.setValue(m.getValue(addr));
79     pc.addOne();
80 }
81
82 private void execRead () {
83     int value = StdIn.readInt();
84     dReg.setValue(value);
85     pc.addOne();
86 }
87
88 private void execWrite (int value) {
89     System.out.println(" " + value);
90     pc.addOne();
91 }
92
93 private void execStore (int addr) {
94     m.setValue(addr, dReg.getValue());
95     pc.addOne();
96 }
97
98 private void execAdd (int addr) {
99     dReg.setValue(m.getValue(addr) + dReg.getValue());
100    pc.addOne();
101 }
102
103
104 private void execGoto (int addr) {
105     pc.setValue(addr);
106 }
107
108
109 private void execGotopt (int addr) {
110     if (dReg.getValue() > 0) {
111         execGoto(addr);
112     } else {
113         pc.addOne();
114     }
115 }
116
117
118 private void execGotoz (int addr) {
119     if (dReg.getValue() == 0) {
120         execGoto(addr);
```

```
121         } else {
122             pc.addOne();
123         }
124     }
125
126     private void execSub (int addr) {
127         dReg.setValue(dReg.getValue() - m.getValue(addr));
128         pc.addOne();
129     }
130
131     private int execStop () {
132         System.out.println("Program terminated normally");
133         pc.addOne();
134         return -1;
135     }
136
137     public void loadProgram(String fileName) {
138         StdIn.setInput(fileName);
139         int counter = 0;
140         while(StdIn.hasNextLine()) {
141             int value = StdIn.readInt();
142             m.setValue(counter, value);
143             counter++;
144         }
145     }
146
147     public void loadInput(String fileName) {
148         StdIn.setInput(fileName);
149     }
150
151     public String toString () {
152         // Put your code here
153         return "D register = " + dReg.getValue() + "\n" + "PC register = " +
pc.getValue() + "\n" + "Memory state:" + "\n" + m.toString();
154     }
155 }
```

```
1 /** Represents a register.
2  * A register is the basic storage unit of the Vic computer. */
3
4 public class Register {
5
6     private int value; // the current value of this register
7
8     /** Constructs a register and sets its value to 0. */
9     public Register() {
10         // Put your code here
11         setValue(0);
12     }
13
14     /** Sets the value of this register.
15      * @param v the value to which the register will be set. */
16     public void setValue(int val) {
17         value = val;
18         // Put your code here
19     }
20
21     /** Increments the value of this register by 1. */
22     public void addOne() {
23         setValue(getValue() + 1);
24         // Put your code here
25     }
26
27     /** Returns the value of this register.
28      * @return the current value of this register, as an int. */
29     public int getValue() {
30         // Put your code here
31         return value;
32     }
33
34     /** Returns a textual representation of the value of this register.
35      * @return Returns the value of this register, as a String. */
36     public String toString() {
37         // Put your code here
38         return "" + value;
39     }
40 }
```

```
1 /** Represents a register.
2  * A register is the basic storage unit of the Vic computer. */
3
4 public class Register {
5
6     private int value; // the current value of this register
7
8     /** Constructs a register and sets its value to 0. */
9     public Register() {
10         // Put your code here
11         setValue(0);
12     }
13
14     /** Sets the value of this register.
15      * @param v the value to which the register will be set. */
16     public void setValue(int val) {
17         value = val;
18         // Put your code here
19     }
20
21     /** Increments the value of this register by 1. */
22     public void addOne() {
23         setValue(getValue() + 1);
24         // Put your code here
25     }
26
27     /** Returns the value of this register.
28      * @return the current value of this register, as an int. */
29     public int getValue() {
30         // Put your code here
31         return value;
32     }
33
34     /** Returns a textual representation of the value of this register.
35      * @return Returns the value of this register, as a String. */
36     public String toString() {
37         // Put your code here
38         return "" + value;
39     }
40 }
```