```java
public class Computer {

    /** This constant represents the size of the memory unit of this Computer
     *  (number of memory registers). */
    public final static int MEM_SIZE = 100;

    /** This constant represents the memory address at which the constant 0 is stored. */
    public final static int LOCATION_OF_ZERO = MEM_SIZE - 2;

    /** This constant represents the memory address at which the number 1 is stored. */
    public final static int LOCATION_OF_ONE = MEM_SIZE - 1;

    // Op-code definitions:
    private final static int WRITE = 9;
    private final static int READ = 8;
    private final static int GOTOP = 7;
    private final static int GOTOZ = 6;
    private final static int GOTO = 5;
    private final static int STORE = 4;
    private final static int LOAD = 3;
    private final static int SUB = 2;
    private final static int ADD = 1;
    private final static int STOP = 0;

    // Put the rest of the op-code definitions here.

    /** The Computer consists of a Memory unit, and two registers, as follows: */
    private Memory m;
    private Register dReg;
    private Register pc;

    public Computer() {
        m = new Memory(MEM_SIZE);
        dReg = new Register();
        pc = new Register();
        reset();
    }

    public void reset() {
        m.reset();
        m.setValue(LOCATION_OF_ZERO, 0);
        m.setValue(LOCATION_OF_ONE, 1);
        pc.setValue(0);
        dReg.setValue(0);
    }

    public void run() {
        int stop = 0;
        while (stop != -1) {
            int value = m.getValue(pc.getValue());

            if (value / 100 == ADD) {
                execAdd(value - 100);
            } else if (value / 100 == SUB) {
                execSub(value - 200);
            } else if (value / 100 == LOAD) {
                execLoad(value - 300);
            } else if (value / 100 == STORE) {
                execStore(value - 400);
```

```
            } else if(value / 100 == GOTO) {
                execGoto(value - 500);
            } else if(value / 100 == GOTOZ) {
                execGotoz(value - 600);
            } else if(value / 100 == GOTOP) {
                execGotop(value - 700);
            } else if(value / 100 == READ) {
                execRead();
            } else if(value / 100 == WRITE) {
                execWrite(dReg.getValue());
            } else {
                stop = execStop();
            }
        }
    }

    // Private execution routines, one for each Vic command
    private void execLoad (int addr) {
        dReg.setValue(m.getValue(addr));
        pc.addOne();
    }

    private void execRead () {
        int value = StdIn.readInt();
        dReg.setValue(value);
        pc.addOne();
    }

    private void execWrite (int value) {
        System.out.println("" + value);
        pc.addOne();
    }

    private void execStore (int addr) {
        m.setValue(addr, dReg.getValue());
        pc.addOne();
    }

    private void execAdd (int addr) {
        dReg.setValue(m.getValue(addr) + dReg.getValue());
        pc.addOne();
    }


    private void execGoto (int addr) {
        pc.setValue(addr);
    }


    private void execGotop (int addr) {
        if (dReg.getValue() > 0) {
            execGoto(addr);
        } else {
            pc.addOne();
        }
    }


    private void execGotoz (int addr) {
        if (dReg.getValue() == 0) {
            execGoto(addr);
```

```java
        } else {
            pc.addOne();
        }
    }

    private void execSub (int addr) {
        dReg.setValue(dReg.getValue() - m.getValue(addr));
        pc.addOne();
    }

    private int execStop () {
        System.out.println("Program terminated normally");
        pc.addOne();
        return -1;
    }

    public void loadProgram(String fileName) {
        StdIn.setInput(fileName);
        int counter = 0;
        while(StdIn.hasNextLine()) {
            int value = StdIn.readInt();
            m.setValue(counter, value);
            counter++;
        }
    }

    public void loadInput(String fileName) {
        StdIn.setInput(fileName);
    }

    public String toString () {
        // Put your code here
        return "D register  = " + dReg.getValue() + "\n" + "PC register = " + pc.getValue() + "\n" + "Memory state:" + "\n" + m.toString();
    }
}
```

```java
/** Represents a register.
 *  A register is the basic storage unit of the Vic computer. */

public class Register {

    private int value;  // the current value of this register

    /** Constructs a register and sets its value to 0. */
    public Register() {
        // Put your code here
        setValue(0);
    }

    /** Sets the value of this register.
     * @param v the value to which the register will be set. */
    public void setValue(int val) {
        value = val;
        // Put your code here
    }

    /** Increments the value of this register by 1. */
    public void addOne() {
        setValue(getValue() + 1);
        // Put your code here
    }

    /** Returns the value of this register.
     * @return the current value of this register, as an int. */
    public int getValue() {
        // Put your code here
        return value;
    }

    /** Returns a textual representation of the value of this register.
     * @return Returns the value of this register, as a String. */
    public String toString() {
        // Put your code here
        return "" + value;
    }
}
```

```java
/** Represents a register.
 *  A register is the basic storage unit of the Vic computer. */

public class Register {

    private int value;  // the current value of this register

    /** Constructs a register and sets its value to 0. */
    public Register() {
        // Put your code here
        setValue(0);
    }

    /** Sets the value of this register.
     * @param v the value to which the register will be set. */
    public void setValue(int val) {
        value = val;
        // Put your code here
    }

    /** Increments the value of this register by 1. */
    public void addOne() {
        setValue(getValue() + 1);
        // Put your code here
    }

    /** Returns the value of this register.
     * @return the current value of this register, as an int. */
    public int getValue() {
        // Put your code here
        return value;
    }

    /** Returns a textual representation of the value of this register.
     * @return Returns the value of this register, as a String. */
    public String toString() {
        // Put your code here
        return "" + value;
    }
}
```