# OOP - Mini Project

## Title: Implementation of Memory with the help of Object - Oriented Principles of Java

In this project, three different types of memories were created using Object-oriented principles of Java. The 3 different types of memories can be characterised as follows:

### 1. Type1 Memory -

| **Feature/ Ports** | **Purpose** |
|---|---|
| Write | write/feed a data at all the address in the memory from the user |
| Read | read the data stored in the memory |

### 2. Type2 Memory -

| **Feature/ Ports** | **Purpose** |
|---|---|
| Write | write/feed a data at all the address in the memory from the user |
| Read | read the data stored in the memory |
| Reset | reset the memory with all registers fed with zero values |

### 3. Type3 Memory -

| **Feature/ Ports** | **Purpose** |
|---|---|
| Write | write/feed a data at all the address in the memory from the user |
| Read | read the data stored in the memory |
| Reset | reset the memory with all registers fed with zero values |

| | |
|---|---|
| display | Display the contents of all the memory registers |

**Key concepts involved:**

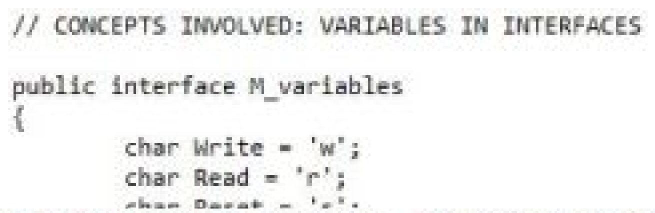
- Classes, objects
- Interfaces, Interfaces can be extended
- Polymorphism (same interface different implementations)
- Inheritance
- Exception Handling, Catching subclass exceptions

The project is made with the help of 5 source codes

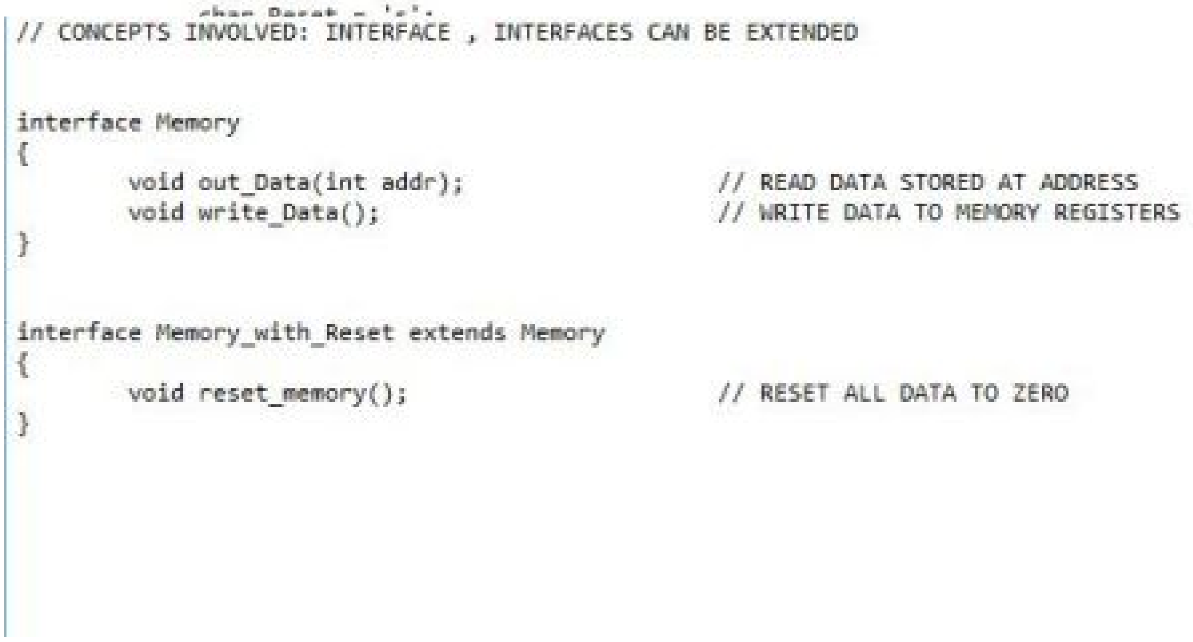
| | |
|---|---|
| M_variables.java | Interface with data members Write, Read and Reset |
| All_interfaces.java | Interface for memories |
| Memory_Implementations.java | Implementation of all interfaces present in All_interfaces.java |
| Project.java | main method present, objects of all the 3 types of memories created |
| drivers_of_Memory.java | All driver classes for the 3 types of memories present |

- M_variables.java -

**Code:**



```
// CONCEPTS INVOLVED: VARIABLES IN INTERFACES

public interface M_variables
{
        char Write = 'w';
        char Read = 'r';
```



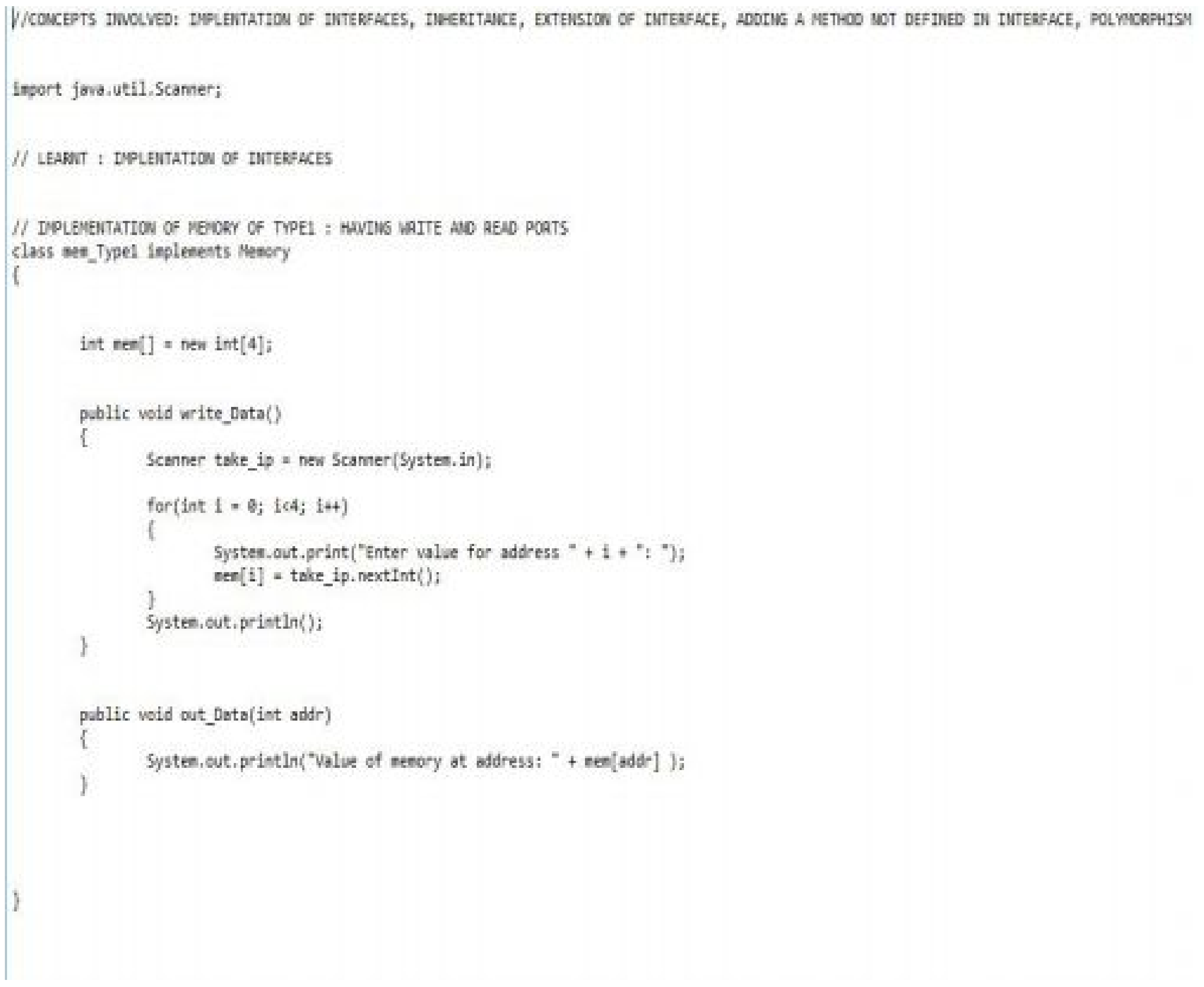
```
// CONCEPTS INVOLVED: INTERFACE , INTERFACES CAN BE EXTENDED


interface Memory
{
        void out_Data(int addr);                    // READ DATA STORED AT ADDRESS
        void write_Data();                          // WRITE DATA TO MEMORY REGISTERS
}


interface Memory_with_Reset extends Memory
{
        void reset_memory();                        // RESET ALL DATA TO ZERO
}
```

**<u>Explanation:</u>**

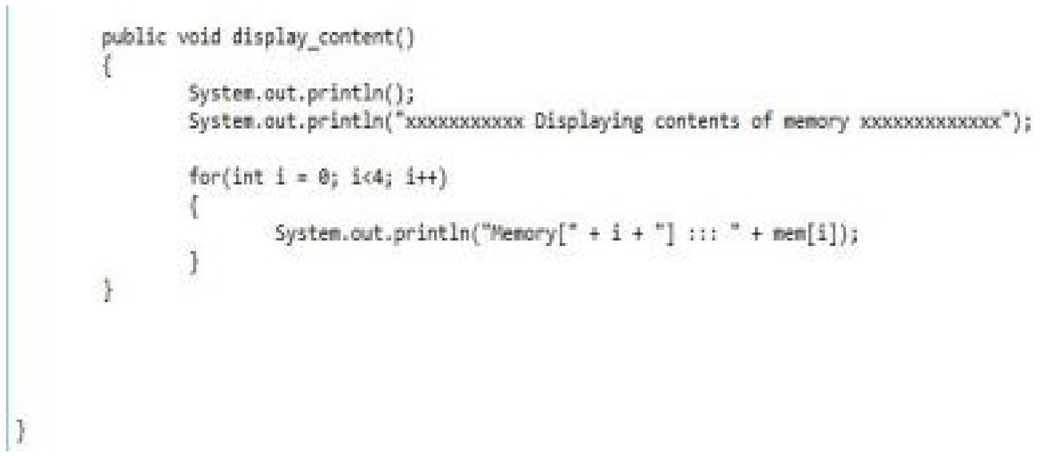
This source file has the implementation of the memories with read, write and reset features. The interface Memory has the features to output the data stored at a particular address in the memory. The interface Memory_with_Reset inherits from the interface Memory and has all the methods of the former but one extra feature (method) to reset the memory.

- Memory_Implementations.java

**<u>Code:</u>**



```java
//CONCEPTS INVOLVED: IMPLENTATION OF INTERFACES, INHERITANCE, EXTENSION OF INTERFACE, ADDING A METHOD NOT DEFINED IN INTERFACE, POLYMORPHISM


import java.util.Scanner;


// LEARNT : IMPLENTATION OF INTERFACES


// IMPLEMENTATION OF MEMORY OF TYPE1 : HAVING WRITE AND READ PORTS
class mem_Type1 implements Memory
{


	int mem[] = new int[4];


	public void write_Data()
	{
		Scanner take_ip = new Scanner(System.in);

		for(int i = 0; i<4; i++)
		{
			System.out.print("Enter value for address " + i + ": ");
			mem[i] = take_ip.nextInt();
		}
		System.out.println();
	}


	public void out_Data(int addr)
	{
		System.out.println("Value of memory at address: " + mem[addr] );
	}




}
```



```
        public void display_content()
        {
                System.out.println();
                System.out.println("xxxxxxxxxx Displaying contents of memory xxxxxxxxxxxx");

                for(int i = 0; i<4; i++)
                {
                        System.out.println("Memory[" + i + "] ::: " + mem[i]);
                }
        }




}
```

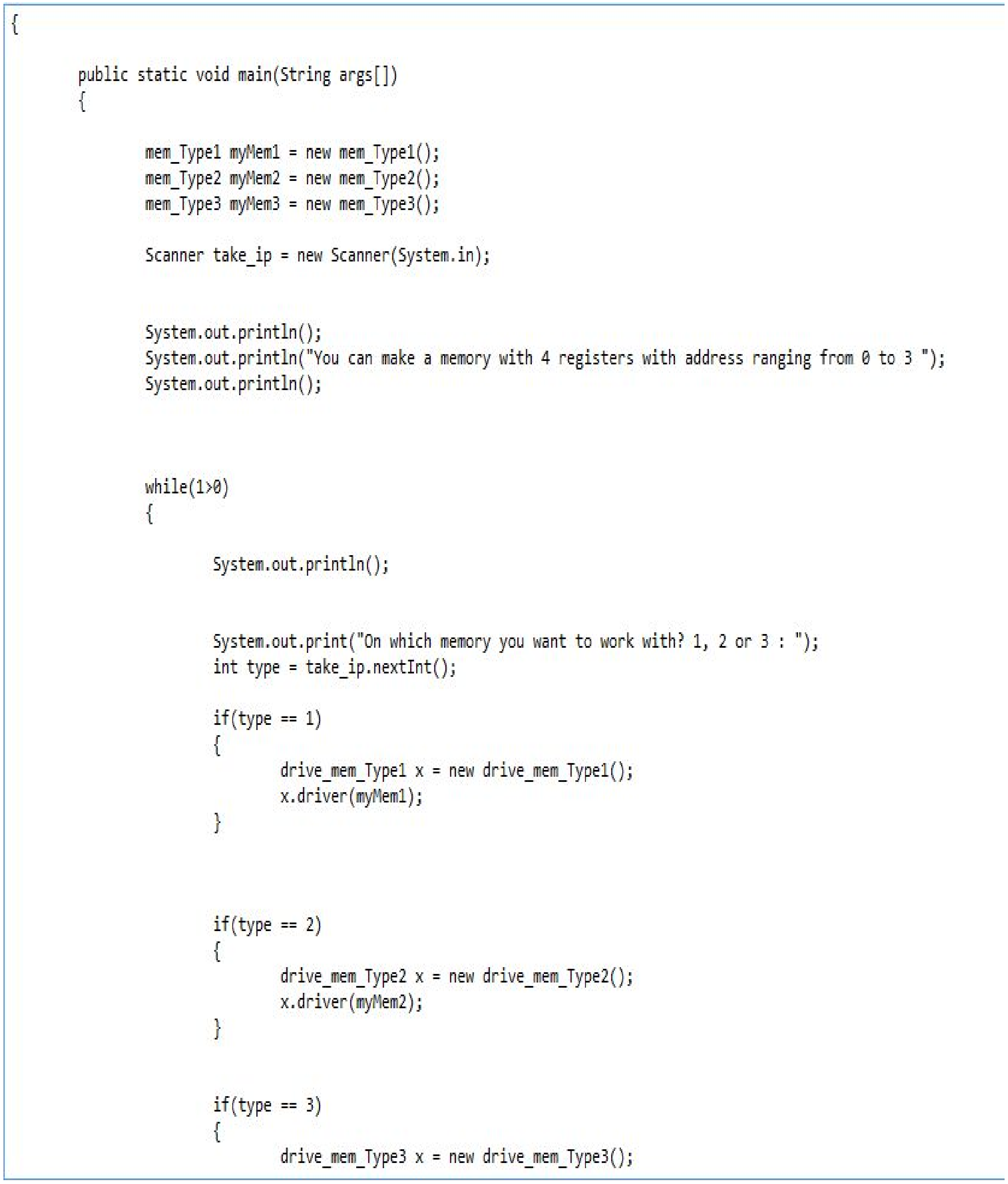
**Explanation:**

This source file has the implementations of all the 3 types of memories. Class mem_Type1 implements Memory and has the functionalities (methods) to write data to all the registers of the memory of type mem_Type1 when called and to read the data stored at a particular address. Class mem_Type2 inherits from mem_Type1 but implements Memory_with_Reset. Hence, it has to define only one method named reset_memory() to reset all the data registers. Class mem_Type3 also implements Memory_with_Reset but has all different implementations of the interface methods than mem_Type2, hence exhibiting the feature of Polymorphism in Java. Also, it has a method named display_content() to display the contents of all the data registers of the memory of type mem_Type3, i.e., it has a method that is not defined in the interface Memory_with_Reset.

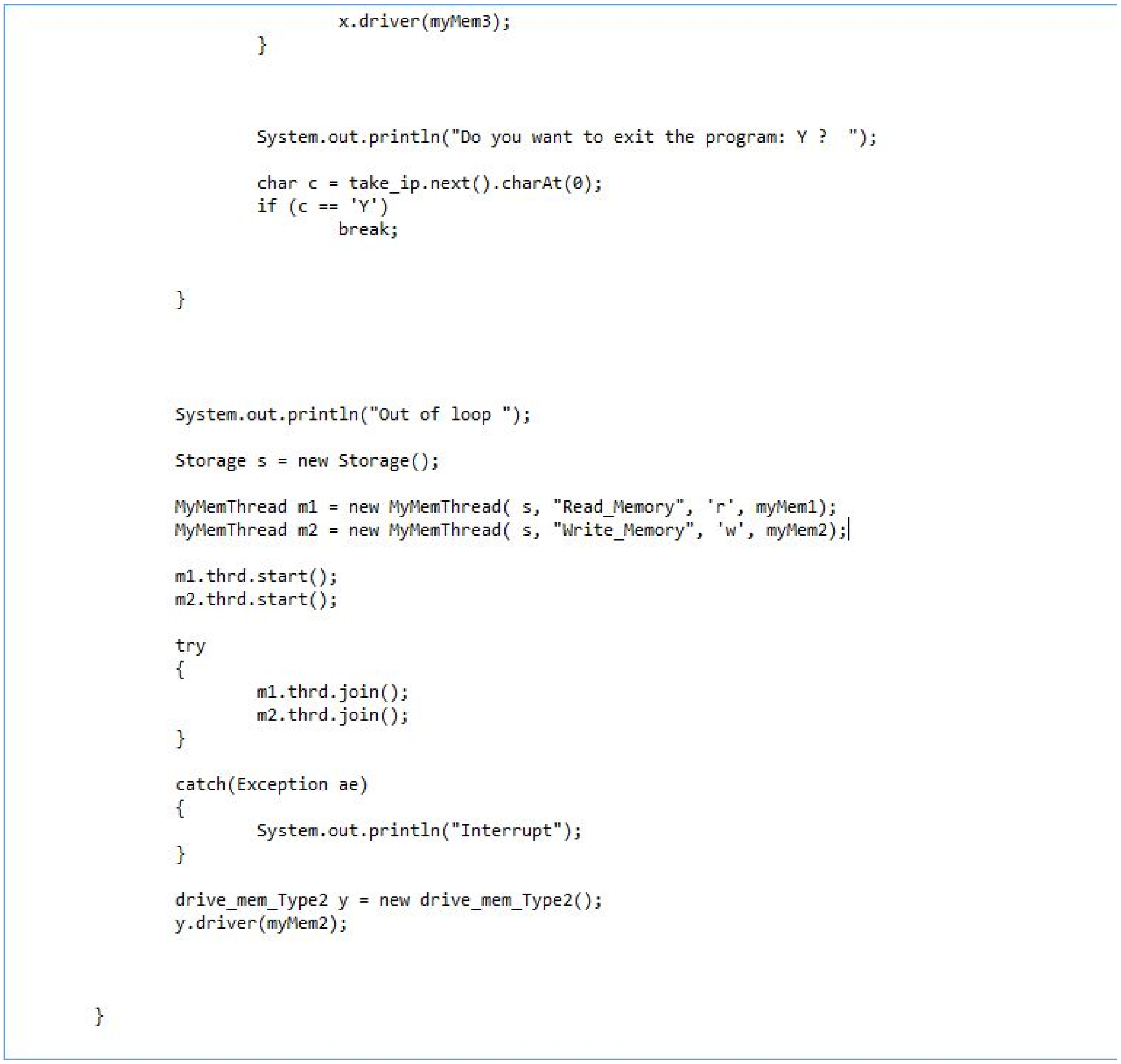
- Project.java

**Code:**

```
// CONCEPTS INVOLVED: CLASSES, OBJECTS, MULTITHREADING

import java.util.Scanner;

class Storage
{
        int val;

        int getVal()
        {
                return val;
        }

        void setVal(int x)
        {
                val = x;
        }
}

class MyMemThread implements Runnable
{
        Thread thrd;
        static Storage s;
        char op;
        mem_Type1 mmry;


        MyMemThread(Storage s, String name, char op, mem_Type1 MEMORY_OBJECT)
        {
                thrd = new Thread(this, name);
                this.s = s;
                this.op = op;
                this.mmry = MEMORY_OBJECT;
        }

        public void run()
        {


                for(int i = 0; i<4; i++)
                {
                        try
                        {
                                synchronized(s)
                                {
                                        if(op == 'r')
                                        {
                                                System.out.println("You're in a memory called:" + thrd.getName());
                                                s.setVal(mmry.out_Data(i));
                                                System.out.println();
                                                s.notify();
                                                s.wait();
                                        }

                                        if(op == 'w')
                                        {
                                                System.out.println("You're in a memory called:" + thrd.getName());
                                                mmry.mem[i] = s.getVal();
                                                System.out.println("Written at address " + i + " " + mmry.mem[i]);
                                                System.out.println();
                                                if(i == 3)
                                                {
                                                        s.notify();
                                                        System.out.println();
                                                        break;
                                                }
                                                s.notify();
                                                s.wait();
                                        }

                                }


                        }

                        catch(InterruptedException ae)
                        {
                                System.out.println("Interrupt");
                        }
                }

        }

}



// CLASS HAVING MAIN METHOD TO BE EXECUTED BY JVM
class project
```



```
{

        public static void main(String args[])
        {

                mem_Type1 myMem1 = new mem_Type1();
                mem_Type2 myMem2 = new mem_Type2();
                mem_Type3 myMem3 = new mem_Type3();

                Scanner take_ip = new Scanner(System.in);


                System.out.println();
                System.out.println("You can make a memory with 4 registers with address ranging from 0 to 3 ");
                System.out.println();



                while(1>0)
                {

                        System.out.println();


                        System.out.print("On which memory you want to work with? 1, 2 or 3 : ");
                        int type = take_ip.nextInt();

                        if(type == 1)
                        {
                                drive_mem_Type1 x = new drive_mem_Type1();
                                x.driver(myMem1);
                        }



                        if(type == 2)
                        {
                                drive_mem_Type2 x = new drive_mem_Type2();
                                x.driver(myMem2);
                        }


                        if(type == 3)
                        {
                                drive_mem_Type3 x = new drive_mem_Type3();
```



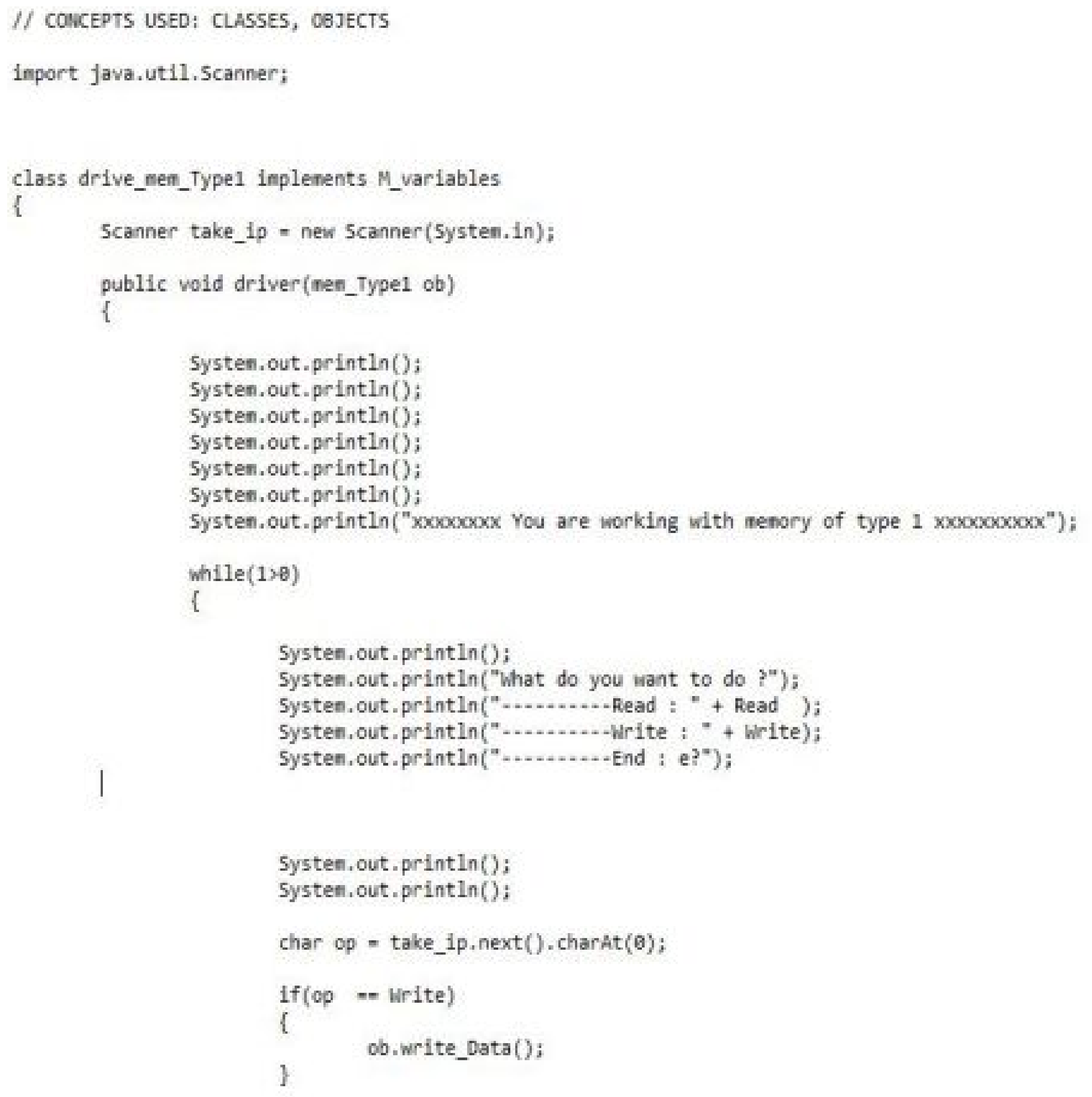
```
            x.driver(myMem3);
        }


        System.out.println("Do you want to exit the program: Y ?  ");

        char c = take_ip.next().charAt(0);
        if (c == 'Y')
            break;


    }



    System.out.println("Out of loop ");

    Storage s = new Storage();

    MyMemThread m1 = new MyMemThread( s, "Read_Memory", 'r', myMem1);
    MyMemThread m2 = new MyMemThread( s, "Write_Memory", 'w', myMem2);

    m1.thrd.start();
    m2.thrd.start();

    try
    {
        m1.thrd.join();
        m2.thrd.join();
    }

    catch(Exception ae)
    {
        System.out.println("Interrupt");
    }

    drive_mem_Type2 y = new drive_mem_Type2();
    y.driver(myMem2);


}
```

**Explanation:**

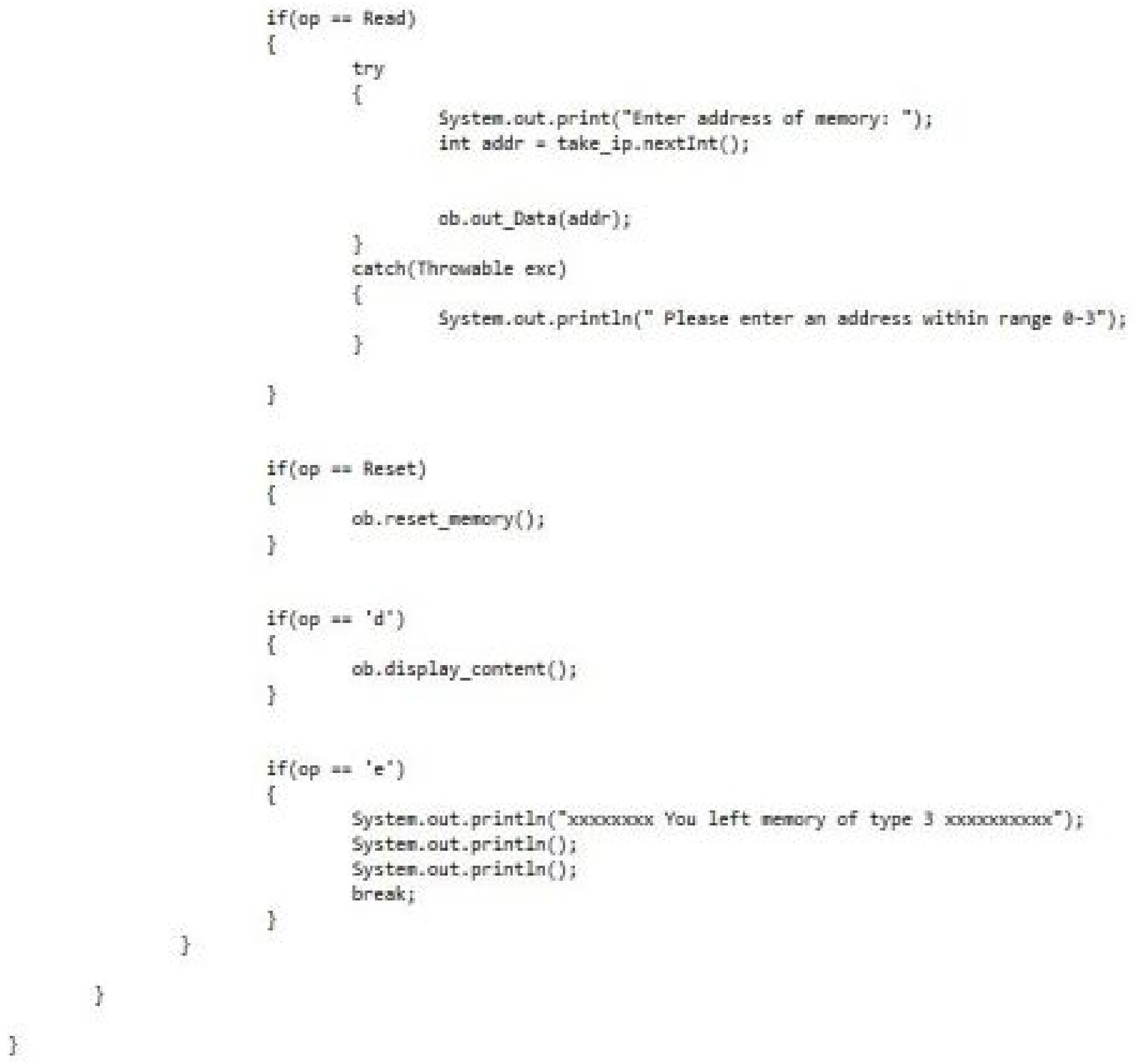
This source file has the main method and contains the objects of all the three types of memories. The main method has the while loop which keeps running the program until the user enters ‘Y’ to end the program. At the beginning of the while loop, the user is asked on which type of memory he wants to work with. For working with memory of Type1, Type2, and Type3, the user needs to enter 1, 2 and 3 respectively. Accordingly, objects of the driver classes of the types of memories are created and the memory objects of that type are passed by calling the driver method which drives the memory and carries on the execution of the program. Multithreading is also used to perform synchronized operations of writing to a memory the data that is read from one memory with every changing address.

- drivers_of_Memory.java

**Code:**



```
// CONCEPTS USED: CLASSES, OBJECTS

import java.util.Scanner;



class drive_mem_Type1 implements M_variables
{
        Scanner take_ip = new Scanner(System.in);

        public void driver(mem_Type1 ob)
        {

                System.out.println();
                System.out.println();
                System.out.println();
                System.out.println();
                System.out.println();
                System.out.println();
                System.out.println("xxxxxxxx You are working with memory of type 1 xxxxxxxxxx");

                while(1>0)
                {

                        System.out.println();
                        System.out.println("What do you want to do ?");
                        System.out.println("----------Read : " + Read  );
                        System.out.println("----------Write : " + Write);
                        System.out.println("----------End : e?");



                        System.out.println();
                        System.out.println();

                        char op = take_ip.next().charAt(0);

                        if(op  == Write)
                        {
                                ob.write_Data();
                        }
```

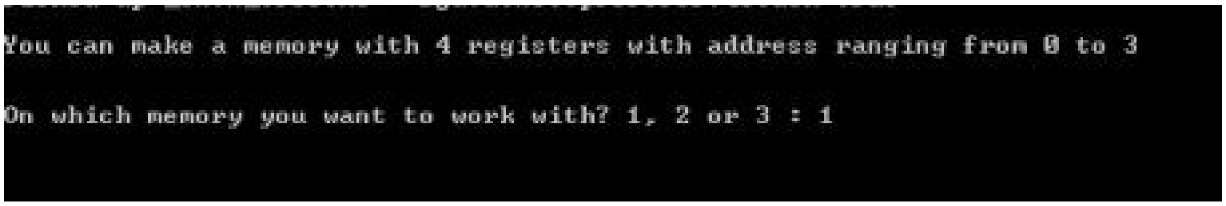


```
            if(op == Read)
            {
                try
                {
                    System.out.print("Enter address of memory: ");
                    int addr = take_ip.nextInt();


                    ob.out_Data(addr);
                }
                catch(Throwable exc)
                {
                    System.out.println(" Please enter an address within range 0-3");
                }

            }

            if(op == Reset)
            {
                ob.reset_memory();
            }


            if(op == 'd')
            {
                ob.display_content();
            }


            if(op == 'e')
            {
                System.out.println("xxxxxxxx You left memory of type 3 xxxxxxxxxx");
                System.out.println();
                System.out.println();
                break;
            }
        }

    }

}
```

**Explanation:**

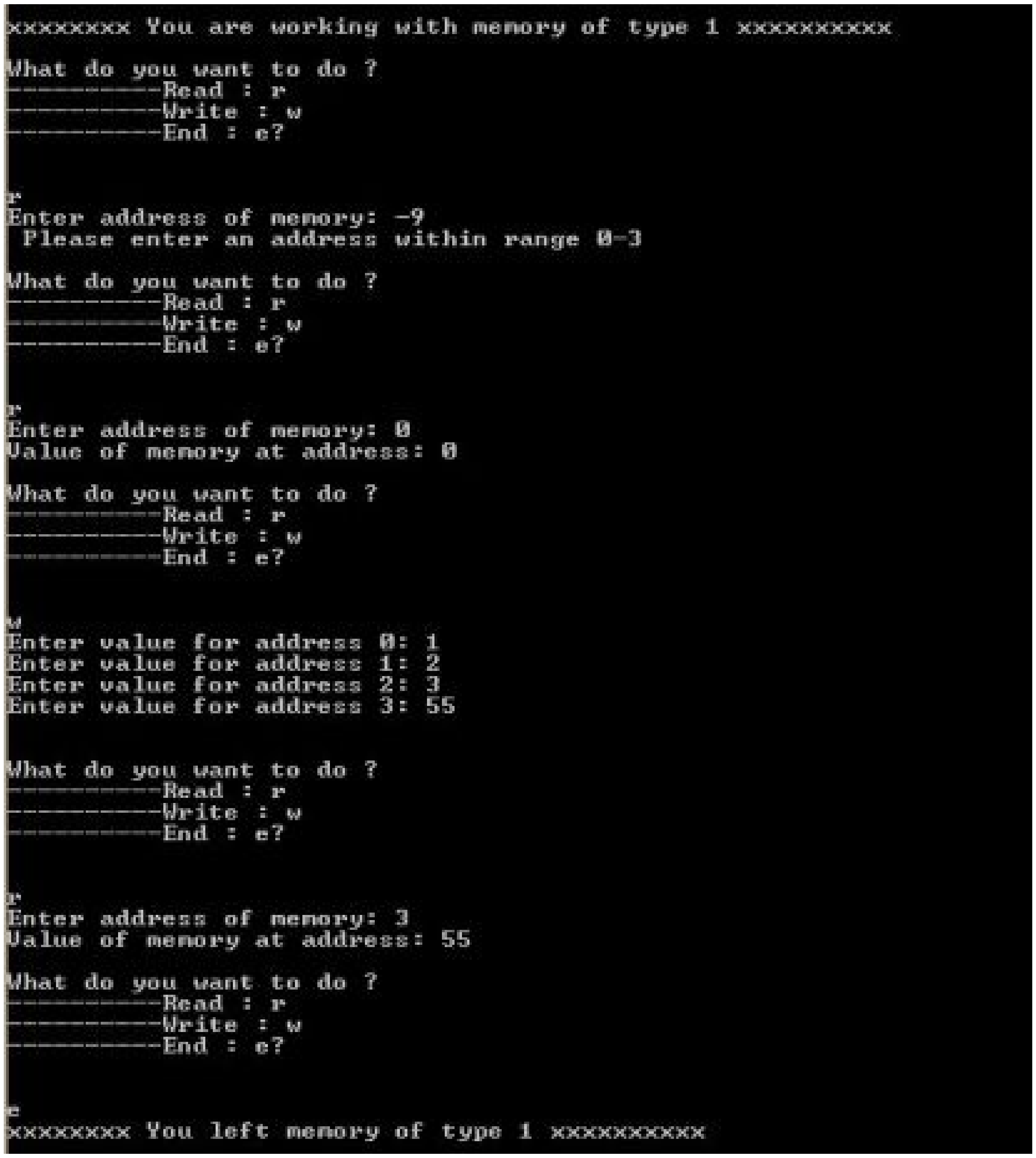
This source file has definitions for all the driver classes of the 3 types of memories whose objects were created and driven in the main source file Project.java . The driver classes implement the interface M_variables. It has a driver method which takes the memory objects as an argument. The driver classes again have a continuous while loop within which the user is asked about what the user wants to do with the memory until the user enters ‘e’ to end working with the memory and the control of execution again comes to the main method asking the user on which type of memory the user desires to work with. Also, exceptions are handled by catching all the possible exceptions when any invalid address is entered.

## Outputs:



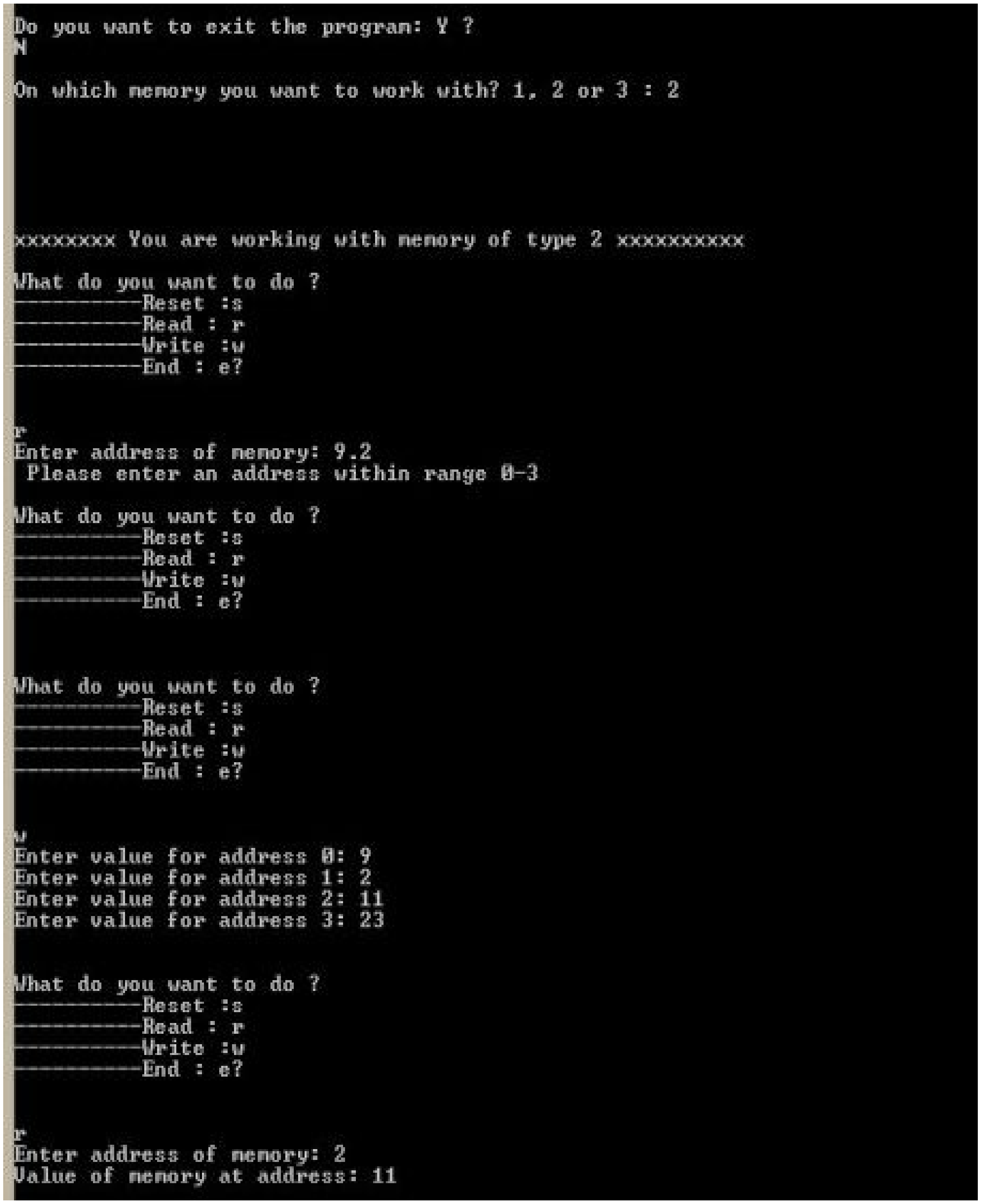
```
You can make a memory with 4 registers with address ranging from 0 to 3

On which memory you want to work with? 1, 2 or 3 : 1
```

**Working with Type1 Memory:**



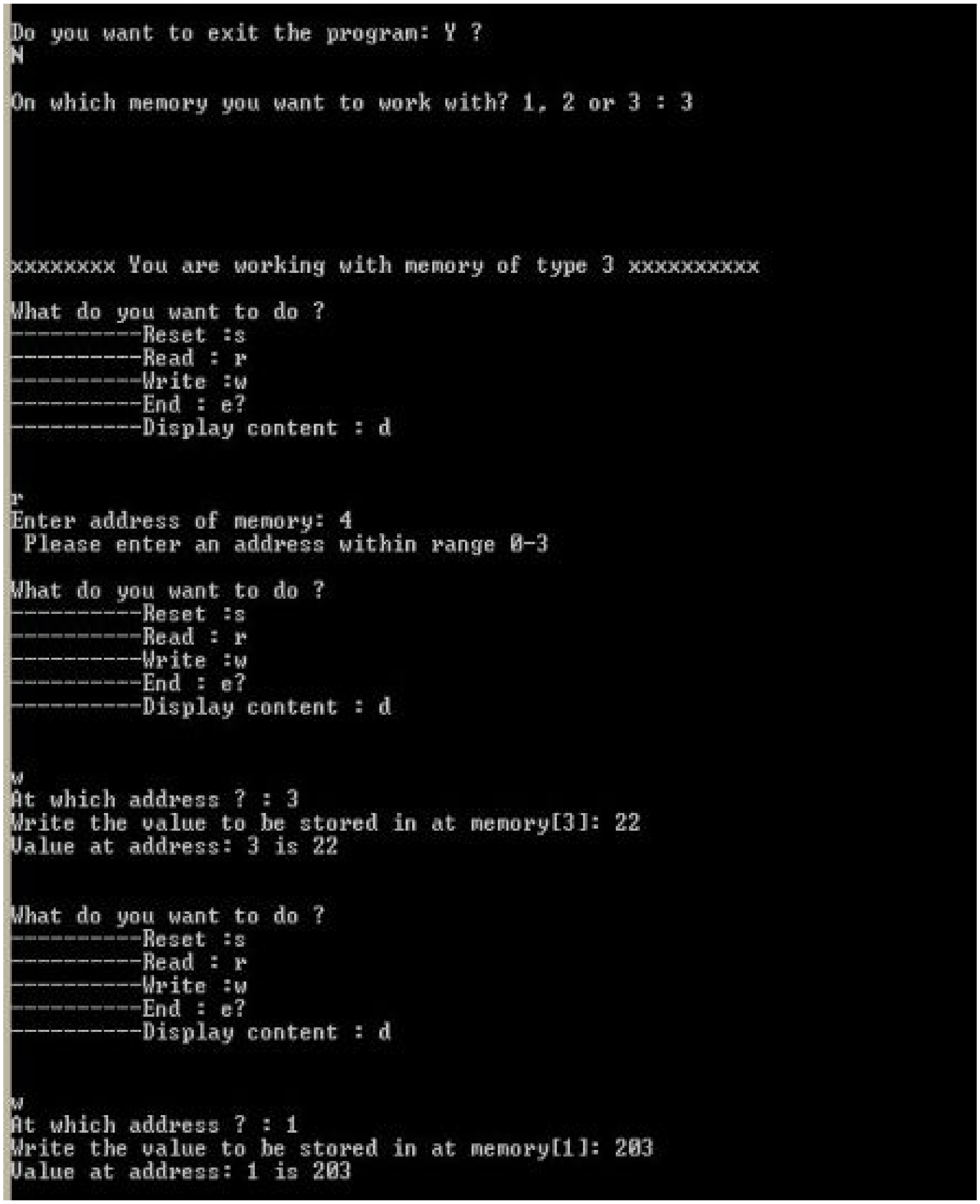
```
xxxxxxxx You are working with memory of type 1 xxxxxxxxxx

What do you want to do ?
-----------Read : r
-----------Write : w
-----------End : e?


r
Enter address of memory: -9
 Please enter an address within range 0-3

What do you want to do ?
-----------Read : r
-----------Write : w
-----------End : e?


r
Enter address of memory: 0
Value of memory at address: 0

What do you want to do ?
-----------Read : r
-----------Write : w
-----------End : e?


w
Enter value for address 0: 1
Enter value for address 1: 2
Enter value for address 2: 3
Enter value for address 3: 55


What do you want to do ?
-----------Read : r
-----------Write : w
-----------End : e?


r
Enter address of memory: 3
Value of memory at address: 55

What do you want to do ?
-----------Read : r
-----------Write : w
-----------End : e?


e
xxxxxxxx You left memory of type 1 xxxxxxxxxx
```

**Working with Type2 Memory:**



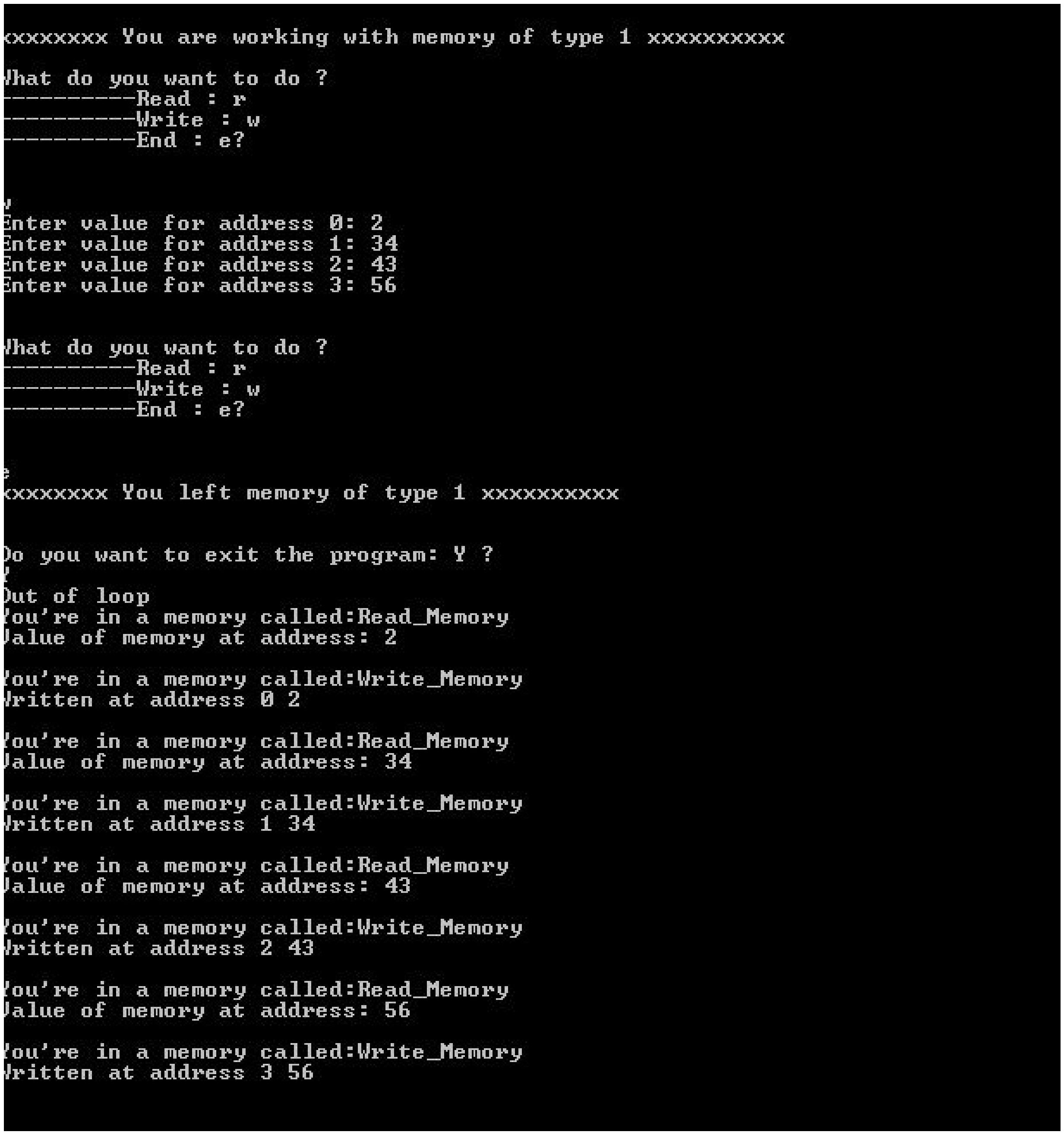
```
Do you want to exit the program: Y ?
N

On which memory you want to work with? 1, 2 or 3 : 2




xxxxxxxx You are working with memory of type 2 xxxxxxxxxx

What do you want to do ?
-----------Reset :s
-----------Read : r
-----------Write :w
-----------End : e?


r
Enter address of memory: 9.2
 Please enter an address within range 0-3

What do you want to do ?
-----------Reset :s
-----------Read : r
-----------Write :w
-----------End : e?



What do you want to do ?
-----------Reset :s
-----------Read : r
-----------Write :w
-----------End : e?


w
Enter value for address 0: 9
Enter value for address 1: 2
Enter value for address 2: 11
Enter value for address 3: 23


What do you want to do ?
-----------Reset :s
-----------Read : r
-----------Write :w
-----------End : e?


r
Enter address of memory: 2
Value of memory at address: 11
```

**Working with Type3 Memory:**



```
Do you want to exit the program: Y ?
N

On which memory you want to work with? 1, 2 or 3 : 3





xxxxxxxx You are working with memory of type 3 xxxxxxxxxx

What do you want to do ?
----------Reset :s
----------Read : r
----------Write :w
----------End : e?
----------Display content : d


r
Enter address of memory: 4
 Please enter an address within range 0-3

What do you want to do ?
----------Reset :s
----------Read : r
----------Write :w
----------End : e?
----------Display content : d


w
At which address ? : 3
Write the value to be stored in at memory[3]: 22
Value at address: 3 is 22


What do you want to do ?
----------Reset :s
----------Read : r
----------Write :w
----------End : e?
----------Display content : d


w
At which address ? : 1
Write the value to be stored in at memory[1]: 203
Value at address: 1 is 203
```

**Synchronization task - Reading data from one memory and simultaneously writing the data to another data as the address of former memory is changed.**



```
xxxxxxx You are working with memory of type 1 xxxxxxxxxx

hat do you want to do ?
----------Read : r
----------Write : w
----------End : e?


nter value for address 0: 2
nter value for address 1: 34
nter value for address 2: 43
nter value for address 3: 56


hat do you want to do ?
----------Read : r
----------Write : w
----------End : e?


xxxxxxx You left memory of type 1 xxxxxxxxxx


o you want to exit the program: Y ?

ut of loop
ou're in a memory called:Read_Memory
alue of memory at address: 2

ou're in a memory called:Write_Memory
ritten at address 0 2

ou're in a memory called:Read_Memory
alue of memory at address: 34

ou're in a memory called:Write_Memory
ritten at address 1 34

ou're in a memory called:Read_Memory
alue of memory at address: 43

ou're in a memory called:Write_Memory
ritten at address 2 43

ou're in a memory called:Read_Memory
alue of memory at address: 56

ou're in a memory called:Write_Memory
ritten at address 3 56
```