Write a program to simulate Memory placement strategies – worst fit in java for generalised with memory fragmentation size

Worst Fit :

```
package P1;

import java.util.Scanner;

class MemoryBlock {
    int size;
    boolean allocated;

    MemoryBlock(int size) {
        this.size = size;
        this.allocated = false;
    }
}

class Process {
    int size;
    int allocatedBlockIndex = -1;
    int fragmentation = 0;

    Process(int size) {
        this.size = size;
    }
}

public class WorstFitMemoryAllocation {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Input memory blocks
        System.out.print("Enter number of memory blocks: ");
        int numBlocks = sc.nextInt();
        MemoryBlock[] memory = new MemoryBlock[numBlocks];

        System.out.println("Enter sizes of memory blocks:");
        for (int i = 0; i < numBlocks; i++) {
            System.out.print("Block " + (i + 1) + ": ");
            memory[i] = new MemoryBlock(sc.nextInt());
        }

        // Input processes
        System.out.print("\nEnter number of processes: ");
        int numProcesses = sc.nextInt();
        Process[] processes = new Process[numProcesses];

        System.out.println("Enter sizes of processes:");
        for (int i = 0; i < numProcesses; i++) {
            System.out.print("Process " + (i + 1) + ": ");
            processes[i] = new Process(sc.nextInt());
        }
```

```
        // Worst Fit Allocation
        for (int i = 0; i < numProcesses; i++) {
            int worstIndex = -1;
            int maxSize = -1;

            for (int j = 0; j < numBlocks; j++) {
                if (!memory[j].allocated && memory[j].size >= processes[i].size) {
                    if (memory[j].size > maxSize) {
                        maxSize = memory[j].size;
                        worstIndex = j;
                    }
                }
            }

            if (worstIndex != -1) {
                processes[i].allocatedBlockIndex = worstIndex;
                processes[i].fragmentation = memory[worstIndex].size -
processes[i].size;
                memory[worstIndex].allocated = true;
            }
        }

        // Output allocation result
        System.out.println("\nAllocation Result:");
        System.out.println("Process\tSize\tBlock\tFragmentation");
        for (int i = 0; i < numProcesses; i++) {
            if (processes[i].allocatedBlockIndex != -1) {
                System.out.println("P" + (i + 1) + "\t" + processes[i].size +
"\tBlock " +
                        (processes[i].allocatedBlockIndex + 1) + "\t" +
processes[i].fragmentation);
            } else {
                System.out.println("P" + (i + 1) + "\t" + processes[i].size +
"\tNot Allocated");
            }
        }

        sc.close();
    }
}
```

Output :

```
Enter number of memory blocks: 5
Enter sizes of memory blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600

Enter number of processes: 4
Enter sizes of processes:
Process 1: 212
Process 2: 417
Process 3: 112
Process 4: 426

Allocation Result:
Process  Size     Block      Fragmentation
P1       212     Block 5         388
P2       417     Block 2         83
P3       112     Block 4         188
P4       426     Not Allocated
```