```
import java.util.Scanner;
class MemoryAllocation {
   // First Fit algorithm
   static void firstFit(int blockSize[], int m, int processSize[], int n) {
      int allocation[] = new int[n];
      for (int i = 0; i < allocation.length; i++)
         allocation[i] = -1;
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
               allocation[i] = j;
               blockSize[j] -= processSize[i];
               break;
            }
         }
      }
      printAllocation(allocation, processSize, n, "First Fit");
   }

   // Best Fit algorithm
   static void bestFit(int blockSize[], int m, int processSize[], int n) {
      int allocation[] = new int[n];
      for (int i = 0; i < allocation.length; i++)
         allocation[i] = -1;
      for (int i = 0; i < n; i++) {
         int bestIdx = -1;
         for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
```

```
            if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])
                bestIdx = j;
        }
      }
      if (bestIdx != -1) {
        allocation[i] = bestIdx;
        blockSize[bestIdx] -= processSize[i];
      }
    }
    printAllocation(allocation, processSize, n, "Best Fit");
}

// Next Fit algorithm
static void nextFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[] = new int[n];
    for (int i = 0; i < allocation.length; i++)
      allocation[i] = -1;

    int lastAllocated = 0;
    for (int i = 0; i < n; i++) {
      for (int j = lastAllocated; j < m; j++) {
        if (blockSize[j] >= processSize[i]) {
          allocation[i] = j;
          blockSize[j] -= processSize[i];
          lastAllocated = j; // Update last allocated block
          break;
        }
      }
```

```
        // If not found, check from the beginning
        if (allocation[i] == -1) {
            for (int j = 0; j < lastAllocated; j++) {
                if (blockSize[j] >= processSize[i]) {
                    allocation[i] = j;
                    blockSize[j] -= processSize[i];
                    lastAllocated = j; // Update last allocated block
                    break;
                }
            }
        }
    }
    printAllocation(allocation, processSize, n, "Next Fit");
}

// Worst Fit algorithm
static void worstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[] = new int[n];
    for (int i = 0; i < allocation.length; i++)
        allocation[i] = -1;

    for (int i = 0; i < n; i++) {
        int worstIdx = -1;
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])
                    worstIdx = j;
            }
```

```
        }
        if (worstIdx != -1) {
            allocation[i] = worstIdx;
            blockSize[worstIdx] -= processSize[i];
        }
    }
    printAllocation(allocation, processSize, n, "Worst Fit");
}

// Print allocation results
static void printAllocation(int allocation[], int processSize[], int n, String method) {
    System.out.println("\n" + method + " Allocation:");
    System.out.println("Process No.\tProcess Size\tBlock No.");
    for (int i = 0; i < n; i++) {
        System.out.print(" " + (i + 1) + "\t\t" + processSize[i] + "\t\t");
        if (allocation[i] != -1)
            System.out.print(allocation[i] + 1);
        else
            System.out.print("Not Allocated");
        System.out.println();
    }
}

// Main method
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.println("Enter number of memory blocks:");
```

```
        int m = sc.nextInt();
        int blockSize[] = new int[m];
        System.out.println("Enter sizes of memory blocks:");
        for (int i = 0; i < m; i++) {
            blockSize[i] = sc.nextInt();
        }

        System.out.println("Enter number of processes:");
        int n = sc.nextInt();
        int processSize[] = new int[n];
        System.out.println("Enter sizes of processes:");
        for (int i = 0; i < n; i++) {
            processSize[i] = sc.nextInt();
        }

        firstFit(blockSize.clone(), m, processSize, n);
        bestFit(blockSize.clone(), m, processSize, n);
        nextFit(blockSize.clone(), m, processSize, n);
        worstFit(blockSize.clone(), m, processSize, n);

        sc.close();
    }
}
```