import java.util.\*Scanner;  
  
    #构造函数，用于初始化作业对象  
    public Job(int id, int arrivalTime, int serviceTime) {  
        this.id = id;  
        this.arrivalTime = arrivalTime;  
        this.serviceTime = serviceTime;  
    }  
}  
  
public class SJFScheduling {  
    public static void main(String[] args) {  
       #示例作业队列  
        Job[] jobs = {  
            job1 = Job("A",0,4)  
            job2 = Job("B",1,6)  
            job3 = Job("C",2,5)  
        };  
  
        # 根据服务时间对作业队列进行排序  
        Arrays.sort(jobs, Comparator.comparingInt(job -> job.serviceTime));  
  
        # 执行作业调度  
        executeJobs(jobs);  
    }  
  
    # 执行作业调度的方法  
    public static void executeJobs(Job[] jobs) {  
        int currentTime = 0; # 当前时间初始化为0  
        double totalWaitingTime = 0; # 总等待时间初始化为0  
        System.out.println("Job\tArrival Time\tService Time\tStart Time\tFinish Time\tTurnaround Time\tWaiting Time");  
  
        # 遍历排序后的作业队列  
        for (Job job : jobs) {  
            # 计算作业的开始时间，要求开始时间不早于作业的到达时间  
            int startTime = Math.max(currentTime, job.arrivalTime);  
            # 计算作业的完成时间  
            int finishTime = startTime + job.serviceTime;  
            # 计算作业的周转时间  
            int turnaroundTime = finishTime - job.arrivalTime;  
            # 计算作业的等待时间  
            int waitingTime = startTime - job.arrivalTime;  
  
            # 更新总等待时间  
            totalWaitingTime += waitingTime;  
  
            # 输出作业的执行情况  
            System.out.println(job.id + "\t\t" + job.arrivalTime + "\t\t\t" + job.serviceTime + "\t\t\t" +  
                               startTime + "\t\t\t" + finishTime + "\t\t\t" + turnaroundTime + "\t\t\t" + waitingTime);  
  
            # 更新当前时间为作业完成时间  
            currentTime = finishTime;  
        }  
  
        # 计算平均等待时间  
        double averageWaitingTime = totalWaitingTime / jobs.length;  
        System.out.println("\nAverage Waiting Time: " + averageWaitingTime);  
    }  
}