```
/*
Student Name : Abhang Rushikesh
Roll NO : BEA-30
 */
// Java program to solve fractional Knapsack Problem
import java.util.Comparator;
import java.util.Arrays;
import java.util.Scanner;
// Itemvalues class
class KnapSack
{
    Double cost;
    double weight, profit, index;
    // item value function
    public KnapSack(int weight, int profit, int index)
    {
        this.weight = weight;
        this.profit = profit;
        this.index = index;
        // on the basis of cost we sort the item
        cost = new Double((double)profit / (double)weight);
    }
}
// the greedy approach
public class FractionalKnapsack
{
    // method to get maximum value
    private static double getMaxValue(int[] weight, int[] profit, int c)
    {
        // length of the array
        int size = weight.length;
        // array of storing all the items
        KnapSack[] iValue = new KnapSack[size];
        for (int j = 0; j < size; j++)
        {
            // storing the items
            iValue[j] = new KnapSack(weight[j], profit[j], j);
        }
        // sorting the items on the basis of cost
        Arrays.sort(iValue, new Comparator<KnapSack>()
        {
            @Override
            public int compare(KnapSack i1, KnapSack i2)
            {
                return i2.cost.compareTo(i1.cost);
            }
        });
        // contains the maximum value that is possible
        // for the given capacity of the knapsack
        double totalVal = 0d;
        for (int j = 0; j < size; j++)
        {
            // obtaining the current weight
            // and value of the ith item
            int currWt = (int)iValue[j].weight;
            int currVal = (int)iValue[j].profit;
            if (c - currWt >= 0)
            {
```

```
                // the current capacity of the knapsack allows the
                // item to be taken as a whole
                c = c - currWt;
                totalVal = totalVal + currVal;
            }
            else
            {
                // when an item can not be picked as a whole
                // we break the item and take a portion of it
                // such that we get the maximum value in the knapsack
                double fraction = ((double)c / (double)currWt);
                totalVal = totalVal + (currVal * fraction);
                c = (int)(c - (currWt * fraction));
                // the knapsack is full
                // no need to proceed further
                break;
            }
        }
        return totalVal;
    }

    // main method
    public static void main(String argvs[])
    {
        Scanner sc=new Scanner(System.in);
        int object,C;
        System.out.println("Enter the Total Objects");
        object=sc.nextInt();
        int weight[]=new int[object];
        int profit[]=new int[object];

        // input arrays
        for(int i=0;i<object;i++)
        {

            System.out.println("Enter the Profit");
            profit[i]=sc.nextInt();
            System.out.println("Enter the weight");
            weight[i]=sc.nextInt();
        }
        System.out.println("Enter the Knapsack capacity");
        C=sc.nextInt();

        // invoking the getMaxValue() method
        double maxVal = getMaxValue(weight, profit, C);
        // printing the result
        System.out.println("Maximum value that can be obtained is = " +
maxVal);
    }
}
```