

ASSIGNMENT-3

```
import java.util.*; // Import Java utility classes (for Scanner, List, Collections, etc.)  
  
// Class to represent an item with value and weight  
class Item {  
    int value; // Value (profit) of the item  
    int weight; // Weight of the item  
  
    // Constructor to initialize an item  
    Item(int value, int weight) {  
        this.value = value;  
        this.weight = weight;  
    }  
}  
  
// Main class that contains the main() method  
public class FractionalKnapsack {  
  
    // Comparator class to sort items based on value-to-weight ratio (in descending order)  
    static class Compare implements Comparator<Item> {  
        public int compare(Item a, Item b) {  
            // Calculate ratio of value/weight for both items  
            double r1 = (double) a.value / a.weight;  
            double r2 = (double) b.value / b.weight;  
  
            // Return descending order (higher ratio first)  
            return Double.compare(r2, r1);  
        }  
    }  
  
    // Function to calculate the maximum value possible within given capacity  
    public static double fractionalKnapsack(int capacity, List<Item> items) {  
        // Step 1: Sort items by their value-to-weight ratio  
        Collections.sort(items, new Compare());  
  
        double totalValue = 0.0; // To store final maximum value  
  
        // Step 2: Iterate over all sorted items  
        for (Item item : items) {  
            // If no capacity remains, break the loop  
            if (capacity == 0)  
                break;  
            // Add the item's value to totalValue if it fits  
            if (item.weight <= capacity) {  
                totalValue += item.value;  
                capacity -= item.weight;  
            }  
        }  
        return totalValue;  
    }  
}
```

```

// Case 1: If item can fit fully into the knapsack
if (item.weight <= capacity) {
    totalValue += item.value;      // Add full value
    capacity -= item.weight;     // Reduce capacity
}
// Case 2: If item can only fit partially
else {
    double fraction = (double) capacity / item.weight; // Fraction that can fit
    totalValue += item.value * fraction;                // Add proportional value
    capacity = 0;                                     // Knapsack is now full
}
}

// Step 3: Return the maximum total value
return totalValue;
}

// MAIN METHOD – Program execution starts here
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in); // Create scanner for user input

    // Step 1: Take number of items from user
    System.out.print("Enter number of items: ");
    int n = sc.nextInt();

    // Step 2: Create a list to store all items
    List<Item> items = new ArrayList<>();

    // Step 3: Input value and weight of each item
    System.out.println("Enter value and weight of each item:");
    for (int i = 0; i < n; i++) {
        int value = sc.nextInt(); // Input value
        int weight = sc.nextInt(); // Input weight
        items.add(new Item(value, weight)); // Add item to list
    }

    // Step 4: Input knapsack capacity
    System.out.print("Enter capacity of knapsack: ");
    int capacity = sc.nextInt();

    // Step 5: Calculate the maximum value using fractionalKnapsack function
    double maxValue = fractionalKnapsack(capacity, items);

    // Step 6: Display final result
}

```

```
        System.out.println("\nMaximum value in the knapsack = " + maxValue);

        sc.close() // Close scanner to prevent resource leak
    }
}
```

✓ Sample Output

```
Enter number of items: 3
Enter value and weight of each item:
60 10
100 20
120 30
Enter capacity of knapsack: 50

Maximum value in the knapsack = 240.0
```