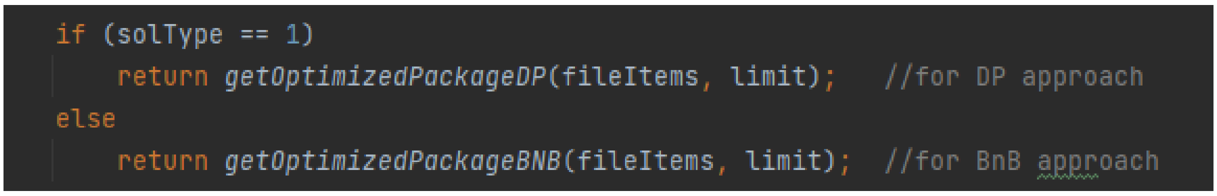
Solving the problem

Knapsack problem is one of the most popular problems in operational research field, and branch and bound algorithm is widely used for solving this type of problems.

Dynamic programming is also commonly used for knapsack problems because of its low time time complexity. (O(NxW) -N number of Items, -W bag's capacity)

In this assigment i involved both solutions and developed two different methods fort hem.

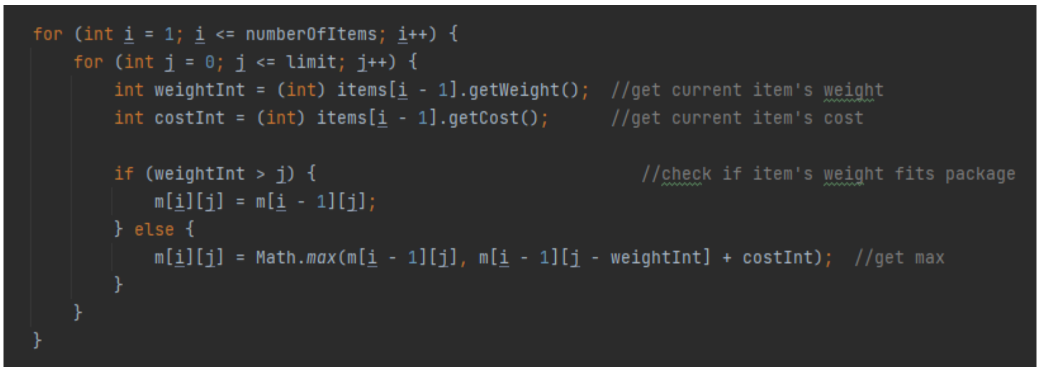
In packer.java's solveProblem method, you can select which approach yo want to solve problem by using solType



```
if (solType == 1)
    return getOptimizedPackageDP(fileItems, limit);    //for DP approach
else
    return getOptimizedPackageBNB(fileItems, limit);  //for BnB approach
```

Dynamic Programming Approach for Knapsack

Main idea is, use a matrix to store solutions of solutions of subproblems. If current item's weight is fitting to bag, check if previous weight is higher than previous object plus current item's cost.



```
for (int i = 1; i <= numberOfItems; i++) {
    for (int j = 0; j <= limit; j++) {
        int weightInt = (int) items[i - 1].getWeight();  //get current item's weight
        int costInt = (int) items[i - 1].getCost();      //get current item's cost

        if (weightInt > j) {                               //check if item's weight fits package
            m[i][j] = m[i - 1][j];
        } else {
            m[i][j] = Math.max(m[i - 1][j], m[i - 1][j - weightInt] + costInt);  //get max
        }
    }
}
```

Branch and Bound Approach for Knapsack

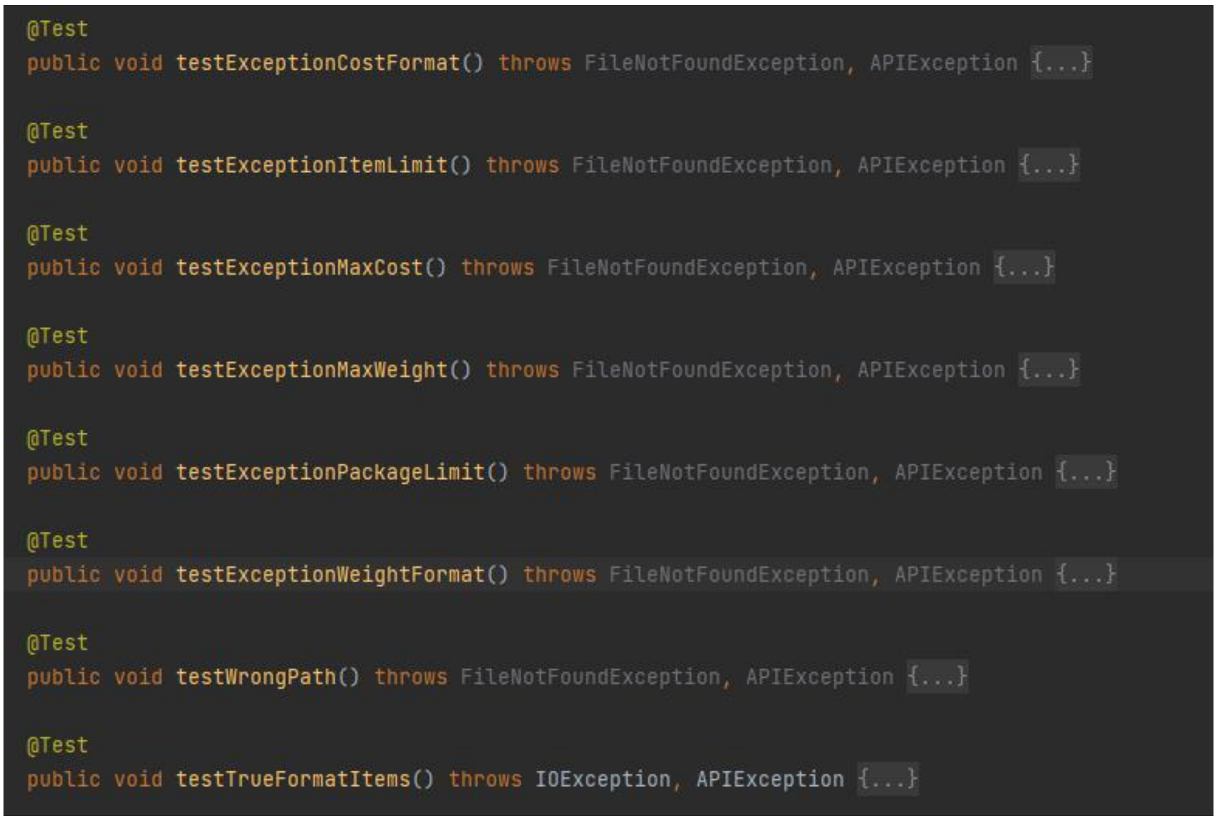
The branch and bound algorithm is similar to backtracking but is used for optimization problems.

I firstly ordered items by their ratio(value/weight)

I created a priority queue as a tree. The root of the tree is the empty knapsack. Each level of the tree can consider adding or not adding the item to the knapsack. The nodes contain the total weight, and value of the knapsack.

Testing

I created 8 test cases. One is true formatted which returns the solution. Other 7 are returning exceptions for different cases.



```java
@Test
public void testExceptionCostFormat() throws FileNotFoundException, APIException {...}

@Test
public void testExceptionItemLimit() throws FileNotFoundException, APIException {...}

@Test
public void testExceptionMaxCost() throws FileNotFoundException, APIException {...}

@Test
public void testExceptionMaxWeight() throws FileNotFoundException, APIException {...}

@Test
public void testExceptionPackageLimit() throws FileNotFoundException, APIException {...}

@Test
public void testExceptionWeightFormat() throws FileNotFoundException, APIException {...}

@Test
public void testWrongPath() throws FileNotFoundException, APIException {...}

@Test
public void testTrueFormatItems() throws IOException, APIException {...}
```