Good.java

```java
package knapsack;

/**
 * 荷物のクラス
 *
 * @author tadaki
 */
public class Good {

    private final int weight;//重量
    private final int value;//価値
    private final String label;//ラベル

    public Good(int weight, int value, String label) {
        this.weight = weight;
        this.value = value;
        this.label = label;
    }

    public int getWeight() {
        return weight;
    }

    public int getValue() {
        return value;
    }

    public String getLabel() {
        return label;
    }

}
```

Knapsack.java

```java
package knapsack;

import java.util.List;
import myLib.utils.Utils;

/**
 *
 * @author tadaki
 */
public class Knapsack implements Cloneable {

    private List<Good> goods;
    private int value;
    private int weight;

    public Knapsack() {
        goods = Utils.createList();
        value = 0;
        weight = 0;
    }

    public void addGood(Good g) {
        if (goods.contains(g)) {
            return;
        }
        goods.add(g);
        value += g.getValue();
        weight += g.getWeight();
    }

    public List<Good> getGoods() {
        return goods;
    }

    public int getValue() {
        return value;
    }

    public int getWeight() {
        return weight;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("[");
        goods.stream().forEachOrdered(g -> {
            sb.append(g.getLabel()).append(" ");
        });
        sb.append("] v=").append(value).append(" w=").append(weight);
        return sb.toString();
    }

    public boolean contains(Good g) {
        return goods.contains(g);
    }

    public boolean remove(Good g) {
        if (goods.contains(g)) {
            value -= g.getValue();
            weight -= g.getWeight();
        }
        return goods.remove(g);
    }

    @Override
    public Knapsack clone() throws CloneNotSupportedException {
        Knapsack r = (Knapsack) super.clone();
        r.goods = Utils.createList();
        goods.stream().forEachOrdered(g -> {
            r.goods.add(g);
        });
        r.value = value;
        r.weight = weight;
        return r;
    }

}
```

AbstractKnapsack.java

```java
package knapsack;

import java.util.List;

/**
 * Knapsack問題解法の抽象クラス
 * @author tadaki
 */
public abstract class AbstractKnapsack {

    protected final List<Good> goods;//荷物の一覧
    protected final int maxWeight;//許容重量
    protected Knapsack knapsack;//ナップザックの最終的状態
    protected boolean debug = true;
    protected int count;

    /**
     * コンストラクタ
     * @param goods 荷物リスト
     * @param maxWeight 許容重量
     */
    public AbstractKnapsack(List<Good> goods, int maxWeight) {
        this.goods = goods;
        this.maxWeight = maxWeight;
        count = 0;
    }

    /**
     * 解法の入り口
     * @return ナップザック内の価値の総和
     * @throws CloneNotSupportedException
     */
    public int doExec() throws CloneNotSupportedException{
        knapsack = doRec(0, maxWeight);
        return knapsack.getValue();
    }

    /**
     * 解法の実装部
     * @param i 荷物番号
     * @param w 残りの重量
     * @return ナップザックの状態
     * @throws CloneNotSupportedException
     */
    abstract protected Knapsack doRec(int i, int w) throws
CloneNotSupportedException;
```

```
    /**
     * ナップサックの中身を得る
     * @return
     */
    public Knapsack getKnapsack() {
        return knapsack;
    }

    /**
     * 操作の工数
     * @return
     */
    public int getCount() {
        return count;
    }

    public void setDebug(boolean debug) {
        this.debug = debug;
    }

}
```

Recursive.java

```
package knapsack;

import java.util.List;

/**
 * 単純な再帰手法
 * @author tadaki
 */
public class Recursive extends AbstractKnapsack {

    public Recursive(List<Good> goods, int maxWeight) {
        super(goods, maxWeight);
    }

    @Override
    protected Knapsack doRec(int i, int w) throws CloneNotSupportedException
{
        if (i == goods.size()) {//他の品物の選択肢は無い
            return new Knapsack();
        }
        count++;
        Good g = goods.get(i);
        if (w < g.getWeight()) {
            return doRec(i + 1, w).clone();//i 番目は使用しない
        }
        //i 番目を使用する場合としない場合の価値の大きいほうを採用
        Knapsack k1 = doRec(i + 1, w).clone();
        Knapsack k2 = doRec(i + 1, w - g.getWeight());
        k2.addGood(g);
        if (k2.getValue() >= k1.getValue()) {
            k1 = k2;
        }
        if (debug) {
            System.out.println("debug:" + k1.toString());
        }
        return k1.clone();
    }

}
```

DynamicalProgramming.java

```java
package knapsack;

import java.util.List;

/**
 * 動的計画法による解法
 * @author tadaki
 */
public class DynamicalProgramming extends AbstractKnapsack {

    private Knapsack[][] q;

    public DynamicalProgramming(List<Good> goods, int maxWeight) {
        super(goods, maxWeight);
        q = new Knapsack[goods.size() + 1][maxWeight + 1];
        for (int i = 0; i < goods.size() + 1; i++) {
            for (int j = 0; j < maxWeight + 1; j++) {
                q[i][j] = null;
            }
        }
    }

    @Override
    protected Knapsack doRec(int i, int w) throws CloneNotSupportedException
{
        if (q[i][w] != null) {
            return q[i][w].clone();
        }
        Knapsack newkn;
        if (i == goods.size()) {//他の品物の選択肢は無い
            newkn = new Knapsack();
        } else {
            count++;
            Good g = goods.get(i);
            if (w < g.getWeight()) {//i 番目は使用しない
                newkn = doRec(i + 1, w);
            } else {
                //i 番目を使用する場合としない場合の価値の大きいほうを採用
                Knapsack k1 = doRec(i + 1, w);
                Knapsack k2 = doRec(i + 1, w - g.getWeight());
                k2.addGood(g);
                if (k2.getValue() >= k1.getValue()) {
                    newkn = k2;
                } else {
                    k1.remove(g);
                    newkn = k1;
                }
            }
        }
        if (debug) {
            StringBuilder sb = new StringBuilder();
            sb.append("debug q[").append(i).append("][");
            sb.append(w).append("]=").append(newkn.toString());
            System.out.println(sb.toString());
        }
        q[i][w] = newkn;
        return q[i][w].clone();
    }

}
```

Sequential.java

```java
package knapsack;

import java.util.List;

/**
 *
 * @author tadaki
 */
public class Sequential extends AbstractKnapsack {

    private Knapsack[][] q;

    public Sequential(List<Good> goods, int maxWeight) {
        super(goods, maxWeight);
        q = new Knapsack[goods.size() + 1][maxWeight + 1];
        for (int i = 0; i < goods.size() + 1; i++) {
            for (int j = 0; j < maxWeight + 1; j++) {
                q[i][j] = new Knapsack();
            }
        }
    }

    @Override
    protected Knapsack doRec(int k, int w) throws CloneNotSupportedException
{
        for (int i = goods.size() - 1; i >= 0; i--) {
            Good g = goods.get(i);
            for (int j = 0; j <= w; j++) {
                count++;
                if (j < g.getWeight()) {
                    q[i][j] = q[i + 1][j].clone();
                } else {
                    Knapsack k1 = q[i + 1][j].clone();
                    Knapsack k2 = q[i + 1][j - g.getWeight()].clone();
                    k2.addGood(g);
                    if (k2.getValue() >= k1.getValue()) {
                        q[i][j] = k2;
                    } else {
                        q[i][j] = k1;
                    }
                }
            }
        }
        return q[0][w];
    }
}
```