

JUnit入門

技法の実践方法

参考文献：現場で使えるソフトウェアテスト -Java編-

発表者 吉富 楓雅

ではお題のテストケース
を考えましょう！



お題のテストケース

- Shippingクラスの単体テストで実施するテストケースの作成
 - * ただしコードカバレッジとして、命令網羅および分岐網羅は100%を満たしたい

お題について

- 引数weightには、荷物の重さを指定する。ただし重量は0kg超30kg以下の範囲で指定する。範囲外の数値が設定された場合は、illegalArgumentExceptionをスローする
- 引数x, y, zには荷物の各辺の長さを指定する。ただし各辺は0cm超であり、かつ3辺の合計が180cm以内の範囲で指定する。範囲外の数値が設定された場合は、illegalArgumentExceptionをスローする
- 指定されたデータから、次ページの表の送料一覧に基づいて送料を算出し戻り値として返却する。なお、表の「サイズ」は「サイズ区分」から決定される。例えば、3辺合計が50cmで重量が15kgの場合は、サイズはMとなる。

送料とサイズ区分

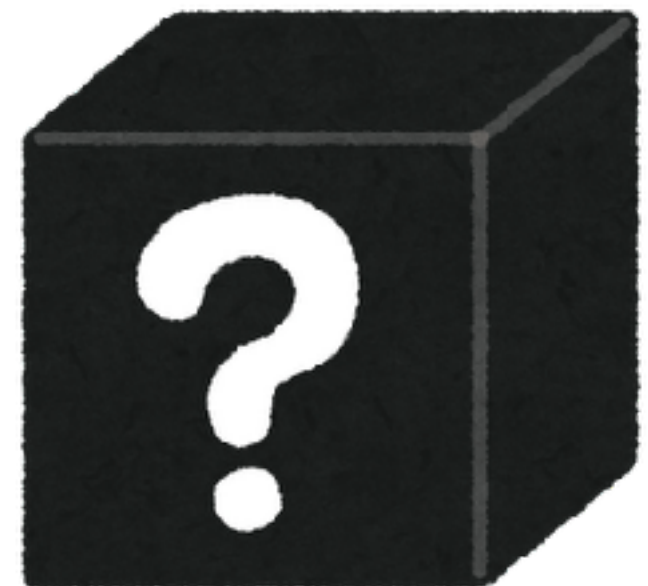
送料一覧

サイズ	価格
S	600円
M	1200円
L	1800円

サイズ区分

サイズ	3辺合計	重量
S	60cm以下	10kg以下
M	120cm以下	20kg以下
L	180cm以下	30kg以下

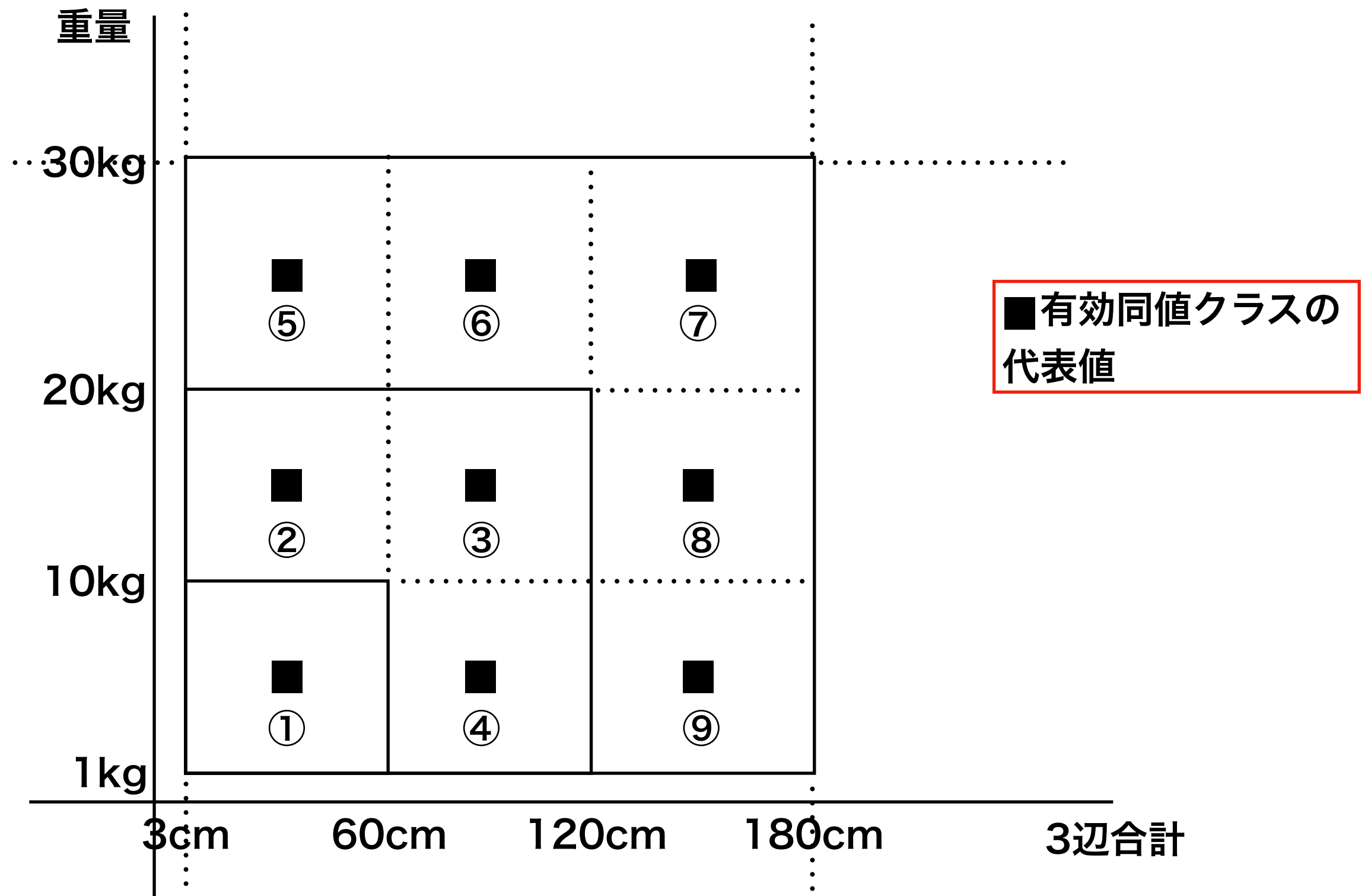
ブラックボックステストの 観点によるテストケースの作成



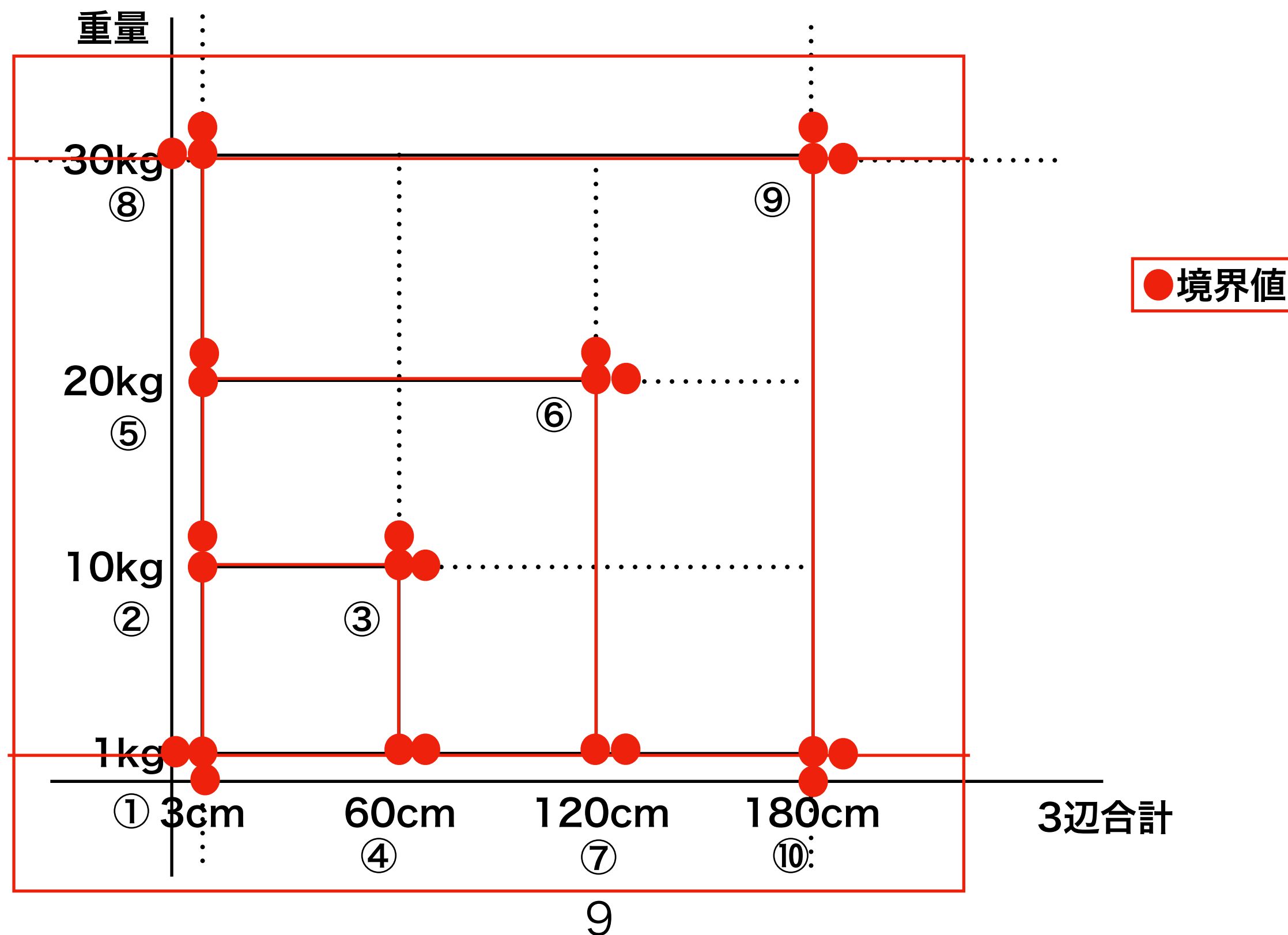
有効値の割り出し

- 荷物の重量と3辺合計に対する制限から次のようになる
 - * 重量は 1 kg以上30kg以下
 - ▶ 重量が0kg超(重量がint型のために1kg以上)
 - * 3辺合計は, 3cm以上180cm以下(ただし各辺は0cm超)
 - ▶ 各辺が0cm超(各辺はint型のために1cm以上)

同値クラス分類(有効値)

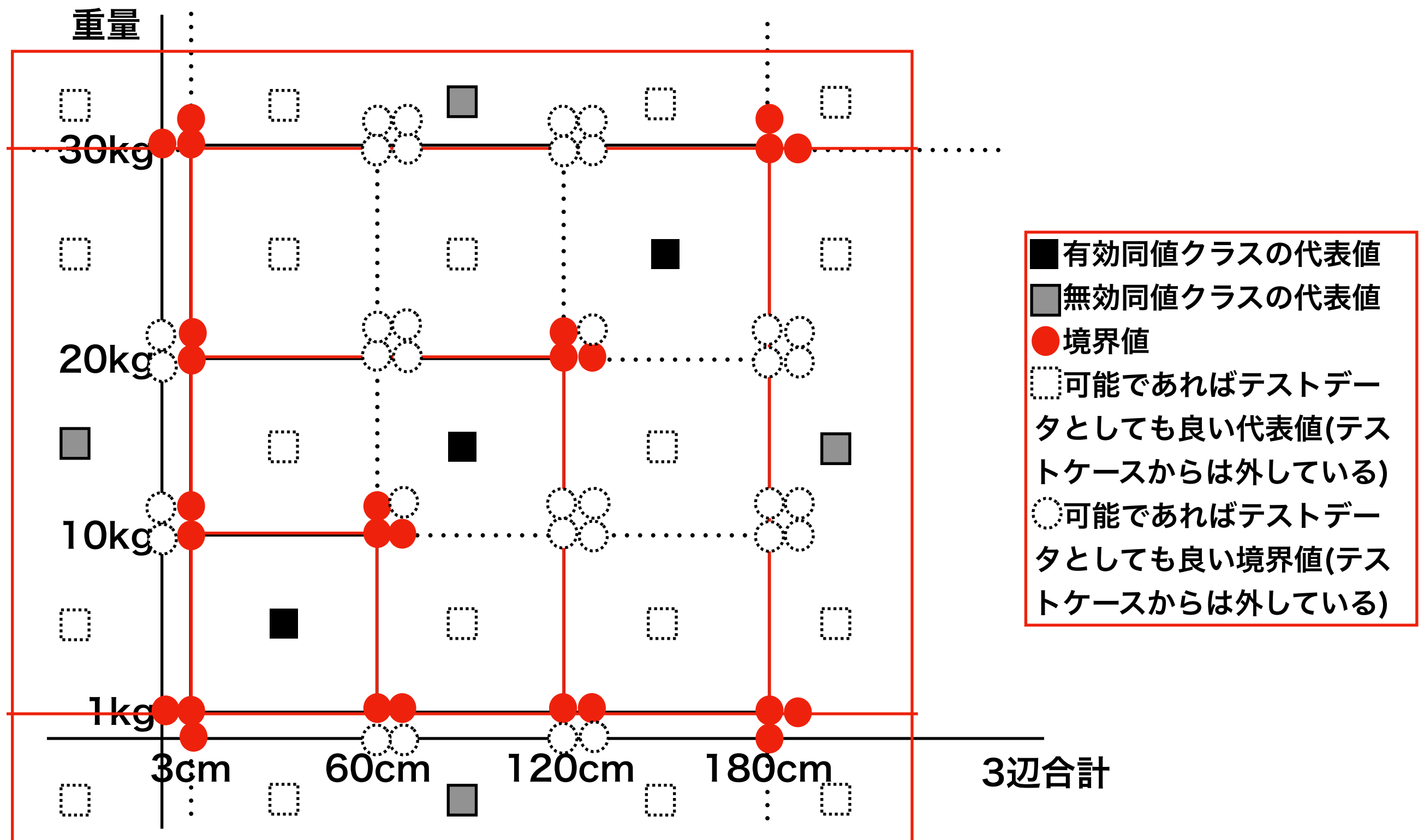


重量と3辺合計による境界値の分類



パターン	入力（テストデータ）	出力 （期待値）	パターン	入力（テストデータ）	出力 （期待値）
1	weight=5, x=10, y=10, z=10	600	21	weight=20, x=1, y=1, z=1	1200
2	weight=15, x=30, y=30, z=30	1200	22	weight=21, x=1, y=1, z=1	1800
3	weight=25, x=50, y=50, z=50	1800	23	weight=20, x=40, y=40, z=40	1200
4	weight=-100, x=30, y=30, z=30	例外	24	weight=21, x=40, y=40, z=40	1800
5	weight=100, x=30, y=30, z=30	例外	25	weight=20, x=40, y=40, z=41	1800
6	weight=15, x=-10, y=-10, z=-10	例外	26	weight=1, x=40, y=40, z=40	1200
7	weight=15, x=100, y=100, z=100	例外	27	weight=1, x=40, y=40, z=41	1800
8	weight=15, x=-10, y=10, z=10	例外	28	weight=30, x=1, y=1, z=1	1800
9	weight=15, x=10, y=-10, z=10	例外	29	weight=31, x=1, y=1, z=1	例外
10	weight=15, x=10, y=10, z=-10	例外	30	weight=30, x=0, y=1, z=1	例外
11	weight=1, x=1, y=1, z=1	600	31	weight=30, x=60, y=60, z=60	1800
12	weight=0, x=1, y=1, z=1	例外	32	weight=31, x=60, y=60, z=60	例外
13	weight=1, x=0, y=1, z=1	例外	33	weight=30, x=60, y=60, z=61	例外
14	weight=10, x=1, y=1, z=1	600	34	weight=1, x=60, y=60, z=60	1800
15	weight=11, x=1, y=1, z=1	1200	35	weight=1, x=60, y=60, z=61	例外
16	weight=10, x=20, y=20, z=20	600	36	weight=0, x=60, y=60, z=60	例外
17	weight=11, x=20, y=20, z=20	1200	37	weight=15, x=0, y=10, z=10	例外
18	weight=10, x=20, y=20, z=21	1200	38	weight=15, x=1, y=10, z=10	1200
19	weight=1, x=20, y=20, z=20	600	39	weight=15, x=10, y=0, z=10	例外
20	weight=1, x=20, y=20, z=21	1200	40	weight=15, x=10, y=1, z=10	1200
			41	weight=15, x=10, y=10, z=0	例外
			42	weight=15, x=10, y=10, z=1	1200

抽出されたテストケース



ホワイトボックステストの 観点によるテストケースの作成



▼リスト5-2 Shipping.java

```
package jp.co.shosisha.example;
```

```
/**
 * 送料計算用クラス。
 *
 * @author 野村 伸
 */
```

```
public class Shipping {
```

```
/**
 * サイズSを表す定数。
 */
```

```
private static final int SIZE_S = 1;
```

```
/**
 * サイズMを表す定数。
 */
```

```
private static final int SIZE_M = 2;
```

```
/**
 * サイズLを表す定数。
 */
```

```
private static final int SIZE_L = 3;
```

```
/**
 * 荷物の送料を計算する。
 *
 * @param weight
 *         重量
 * @param x
 *         荷物の縦幅
 * @param y
 *         荷物の幅
 * @param z
 *         荷物の高さ
 * @return 荷物の送料
 */
```

```
public int calculate(int weight, int x, int y, int z) {
```

```
    // 仕様1に準拠
```

```
    if (weight <= 0 || weight > 30) { 分岐①  
        throw new IllegalArgumentException(); 命令①  
    }
```

```
    // 仕様2に準拠
```

```
    int total = x + y + z; 命令②  
    if (x <= 0 || y <= 0 || z <= 0 || total > 160) { 分岐②  
        throw new IllegalArgumentException(); 命令③  
    }
```

```
    // 仕様3に準拠
```

```
    // 重量だけから求めたサイズと  
    // 3辺の合計から求めたサイズを比較して  
    // 大きいサイズのものが荷物のサイズとなる。
```

```
    int weightSize = getWeightSize(weight);  
    int totalSize = getTotalSize(total);  
    int size = Math.max(weightSize, totalSize);
```

```
    int price;
```

```
    switch (size) {
```

```
        case SIZE_S: 分岐③
```

```
            price = 500;  
            break;
```

```
        case SIZE_M: 分岐④
```

```
            price = 1200;  
            break;
```

```
        default: 命令④
```

```
            price = 1800;  
            break;
```

```
    } 命令⑤
```

```
    return price;
```

```
/**
 * 重量をベースとしたサイズを取得する。
 *
 * @param weight
 *         重量
 * @return 重量だけから求めたサイズ
 */
```

```
private int getWeightSize(int weight) {
```

```
    if (weight <= 10) { 分岐⑥
```

```
        return SIZE_S; 命令⑥
```

```
    } else if (weight <= 20) { 分岐⑦
```

```
        return SIZE_M; 命令⑦
```

```
    } else { 命令⑧
```

```
        return SIZE_L; 命令⑨
```

```
/**
 * 3辺の合計をベースとしたサイズを取得する。
 *
 * @param total
 *         3辺の合計
 * @return 3辺の合計から求めたサイズ
 */
```

```
private int getTotalSize(int total) {
```

```
    if (total <= 50) { 分岐⑩
```

```
        return SIZE_S; 命令⑩
```

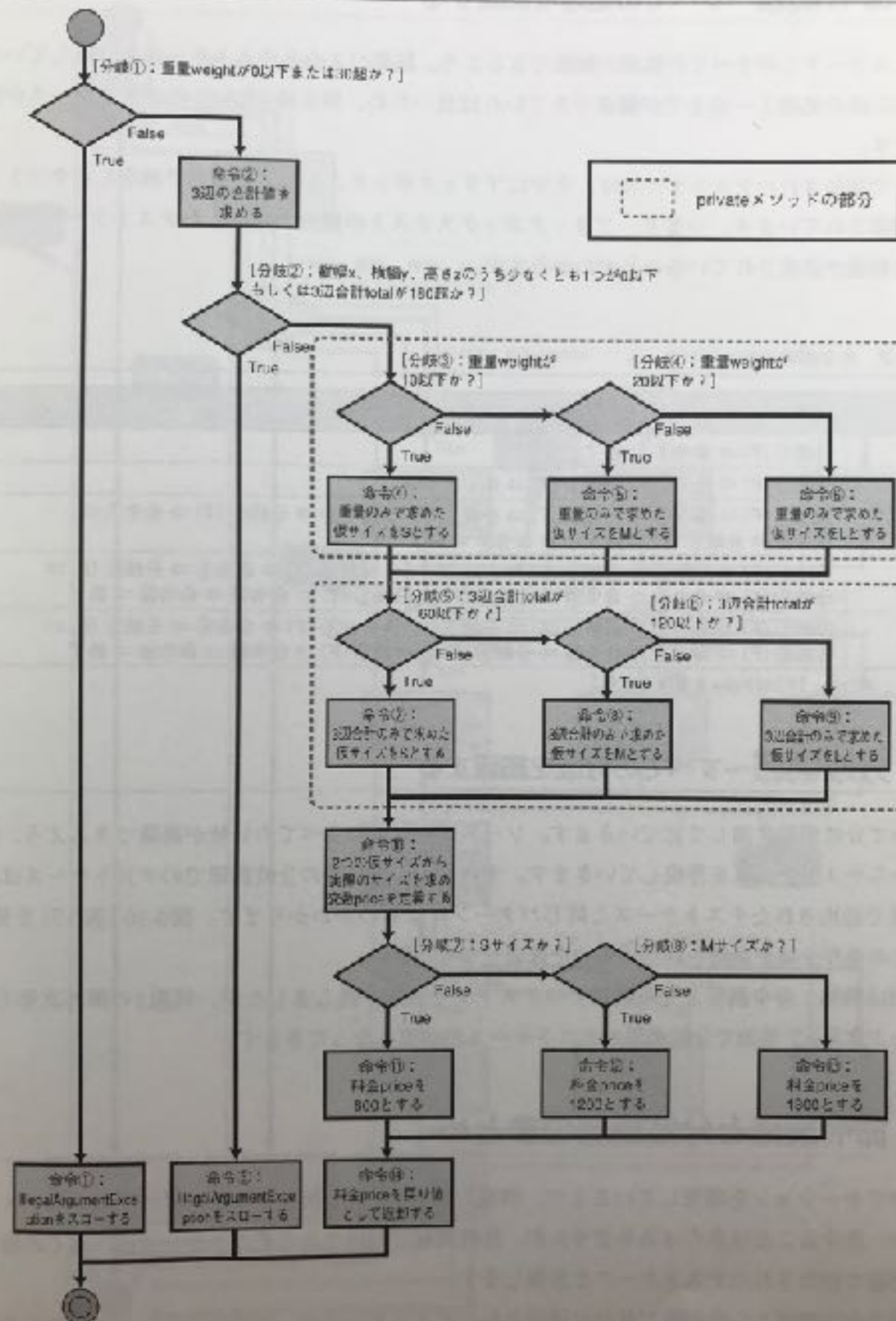
```
    } else if (total <= 120) { 分岐⑪
```

```
        return SIZE_M; 命令⑪
```

```
    } else { 命令⑫
```

```
        return SIZE_L; 命令⑬
```

▼図5-15 制御パス



命令網羅

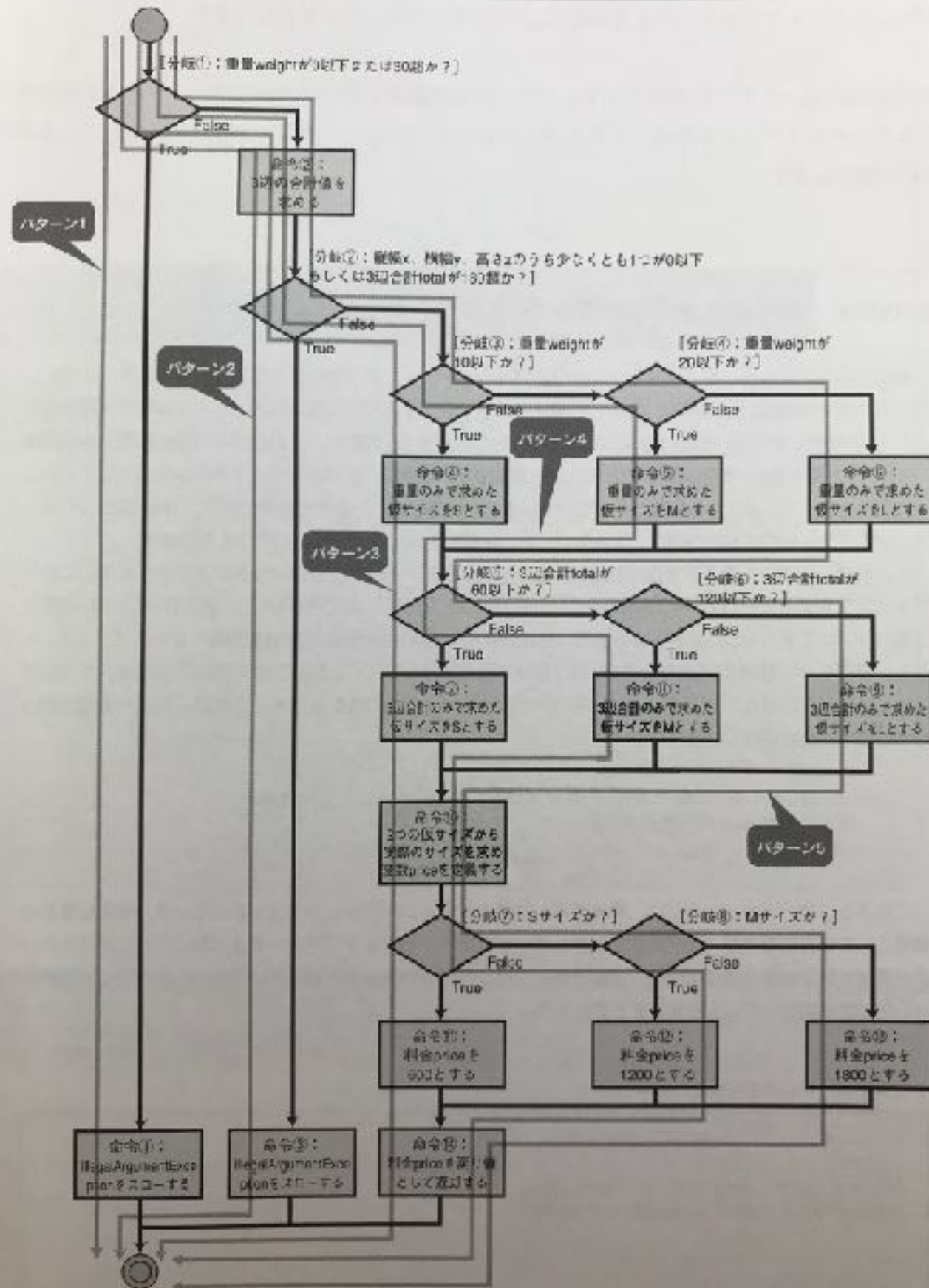
全ての処理を網羅する

▼表5-7 命令網羅でのテストケース

パターン	パス
1	分岐① (T) ⇒ 命令① ⇒ 終了
2	分岐① (F) ⇒ 命令② ⇒ 分岐② (T) ⇒ 命令③ ⇒ 終了
3	分岐① (F) ⇒ 命令② ⇒ 分岐② (F) ⇒ 分岐③ (T) ⇒ 命令④ ⇒ 分岐⑤ (T) ⇒ 命令⑦ ⇒ 命令⑩ ⇒ 分岐⑦ (T) ⇒ 命令⑪ ⇒ 命令⑭ ⇒ 終了
4	分岐① (F) ⇒ 命令② ⇒ 分岐② (F) ⇒ 分岐③ (F) ⇒ 分岐④ (T) ⇒ 命令⑤ ⇒ 分岐⑤ (F) ⇒ 分岐⑥ (T) ⇒ 命令⑧ ⇒ 命令⑩ ⇒ 分岐⑦ (F) ⇒ 分岐⑧ (T) ⇒ 命令⑫ ⇒ 命令⑭ ⇒ 終了
5	分岐① (F) ⇒ 命令② ⇒ 分岐② (F) ⇒ 分岐③ (F) ⇒ 分岐④ (F) ⇒ 命令⑥ ⇒ 分岐⑤ (F) ⇒ 分岐⑥ (F) ⇒ 命令⑨ ⇒ 命令⑩ ⇒ 分岐⑦ (F) ⇒ 分岐⑧ (F) ⇒ 命令⑬ ⇒ 命令⑭ ⇒ 終了

【*】「T」はTrue、「F」はFalseを意味します。

▼図5-16 命令網羅でのテストケース



分岐網羅 全ての分岐を網羅する

- ソースコード内のすべての分岐に対して、真と偽となるパターンを 1 回以上実行する
- 今回は命令網羅で抽出されたテストケースと同じになる

テストケース作成のまとめ

- 今回はブラックボックステストで全てのテストケースを満たすことができた
 - * 実際の現場ではこのような例は少ないので、その時の状況に応じてテストケースを追加する
- 命令網羅や分岐網羅が100%でも安心してはいけない
 - * 仕様の通りに動作していなくても、プログラムとしては正常に動作してしまう場合がある