

# 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

# 目次

- お題
- プロダクトコードの作成
- テストケース作成の進め方
- テストケースを作成するための技法
- テストの記述

お題



# “荷物の送料を計算するコード”

次の条件1～5に基づいてコードを作成せよ

それでは、ここからは具体的な題材を利用して、プロセスと技法についての理解を深めていきましょう。まずは、次の例題に目を通してください。

## 例題1

以下の条件1～5に基づいて、「荷物の送料を計算する」ソースコードを作りなさい。

**条件1** 作成するクラスは、表Aに従ったクラス定義とする。

▼表A クラス定義

|          |                                                 |        |            |
|----------|-------------------------------------------------|--------|------------|
| パッケージ名   | jp.co.shoeisha.example                          |        |            |
| クラス      | Shipping                                        |        |            |
| クラスの修飾子  | public                                          |        |            |
| メソッド     | int calculate (int weight, int x, int y, int z) |        |            |
|          | 引数                                              | weight | 荷物の重さ (kg) |
|          |                                                 | x      | 荷物の縦幅 (cm) |
|          |                                                 | y      | 荷物の横幅 (cm) |
|          |                                                 | z      | 荷物の高さ (cm) |
|          | 戻り値                                             | 送料 (円) |            |
| メソッドの修飾子 | public                                          |        |            |

**条件2** calculate()メソッドは、次の3つの仕様に基づいて作成すること。privateメソッドは、自由に定義しても良い。

- ①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
- ②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
- ③指定されたデータから、表B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Bのサイズは、表Cのサイズ区分から決定される。たとえば、3辺合計が50cmで重量が15kgの場合は、サイズはMとなる

▼表B 送料一覧

| サイズ | 価格    |
|-----|-------|
| S   | 600円  |
| M   | 1200円 |
| L   | 1800円 |

▼表C サイズ区分

| サイズ | S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 |

**条件3** Javadocを記述し、Sunのコーディング規約に沿っていること。ただしJavadocコメントの最初の一文の終わりには句点「。」を利用する。また、Sunのコーディング規約4章「インデント」の冒頭に  
あるインデントに関してのみ、4個の半角スペースに相当するタブを利用する形に読み替える（Eclipseの  
デフォルトフォーマットルールと同等）。

**条件4** 静的解析ツール（CheckstyleおよびFindBugs）のチェックもパスすること。なお、Checkstyle  
で利用するチェックルールは、「Sun Checks (Eclipse)」から表Dで示した5つのチェックルールを変更  
したものとする（Checkstyleのルール内容に関しては、巻末付録B「Checkstyle／FindBugs」ルール  
一覧）を参照）。

▼表D 変更するチェックルール

| グループ名            | モジュール名               | 変更内容                        |
|------------------|----------------------|-----------------------------|
| Javadoc Comments | Package Html         | チェック対象外とする                  |
| Javadoc Comments | Style Javadoc        | checkFirstSentenceをfalseに変更 |
| Coding Problem   | Magic Number         | チェック対象外とする                  |
| Class Design     | Design for Extension | チェック対象外とする                  |
| Miscellaneous    | Final Parameters     | チェック対象外とする                  |

**条件5** 次のレビュー観点を参考にソースコードをレビューすること。

- サイズ区分を決める3辺合計や重量の条件で指定された以上／下、以下／未満といった細かな表現が正しくプログラミングされているか？
- 異常値が入力されたときの例外処理がプログラミングされているか？
- クラス名やメソッド名が正しいか？
- ソースコードに妥当なコメントが記述されているか？
- Javadocコメントの最初の一文の終わりには句点「。」が利用されているか？
- CheckstyleとFindBugsでのエラーがすべて解消されているか？

# プロダクトコード の作成



# プロジェクトの作成

- プロジェクト名やパッケージ名も忠実に従ってください
  - \* 自分勝手にプロジェクトを作成しない
    - ▶ 統一性が無くなればトラブルが発生する原因となる

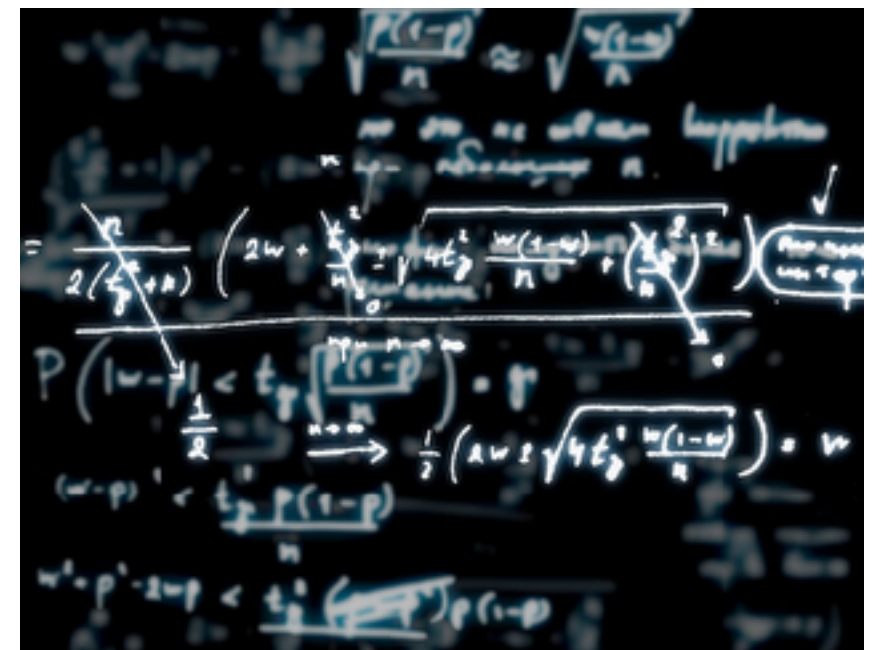
# コードの生成

- Javaの書き方に従ってコードを記述してください
  - \* インデントなどにも注意して, わかりやすいコードを記述することに努める
- privateとpublicも使い分けてください
  - \* 計算式はprivateでも構わないと思いますが, 数値など他のプロジェクトで使われるとまずいものはprivateにしてください

# コードについて

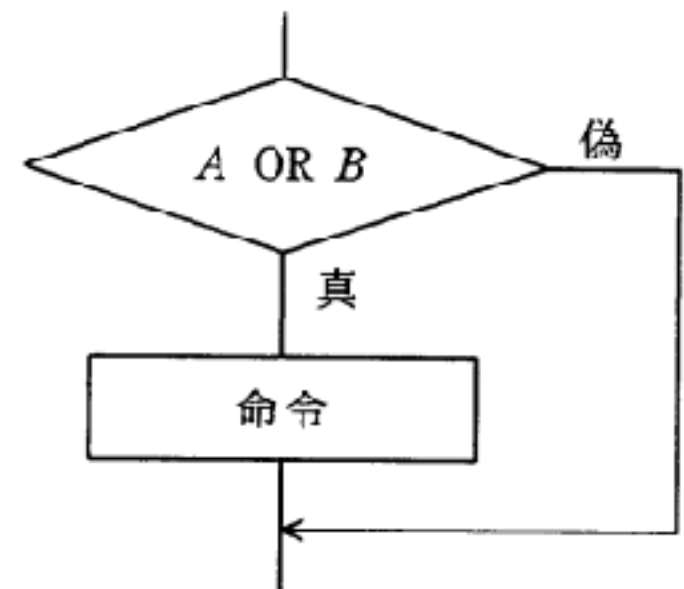
- 模範解答に当たるコードは以下のURLからダウンロードすることも可能です. 答え合わせに使ってください
  - \* <http://www.seshop.com/book/download/>
    - ▶ ISBNを聞かれますので, 「978-4-7981-1463-7」と入力してください
- 以降のテストコードにつきましても, 圧縮ファイルに含まれていますので, ご覧になってください

# しばらく理論を お話しします



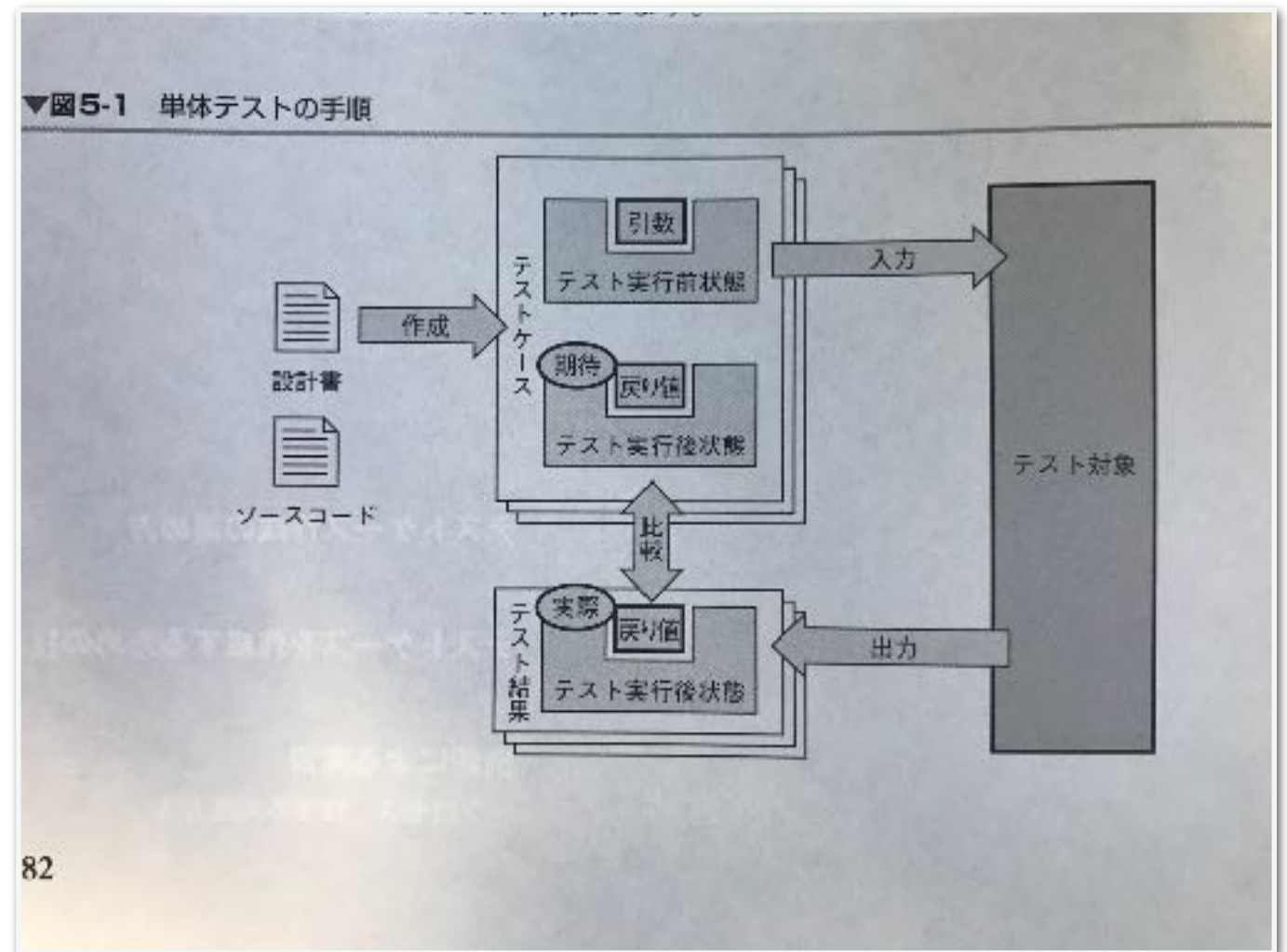
The image shows handwritten mathematical derivations on a dark background. At the top, it states  $\sqrt{\frac{P(1-P)}{n}} \approx \sqrt{\frac{w(1-w)}{n}}$ . Below this, there is a complex fraction involving  $\frac{n}{2(\frac{1}{2} + h)}$  and a large expression in parentheses. The main part of the derivation shows the probability  $P\left(\left|w - \frac{1}{2}\right| < \frac{1}{2} \sqrt{\frac{P(1-P)}{n}}\right) = P$  being approximated by  $\frac{1}{2} \left(1 + \sqrt{\frac{4t_p^2}{n} \frac{w(1-w)}{n}}\right) = w$ . The final line shows  $w^2 - P^2 - 2wP < \frac{t_p^2}{n} P(1-P)$ .

# テストケース作成 の進め方



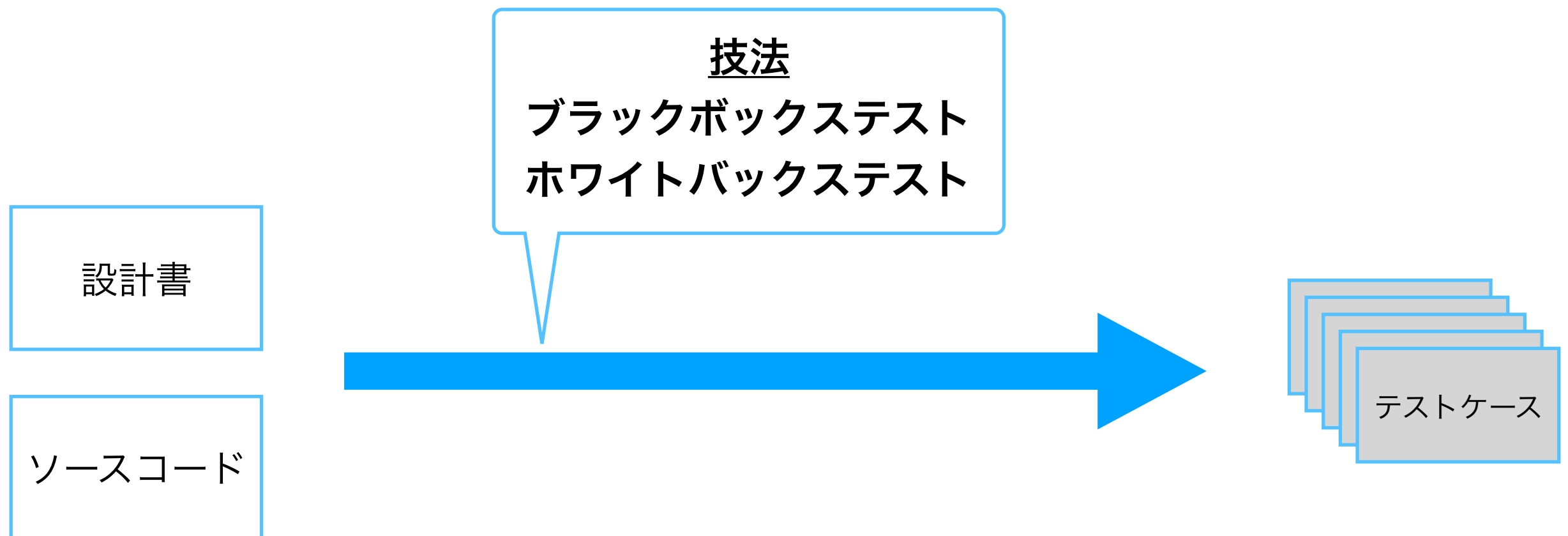
# 単体テスト

- まずテスト実行前状態を作り, その上で入力をテスト対象に与えて実行させる
- その結果, テスト対象はテスト実行後状態を作り, 出力を返す
- 最後に, これらの出力とテスト実行後状態があらかじめ設計書から計算した期待する出力と期待するテスト実行後状態と一致しているかを比較・検証する

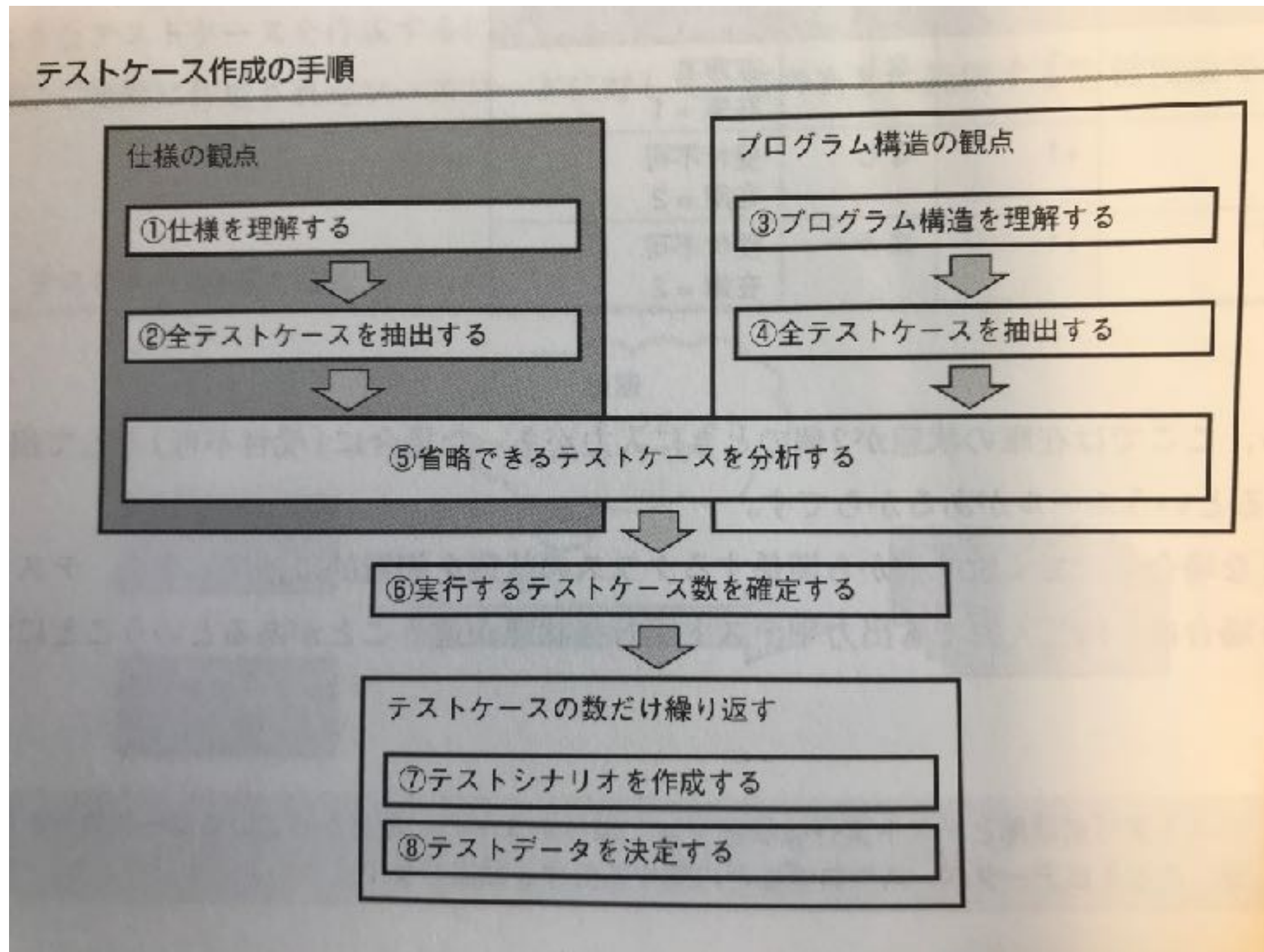


# テストケース

- ・ テストケース作成と技法



# テストケースの作成手順



# テストケースを作成 するための技法



# ブラックボックステスト

- ある入力に対してどのような出力があったかという点にのみ注目してテストケースを作成する

- \* テストケース網羅手法

- ▶ 同値分割法
- ▶ 境界値分析法

# 同値分割法

- 仕様からデータを意味のあるグループに分類し、各グループから代表値を選ぶ
  - \* 同じ意味を持つデータばかりに偏ったテストケースになることが避けられる
    - ▶ その結果効率よくテストが実施できる
- 例：int型を入力として成人であるかをチェックするようなメソッド
  - \* 引数の値が0以上20未満を「未成年」20以上を「成年」と判断する
  - \* 代表値を効率よく選ぶにはどうすれば良いか？

# 代表値の選び方

- ・ 「代表値を6つ選べ」という状況である場合

- ・ 良い例

- \* 年齢として無効： -10, -5

- \* 未成年： 5, 10

- \* 成年： 25, 30

- ・ 悪い例

- \* 年齢として無効： なし

無効になるデータが全くないのはまずい

- \* 未成年： 1, 2, 3, 4, 5

- \* 成年： 25

# 境界値分析法

- 同値分割を行なった後に, 同値クラス間の境界値をテストデータとして選ぶ手法

\* 境界値はソースコードの細かいミスを発見しやすい

正しい条件文

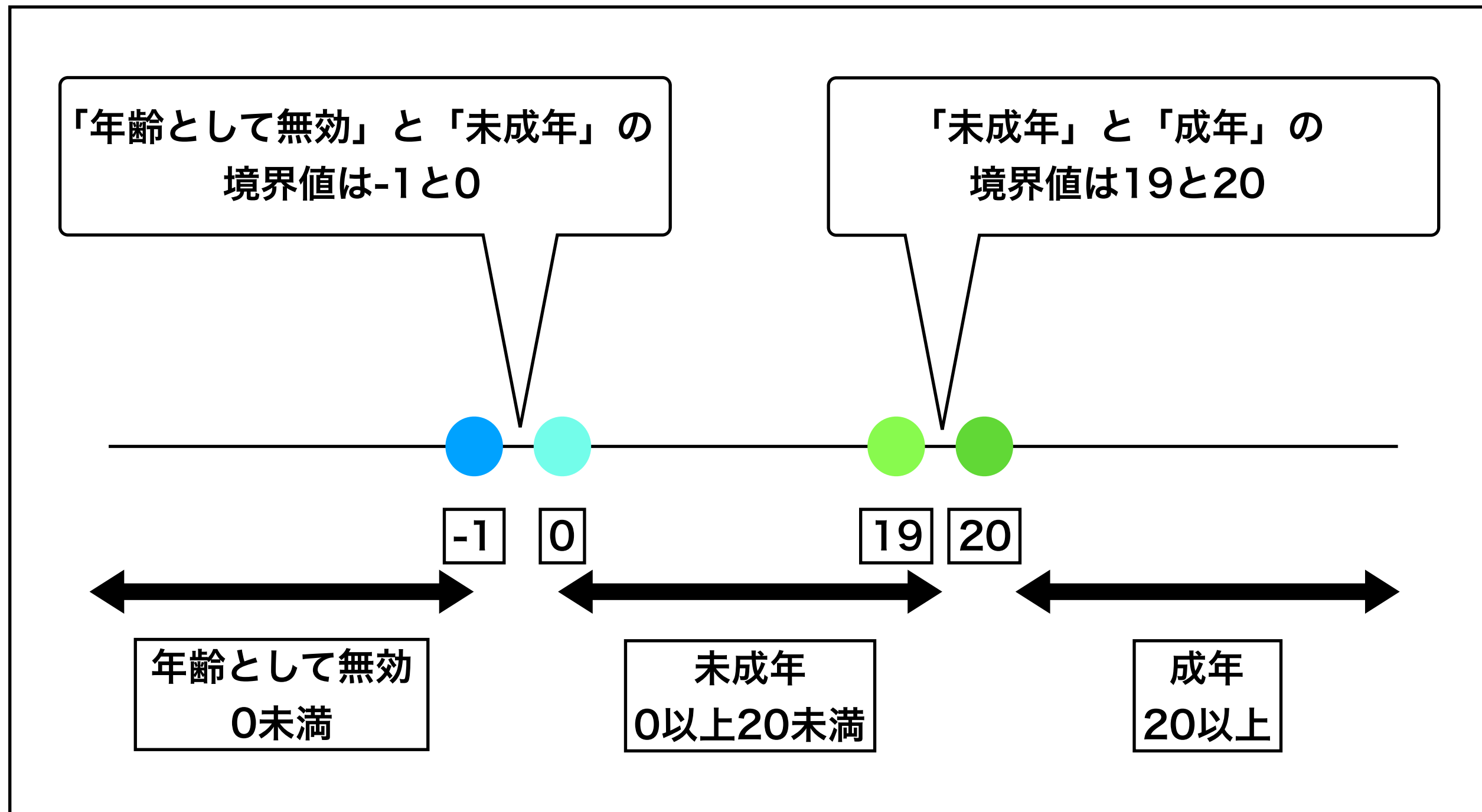
```
if (x >= 20) {  
    ///  
}
```

誤った条件文

```
if (x > 20) {  
    ///  
}
```

xに25や50など, 20から離れた値を入れてもこのバグは検出できない

# 境界値の選び方



# ホワイトボックステスト

- ブラックボックスだけでは確認できない「仕様として決めていないのに勝手にプログラミングしてしまっている処理ロジック」が確認できる
  - \* 制御構造に注目して、その処理ロジックを確認する
- 単体テストでは、特にソースコードに対するカバレッジとして、「コードカバレッジ」を利用する
- もっともよく使われるのは「制御パステスト」

# コードカバレッジ

|          |                                                                                           |
|----------|-------------------------------------------------------------------------------------------|
| 命令網羅C0   | テストにより実行された命令文の割合。ソースコード内のすべての命令を1回以上実行すると命令網羅100%となる。                                    |
| 分岐網羅C1   | テストにより実行された分岐の割合。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回以上実行すると分岐網羅100%となる。                      |
| 条件網羅C2   | テストにより実行された条件の割合。ソースコード内のすべての条件に対して、真と偽となるパターンを1回以上実行すると条件網羅100%となる。                      |
| 複合条件網羅C2 | テストにより実行された条件の組み合わせの割合。ソースコード内の各分岐におけるすべての条件に対して、真と偽のすべての組み合わせパターンを1回以上実行すると複合条件網羅100%となる |

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

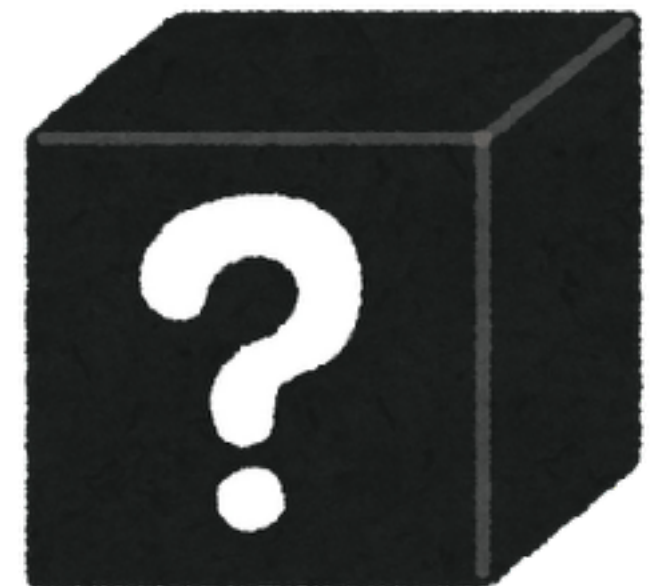
## 送料一覧

| 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 |

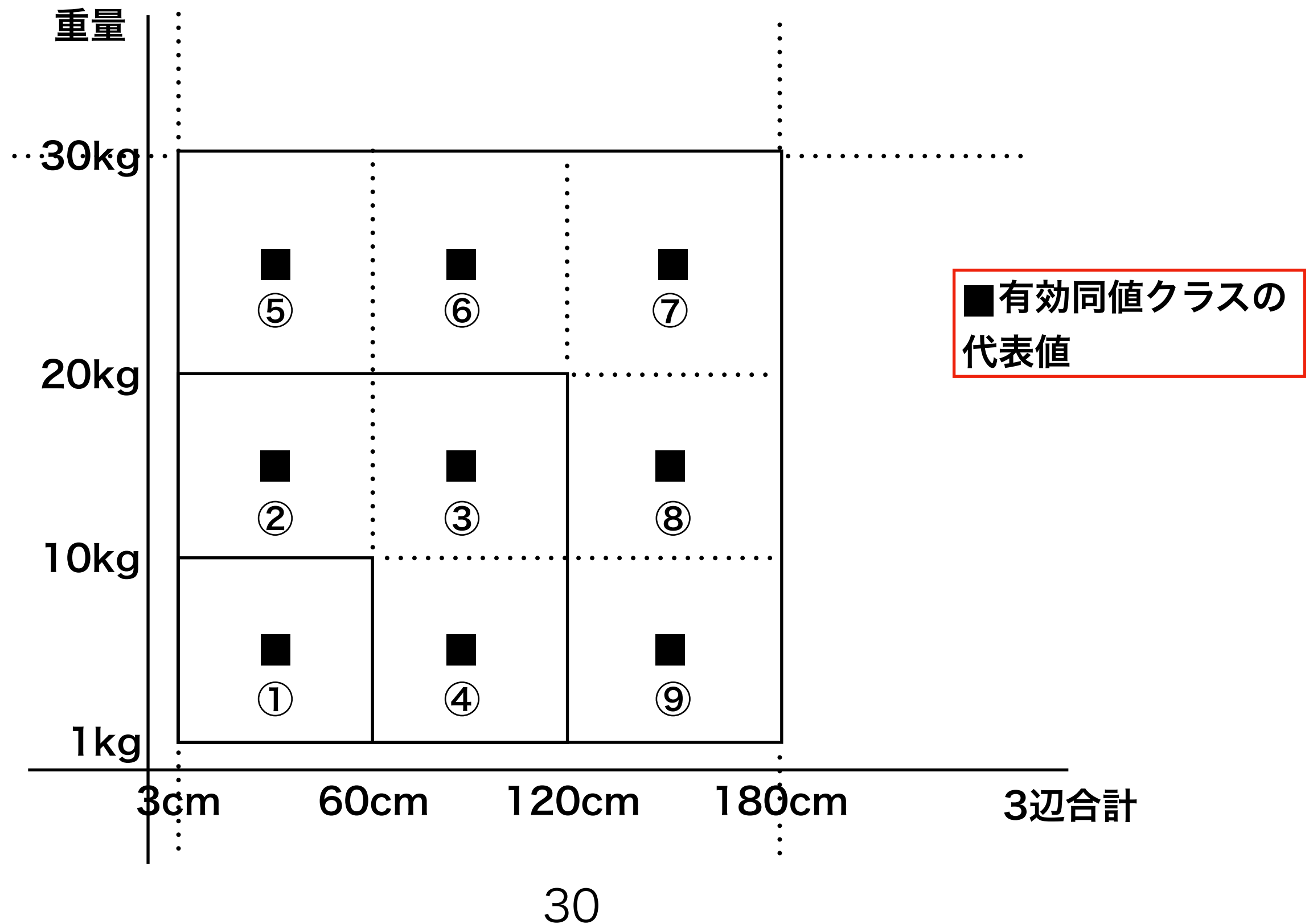
# ブラックボックステストの 観点によるテストケースの作成



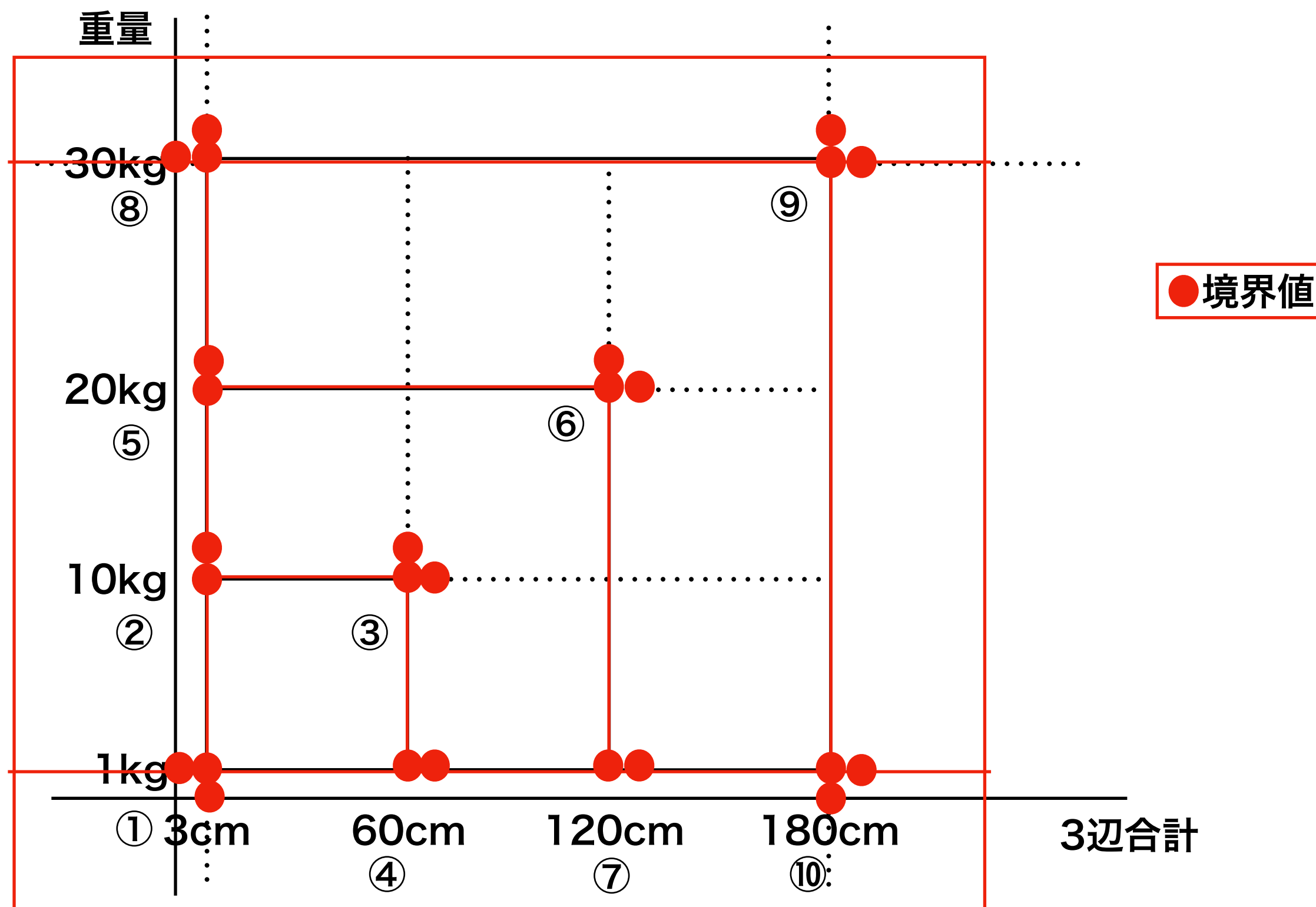
# 有効値の割り出し

- 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)

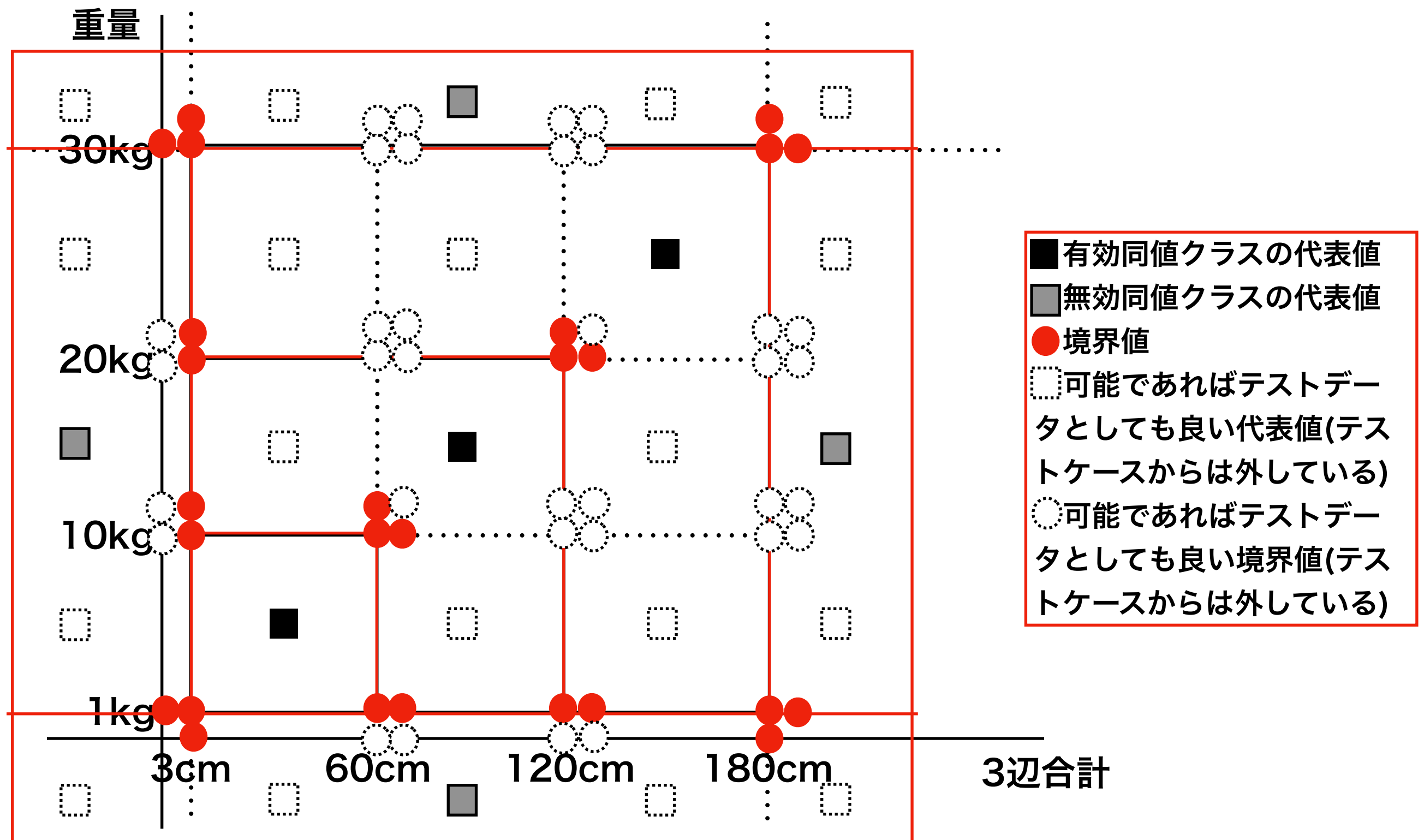


# 重量と3辺合計による境界値の分類



| パターン | 入力（テストデータ）                     | 出力<br>（期待値） | パターン | 入力（テストデータ）                  | 出力<br>（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;
```

```
    }
```

```
    return price; 命令⑤
```

```
/**
```

```
 * 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③
```

```
        return SIZE_S; 命令④
```

```
    } else if (weight <= 20) { 分岐④
```

```
        return SIZE_M; 命令⑤
```

```
    } else {
```

```
        return SIZE_L; 命令⑥
```

```
    }
```

```
}
```

```
/**
```

```
 * 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⑦
```

```
        return SIZE_S; 命令⑦
```

```
    } else if (total <= 120) { 分岐⑧
```

```
        return SIZE_M; 命令⑧
```

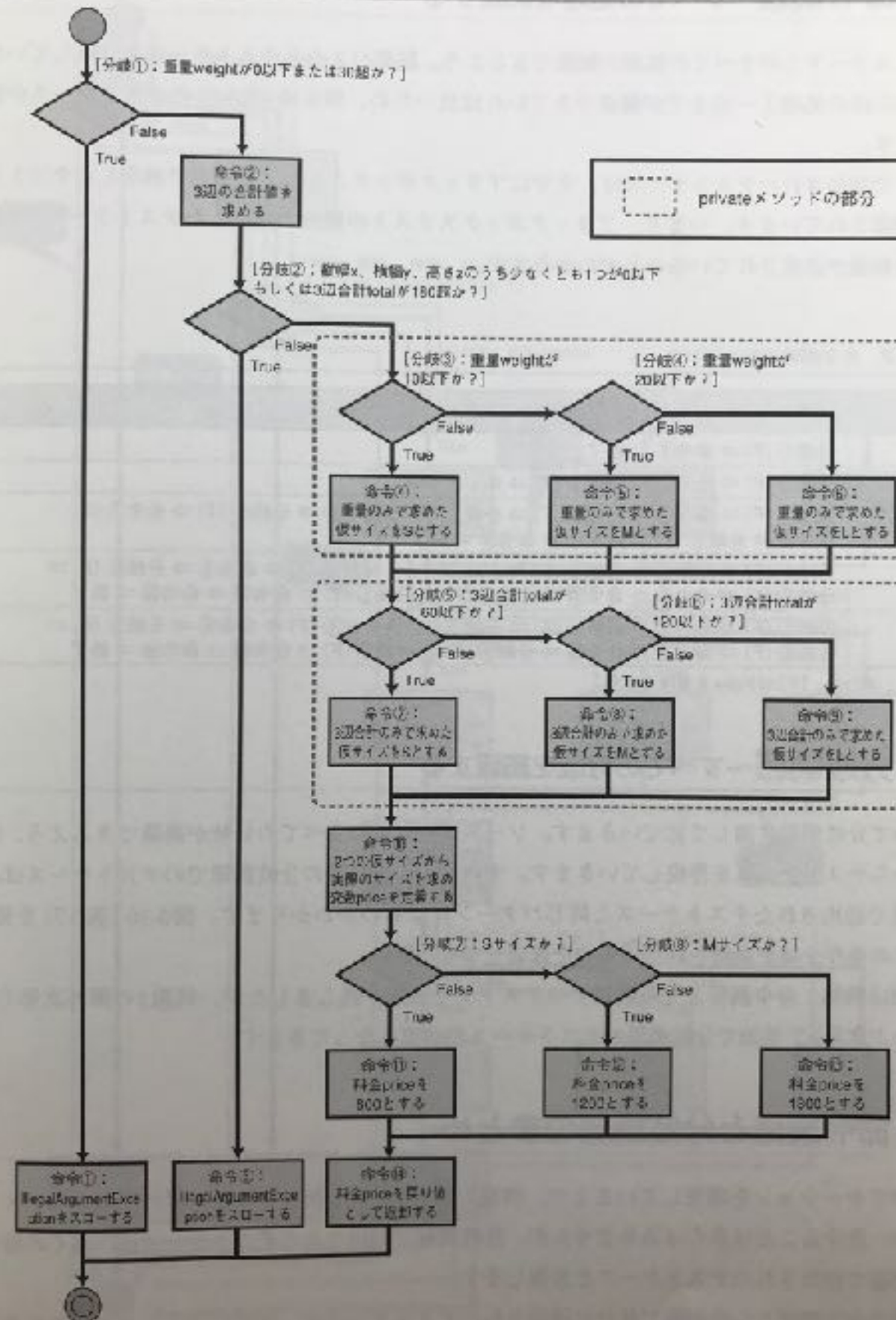
```
    } else {
```

```
        return SIZE_L; 命令⑨
```

```
    }
```

```
}
```

▼図5-15 制御パス



# 命令網羅

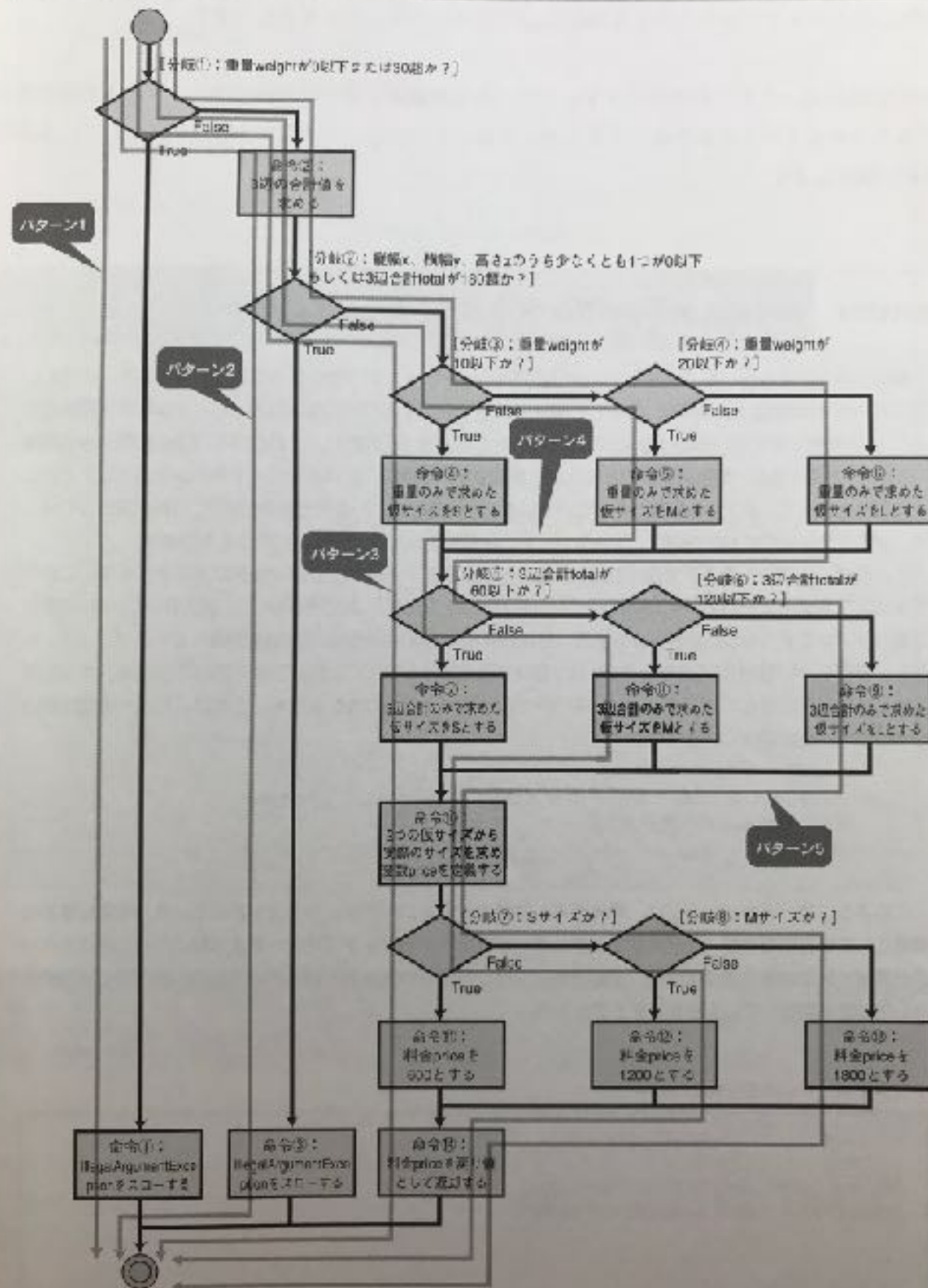
全ての処理を網羅する

▼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

# テストの記述



# しばらく理論を お話しします

The image shows handwritten mathematical derivations on a dark background. At the top, it states  $\sqrt{\frac{P(1-P)}{n}} \approx \sqrt{\frac{w(1-w)}{n}}$ . Below this, there is a complex fraction involving  $\frac{n}{2(\frac{1}{2} + h)}$  and a large expression in parentheses. The main part of the derivation shows the probability  $P\left(\left|w - \frac{1}{2}\right| < t_p \sqrt{\frac{P(1-P)}{n}}\right) = P$  being approximated as  $\frac{1}{2} \left(1 + w t_p \sqrt{4 t_p^2 \frac{w(1-w)}{n}}\right) = w$ . The final line shows  $w - p < \frac{t_p^2}{n} P(1-P)$ .

# テストイングフレームワークとは

- テストのフレームワークのこと
  - \* フレームワーク: 共通部分を取り出して複数のプロジェクトで再利用可能にしているもの
    - ▶ この共通基盤部分を提供するフレームワークを使用することが多い. 例: Struts, Springなど

# フレームワークを利用するメリット

フレームワークで提供されているクラスライブラリの  
再利用による, 開発効率の向上

品質が保証されたフレームワークの使用による,  
アプリケーション全体の品質の向上

フレームワークが提供する共通基盤部分の高度な知識を必要と  
しないことによる, 開発容易性の向上

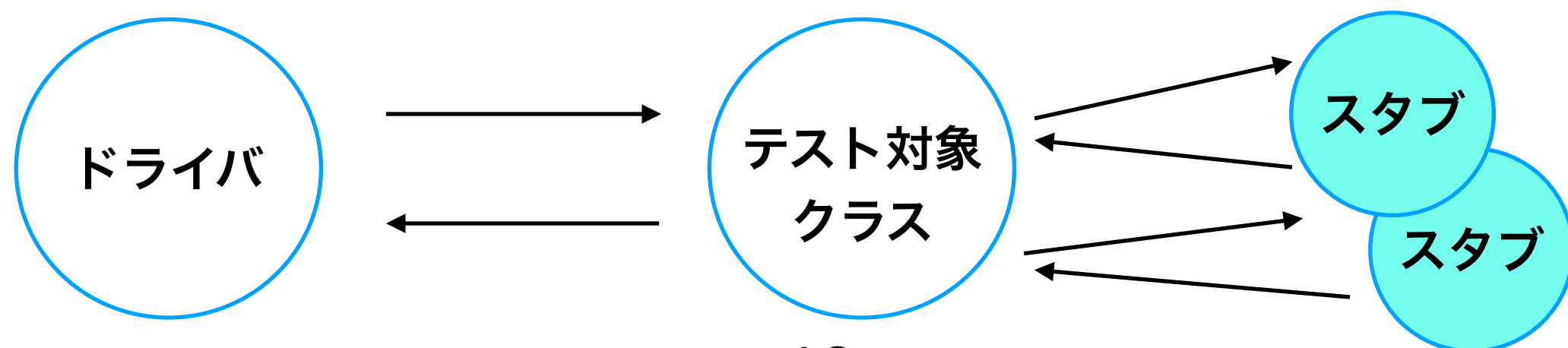
独自に開発する部分のプログラミングパターン定型化による,  
統一性, メンテナンス性などの向上

# xUnit

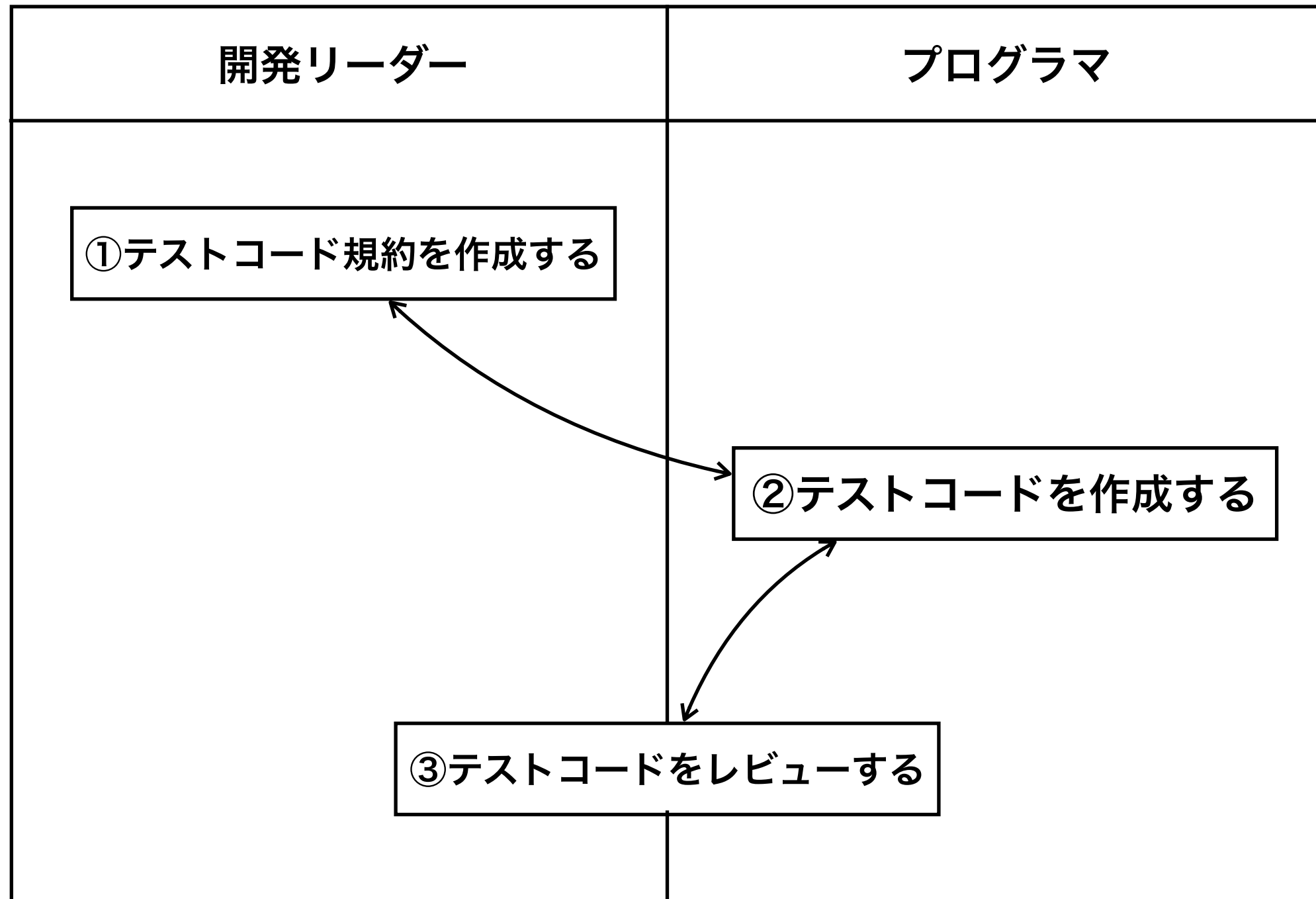
- 単体テスト用のテストフレームワークは一般的に「xUnit」と呼ばれる
  - \* ほとんどのプログラミング言語に対してフレームワークが用意されている
    - ▶ 今回はJUnitを用いて単体テストを進める
- xUnitは以下のURLから一覧をみることができます
  - \* <https://ja.wikipedia.org/wiki/ユニットテスト・フレームワーク一覧>

# ドライバとスタブ

- テスト対象のソースコードを呼び出す側のソースコードがまだ開発されていない場合がある
  - \* このような場合でも単体テストが進められるように、まだ開発されていないソースコードの代わりにするソースコードを用意する
- **ドライバ**: テスト対象のソースコードを呼び出す代替ソースコード
- **スタブ**: テストコードに呼び出される代替ソースコード



# テスト実装の作業フロー



# テストコード規約を作成する

**担当：開発リーダー**

- 必ず作成するように心がける
  - \* 場当たり的にテストコードを書いているとテストコードの保守性が低下する
    - ▶ 仕様変更時などにテストコードを修正する際に手間がかかる
- JUnitはテスト結果の報告機能を備えているため、どのようなテストを実施したのかの証拠として、テストコードをユーザに提出してくれる
  - \* テストコードでもソースコード作成時と同様にしっかりと規約を作成し、品質を確保するようにする

# テストコードを作成する

---

担当：プログラマ

- テストコードの作成は似た内容をコピーする傾向が高い
  - \* テストコードをコピー先のテスト用に修正する際にバグを作り込んでしまうことがあるので十分注意する
- テストコードを静的解析ツールでレビューする場合は、コーディング規約とテストコード規約の差をチェックルールの設定に反映するのを忘れないようにする

# テストコードをレビューする

---

**担当：開発リーダー・プログラマ**

- バグのあるテストコードを利用してソースコードをテストしても意味がないため、必ずレビューを行うようにする
- 全てのテストコードをレビューできないときは、以下の方針でテストコードをピックアップしてレビューする
  - \* テストコードを作成したプログラマごとに、最低1つのテストコードは必ずレビューする
  - \* アプリケーションが提供する機能別に、その機能の中心となるソースコードに対するテストコードは必ずレビューする
  - \* プロジェクトとして初めて提供する機能を実現するソースコードに対するテストコードは必ずレビューする
  - \* プロジェクトメンバーが難易度の高いと指定したソースコードに対するテストコードは必ずレビューする

# 参考：テストファースト

- JUnitを使用した開発手法として「テストファースト」という手法がある
  - \* テスト対象のソースコードをプログラミングする前にテストコードを書き, そのテストコードが成功するようにソースコードをプログラミングする
    - ▶ テストファーストを取り入れた開発手法を「テスト駆動開発(TDD: Test Driven Development)」と呼ぶ.
- ただしテストファーストでのテストとソースコードが記述できたら単体テストが終了したことはない
  - \* この手法はあくまで仕様通りに動くかどうかを確認するか確認するためのもの
  - \* 単体テストは別途作成する必要がある

ではお題のテストを記述  
していきましょう！



# JUnitの基礎知識

- 基礎知識についてはスライド「JUnit入門【テストコードを書く】」を参考にしてください
- \* 今回は前回の知識を実践的に使用することを考えていきます

## 6-3

# 例題による実習 ～プロセス／技法の実践方法

それでは、実際にテストケースを実行するテストコードを作成してみましょう。まずは、次の例題を見てください。

### 例題3

第4章の例題1で作成したShippingクラス(Shipping.java)に対し、第5章の例題2で作成したテストケースを実行するテストコード「ShippingTest.java」を作成しなさい。

なお、テストコードは以下のテストコード規約とテストコードレビュー観点に従って作成およびレビューすること。

#### テストコード規約

##### ●命名規約

- パッケージ名……テスト対象クラスと同一パッケージにする
- テストクラス名……「テスト対象クラス名 + Test」とする
- テストメソッド名……「test + テスト対象メソッド名 + 連符」とする

##### ●その他

- テストメソッドの宣言部分にはthrows Exceptionを記述すること
- 1つのテストメソッドには1つのテストケースだけをプログラミングすること

#### テストコードレビュー観点

- テストコード規約で決めた事項が守られていること
- テストの事前状態や引数、検証内容がテストケースの内容と一致していること
- テストコードの処理ロジックに誤りがないこと
- テストメソッドの実行順序によって結果が変わるような記述になっていないこと
- 過度に複雑な記述をしていないこと
- テストの内容を表わすコメントが適度に記述されていること



各例題の解答例は、巻末付録Cに掲載しています。また、Shipping.javaとテストケースデータは、<http://www.seehcp.com/book/download/>からダウンロードできます。また例題1(P.59)、例題2(P.69)を解いていない方は、これらを利用して本例題に取り組んでみてください。

# Eclipseによって自動生成されたテストコード

```
public ip.co.shoeisha.example;↓
↓
import static org.junit.Assert.*;↓
import org.junit.Test;↓
↓
public class ShippingTest {↓
^   ↓
^   @Test↓
^   public void testCalculate() {↓
^   ^   fail("まだ実装されていません");↓
^   }↓
}↓
```

※おかしなエラーが出ていますが、無視してください

# テストコードの修正

- 修正が必要なポイント
  - \* 「org.junit.Assert.\*」のstaticインポート文を削除する
  - \* テストクラスの宣言部分でTestCaseクラスを継承する
  - \* テストメソッドの宣言部分でthrows Exceptionを記述する

# 修正後

```
public jp.co.shoeisha.example;↓
↓
import junit.framework.TestCase;↓
import org.junit.Test;↓
↓
public class ShippingTest extends TestCase{↓
    ↓
    @Test↓
    public void testCalculate() throws Exception{↓
        fail("まだ実装されていません");↓
    }↓
}
```

例外をスローする

# テストメソッドの中身のプログラミング

- お題でプログラミングするテストメソッドには2つのパターンがある
  - \* 正常に値を返すパターン
  - \* 例外が発生するパターン

# 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 |

| パターン | 入力（テストデータ）                     | 出力<br>（期待値） | パターン | 入力（テストデータ）                  | 出力<br>（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        |

# 正常に値を返すパターン

|   |                            |     |
|---|----------------------------|-----|
| 1 | weight=5, x=10, y=10, z=10 | 600 |
|---|----------------------------|-----|

```
/**↓
 * 重さが「Sサイズ任意値」で3辺合計が「Sサイズ任意値」のとき、↓
 * サイズ区分が「Sサイズ」の料金出力されるか確認する。↓
 *↓
 * @throws Exception↓
 */↓
@Test↓
public void testCalculate01() throws Exception {↓
    int weight = 5;↓
    int x = 10;↓
    int y = 10;↓
    int z = 10;↓
    Shipping sp = new Shipping();↓
    int result = sp.calculate(weight, x, y, z);↓
    assertEquals(600, result);↓
}↓
```

# 例外が発生するパターン

|   |                                  |    |
|---|----------------------------------|----|
| 4 | weight=-100, x=30, y=30,<br>z=30 | 例外 |
|---|----------------------------------|----|

```
/**  
 * 重さが「Sサイズ最小値未満」で3辺合計が「Mサイズ任意値」のとき、  
 * IllegalArgumentExceptionがthrowされるか確認する。  
 *  
 * @throws Exception  
 */  
@Test  
public void testCalculate04() throws Exception {  
    int weight = -100;  
    int x = 30;  
    int y = 30;  
    int z = 30;  
    Shipping sp = new Shipping();  
    try {  
        sp.calculate(weight, x, y, z);  
        fail();  
    } catch (IllegalArgumentException e) {  
        // 例外が発生すれば成功  
    }  
}
```

# コードについて

- 模範解答に当たるコードは以下のURLからダウンロードすることも可能です. 答え合わせに使ってください
  - \* <http://www.seshop.com/book/download/>
    - ▶ ISBNを聞かれますので, 「978-4-7981-1463-7」と入力してください
- 解答は4 2通りのテストケースについて記述されているので非常に長いです

# レビューの練習

- 以下のテストコードには規約に従っていない記述や誤った記述がいくつかあります。それを見つけて修正してみましょう！

```
1 package jp.co.shoeisha.example;
2
3 import junit.framework.TestCase;
4
5 import org.junit.Test;
6
7 public class ShippingTest extends TestCase {
8
9     @Test
10    public void testCalculate01() {
11        int weight = 5;
12        int x = 10;
13        int y = 10;
14        int z = 10;
15        Shipping sp = new Shipping();
16        int result = sp.calculate(weight, x, y, z);
17        assertEquals(result, 600);
18    }
19
20    @Test
21    public void testCalculate04() throws Exception {
22        int weight = -100;
23        int x = 30;
24        int y = 30;
25        int z = 30;
26        Shipping sp = new Shipping();
27        try {
28            sp.calculate(weight, x, y, z);
29        } catch (IllegalArgumentException e) {
30            fail();
31        }
32    }
33
34 }
```

# 解答

- 1つ目のテストメソッドにthrows Exceptionがない
- 1つ目のテストメソッドのassertEquals()メソッドの引数が逆
- 2つ目のテストメソッドでのfail()メソッドの位置が違う(正しくはtryブロックの中)