3-1　ホワイトボックステストとは

ホワイトボックステストは、プログラムの内部構造に着目してテストケースを設計するテストです。なお、テストケースの設計には、システム／ソフトウェアやプログラミング言語に関する詳しい知識が必要です。

ホワイトボックステストには、次の二つの方法があります。

テスト方法	概　　要
制御フローテスト	開発プロジェクトによって定義された**網羅基準**に従い、プログラムの制御パスが正しいかを検証する
データフローテスト	プログラムの実行において変数の取扱い（定義→使用→消滅）が正しいかを検証する

※本書では、制御フローテストのみ解説します。

制御フローテストは、次の流れでテストケースを設計します。

①プログラムから制御フロー図を作成する
②制御フロー図から、網羅基準を満たす制御パスを読み取る
③テスト項目（制御パス）に対応するテストケース（操作手順、テストデータ、期待結果）を作成する

また、**網羅基準**とは、すべての制御パスから実際にテストを実施する制御パスを選定する際に適用する基準のことです。**カバレッジ基準**とも言います。

網羅基準	概　　要
命令網羅（C0網羅）	すべての命令を１回以上実行する（詳細は後述）
分岐網羅（C1網羅）	分岐の単一条件または複合条件について、真と偽の判定をそれぞれ１回以上実行する（詳細は後述）
条件網羅（C2網羅）	分岐の複合条件を構成する各要素について、真と偽の判定をそれぞれ１回以上実行する（詳細は後述）
全パス網羅	すべての制御パスを対象とするため、非現実的である

3-2　制御フロー図の作成

制御フロー図は、**ノード**と**有向エッジ**の記号のみ用いて作成できるため、プログラムの制御構造を簡単に整理できます。

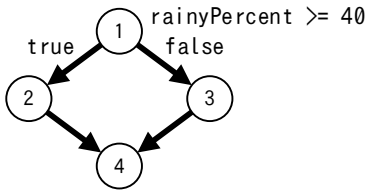
記　号		説　明
ノード	ノード番号	・順次構造の命令群を示す ・分岐構造や繰返し構造の条件判定を示す
有向エッジ	↓	・プログラム実行の流れを示す

それでは、プログラムと制御フロー図の対応関係を見てみましょう。

〔プログラム：WeatherForecast.java〕

ノード	ソースコード
	`package example3_1;`
	`public class WeatherForecast {`
	`  public static boolean check(int rainyPercent) {`
	`  boolean bringUmbrella;`
①	`  if(rainyPercent >= 40) {  //降水確率40%以上の場合`
②	`  bringUmbrella = true;  //傘を持って行く`
	`  } else {`
③	`  bringUmbrella = false;  //傘を持って行かない`
	`  }`
④	`  return bringUmbrella;`
	`  }`
	`}`

〔制御フロー図：WeatherForecastクラスのcheckメソッド〕



制御フロー図を作成するときのポイント

- プログラムに連番のノード番号を付与する
  同時初期化を伴わない変数の定義、ブロックの閉じ中括弧などには、ノード番号を付与しない
- 分岐構造の条件判定のノードに条件を併記する
- 条件判定のノードから伸ばす有向エッジにtrueまたはfalseを併記する

3-3　命令網羅（C0網羅）

命令網羅（C0網羅、ステートメントカバレッジ）は、すべての命令を１回以上実行する網羅基準です。始めに、プログラムから制御フロー図を作成します。

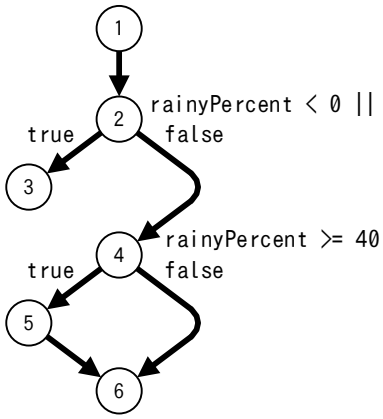
〔プログラム：WeatherForecast.java〕

ノード	ソースコード

	package example3_2;
	public class WeatherForecast {
	　public static boolean check(int rainyPercent) throws RainyException {
	① 　　boolean bringUmbrella = false;　//傘を持って行かない
	② 　　if(rainyPercent < 0 || rainyPercent > 100) {
	③ 　　　throw new RainyException(”降水確率が無効な値です”);
	　　}
	④ 　　if(rainyPercent >= 40) {　　//降水確率40%以上の場合
	⑤ 　　　bringUmbrella = true;　　//傘を持って行く
	　　}
	⑥ 　　return bringUmbrella;
	　}
	}

	package example3_2;
	public class RainyException extends Exception {
	　public RainyException(String reason) {
	　　super(reason);
	　}
	}

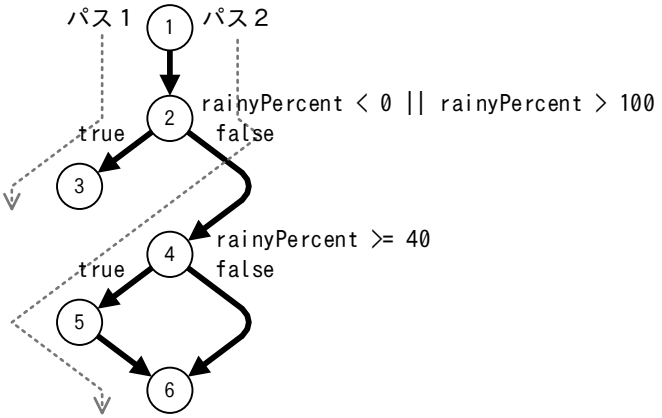
〔制御フロー図：WeatherForecastクラスのcheckメソッド〕



制御フロー図を作成するときのポイント
・例外スロー命令のノードから伸ばす有向エッジは記載しない

次に、制御フロー図から、命令網羅の制御パスを読み取ります。

〔制御パス：命令網羅〕



パス１　①→②→③

パス２　①→②→④→⑤→⑥

最後に、テスト項目（制御パス）に対応するテストケース（テストケースNo、テストデータ、期待結果）を作成します。なお、各テスト項目のテストデータrainyPercentは、任意の値を設定しています。

〔テストケース〕

テストケースNo	テスト項目	テストデータ	期待結果
1	①→②→③	rainyPercent：120	RainyException
2	①→②→④→⑤→⑥	rainyPercent：　60	戻り値：true

計算上は網羅率100％ですが、「④→⑥」の流れを確認していないので、ノード④の条件記述ミスは発見できません。