ツールの使用手順

目次

➢ 全体のワークフロー

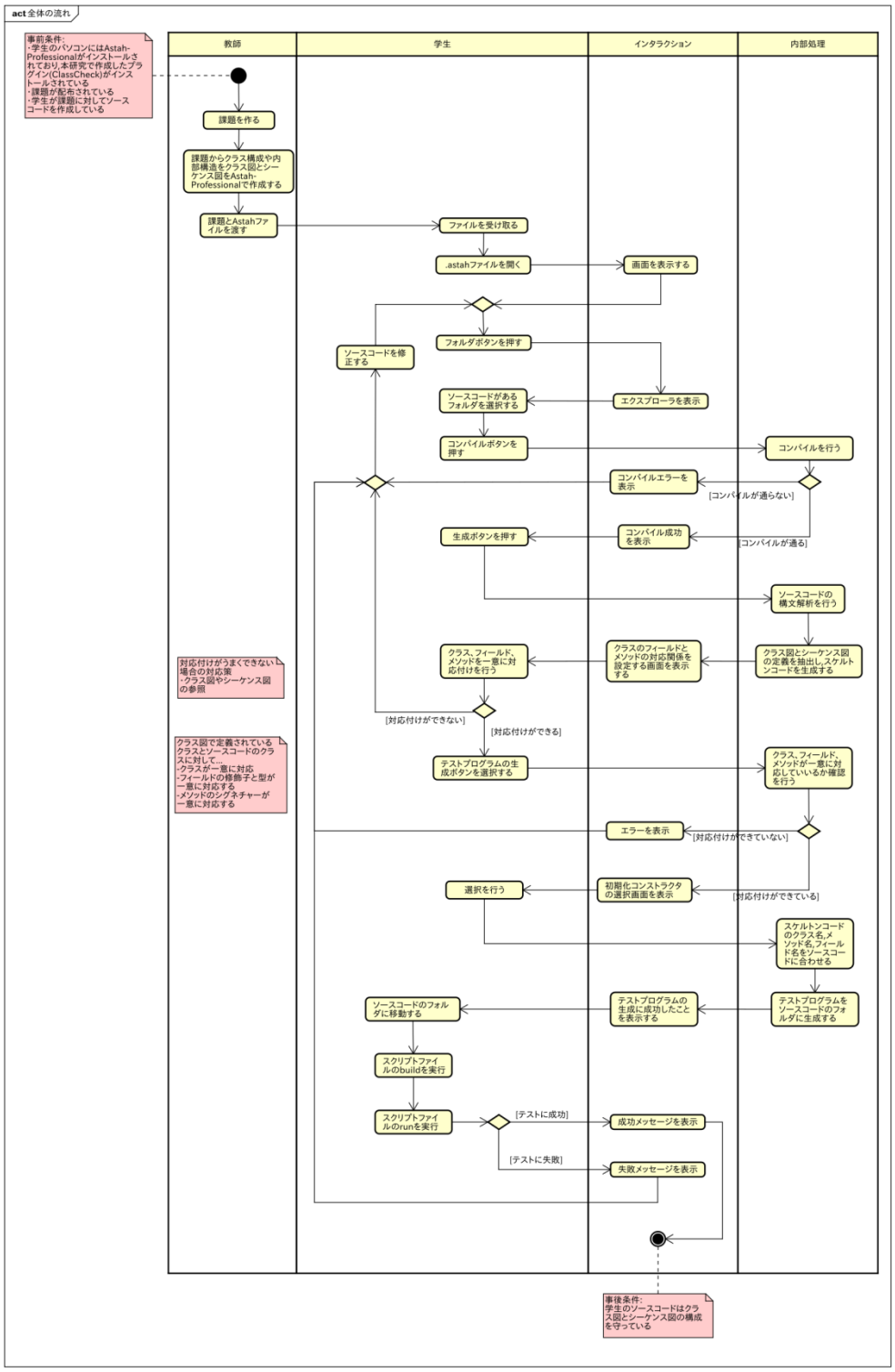

図 1：全体のワークフロー

このツールの使用方法については全体の流れを表した図 1 と実際のツールの画面を対応させながら説明を行う.

➢ 事前条件

事前条件:
・学生のパソコンにはAstah-Professionalがインストールされており,本研究で作成したプラグイン(ClassCheck)がインストールされている
・課題が配布されている
・学生が課題に対してソースコードを作成している

図 2:事前条件

■ 教師

- 課題に対して満たしてほしいクラス構成や内部構造を astah-professional を用いてクラス図とシーケンス図で表現を行う
- 課題を配布している

■ 学生

- astah-professional がインストールされている
- プラグイン(ClassCheck)がインストールされている
- 課題に対してソースコードを作成している

➢ ソースコードのフォルダを指定

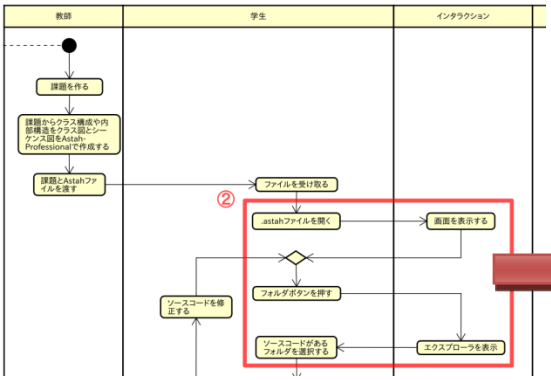

図 3:ソースコードのフォルダを指定

学生は課題に対してクラス図とシーケンス図を定義した asta ファイルを開き,ソースコードがあるフォルダを指定する

図 4:ソースコードのフォルダ選択

➢ コンパイル

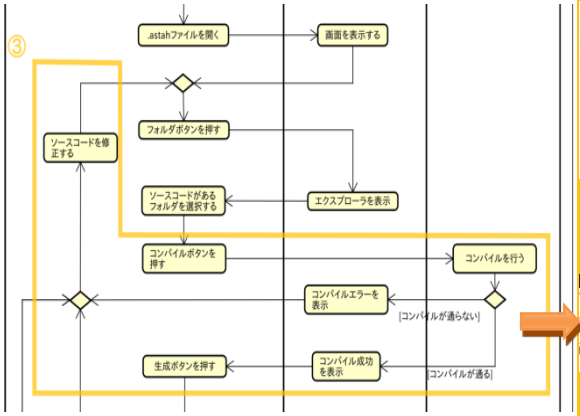

図 4:コンパイル

本ツールはソースコードが文法的に合っているものに対して、テストプログラムが生成可能なので、まずソースコードのコンパイルを行う.

コンパイル成功の場合は成功メッセージが表示されるが、失敗の場合はエラーメッセージが表示される.その際、学生はソースコードのコンパイルが成功するまで、ソースコードの修正を行う.また、外部ライブラリを参照するときは jar ファイルがあるパスを指定し、エンコードする場合は文字コードを指定する.

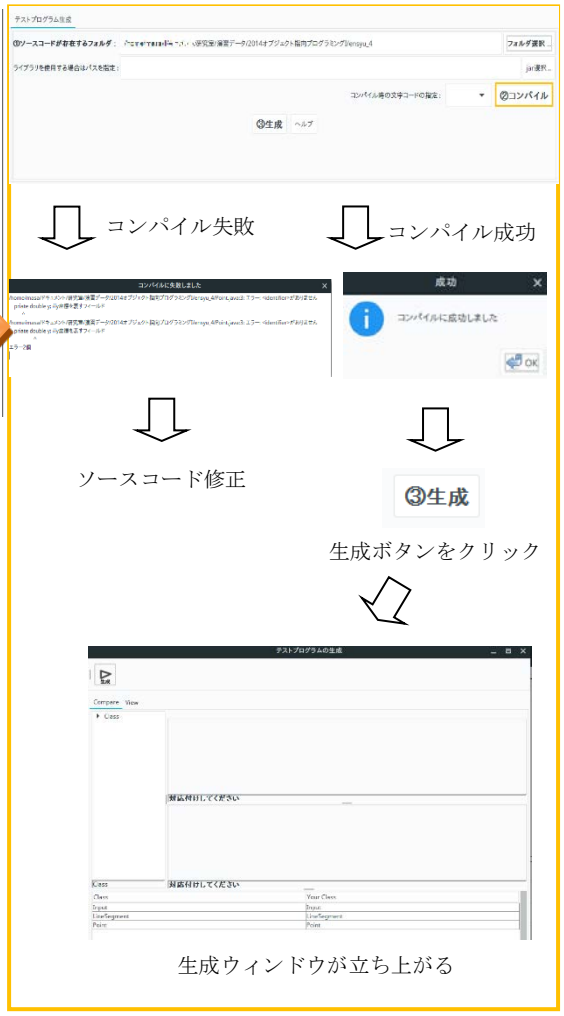

図 6:コンパイルから生成ウィンドウ

そして、生成ボタンを押してテストプログラムの生成ウィンドウの画面が立ち上げる.生成ウィンドウの各部分の役割は以下の図のようになる.

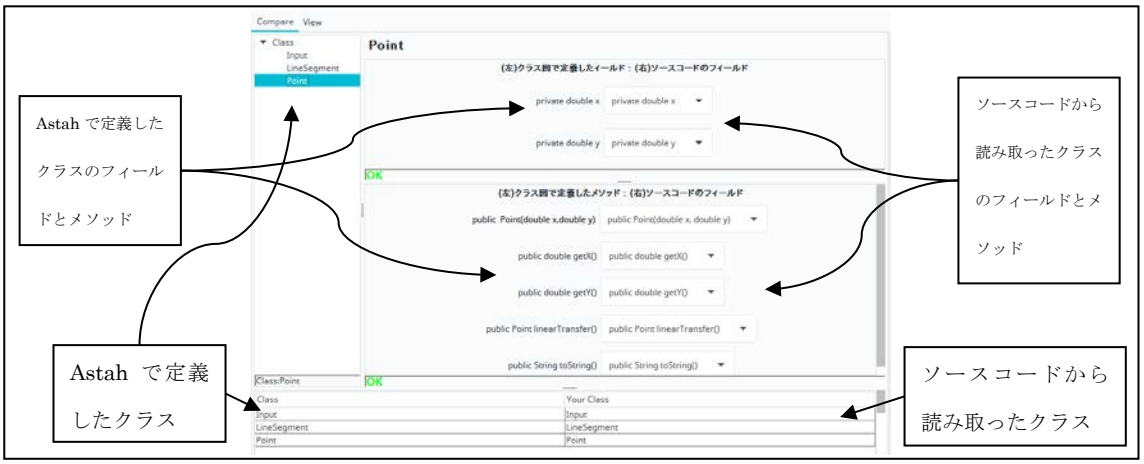

図 7: .生成ウィンドウの各部分の役割

➢ クラス図とソースコードを対応させる

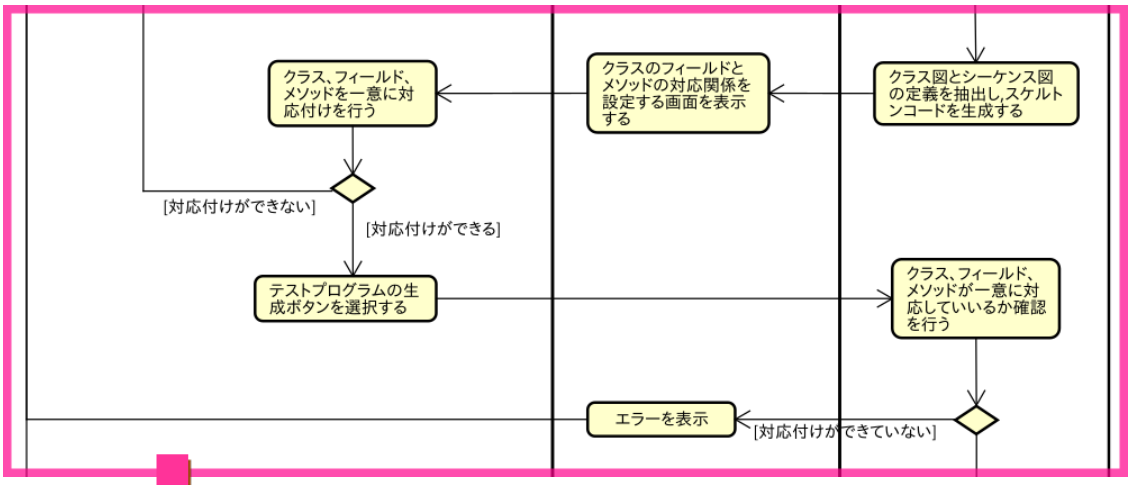

図 8: クラス図とソースコードを対応

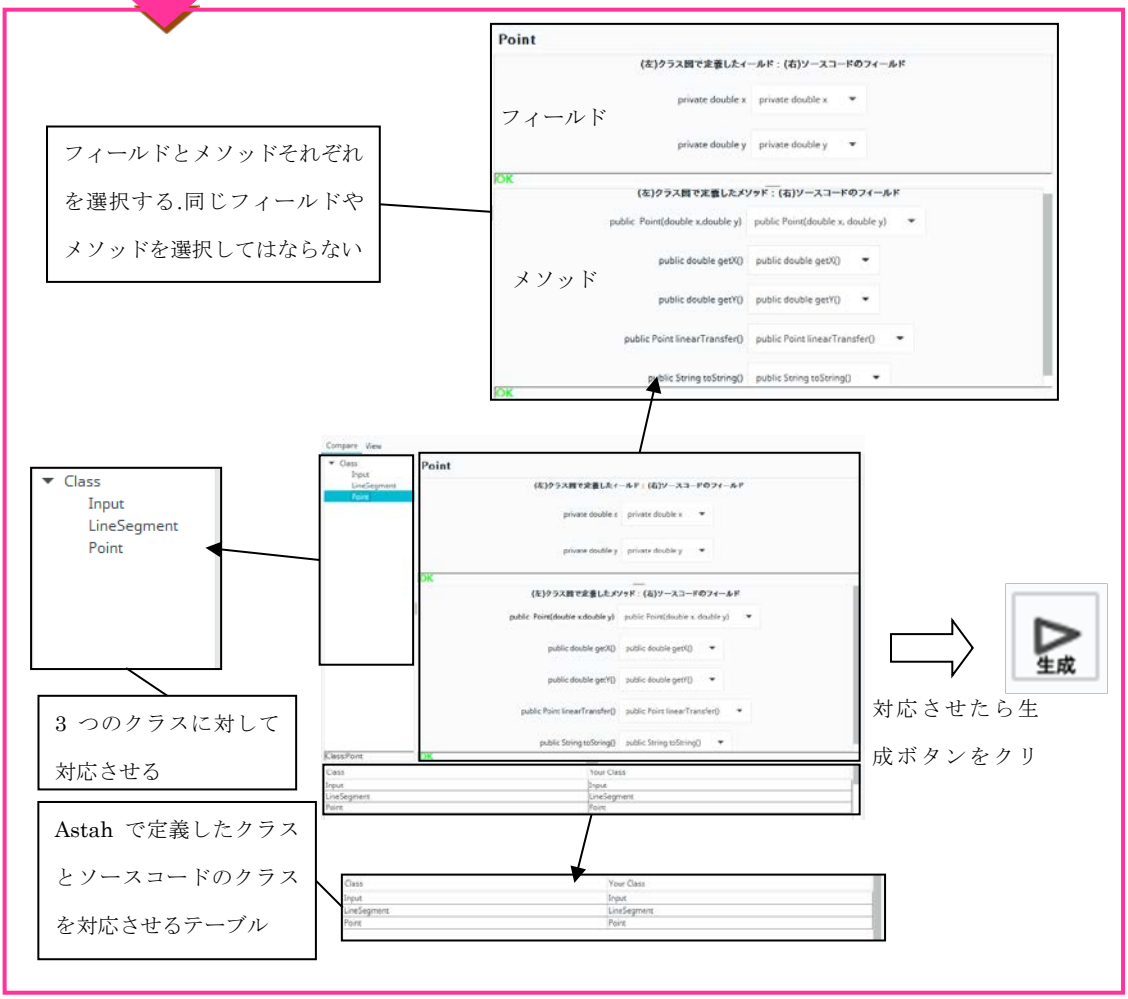

図 9:生成ウィンドウ

生成ウィンドウ(図 9)ではクラス図で定義したクラスとソースコードにあるクラスを対応させることができれば生成ボタンをクリックする.その際に、以下の条件に当てはまる場合

は生成ボタンをクリックするとエラーウィンドウが立ち上がる(図 10)

- テーブルに同じクラスが複数選択されている
- テーブルにクラスが選択されていない
- 同じフィールドやメソッドが選択されている
- フィールドやメソッドが選択されていない

また、フィールドやメソッドの選択リストとして以下の条件に当てはまる場合は選択リストのアイテム候補として選ばれない.

- フィールドの型が異なる
- フィールドの修飾子が異なる
- メソッドの返り値の型が異なる
- メソッドの引数の個数が異なる
- メソッドの引数に対してそれぞれの型が異なる

図 10:エラーウィンドウ(同じフィールドを選択した場合)

➢ テストプログラムの生成

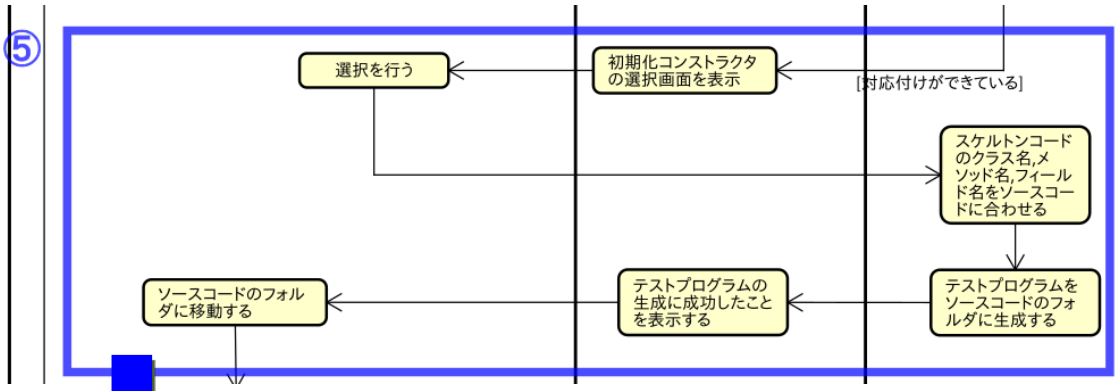

図 11:テストプログラムの生成

図 12:初期化コンストラクタの指定からテストプログラムの生成

図 9 の生成ボタンをクリックすると図 12 の初期化コンストラクタの指定を行うウィンドウが立ち上がり設定をする。テストコードの確認画面が表示され ok ボタンをクリックするとテストプログラムの生成が成功したことを告げるウィンドウが立ち上がる。

成功する test フォルダを作成し以下のようなファイル階層になる。

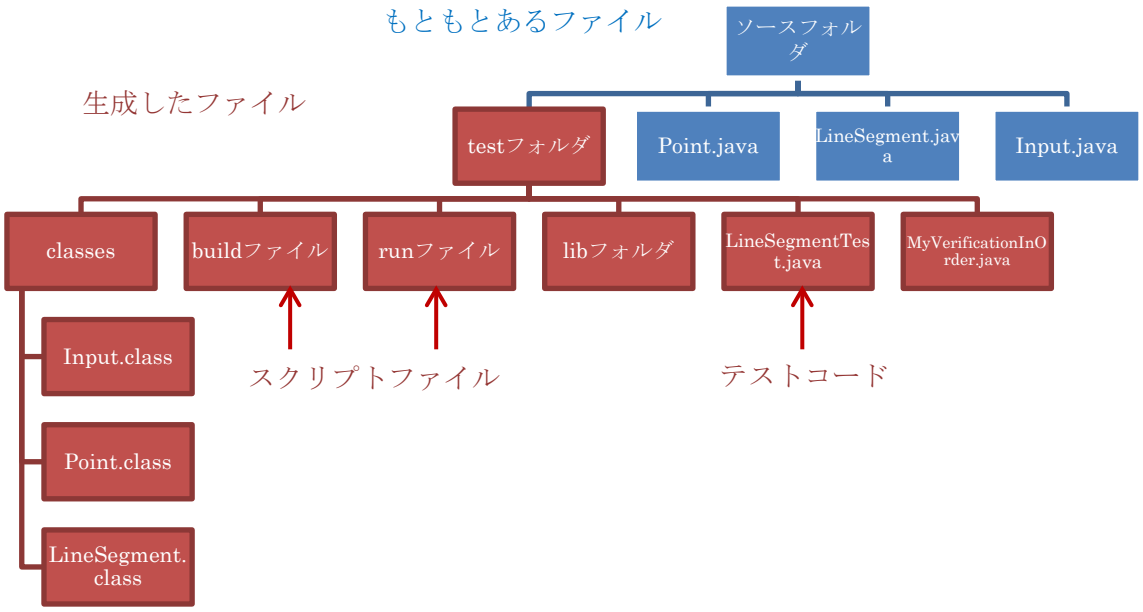

図 13:ファイル階層

➢ **テストコードのコンパイル、実行**

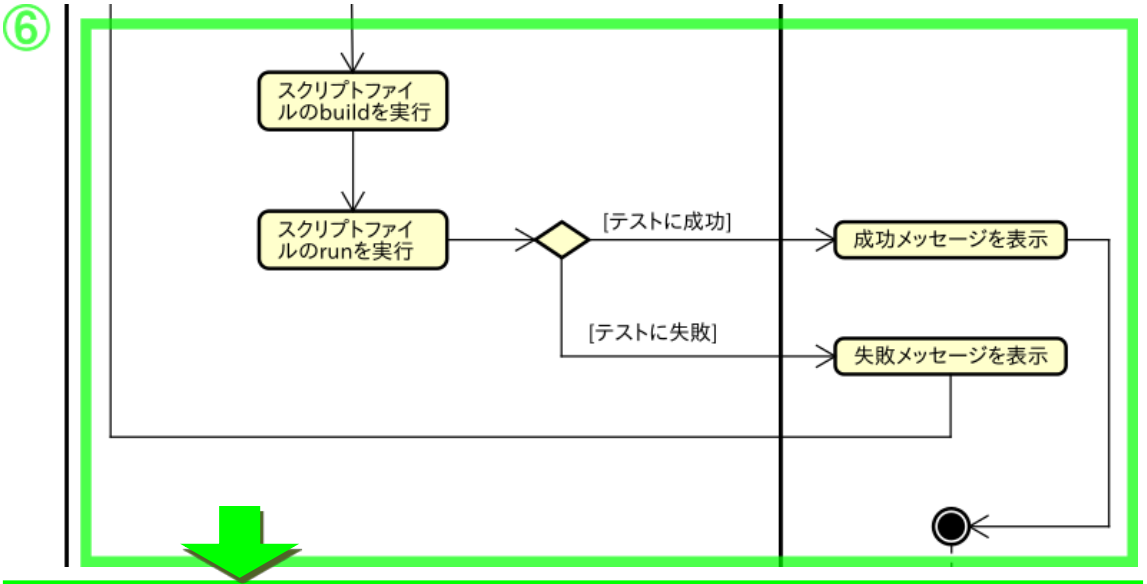

図 14:コンパイルと実行(Linux)

テストプログラムの生成を行うと図 13 のようなファイルが新たに生成されるが、この中に 2 つのスクリプトファイルが存在する。build ファイルはテストプログラムのコンパイルを行い,run ファイルはテストプログラムの実行を行うスクリプトファイルである。また、Linux 環境ではシェルスクリプトファイル、Windows 環境では bat ファイルである。テストプログラムは astah ファイルに定義されている図 15 のようなシーケンス図をもとにしている。図 15 をもとにプログラムの実行を行うと以下のような図 16、17 のような実行結果が標準出力される。

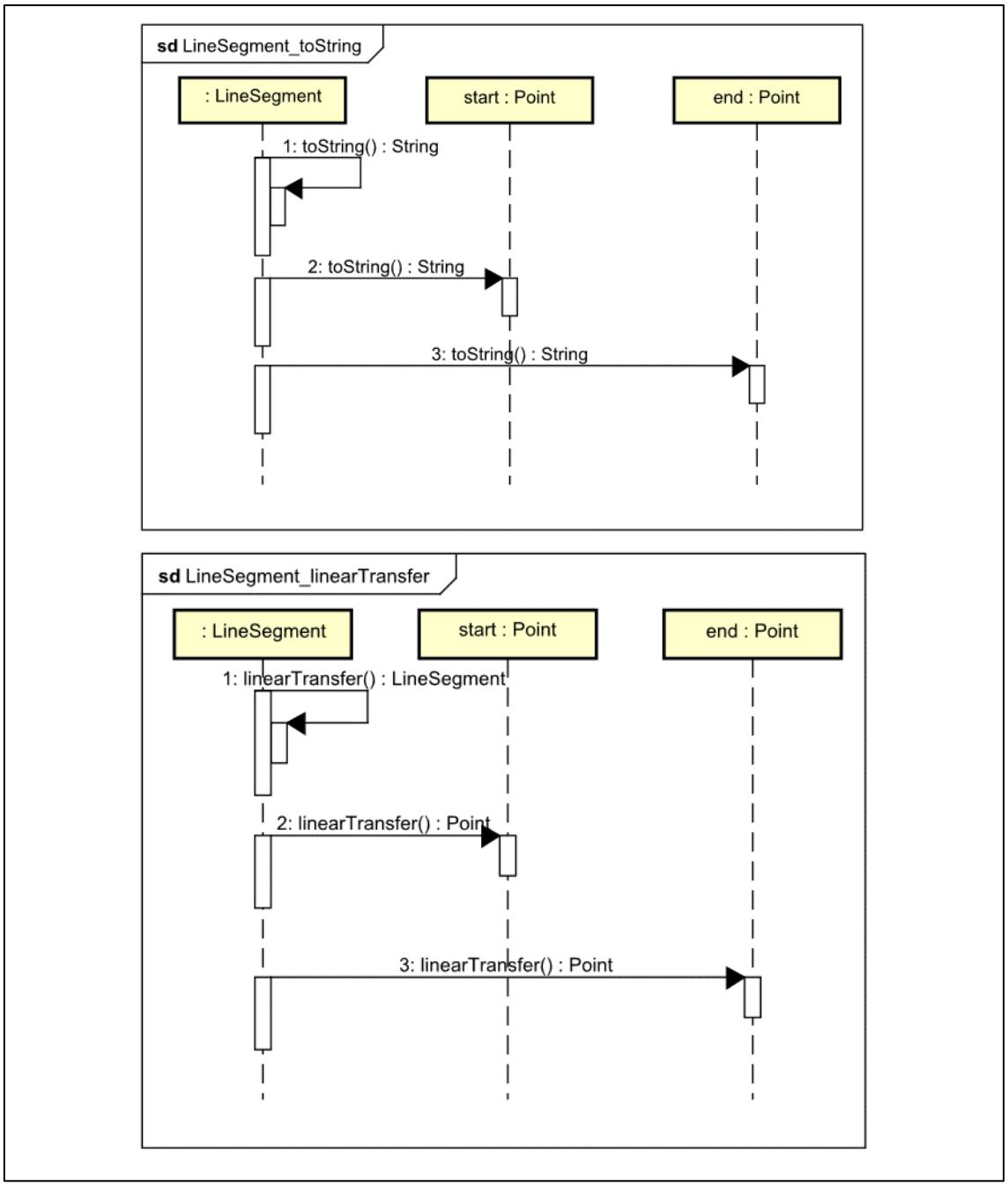

図 15:シーケンス図

テストするソースコード(一部抜粋):

```
public class LineSegment{
        private Point start;
        private Point end;
    public LineSegment linearTransfer(){
        Point transfered_start = start.linearTransfer();
        Point transfered_end = end.linearTransfer();
        LineSegment transferedLine = new
        LineSegment(transfered_start,transfered_end);
        return transferedLine;
    }
    public String toString(){
                return start  + "->" + end;
    }
}
```

実行結果:

```
JUnit version 4.12
.++++++ クラス「LineSegment」 の 「public LineSegment linearTransfer()」
メソッド のテストに成功しました
参考シーケンス図（テストプログラムの参考元）:[LineSegment_linearTransfer]

.++++++ クラス「LineSegment」 の 「public String toString()」 メソッド の
テストに成功しました
参考シーケンス図（テストプログラムの参考元）:[LineSegment_toString]

Time: 0.052

OK (2 tests)
```

図 16:実行結果（成功事例）

テストするソースコード(一部抜粋):

```
public class LineSegment{
        private Point start;
        private Point end;
    public LineSegment linearTransfer(){
        Point transfered_end = end.linearTransfer();
        Point transfered_start = start.linearTransfer();
        LineSegment transferedLine = new
        LineSegment(transfered_start,transfered_end);
        return transferedLine;
    }
    public String toString(){
                return start  + "->" + end;
    }
}
```

実行結果:

```
JUnit version 4.12
.E.++++++ クラス「LineSegment」 の 「public String toString()」 メソッド
のテストに成功しました
参考シーケンス図（テストプログラムの参考元）:[LineSegment_toString]

Time: 0.056
There was 1 failure:
1) linearTransfer_Test(LineSegmentTest)
java.lang.AssertionError: ++++++ クラス「LineSegment」 の 「public
LineSegment linearTransfer()」 メソッド のテストに失敗しました!!!
===>メソッドの呼び出しが足りないか、順番が守られていない可能性があります
参考にして欲しいシーケンス図（テストプログラムの参考
元）:[LineSegment_linearTransfer]
        at
LineSegmentTest.linearTransfer_Test(LineSegmentTest.java:183)

FAILURES!!!
Tests run: 2,  Failures: 1
```

図 17:実行結果(失敗事例)

テストプログラムはあるメソッドをシーケンス図の振る舞い系列(図 15)で表したものとし

てコーディングされているので、もし図 16 のようにテストが失敗する場合は、失敗メッセージに表示されているメッセージを参考にソースコードを修正することになる。一方、テストが成功した場合はシーケンス図で定義されている呼び出し系列を守っているため、意図したとおりの動きをソースコード内で行っているということが言える。

- **別機能**

本ツールでは「クラス図とソースコードを対応させる」(p5)場面で、ソースコードの修正を行うための機能を実装している。

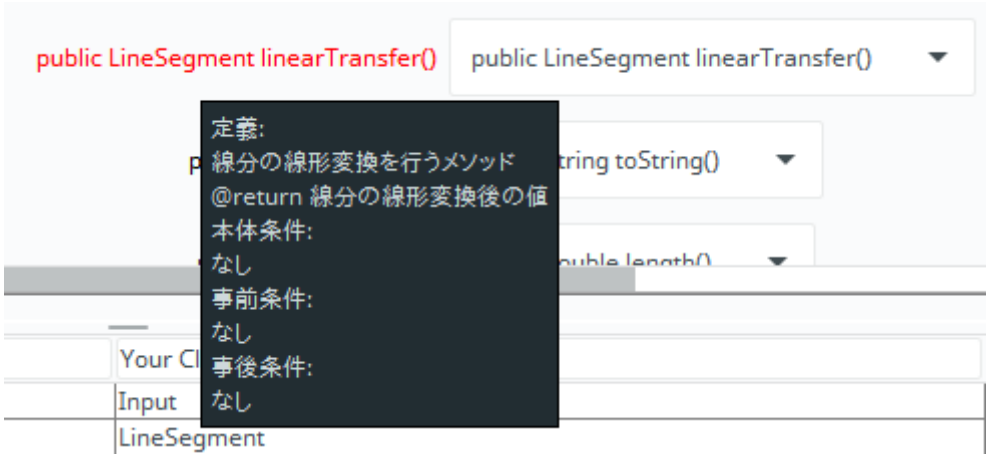

図 18:メソッドの定義の表示

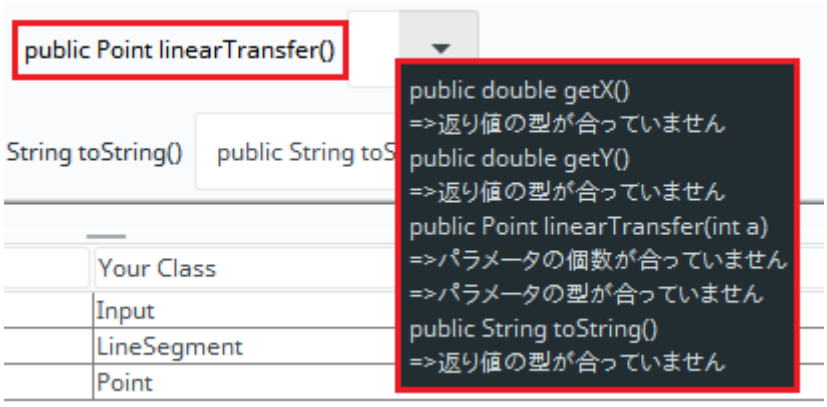

図 19:メソッドを比較し候補を表示

クラス図で定義したあるクラスのメソッドの定義を表示させたい場合は、図 17 のようにメソッド名のラベルにマウスカーソルを合わせると定義などがポップアップ表示される。これは、astah-professional の編集機能で、クラスのメソッドの定義について記述してある内容を参照したものである。

図 18 はメソッドを選択するリストの中に一つもアイテム候補が無い場合に表示されるものである。アイテム候補として選ばれるものは、「クラス図とソースコードを対応させる」(p6)の条件で紹介したとおりで、修飾子、返り値の型、引数の個数、それぞれの引数の型が、メソッドのラベル（図 18 では public Point linearTransfer()）と一致している場合のみで

ある。そのため、ここではソースコードに定義されているそれぞれのメソッドと比較した際に何が異なるのかということをヒントとしてポップアップ表示を行っている。また、フィールドも同様にソースコードとの差異をポップアップ表示する。

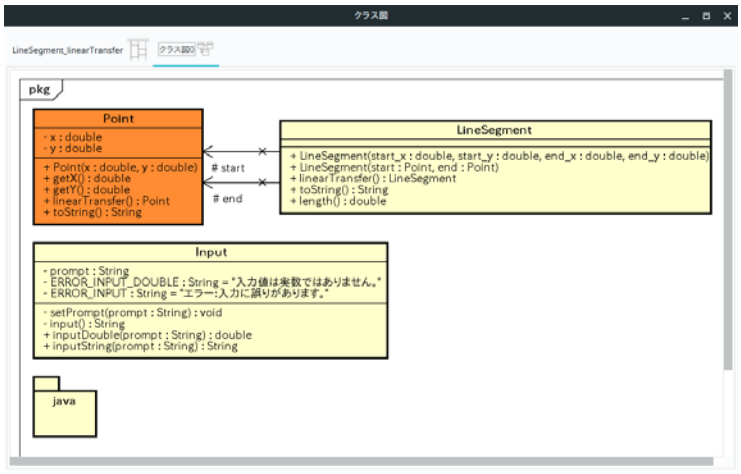

図 20:クラス図

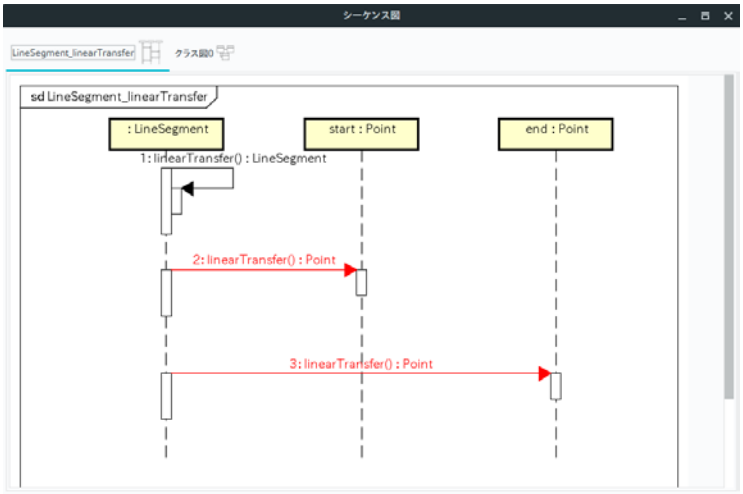

図 21：シーケンス図

メソッドラベルにマウスカーソルを合わせると、図 17 のようにメソッドラベルが赤くなるが、これをクリックすると図 19、図 20 のようなウィンドウが表示される。これらもソースコードの修正のヒントとなるように実装したものであり、図 19 はクリックしたメソッドがどのクラスで定義されているかということについて、クラス図を用いてハイライトしており、図 20 はシーケンス図でそのメソッドが使われている場合はそのメッセージの箇所にハイライトを行っている。

クラス図はハイライトされているクラスの依存関係や自身のもつフィールドやメソッドに着目してほしい。また、シーケンス図ではハイライトされたメッセージがどのメッセージから呼ばれているのか注目してもらいたい。

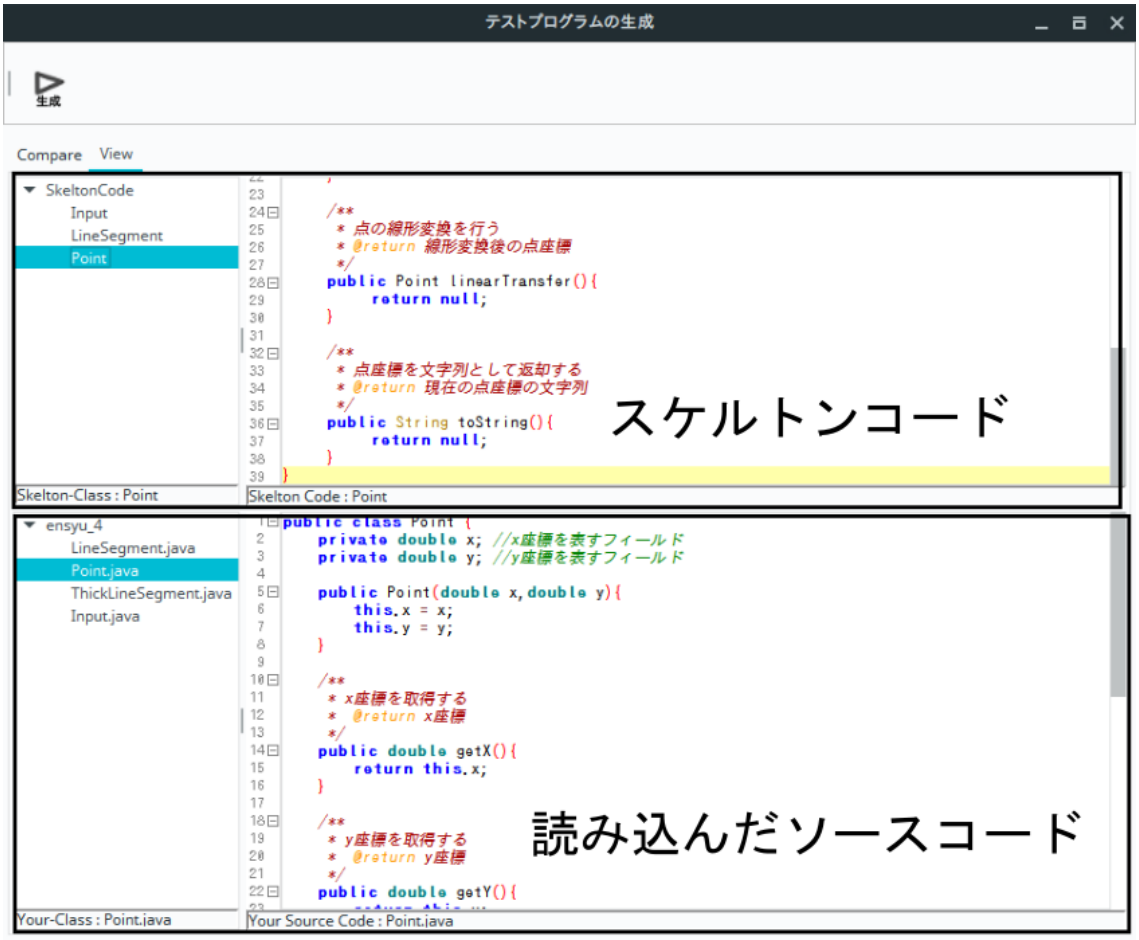

図 22:スケルトンコードと読み込んだソースコードの比較

図 21 は「クラス図とソースコードを対応させる」(p5)の生成ウィンドウで View タブに切り替えると、astah で定義したクラス図とシーケンス図から生成したスケルトンコードと読み込んだソースコードが上下に表示される。ここでは、クラス図とシーケンス図を反映させたスケルトンコードに落とすことにより、ソースコードレベルで比較を行うことができる。