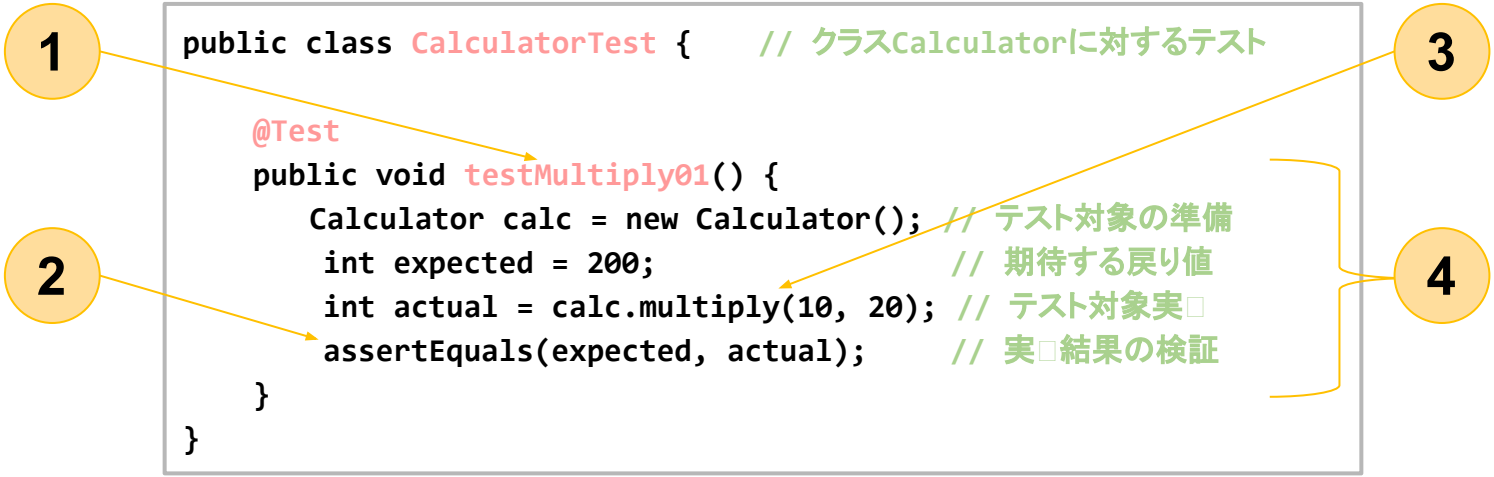
テストコードの設計

- 良いテストコードの設計
- 悪いテストコードの設計

良いテストコードの設計

良いテストコードの設計



① そのテストが何をしているのか他の開発者が見ても分かる**可動性が　高いメソッド名**を記述する
② 1つのテストメソッドに**1つのテスト項目(asser文)**を記述する
③ テスト対象メソッドを1回だけ呼び出されている
④ テストの処理内容が**複雑でなく、外部との依存関係**がない

悪いテストコードの設計

テストメソッド内に複数のassert文が存在する

- 各assert文は異なる条件をテストするが、開発者へ各assert文のエラーメッセージは提供されない
- assert文の1つが失敗した場合、失敗の原因を特定するが困難である

```
@Test
public void testCloneNonBareRepoFromLocalTestServer () {
   .....
   assertThat(readmeFile, exists());
   assertThat(readmeFile, ofLength(-1));
   assertThat(readmeFile, ofLength(1));
}
```

複数のassert文が存在する

テストメソッド名が初期状態(意味のない名前)である

- テスティングフレームを使用した場合、クラス・メソッド名が初期状態(意味がない名前)である
- 可読性の向上のために適切な名前に変更する必要がある

```
@Test
public void test() {
      .....
}
```

テストメソッド名が初期状態のままである

テストメソッド内に複数の制御文が含まれている

- テストの成功・失敗は制御フロー内にあるassert文に基づくためテスト結果を予測できない
- 条件分岐が多く複雑なテストコードは可読性を下げる

```
@Test
public void testSpinner() {
      if (resultObject instanceof EventsModel) {
        if (result.testSpinner.runTest) {
           .....
        else{
           assertEquals(getCount(),size());
          }
      }
}
```

複数の制御文が存在する

テスト対象クラスの複数のメソッドを呼び出す

- 1つのテストメソッドで複数のメソッドをテストすると、他の開発者は何をテストしているかについて混乱が生じる

```
@Test
public void NmeaSentence_GPGSA_ReadValidValues(){
    .....
    assertThat("GPGSA", nmeaSentence.getPdop(), is("2"));
    assertThat("GPGSA", nmeaSentence.getHdop(), is("1"));
    assertThat("GPGSA", nmeaSentence.getVdop(), is(“-2"));
}
```

テスト対象コードのメソッドが複数回呼び出される

テストメソッド内に例外処理が含まれている

- 例外処理は，対象コードに記述すべきで、テストコード内では正しく例外処理が行われるかを確認すべきである

```
@Test
public void realCase() {
    .....
    try {
        a.compute();
    } catch (CalculationException e) {
        Assert.fail(e.getMessage());
    }
    .....
}
```

テストメソッド内に例外処理の記述が存在する

テストメソッド内で，外部リソースを利用する

- テストメソッド内だけでなく外部ファイルなど、外部リソースを使用すると見えない依存関係が生じる
- 誰かが、外部ファイルを削除するとテストが失敗してしまう

```
public void testPersistence() {
    .....
    File tempFile = File.createTempFile("systemstate-", ".txt");
    .....
}
```

外部にファイルを生成し，テストプロセスで利用している