プログラミングII

課題レポート4: リファクタリングを通して継承に慣れよう

編集日:2022-11-27
報告者:e225717 高嶺拓矢
協力者:なし

ステップ0: コードの準備

リポジトリは以下の場所である。report3と同じリポジトリである。 https://github.com/medicine-t/prog2_rep3

ステップ1:EnemyクラスとHeroクラスの重複をどうにかしたい。

1. `gradle test`の結果

```
$ gradle test

BUILD SUCCESSFUL in 909ms
3 actionable tasks: 2 executed, 1 up-to-date
```

2. 今回のコード修正を通して、気づいたことを200字以上で報告せよ。
   LivingThingに主要な機能をまとめたことで、HeroとEnemyの内容を比較したときに相違点がより明確に示されていた。また、スーパークラスを作成する前はほとんど同じ内容が書かれており、特に同様のgetter/setterが両方のコードにあるなどでコードが冗長という印象を受けたが、共通する機能をまとめることでたしかに保守性が向上していると感じた。`@Override`アノテーションを意識的につけることで、意図しないオーバーライドの防止や引数を間違えるなどのミスを防止できることは、開発期間が長期になる程に役に立つ機能なのだろうと感じた。(271字)

ステップ2: Heroクラスの上位職を作ってみよう

1. テストコード

```
package jp.ac.uryukyu.ie.e225717;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class WarriorTest {

    /**
     * Warrior.attackWithWeaponSkillでの攻撃によるHP減少がattackの150%になっているかの確認
     * 検証はWarriorの攻撃に十分な回数耐えることのできるEnemyを生成して行う。
     * attackWithWeaponSkillの仕様から、与えるダメージはattack * 1.5 (== attack * 3 / 2)である
     * Enemyの攻撃を受ける前後でのHPの減少分が先述の計算によるダメージと等しいことを期
```

```
待する
     */
    @Test
    void warriorWeaponTest() {
        Warrior demoWarrior = new Warrior("デモ戦士", 100, 10);
        Enemy slime = new Enemy("スライムもどき", 100, 0);
        for (int index = 0; index < 3; index++) {
            int estimatedDiff = demoWarrior.getAttack() * 3 / 2;
            int beforeAttackHitPoint = slime.getHitPoint();
            demoWarrior.attackWithWeponSkill(slime);
            int afterAttackHitPoint = slime.getHitPoint();
            assertEquals(estimatedDiff, beforeAttackHitPoint -
afterAttackHitPoint);
        }
    }
}
```

2. `gradle test`の結果

```
gradle test --rerun-tasks

BUILD SUCCESSFUL in 754ms
3 actionable tasks: 3 executed
```

動作確認時の影響により`gradle test`ではすべてup-to-dateとなっており実行されなかったためオプション`--rerun-tasks`を利用して実際にテストを通過したことを示した。

3. テストコードの解説　　Warriorの攻撃によるHPの減少分がWarrior.attackの150%(attack * 3 / 2)と等しいかどうかを検証する。少なくとも3回の攻撃をテスト中に行うため、Enemyの初期HPはこの攻撃に十分耐えることのできる値が望ましく、今回は100と設定した。
   増減分の評価は、事前にWarriorから計算した値と、Enemyの被攻撃前後のHPの増減の値の比較によって行った。

オプション: サブクラスのオブジェクトをスーパークラス型に代入した場合？

(1) できた
(2) できた (3) できなかった

```
Task :app:compileJava FAILED
/Users/takuya/prog/prog2_rep3/app/src/main/java/Main2.java:7: エラー: 不適合
な型: HeroをWarriorに変換できません:
        Warrior hero = new Hero("勇者", 10, 5); // (a)
                       ^
エラー1個

FAILURE: Build failed with an exception.
```

(4) できなかった

```
Task :app:compileJava FAILED
/Users/takuya/prog/prog2_rep3/app/src/main/java/Main2.java:16: エラー: シンボルを見つけられません
            hero.attackWithWeponSkill(enemy); // (b)
                ^
  シンボル:   メソッド attackWithWeponSkill(Enemy)
  場所: タイプHeroの変数 hero
エラー1個
```

これらの結果から、サブクラスのオブジェクトをスーパークラスに代入した場合は、スーパークラスの要素以外は無視されることがわかった。そのため、スーパークラスに代入したあとにサブクラスで新たに定義された値やメソッドを利用することはできないようだ。

(5)以下が出力された

```
jp.ac.uryukyu.ie.e225717.Warrior@33909752
```

この結果は`System.out.println(Object x)`を呼び出したことで出力するものとして`String.valueOf(Object obj)`が呼ばれ、`obj`がnullでなかったため`Object.toString()`が呼ばれた結果、`getClass().getName() + '@' + Integer.toHexString(hashCode())`の結果が返されたためこの結果が出力された。