勉強会: Java の Optinal クラスを使ってみる ver 1.0

2021-12-09

株式会社キングプリンターズ システム課

想定する対象

- Java の初学者

目的

- Java の Optinal クラスを使用してみます
- その前提となる `キャスト` 、 `ジェネリックス` の基礎知識を学びます

動作環境※任意

- JDK, Maven を使用すればソースコードを動作(単体テスト)させることが出来ます

```
$ java -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.202-b08, mixed mode)
```

```
$ mvn -version
Apache Maven 3.6.1 (d66c9c0b3152b2e69ee9bac180bb8fcc8e6af555; 2019-04-05T04:00:29+09:00)
```

目次

型とは何でしょうか？ 🤔

指定した型の変数でインスタンスを受け取るとは 🤔

- このようなシンプルな POJO クラスをエンティティクラスとします

```java
/**
 * Person クラス
 */
@Getter
@RequiredArgsConstructor
public class Person {
    /**
     * ID
     */
    @NonNull
    private long id;
    /**
    * 名前
    */
    @NonNull
    private String name;
    /**
     * 性別 1:男性、2:女性
     */
    @NonNull
    private int gender;
    /**
     * 出生日
     */
    @NonNull
    private Date born;
}
```

指定した型の変数でインスタンスを受け取るとは 🤔

- 以下のケースでは o インスタンスのクラス名は何になるでしょうか？

```
// Person 型のインスタンスを作成します
Person p = new Person(1, "テスト太郎", 1, new SimpleDateFormat(DATE_FORMAT).parse("1996-07-31"));

// Object 型として変数に取得します
Object o = (Object) p;

// ここで o のクラス名は何でしょうか？
```

🤔

指定した型の変数でインスタンスを受け取るとは 🤔

- o のクラス名は Person 型なのです 😋

```
// Person 型のインスタンスを作成します
Person p = new Person(1, "テスト太郎", 1, new SimpleDateFormat(DATE_FORMAT).parse("1996-07-31"));

// Object 型として変数に取得します
Object o = (Object) p;

// ここで o のクラス名は何でしょうか？

// o 変数のクラス名を取得します ※ ? はワイルドカード
Class<?> calzz = o.getClass();

// Object 型 の変数で受けても o インスタンスは Person 型なのです
assertEquals(Person.class, calzz);
```

指定した型の変数でインスタンスを受け取るとは 🤔

- ここで 😰😰？？となった方も安心して下さい！
- これから出来るだけ分かりやすく説明していきます 😊

指定した型の変数でインスタンスを受け取るとは 🤔

- このようなリポジトリクラスのメソッドがあるとします
- 戻り値を Person 型で返しています

```
/**
 * 引数 id の Person オブジェクトを返します
 * @param personId
 * @return Person オブジェクトを返します
 */
public Person findByPersonId(long personId) {
    for (Person p : personList) {
        if (p.getId() == personId) {
            return p;
        }
    }
    return null;
}
```

指定した型の変数でインスタンスを受け取るとは 🤔

- さらに先ほどのリポジトリクラスを使用するサービスクラスがあります

```
/**
 * Person オブジェクトを Object 型のオブジェクトとして返します
 * @param personId Person オブジェクトの ID が提供されます
 * @return Object 型のオブジェクトを返します
 */
public Object getObject(long personId) {
    return personRepository.findByPersonId(personId);
}
```

- ここでの戻り値は Person から Object へ自動的に `アップキャスト` されています

指定した型の変数でインスタンスを受け取るとは 🤔

- このように手元のコードで `Object` 型と思いがちですが
- 受け取る変数の型が `Object` 型なだけで `実際の型は違う` 場合があります
- むしろそのオブジェクトの `実際の型` はソースコードだけでは分からない場合が多いです

```
// サービスクラスのオブジェクトから Object 型として変数に取得します
Object o = service.getObject(1);

// o 変数のクラス名を取得します ※ ? はワイルドカード
Class<?> calzz = o.getClass();

// ID:1 のオブジェクトは Person 型なので

// Object 型 の変数で受けても o インスタンスは Person 型なのです
assertEquals(Person.class, calzz);
```

指定した型の変数でインスタンスを受け取るとは 🤔

- ここで Person クラスのサブクラスを加えます

```
/**
 * 魔法使いクラス
 */
@Getter
public class Magician extends Person{
    /**
     * マジックポイント
     */
    @NonNull
    private long mp;
    /**
     * コンストラクタ
     * @param id     IDが提供されます
     * @param name   名前が提供されます
     * @param gender 性別が提供されます
     * @param born   出生日が提供されます
     * @param mp     マジックポイントが提供されます
     */
    public Magician(long id, String name, int gender, Date born, long mp) {
        // スーパークラスのコンストラクタを呼び出します
        super(id, name, gender, born);
        // マジックポイントを初期化します
        this.mp = mp;
    }
}
```

指定した型の変数でインスタンスを受け取るとは 🤔

- またこれらエンティティオブジェクトのリストを以下のように作成しました

```
/**
 * テスト用の Person or サブクラスのオブジェクトのリストを作成します
 */
list.add(new Person(1, "テスト太郎", 1, new SimpleDateFormat(DATE_FORMAT).parse("1996-07-31")));
list.add(new Person(2, "テスト次郎", 1, new SimpleDateFormat(DATE_FORMAT).parse("1999-03-04")));
list.add(new Magician(3, "マジシャン三郎", 1, new SimpleDateFormat(DATE_FORMAT).parse("2001-12-07"), 30));
list.add(new Magician(4, "エレガント弥生", 2, new SimpleDateFormat(DATE_FORMAT).parse("2009-04-01"), 50));
```

指定した型の変数でインスタンスを受け取るとは 🤔

- ID:3 のオブジェクトを `Object型の変数` に取得します

```
// サービスクラスのオブジェクトから Object 型として変数に取得します
Object o = service.getObject(3);

// o 変数のクラス名を取得します ※ ? はワイルドカード
Class<?> calzz = o.getClass();

// ID:3 のオブジェクトは Magician 型です

// Object 型 の変数で受けても o インスタンスは Magician 型です
assertEquals(Magician.class, calzz);
```

- このように o インスタンスの `実際の型` は Magician 型です

指定した型の変数でインスタンスを受け取るとは 🤔

ここまでのまとめ

- Java では全てのオブジェクトのスーパークラスとして `Object` クラスが定義されています
- その `Object型` の変数に全ての型のオブジェクトを参照させることが出来ます
- しかしコード上では `Object型` に見えても `実際の型` は違います

指定した型の変数でインスタンスを受け取るとは 🤔

- ではオブジェクトを `実際の型` として使うのはどうすれば良いのでしょうか？ 🤔

指定した型の変数でインスタンスを受け取るとは 🤔

- `ダウンキャスト` します 😋

指定した型の変数でインスタンスを受け取るとは 🤔

- 以下の例では `Object型` の変数 o を `Person型` の変数 p に `ダウンキャスト` しています

```
// Object 型として変数に取得します
Object o = service.getObject(1);

// Person 型に 'キャスト' します
Person p = (Person) o;

// Person クラスの getName メソッドが使えます
assertEquals("テスト太郎", p.getName());
```

- くれぐれもそもそもの変数 o インスタンスの `実際の型` は `Person型` です 😋

指定した型の変数でインスタンスを受け取るとは 🤔

- 正直、他の動的型付けであるスクリプト言語に慣れた方には意味不明だと思います 😣
- このように静的型付け言語である Java では `キャスト` という機能が必要なケースがあります
- 実際には `インスタンスの型` を変換しているわけではありません
- インスタンスの `実際の型` が Person 型であるオブジェクトを、ある場面では `Object型` として扱い別の場面では `Person型` として扱っているのです

指定した型の変数でインスタンスを受け取るとは🤔

- なぜそんなことをする必要があるのですか？ 🤔

指定した型の変数でインスタンスを受け取るとは 🤔

- 理由の一つとしてはポリモーフィズムを実現するためです 😋

指定した型の変数でインスタンスを受け取るとは 🤔

- ここではポリモーフィズムを深く詳しくは説明しません
- ただ次の例で少しでも `なるほど` と感じて頂けたら幸いです 😋

指定した型の変数でインスタンスを受け取るとは 🤔

- 次に ID:3 のマジシャン三郎の名前を表示させるにはどうすれば良いのでしょうか？ 🤔

```
// Object 型として変数に取得します
Object o = service.getObject(3);

// ID:3 は Magician 型なので
// Magician 型に 'キャスト' します
Magician m = (Magician) o;

// Magician は Person クラスのサブクラスなので getName メソッドが使えます
assertEquals("マジシャン三郎", m.getName());
```

- このように `Object型` の o を `Magician型` の m オブジェクトとして `ダウンキャスト` すれば名前が取れます
- `Magician型` は `Person型` のサブクラス(継承)しているので getName() メソッドが使えます

指定した型の変数でインスタンスを受け取るとは 🤔

- ちょっと待ってください！ 😣 それでは困ることになります！ 😣
- 先ほどの例のようにインスタンスそのものの `具体的な型` で変数受けをするとコード記述者は `具体的な型` を知る必要があります
- 例えばフレームワークなど利用者が拡張して使用する場合、フレームワークの制作者は未来にユーザーが記述する `具体的な型` を知ることは出来ません

指定した型の変数でインスタンスを受け取るとは 🤔

- そこで `具体的な型` を知らなくても良いようにコード上では `抽象的な型` の変数でインスタンスを受け操作します
  - (※ここでの説明では抽象クラスのことではありません)

指定した型の変数でインスタンスを受け取るとは 🤔

- この例では `Magician型` の三郎を `Person型` のオブジェクトとして扱うことに意味があるのです 😋

```
// Object 型として変数に取得します
Object o = service.getObject(3);

// ID:3 は Magician 型ですが Person 型に 'キャスト' します
Person p = (Person) o;

// Person クラスの getName メソッドが使えます
assertEquals("マジシャン三郎", p.getName());

// このような性質を 'ポリモーフィズム' と言います
// Person 型を継承するオブジェクトは Person 型として扱えます
// フレームワークの内部ではこのような仕組みを多用しています
```

指定した型の変数でインスタンスを受け取るとは 🤔

- この例をフレームワークとすると `名前` を使う必要があった場合 `Person型` の変数で受け取れば動作に問題ありません
- 実際の p `インスタンス` の型が何であるかはフレームワークの制作者は気にしていません
- フレームワークの利用者が `Person型` を継承したサブクラスを実装すればフレームワークは良きに扱ってくれます 😋

```
// もちろんはじめから Person 型として取得出来ます
// サービスクラスのオブジェクトから Person 型として変数に取得します
Person p = service.getPerson(3);

// Person クラスの getName メソッドが使えます
assertEquals("マジシャン三郎", p.getName());
```

指定した型の変数でインスタンスを受け取るとは 🤔

ここまでのまとめ

- Java では継承という仕組みでポリモーフィズムの一部を実現しています
- フレームワークなどのコード上ではオブジェクト達は `具体的` な `実際のクラス(型)` ではなく、 `抽象的` な `キャストされたクラス(型)` で記述されます
    - しかしプログラム内で動作しているのは `実際のクラス` の型の `インスタンス` です

指定した型の変数でインスタンスを受け取るとは 🤔

ここまでのまとめ

- 逆に言えばコード上では `インスタンス` を `実際のクラス(型)` ではなく `抽象的なクラス(型)` として扱うことが出来ます
- ここでは `その時に相応しい型` に `キャスト` することによりフレームワークが動作している程度のシンプルな理解で十分です
    - 初期段階の実務では `誰かに継承させる抽象クラス` より `何かを継承した具象クラス` を書くことが多いと思います

ジェネリックスとは 🤔

- 簡単に説明すると型パラメータをとるクラスのことをジェネリックスクラスと言います
- Java での一番シンプル例は `List<T>` です

```
List<String> list = new ArrayList<String>();
```

- ここでは `String型` のパラメータを設定した `ArrayList型` のインスタンスを `List型(※インタフェース)` の変数で受けています
- とても Java っぽいコードです 😋

ジェネリックスとは 🤔

- 次の例では `型パラメータ` を使用していない `List型` のオブジェクトを作成した例です
- ループ処理で `キャスト` が発生しています

```
// リストを作成します
List list = new ArrayList();

// テスト用のオブジェクトをリストに追加します
list.add(new Person(1, "テスト太郎", 1, new SimpleDateFormat(DATE_FORMAT).parse("1996-07-31")));
list.add(new Magician(3, "マジシャン三郎", 1, new SimpleDateFormat(DATE_FORMAT).parse("2001-12-07"), 30));

// エンティティオブジェクトの名前を表示する為には
for (Object o : list) {
    // キャストする必要があります
    Person p = (Person) o;
    // 標準出力に表示します
    System.out.println(METHOD() + p.getName());
}

// 型パラメータを指定しない List では一旦 Object 型のオブジェクトとしてしか受け取れません
```

ジェネリックスとは 🤔

- 次の例は `型パラメータ` を使用した `List型` のオブジェクトを作成した例です
- ループ処理で `キャスト` する必要がありません

```
// 型パラメータを指定してリストを作成します
List<Person> list = new ArrayList<Person>();

// テスト用のオブジェクトをリストに追加します
list.add(new Person(1, "テスト太郎", 1, new SimpleDateFormat(DATE_FORMAT).parse("1996-07-31")));
list.add(new Magician(3, "マジシャン三郎", 1, new SimpleDateFormat(DATE_FORMAT).parse("2001-12-07"), 30));

// エンティティオブジェクトの名前を表示します
for (Person p : list) {
    // 標準出力に表示します
    System.out.println(METHOD() + p.getName());
}

// Person 型の List からは Person 型のオブジェクトとして利用出来ます
// 危険なキャストも使わないので安全なコードになります
```

ジェネリックスとは 🤔

ここまでのまとめ

- 簡単に言えば `型パラメータ` を使用するクラスを使うことです
- 初期の業務では `型パラメータ` を取る抽象的なクラスを設計することはまれだと思います
- ジェネリックス自体は抽象的・汎用的なコード記述を実現する為に存在します

基礎知識を学んで 🤔

- `キャスト`、`ジェネリックス`がどんなものかをなんとなく掴むことが出来ましたでしょうか？
- どちらも `抽象的` で `汎用的` なコード記述を実現する為に存在します
- 特に Java のプログラムはソースコードを見るだけでは実態がつかみにくいです
  - ぜひデバッガでブレークして変数 `インスタンス` の `実際の型` を確認してみてください！ 😋

Optional を使ってみる 🤔

- ようやく本題にたどり着きました！
- それでは Java の Optional についてみて行きたいと思います 😋

Optional を使ってみる 🤔

- Java の Optional クラスとは
  - Optional 型は値をラップしその値が null かもしれないことを表現するクラスです
- 例えばこのようにラップします
  - ※ジェネリックスの型パラメータを指定することをラップすると表現しています

```
// Person 型の型パラメータを設定した Optional 型のオブジェクトを作成します
Optional<Person> pOpt = Optional.ofNullable(service.getPerson(2));
```

Optional を使ってみる 🤔

- なぜそのように値とラップすることが必要なのでしょうか？ 🤔
  - 次の例を見ながら考えて行きましょう！ 😋

Optional を使ってみる 🤔

- このようなコードは誰でも書いてしまいがちです

```
// サービスオブジェクトから ID:7 を取得しようとしています
Person p = service.getPerson(7);

// 注意 ID:7 のオブジェクトは存在しません

// エンティティオブジェクトの名前を表示してみます
System.out.println(METHOD() + p.getName());

// java.lang.NullPointerException が発生します！
```

- サービスオブジェクトから null が返された場合 NullPointerException が発生します 😫

Optional を使ってみる 🤔

- こちらはオブジェクトが null でないかチェックした例です

```
// サービスオブジェクトから ID:7 を取得しようとしています
Person p = service.getPerson(7);

// オブジェクトが null じゃないか確認します
if (p != null) {
    // p が null なのでこの処理は実行されません
    System.out.println(METHOD() + p.getName());
}
```

- null 判定が非常に有効であることが分かります

Optional を使ってみる 🤔

- なせこのようなことが起こるのでしょうか？
  - Java の場合変数(※参照型)には null が代入出来てしまうからです
  - ※ Java には Null非許容型の参照型の変数はありません
    - つまり Java は Null安全な言語ではありません 😓

Optional を使ってみる 🤔

- 気を取り直して Optional 型でラップしてから使用した例を見てみます

```
// Person 型の型パラメータを設定した Optional 型のオブジェクトを作成します
Optional<Person> pOpt = Optional.ofNullable(service.getPerson(7));

// オブジェクトが null ではない場合の記述方法を書かざるを得なくなります
pOpt.ifPresent(
    p -> System.out.println(METHOD() + p.getName())
);
```

- この例では Optional の `ifPresent()` メソッドを通してオブジェクトを操作することになります

Optional を使ってみる 🤔

- 次にオブジェクトが null でないかチェックしてから使用する例に、値がない場合の処理を追加します
    - ベタに記述します

```
// サービスオブジェクトから ID:7 を取得しようとしています
Person p = service.getPerson(7);

// オブジェクトが null じゃないか確認します
if (p != null) {
    // p が null なのでこの処理は実行されません
    System.out.println(METHOD() + p.getName());
} else {
    System.out.println(METHOD() + "ないです");
}
```

- 非常にオーソドックスなコードです
    - ※Java なのでこのコードが良くないわけではありません

Optional を使ってみる 🤔

- 前の例を Optional を使って記述してみます

```
// Person 型の型パラメータを設定した Optional 型のオブジェクトを作成します
Optional<Person> pOpt = Optional.ofNullable(service.getPerson(5));

// デフォルトの Person オブジェクトを作成しておく
// "ないです" を表示させたいだけで他の値に意味はありません
Person defaultP = new Person(0, "ないです", 0, new SimpleDateFormat(DATE_FORMAT).parse("1970-01-01"));

// Optional 型のオブジェクトから Person 型のオブジェクトを取得
Person p = pOpt.orElse(
    // 値がなければこちらが取得される
    defaultP
);

// エンティティオブジェクトの名前を表示
System.out.println(METHOD() + p.getName());
```

- ※ Java 8 には ifPresentOrElse が存在しないのでこのような無様な書き方になりました 😓

Optional を使ってみる 🤔

- 実際には手元のコードでオブジェクトを Optional 型でラップしてから使用したケースはまれだと思います
- 他オブジェクトなどが `Optional型の戻り値` を返してくることが想定されます

```
// サービスオブジェクトから Optional 型のオブジェクトを取得します
Optional<Person> pOpt = service.getOptional(7);

// 事前にわざわざ値の存在をチェックをする
if (pOpt.isPresent()) { // ※ Java 8 には ifPresentOrElse が存在しないのでベタに書きます
    // Optional 型のオブジェクトから Person オブジェクトを取得
    // ※ null ではないことは保障されている
    Person p = pOpt.get();
    // エンティティオブジェクトの名前を表示
    System.out.println(METHOD() + p.getName());
} else {
    System.out.println(METHOD() + "ないです");
}
```

- 実際には if, else を使った方が分かりやすいコードになることも多いです

Optional を使ってみる 🤔

- しかし Optional 型を返して来る外部オブジェクトも信用出来ないケースがあります

```
// サービスオブジェクトから Optional 型のオブジェクトを取得します
Optional<Person> pOpt = service.getOptionalAsNull(7);

// そもそもこの pOpt が null かどうかチェックしないと意味がないのでは？

// 上のコードと同じですが Optional が null の可能性はあります
if (pOpt.isPresent()) {
    Person p = pOpt.get();
    System.out.println(METHOD() + p.getName());
} else {
    System.out.println(METHOD() + "ないです");
}
```

- 当然 java.lang.NullPointerException が発生します！

Optional を使ってみる 🤔

ここまでのまとめ

- あるオブジェクトから戻り値が `Optional型` で返される場合 null を意識した書き方にならざるを得ないので有効です
- しかし、その `Optional型` の戻り値が `100パーセント null ではない` と言えないのでそれだけでは不十分だと思います
    - ※ Java 言語の場合です

Optional を使ってみる 🤔

- `Optional型` を使用することで出来ること・出来ないことが少しずつ分かつて来たと思います
- 次にリストを扱う場合を考えてみましょう 😋

Optional を使ってみる 🤔

- 他のオブジェクトからリストを取得してループ処理する例です

```
// サービスオブジェクトからエンティティオブジェクトのリストを取得します
List<Person> list = service.getAll();

// エンティティオブジェクトの名前を表示します
for (Person p : list) {
    // 標準出力に表示します
    System.out.println(METHOD() + p.getName());
}
```

- list オブジェクトのチェックをしなくても大丈夫でしょうか？ 🤔

Optional を使ってみる 🤔

- サービスオブジェクトから `空のリスト` 返る場合を見てみます

```
// サービスオブジェクトからエンティティオブジェクトのリストを取得します
List<Person> list = service.getAllAsEmpty(); // `空` のリストが返ります

// エンティティオブジェクトの名前を表示します
for (Person p : list) {
    // 標準出力に表示します
    System.out.println(METHOD() + p.getName());
}
```

- 空のリストだとループ処理が走らないのでエラーは起きません 😋

Optional を使ってみる 🤔

- 次にサービスオブジェクトから `null` が返る場合を見てみます

```
// サービスオブジェクトからエンティティオブジェクトのリストを取得します
List<Person> list = service.getAllAsNull();

// エンティティオブジェクトの名前を表示します
for (Person p : list) {
    // 標準出力に表示します
    System.out.println(METHOD() + p.getName());
}
```

- null の list オブジェクトを操作すると java.lang.NullPointerException が発生します！ 😫

Optional を使ってみる 🤔

- リストを操作する場合にも場合によっては `nullチェック` が必要そうです

Optional を使ってみる 🤔

- リストを `Optional型` クラスでラップしてみます

```
// サービスオブジェクトからエンティティオブジェクトのリストを取得します
Optional<List<Person>> listOpt = service.getAllAsOptional();

// エンティティオブジェクトの名前を表示します
listOpt.ifPresent(list -> list.stream().forEach(
    // 標準出力に表示します
    p -> System.out.println(METHOD() + p.getName())
));
```

- このように null を意識したコードを書く必要があります
  - ※記述方法が難しいので別機会で説明する予定です

Optional を使ってみる 🤔

- しかし Optional 型のオブジェクト自体が null の場合では

```
// Optional 型のオブジェクト自体が null の場合
Optional<List<Person>> listOpt = null;

// エンティティオブジェクトの名前を表示します
listOpt.ifPresent(list -> list.stream().forEach(
    // 標準出力に表示します
    p -> System.out.println(METHOD() + p.getName())
));
```

- 当然 java.lang.NullPointerException が発生します！ 😫

Optional を使ってみる 🤔

ここまでのまとめ

- Java は Null安全な言語ではないのでどう頑張っても Null安全には出来ません
- 残念ですがこの例は Optional クラスのメリット・デメリットを考慮してバランスを見て使用していくのが良いかと感じました
- また、完全な Null安全を言語的に備えている言語ではそうではないと思います

全体のまとめ

- ここでは Java の `Optical` クラス・機能を学ぶために `キャスト` 、 `ジェネリックス` の基礎的な内容を紹介しました
- このようにある `技術要素を学ぶために他の技術の理解が不可欠である` ということは多いと思います
- 少しづつでも良いので様々な技術的要素を `関連付けて` 学んでいくのが良いと思いました

ありがとうございました