

今週の一コマ画像



図0...巨人の方の上に乗る(酔拳ver.)

APJ06

学籍番号: 744366

氏名: 岸 典樹

2020年10月31日

# 第1章 ジェネリクスを用いたライブラリについて

本章ではJava SE APIの中からジェネリクスを用いたライブラリ “AbstractMap<K,V>”について設計図や例題プログラムを援用して論ずる。また、ワイルドカードやパラメータが付随したジェネリクスについても論ずる。

## 1.1 AbstractMap<K,V>の簡単な説明

AbstractMapはMapインターフェースのスケルトン実装を提供し、このインターフェースを実装するのに必要な作業量を最小限に抑えるためのライブラリである。AbstractMap<K,V>の所在地は、java.util.AbstractMap<K,V>である。

## 1.2 AbstractMap<K,V>の設計図

AbstractMap<K,V>のUMLを以下に記す。

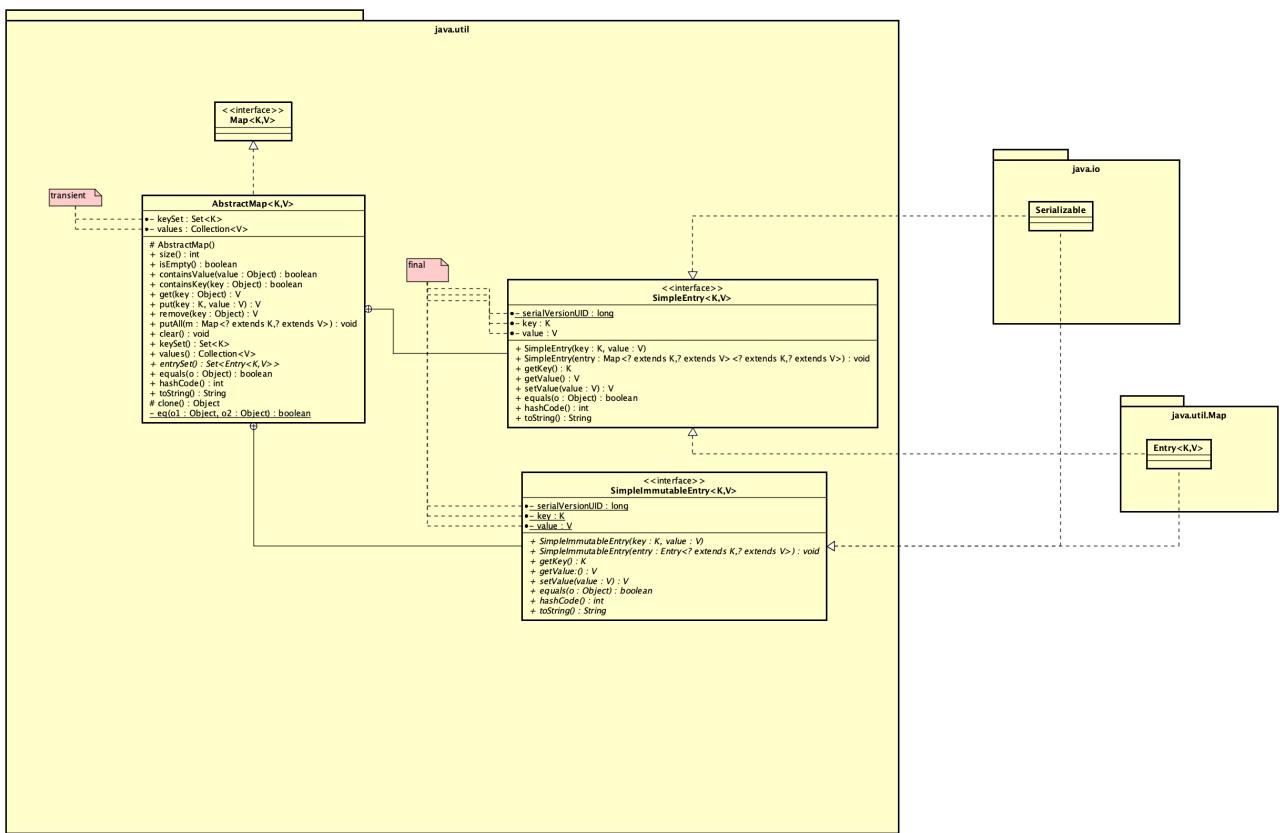


図1.2.1 java.util.AbstractMap<K,V>のUML

## 1.3 AbstractMap<K,V>の例題プログラム

AbstractMapはabstractであるため、(第2回講義で展開したsrcの中にいたとして).`/java/util/HashMap.java`が良い例になる。2000行を超えるプログラムであるため、一部を抜粋する。

```
138  public class HashMap<K,V> extends AbstractMap<K,V>{
139      ... implements Map<K,V>, Cloneable, Serializable {
140      ...
141      ... private static final long serialVersionUID = 362498820763181265L;
142  }
```

図1.3.1 HashMapの定義部。AbstractMapが継承されている。

## 1.4 AbstractMapについて

先述したとおり、AbstractMapはMapインターフェースのスケルトン実装を提供し、このインターフェースを実装するのに必要な作業量を最小限に抑えるためのライブラリである。実際に、「java.util.○○Map」にマッチするAPIにおいてはTreeMap, IdentityHashMap, EnumHashMap, WeakHashMap, ConcurrentHashMap, ConcurrentSkipListMap, HashMapに使用されている。

## 1.5 仮型パラメータやワイルドカードを用いた境界型パラメータについて

Abstract.javaにはKやV、? extends K, ? extends Vなど、いくつかの特殊な型が登場した。KeyやValueを略したK, V、Typeを略したTなど、大文字アルファベット1文字で表す仮型パラメータと、? extends Tで表す境界型パラメータである。前者は、コンストラクタで与えられたどんな型でも柔軟に対応でき、同じ仮型パラメータ同士で型を共有できる。一方?はワイルドカードであり、どのような型でも使用できる。Abstract.javaにおいてよく出てきたジェネリクスが<? extends T>であり、Tのサブクラスであればいかなる型でも受け付けるというものである。Integer用のジェネリクス、String用のジェネリクス、またはInteger, Stringペアのジェネリクス、String, Integerペアのジェネリクス…など、全てのパターンでプログラムを組むと再利用性に欠け、プログラムが非常に膨大なものになってしまうため、このような仮の型やワイルドカードが設置されているものだと考察する。

## 第2章 Bevyの実装

本章では群を表す「Benvy」を作る際に、継承と集約のそれぞれを用いた設計図とスケルトンアーカイブについて、期待する結果を得るためのプログラムを実装し、当該コードを示すとともに、その長短を論ずる。

### 2.1 BevyByAggregationの変更点

BenvyByAggregationの変更点をいかに記す。



The screenshot shows a dark-themed terminal window on a Mac OS X desktop. In the top-left corner, there are three colored window control buttons: red, yellow, and green. The main area of the terminal contains the following Java code:

```
/**  
 * 自分自身の内の指定された位置(1オリジン、0オリジンではない)にある要素を応答する。  
 * @param anIndex 指定位置(インデックス:1オリジンであることに注意)  
 * @return 指定された位置にある要素  
 */  
public Element at(Integer anIndex)  
{  
    this.examineIndex(anIndex);  
  
    // return null; // ちゃんと実装してください。  
    return this.arrayList.get(anIndex - 1);  
}
```

図2.1.1 BevyByAggregationのatの変更点

```
/*  
 * 自分自身の内の指定された位置(1オリジン、0オリジンではない)にある要素を、指定された要素で置き換える。  
 * @param anIndex 指定位置(インデックス:1オリジンであることに注意)  
 * @param anElement 置き換える要素  
 * @return 指定された位置に以前(置き換える前に)あった(元々の)要素  
 */  
public Element atPut(Integer anIndex, Element anElement)  
{  
    this.examineIndex(anIndex);  
    Element anElementBeforeChange = this.arrayList.get(anIndex - 1);  
    this.arrayList.set(anIndex - 1, anElement);  
  
    // return null; // ちゃんと実装してください。  
    return anElementBeforeChange;  
}
```

図2.1.2 BevyByAggregationのatPutの変更点

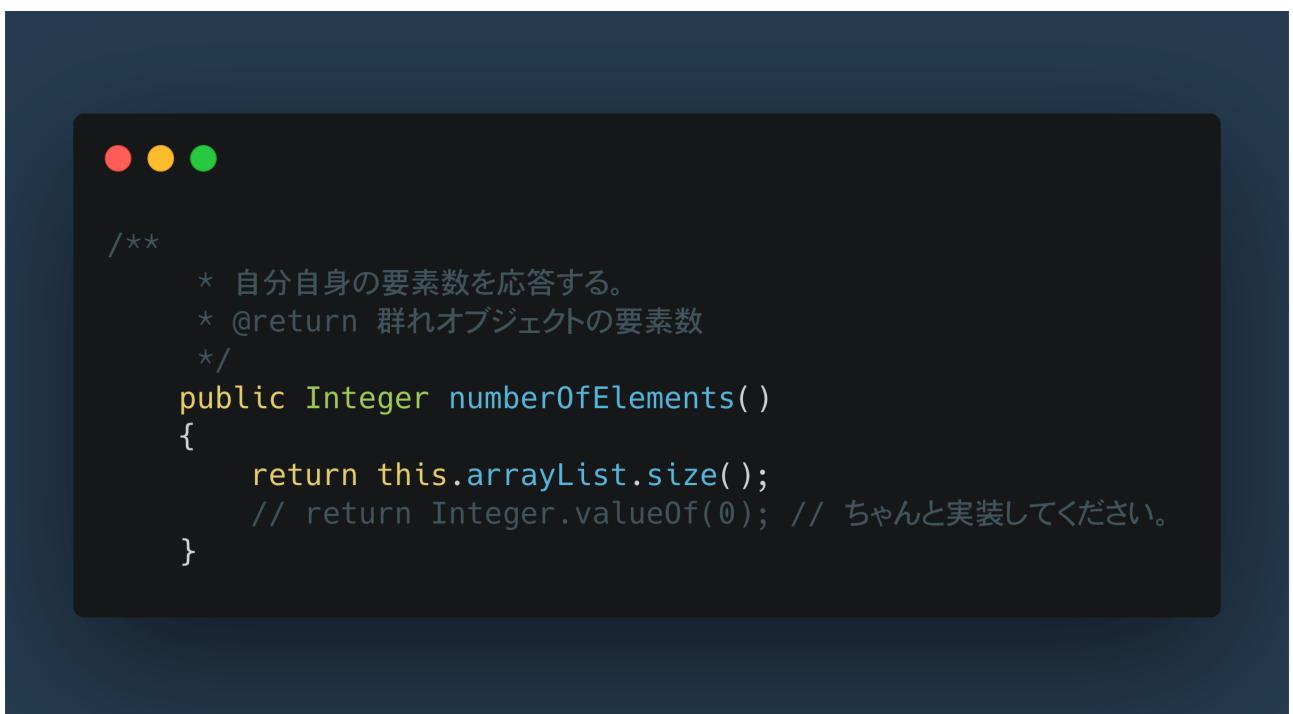
```
/*  
 * 自分自身の最後に指定された要素を追加する。  
 * @param anElement 追加したい要素  
 * @return 要素を追加できたか否かの真偽  
 */  
public boolean addElement(Element anElement)  
{  
    return this.arrayList.add(anElement);  
    // return false; // ちゃんと実装してください。  
}
```

図2.1.3 BevyByAggregationのaddElementの変更点



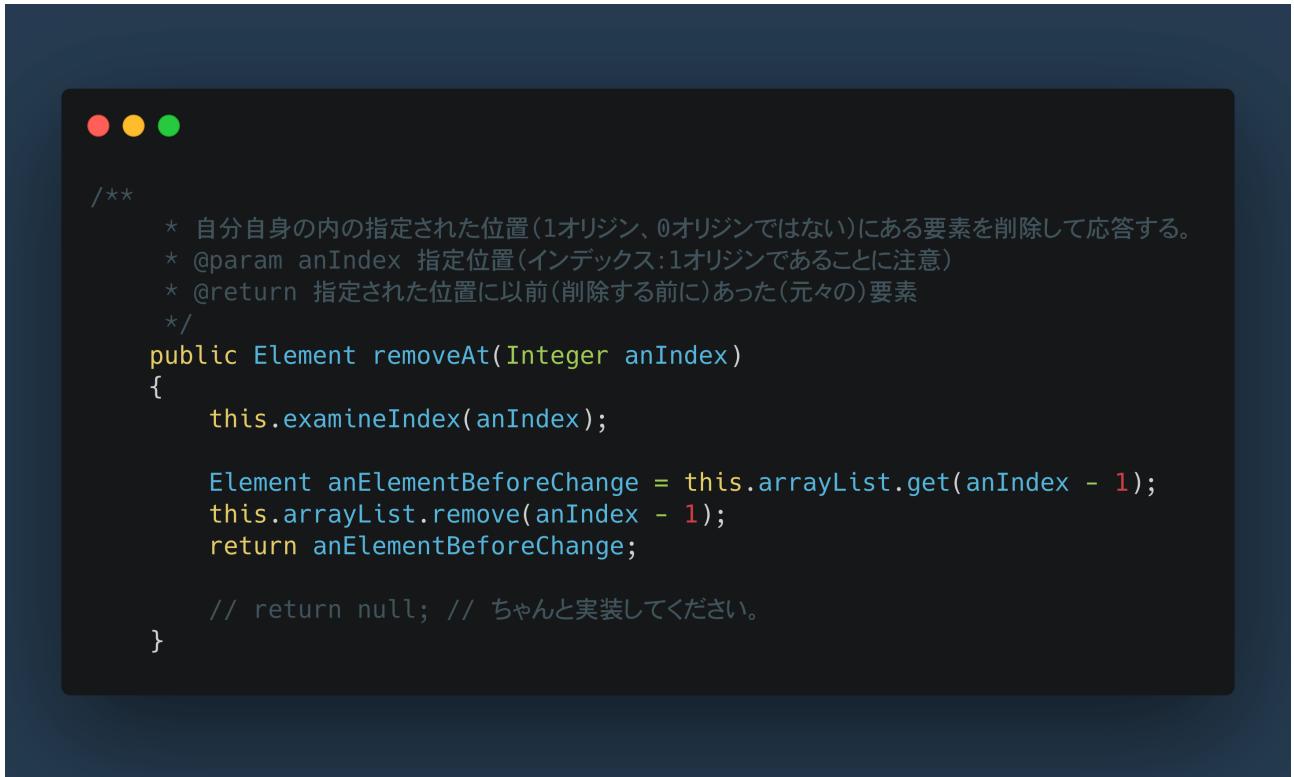
```
/** * 自分自身が空かどうかを応答する。 * @return 空かどうかの真偽 */ public boolean isEmpty() {    return this.arrayList.isEmpty();    // return true; // ちゃんと実装してください。 }
```

図2.1.4 BevyByAggregationのisEmptyの変更点



```
/** * 自分自身の要素数を応答する。 * @return 群れオブジェクトの要素数 */ public Integer numberOfElements() {    return this.arrayList.size();    // return Integer.valueOf(0); // ちゃんと実装してください。 }
```

図2.1.5 BevyByAggregationのnumberOfElementsの変更点



```
/**  
 * 自分自身の内の指定された位置(1オリジン、0オリジンではない)にある要素を削除して応答する。  
 * @param anIndex 指定位置(インデックス:1オリジンであることに注意)  
 * @return 指定された位置に以前(削除する前に)あった(元々の)要素  
 */  
public Element removeAt(Integer anIndex)  
{  
    this.examineIndex(anIndex);  
  
    Element anElementBeforeChange = this.arrayList.get(anIndex - 1);  
    this.arrayList.remove(anIndex - 1);  
    return anElementBeforeChange;  
  
    // return null; // ちゃんと実装してください。  
}
```

図2.1.6 BevyByAggregationのremoveAtの変更点

## 2.2 BevyByInheritanceの変更点



```
/**  
 * 自分自身の内の指定された位置(1オリジン、0オリジンではない)にある要素を応答する。  
 * @param anIndex 指定位置(インデックス:1オリジンであることに注意)  
 * @return 指定された位置にある要素  
 */  
public Element at(Integer anIndex)  
{  
    this.examineIndex(anIndex);  
  
    // return null; // ちゃんと実装してください。  
    return super.get(anIndex - 1);  
}
```

図2.2.1 BevyByInheritanceのatの変更点

```
/*  
 * 自分自身の内の指定された位置(1オリジン、0オリジンではない)にある要素を、指定された要素で置き換える。  
 * @param anIndex 指定位置(インデックス:1オリジンであることに注意)  
 * @param anElement 置き換える要素  
 * @return 指定された位置に以前(置き換える前に)あった(元々の)要素  
 */  
public Element atPut(Integer anIndex, Element anElement)  
{  
    this.examineIndex(anIndex);  
  
    // return null; // ちゃんと実装してください。  
    // Element anElementBeforeChange = super.get(anIndex - 1);  
    Element anElementBeforeChange = super.get(anIndex - 1);  
    super.set(anIndex - 1, anElement);  
    return anElementBeforeChange;  
}
```

図2.2.2 BevyByInheritanceのatPutの変更点

```
/*  
 * 自分自身の最後に指定された要素を追加する。  
 * @param anElement 追加したい要素  
 * @return 要素を追加できたか否かの真偽  
 */  
public boolean addElement(Element anElement)  
{  
    // return false; // ちゃんと実装してください。  
    return super.add(anElement);  
}
```

図2.2.3 BevyByInheritanceのaddElementの変更点

```
/*  
 * 自分自身が空かどうかを応答する。  
 * @return 空かどうかの真偽  
 */  
@Override  
public boolean isEmpty()  
{  
    // return true; // ちゃんと実装してください。  
    return super.isEmpty();  
}
```

図2.2.4 BevyByInheritanceのisEmptyの変更点

```
/*  
 * 自分自身の要素数を応答する。  
 * @return 群れオブジェクトの要素数  
 */  
public Integer numberOfElements()  
{  
    // return Integer.valueOf(0); // ちゃんと実装してください。  
    return super.size();  
}
```

図2.2.5 BevyByInheritanceのnumberOfElementsの変更点

```
/*  
 * 自分自身の内の指定された位置(1オリジン、0オリジンではない)にある要素を削除して応答する。  
 * @param anIndex 指定位置(インデックス:1オリジンであることに注意)  
 * @return 指定された位置に以前(削除する前に)あった(元々の)要素  
 */  
public Element removeAt(Integer anIndex)  
{  
    this.examineIndex(anIndex);  
  
    // return null; // ちゃんと実装してください。  
    Element anElementBeforeChange = super.get(anIndex - 1);  
    super.remove(anIndex - 1);  
    return anElementBeforeChange;  
}
```

図2.2.6 BevyByInheritanceのremoveAtの変更点

## 2.3 双方のUML

以下に、配布された2つのBevyのUMLを記す。

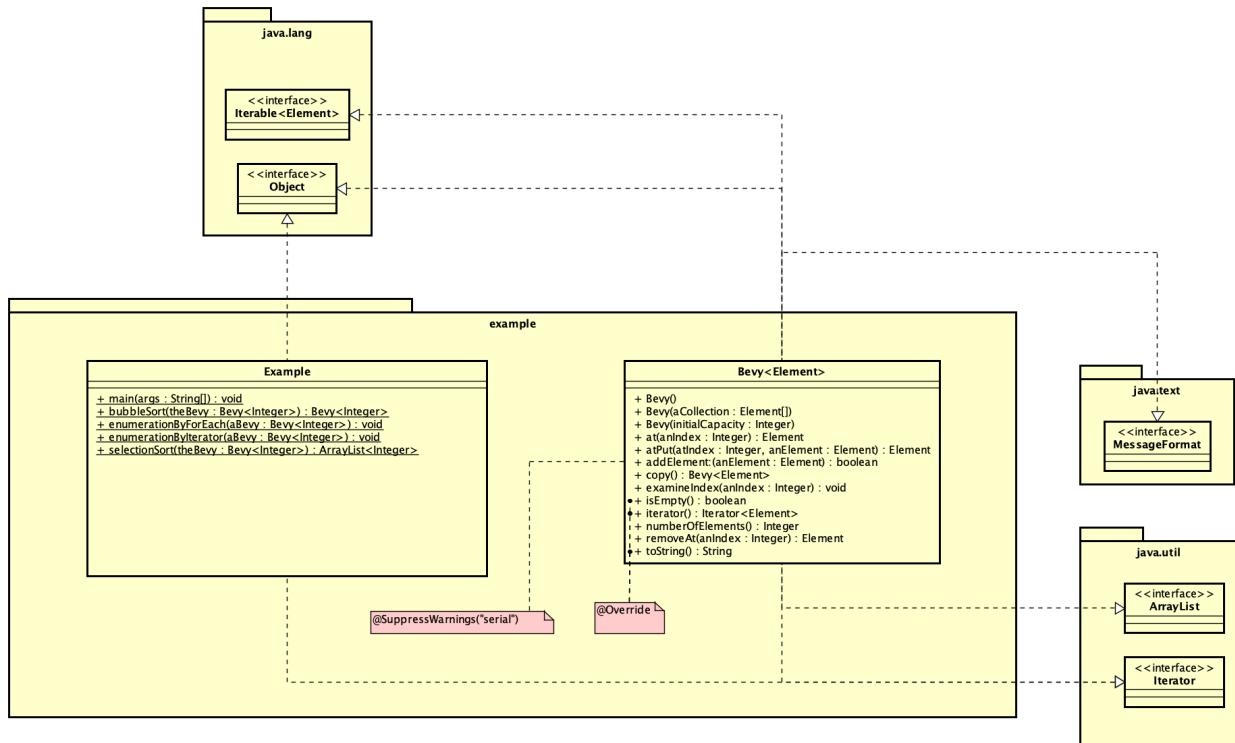


図2.3.1 繙承を用いたBevyのUML

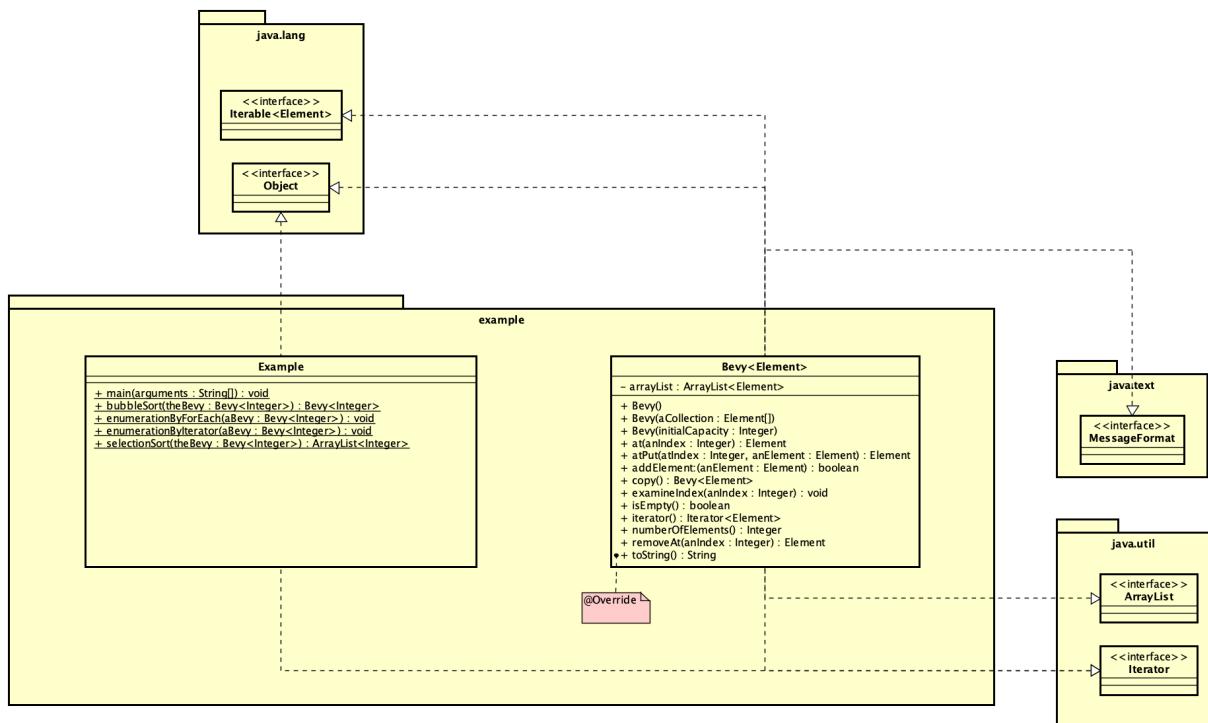


図2.3.2 集約を用いたBevyのUML

## 2.4 繙承を用いて作ったプログラムの長所・短所

継承を用いてあるオブジェクトを引き継ぐことによってることで、無駄なフィールドの定義がなくなり、コードによりまとまりが持てるようになった。しかし、UML上では「AはBの一種である」と言う意味で表現されるため、便宜上好ましくない場合が多い。そして、1つのスーパークラスしか継承できないことが、Javaにおける最大の欠点である。一方、集約はあるクラスを他のクラスと組み合わせて構成した関係を指し、今度のプログラムにおいてはthis.ArrayListとBevyでElement型を共有している。集約はスーパークラスを別に定義できる他フィールドを外部から直接操作できるなどのメリットがある代わり、少しばかりコードが散らばる弱点を持つ。

# 第3章 自由記述

## 3.1 応用プログラミング(\*)について

応用プログラミングを学ぶことについて、感銘を受けました。オブジェクト指向プログラミングを応用的に学ぶ手段としてJavaを例に学んでいくことがこの授業の真意だと考えていますが、そう考えると、エンジニアの就職活動に疑問が生じます。と言いますのも、求人やスキルチェックシートにはJavaをどれくらい使えるか、Pythonをどれくらい使えるか、Rubyをどれくらい使えるかなど、各言語におけるプログラミング能力を問う項目がありますが、この概念自身に疑問を持ちます。もっとも、プログラミング初心者という観の中においては幾つ言語を使えるかが一つのヒエラルキーになっているように感じますが、達者になるにつれてこの概念の意味が薄れしていくよう思います。つまり、ドキュメントを見ないでも方言を覚えている言語がいくつあるのかを問いただすのは即戦力を求める企業にとって良いことではあるが、本来プログラマ(ないしはエンジニア)に求められる能力はどのパラダイムについて熟知しているかではないのかと疑問に思っています。

## 3.2 インスタンス名「a<ClassName>」について

純粋な疑問ではありますが、インスタンスのネーミングについて、英語的な意味でaInstance, an Instance, theInstanceとネーミングすることに馴染みがないのですが、より達者な方が書かれたプログラムはこう言ったネーミングの方が多いのでしょうか。私自身が身を投じてきた環境に凄腕のプログラマは(使わせていただいているAPIや言語の開発者...などを除いて)一人もいなかったため、当該の書き方について、純粋に疑問に思います。

## 3.3 Javaから入ることについて

私は私立大商学園高等学校(大阪)出身です。大商学園はしばしばAO入試で京産に合格する生徒がいますが、中でも私が所属していた普通科情報コースでは1週間に合計10時間近く情報にまつわる授業があり、プログラミングに関しては3年間で順番にJava, HTML, CSS, Scratch, PHPを学びます。中でもJavaは3年かけてやっとEclipseを使用してボタンをぽちぽちするだけのGUIプログラムを作る(アプリケーション化は教わらない所まで到達しますが、オブジェクト指向については愚か、ジェネリクスやメソッドやフィールドの意味についても触れられませんでした。自ら的好奇心で色々触っていると、System.out.printf()すら使用できないバージョンであり、当時の私としてはいろんな鬱憤が溜まったのをよく覚えています(笑)

大学に入って発展プログラミングまではほとんど授業に出席せず高い成績を維持できる技能の貯金はありました。C言語(およびオブジェクト指向プログラミングのパラダイムを持たない言語)に触れたことがなかったため、メモリの操作やポインタに関してはJavaの貯金で新たに勉強することとなりました。

## 3.4 TypeScriptのジェネリクスについて

先日、SNSでこのような記事を見つけました。どうやらTypeScriptはSQLでインターフェースを定義できるようです。これがどのようなメリットをもたらすのかという議論はありますが、こういった小技を発見するエンジニアはすごいなあと感心します。

<https://twitter.com/suin/status/1309289448270491650>

### 3.5 参考文献

Java SE AbstractMap

<https://docs.oracle.com/javase/jp/8/docs/api/java/util/AbstractMap.html>