

今週の一コマ画像



図0...推奨されない型に代入されたウイスキー

APJ07

学籍番号: 744366

氏名: 岸 典樹

2020年11月7日

第1章 ジェネリクス、抽象クラス、インターフェースなどについて	3
1.1 ジェネリクスについて	3
1.2 抽象クラス・インターフェースについて	4
1.3 配列リストを部分とするシーケンスからの継承で作成したスタックとキューのプログラム	4
第2章 列挙型について	12
2.1 列挙型について	12
2.2 サンプルプログラム	12
第3章 自由記述	14
3.1 改めて、Javaとは…を拝読させていただきました。	14
3.2 都会の大根	14
3.3 趣味で作りたいもの	14
3.4 参考文献	14

第1章 ジェネリクス、抽象クラス、インターフェースなどについて

本章ではジェネリクス、抽象クラス、インターフェース、それぞれが何であるのか、いつ具備されたのか、なぜ導入されたのか、などなどについて配列リストを部品とするシーケンスからの継承でスタックとキューのプログラムを作り上げ、論じる。

1.1 ジェネリクスについて

ジェネリクスはJDK1.5から導入された機能であり、型を変数にすることで型推論を可能とした。ジェネリクス関連のAPIはJava SEのjava.utilに内包されており、変数を定義する際は

“コレクション名<型> 変数名 = 値;”

と記述する。また、クラスやメソッドを定義する際には型にE, T, K, Vなどの存在しないクラス名を充てることができ、これを仮型変数と呼ぶ。さらに、? extends Objectなどのワイルドカード記法も可能なことからObjectクラスを継承しているすべてのクラスを受け入れる…などの記述も可能したことから、これまで以上に型については柔軟に対応できる。また、メソッドが豊富なため、コレクションの長さを変えたり削除した要素を追加したりコレクションの途中にデータを挿入したりするなど、複雑なポインタ操作を要求される処理が標準装備されているため、より柔軟なプログラムをお手軽に組むことができる。のちのJDK1.8からはラムダ式が導入され、ジェネリクスを含め、登場から18年でやっとオブジェクト指向プログラミングになれた。

=) ヨッキはビールを入れるもの 飲み物だったら何入れてもいいじゃない!



型指定

Jokki<Beer> drinks = new Jokki<>();



型推論・ワイルドカード

Jokki<? extends Drink> drinks = new Jokki<>();

図1.1.1 型指定と型推論・ワイルドカードの違いをお酒で例えるとこうなる

1.2 抽象クラス・インターフェースについて

抽象クラス・インターフェースはそれぞれJDK1.8から実装された機能である。

抽象クラスはJavaではabstract修飾子が使用されており、これを使用することでメソッドのオーバーライドを強制することができる。例えば、a, b, c, d, eと言うメソッドがあって、これらをX, Y, Zの3つのクラスで使用する場合、a, b, cは同じだがd, eの内容だけそれぞれのクラスで書き方が異なる場合がある。この場合にコードの記述を最適化するシステムが、抽象クラスである。インターフェースはおよそ構造体とよく似ており、どのようなフィールド、どのようなメソッドが用意されるのかを事前に定義し、ある規格を儲けるためのシステムである。例えば、先程のa, b, c, d, eがそれぞれウォーターフォールモデルの要件定義、外部設計、内部設計、プログラミング、テストを意味するメソッドであり、X, Y, Zがそれぞれ会社を表すとしたら、共通の設計書は定義されていてもプログラミングやテストの作業は各社独自の方法を取り入れていきたい場合がある。その際に、インターフェースは役立つのではないかと考える。

1.3 配列リストを部分とするシーケンスからの継承で作成したスタックとキューのプログラム

次頁以降に、作成したスケルトンから作成したプログラムを記載する。

```
Example.java Queue.java Stack.java
1 package example;
2
3 /**
4 * 例題プログラム:この例題を改変して大きなプログラムを作る足がかりにしてください。
5 */
6 public class Example extends Object{
7
8     /**
9      * 例題のメインプログラム。
10     * @param arguments コマンドの引数列(文字列の配列)
11     */
12     public static void main(String[] arguments){
13
14         Example.example1(); // Sequenceの例題プログラム
15         Example.example2(); // Stackの例題プログラム
16         Example.example3(); // Queueの例題プログラム
17
18         return;
19     }
20
21     /**
22      * Sequenceの例題プログラム。
23      */
24     public static void example1(){
25
26         String aString = Thread.currentThread().getStackTrace()[1].getMethodName();
27         System.out.printf("%s: ", aString);
28
29         Sequence<Integer> aSequence = new Sequence<Integer>();
30         aSequence.addFirst(100);
31         aSequence.addLast(200);
32         aSequence.addLast(300);
33         System.out.printf("%s%n", aSequence.getClass().getName());
34         System.out.printf("%s%n", aSequence);
35
36         Integer anInteger = null;
37         anInteger = aSequence.removeFirst();
38         System.out.printf("%s, removed: %s%n", aSequence, anInteger);
39         anInteger = aSequence.removeLast();
40         System.out.printf("%s, removed: %s%n", aSequence, anInteger);
41
42         return;
43     }
44 }
```

図1.3.1 Example.java①

```
45  */  
46  * Stackの例題プログラム。  
47  */  
48  public static void example2()  
49  {  
50  String aString = Thread.currentThread().getStackTrace()[1].getMethodName();  
51  System.out.printf("%s: ", aString );  
52  
53  Stack<Integer> aStack = new Stack<Integer>();  
54  aStack.push(100);  
55  aStack.push(200);  
56  aStack.push(300);  
57  System.out.printf("%s%n", aStack.getClass().getName());  
58  System.out.printf("%s%n", aStack.getOwnItems());  
59  
60  Integer anInteger = null;  
61  anInteger = aStack.pop();  
62  System.out.printf("%s, popped: %s%n", aStack.getOwnItems().toString(), anInteger);  
63  anInteger = aStack.pop();  
64  System.out.printf("%s, popped: %s%n", aStack.getOwnItems().toString(), anInteger);  
65  
66  return;  
67 }  
68  
69  */  
70  * Queueの例題プログラム。  
71  */  
72  public static void example3()  
73  {  
74  String aString = Thread.currentThread().getStackTrace()[1].getMethodName();  
75  System.out.printf("%s: ", aString );  
76  
77  Queue<Integer> aQueue = new Queue<Integer>();  
78  aQueue.enqueue(100);  
79  aQueue.enqueue(200);  
80  aQueue.enqueue(300);  
81  System.out.printf("%s%n", aQueue.getClass().getName());  
82  System.out.printf("%s%n", aQueue.getOwnItems());  
83  
84  Integer anInteger = null;  
85  anInteger = aQueue.dequeue();  
86  System.out.printf("%s, dequeued: %s%n", aQueue.getOwnItems().toString(), anInteger);  
87  anInteger = aQueue.dequeue();  
88  System.out.printf("%s, dequeued: %s%n", aQueue.getOwnItems().toString(), anInteger);  
89  
90  return;  
91 }  
92 }  
93
```

図1.3.2 Example.java②

```
1 package example;
2
3 /**
4 * スタック(積み上げ) : シーケンス(順序のある集まり)example.Sequenceを継承し、スタック(積み上げ)の機能(pussh
5 ュやポップ)を実現する。
6 */
7 public class Stack<Element> extends Sequence<Element> implements Iterable<Element> {
8
9     private Sequence<Element> contents;
10
11    /**
12     * スタックのコンストラクタ。
13     */
14    public Stack() {
15        this.contents = new Sequence<Element>();
16        return;
17    }
18
19    /**
20     * 自分自身(積み上げ)に指定された要素をpussh(追加)する。
21     * @param anElement 追加したい要素
22     */
23    public void push(Element anElement) {
24        this.contents.addLast(anElement);
25        System.out.println(">>" + this.contents);
26        return;
27    }
28
29    /**
30     * 自分自身(積み上げ)の最初の要素をpop(削除)し、pop(削除)した要素を応答する。
31     * @return 削除された要素
32     */
33    public Element pop() {
34        Element anRemovedElement = this.contents.last();
35        this.contents.removeLast();
36        return anRemovedElement;
37        // return null;
38    }
39
40    public Sequence<Element> getOwnItems() {
41        return this.contents;
42    }
43
44}
45
46
47
```

図1.3.3 Stack.java

```
1 package example;
2
3 /**
4 * キュー(待ち行列)：シーケンス(順序のある集まり)example.Sequenceを継承し、キュー(待ち行列)の機能(エンキュー  
やデキュー)を実現する。
5 */
6 public class Queue<Element> extends Sequence<Element> implements Iterable<Element>
7 {
8     private Sequence<Element> contents;
9
10    public Queue()
11    {
12        this.contents = new Sequence<Element>();
13        return;
14    }
15
16    /**
17     * 自分自身(待ち行列)から要素を取り出して応答する。
18     * @return 取り出した要素
19     */
20    public Element dequeue()
21    {
22        Element anRemovedElement = this.contents.first();
23        this.contents.removeFirst();
24        return anRemovedElement;
25        // return null;
26    }
27
28    /**
29     * 自分自身(待ち行列)に指定された要素を入れる。
30     * @param anElement 追加したい要素
31     */
32    public void enqueue(Element anElement)
33    {
34        this.contents.addLast(anElement);
35        return;
36    }
37    public Sequence<Element> getOwnItems() {
38        return this.contents;
39    }
40}
41
```

図1.3.4 Queue.java

```

1 package example;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5 import java.util.List;
6
7 /**
8 * シーケンス(順序のある集まり):配列リストjava.util.ArrayListのインスタンスを1つ集約(aggregation)して、イ
9 * ンスタンス変数contents(私的フィールドcontents)に保持し、シーケンス(順序のある集まり)の機能(たとえば、要素の
10 * 追加や削除や列挙など)をcontentsへ委譲(delegation)を用いて実現する。-
11 */
12 public class Sequence<Element> extends Object implements Iterable<Element> {
13
14     /**
15      * シーケンス(順序のある集まり)の要素を記憶するフィールド。要素はジェネリクス(Generics)を用いて型変
16      * 数Elementで表現する。-
17      */
18
19     protected ArrayList<Element> contents;
20
21     /**
22      * シーケンス(順序のある集まり)のコンストラクタ。-
23      */
24
25     public Sequence() {
26
27         /**
28          * 自分自身の最初に指定された要素を追加する。-
29          * @param anElement 追加したい要素
30          */
31         public void addFirst(Element anElement) {
32
33             // ちゃんと実装してください。-
34             this.contents.add(0, anElement);
35
36         }
37
38         /**
39          * 自分自身の最後に指定された要素を追加する。-
40          * @param anElement 追加したい要素
41          */
42         public void addLast(Element anElement) {
43
44             // ちゃんと実装してください。-
45             this.contents.add(anElement);
46
47         }
48
49         /**
50          * 自分自身の最初の要素を応答する。-
51          * @return 最初の要素
52          */
53         public Element first() {
54
55             // ちゃんと実装してください。-
56             return this.contents.get(0);
57
58         }
59

```

図1.3.5 Sequence.java①

```
60  > /**
61  > * @param index オリジンの添字indexで指定された要素を応答する。
62  > */
63  > * @return オリジンの添字Indexで指定された要素
64  > */
65  > public Element get(Integer index) {
66  > }
67  > // ちゃんと実装してください。
68  > return this.contents.get(index);
69  > // return null;
70  > }
71  >
72  > /**
73  > * 自分自身が空かどうかを応答する。
74  > */
75  > * @return 空かどうかの真偽
76  > */
77  > public boolean isEmpty() {
78  > }
79  > // ちゃんと実装してください。
80  > return this.contents.isEmpty();
81  > // return true;
82  >
83  > /**
84  > * 自分自身のイテレータを応答する。
85  > */
86  > * @return ArrayListのインスタンスのイテレータ
87  > */
88  > public Iterator<Element> iterator() {
89  > }
90  > return this.contents.iterator();
91  >
92  > /**
93  > * 自分自身の最後の要素を応答する。
94  > */
95  > * @return 最後の要素
96  > */
97  > public Element last() {
98  > }
99  > // ちゃんと実装してください。
100 > return this.contents.get(this.getLastIndex());
101 > // return null;
102 >
103 > private int getLastIndex() {
104 > }
105 > return this.contents.size() - 1;
106 >
107 > /**
108 > * 自分自身の最初の要素を削除し、削除した要素を応答する。
109 > */
110 > * @return 削除された要素
111 > */
112 > public Element removeFirst() {
113 > }
114 > // ちゃんと実装してください。
115 > Element aRemoveElement = this.first();
116 > this.contents.remove(0);
117 > return aRemoveElement;
118 > }
119 >
```

図1.3.6 Sequence.java②

```
120 >> /**
121 >> * 自分自身の最後の要素を削除し、削除した要素を応答する。-
122 >> * @return 削除された要素-
123 >> */
124 >> public Element removeLast()-
125 >> {
126 >>     // ちゃんと実装してください。-
127 >>     Element aRemoveElement = this.last();
128 >>     this.contents.remove(this.getLastIndex());
129 >>     return aRemoveElement;
130 >>     // return null;
131 >> }
132 >> /**
133 >> * 自分自身の要素数を応答する。-
134 >> * @return 群れオブジェクトの要素数-
135 >> */
136 >> public Integer size()-
137 >> {
138 >>     // ちゃんと実装してください。-
139 >>     int sizeAsAnInt = this.contents.size();
140 >>     return Integer.valueOf(sizeAsAnInt);
141 >>     // return Integer.valueOf(0);
142 >> }
143
144 >> /**
145 >> * 自分自身を文字列にして応答する。-
146 >> * @return 群れオブジェクトの要素を列挙した文字列-
147 >> */
148 >> @Override-
149 >> public String toString()-
150 >> {
151 >>     StringBuffer aBuffer = new StringBuffer();
152 >>     Class<?> aClass = this.getClass();
153 >>     aBuffer.append(aClass.getName());
154 >>     aBuffer.append("[");
155 >>     boolean firstTime = true;
156 >>     for (Element anElement : this)-
157 >>     {
158 >>         if (firstTime) { firstTime = false; } else { aBuffer.append(", "); }
159 >>         aBuffer.append(anElement.toString());
160 >>     }
161 >>     aBuffer.append("]");
162
163 >>     return aBuffer.toString();
164 >> }
165 >> }
```

図1.3.7 Sequence.java③

第2章 列挙型について

本章では列挙型が何であるのか、いつ具備されたのか、なぜ導入されてたのかなどについて、適切なプログラムを例示しながらまとめ、論じる。

2.1 列挙型について

列挙型はJDK1.5から導入された機能であり、型保証なし、名前空間なし、脆弱、出力値に情報価値がないなどの問題に回避するために生まれた。例えば以下のプログラム2.1.1の場合、enumを使用すればプログラム2.1.2のように書き換えることができる。

```
private static final int SEASON_WINTER = 0;
private static final int SEASON_SPRING = 1;
private static final int SEASON_SUMMER = 2;
private static final int SEASON_FALL = 3;
```

プログラム2.1.1 enumを使用しないプログラム

```
enum Season { WINTER, SPRING, SUMMER, FALL }
```

プログラム2.1.2 enumを使用したプログラム

2.2 サンプルプログラム

いかに、サンプルプログラムを例示する。このプログラムは、enumを使用して、呼び出された列挙子に対応するURLを出力する。

```

Example.java
1 package example;
2
3 /**
4 * MyUrlクラスをインポート
5 */
6 import static example.MyUrl.*;
7
8 /**
9 * 例題プログラム:APPLE, AOK, KSU、それぞれ呼び出したクラスに対応するURLを出力する
10 */
11 public class Example extends Object {
12
13     /**
14      * 例題のメインプログラム
15      * @param arguments 引数の文字列の配列
16      */
17     public static void main(String[] arguments) {
18
19         System.out.println("= Apple =====");
20         MyUrl aMyURL = MyUrl.APPLE;
21         System.out.println(aMyURL.getURL());
22
23         System.out.println("= AOK =====");
24         MyUrl aMyURL1 = MyUrl.AOK;
25         System.out.println(aMyURL1.getURL());
26
27         System.out.println("= KSU =====");
28         MyUrl aMyURL2 = MyUrl.KSU;
29         System.out.println(aMyURL2.getURL());
30
31     }
32 }

```

```

MyUrl.java
1 package example;
2
3 public enum MyUrl {
4     APPLE("Apple", "https://www.apple.com/jp/"),
5     AOK("青木先生", "http://www.cc.kyoto-su.ac.jp/~atsushi/index-j.html"),
6     KSU("京都産業大学", "http://www.kyoto-su.ac.jp/");
7
8     private String keyword;
9     private String url;
10
11     private MyUrl(String aKeyword, String anURL) {
12         this.keyword = aKeyword;
13         this.url = anURL;
14     }
15
16     public String getKeyword() {
17         return this.keyword;
18     }
19
20     public String getURL() {
21         return this.url;
22     }
23 }
24

```

図2.2.1 サンプルプログラム

第3章 自由記述

3.1 改めて、Javaとは…を拝読させていただきました。

レポートを進めるにあたりいくつか不明な点があったので、先生のホームページからJavaとは…を改めて拝読させていただきました。他のどの第一次資料よりもわかりやすくまとめられており、たった数枚のスライドでJavaをトップダウンで見下ろせるようになった気がします。ランチタイムトークを担当されている某インフラ系の先生もトップダウンで論理的に考えられているようですが、やはりその要素は大事で、例えば都会で道に迷った時、地面から景色を見渡しても周りのビルに遮られて今いる場所がどこかわかりません。少し高い位置、例えば東京タワーの上から見渡すことで、目的地やその周辺にあるものが広く見渡せるため、トップダウンで論理的に観察する力は今後身につけなければならないと痛感しました。新しく言語を学ぶ際もそうです。基本その方言はどうでもよく、パラダイムを理解し、言語の特徴を理解していれば、あとはドキュメントを参考に書くだけなので最小限の勉強法で適切なプログラムが書けるようになりますよね？と頭の中で言い聞かせています..

Javaを久しく触ってこなかったため、プロ言の総理大臣課題でもそのような方法を取り入れていこうと思います。この凝り固まった頭から枷を外さなければ…

3.2 都会の大根

今朝方(11/9)ニュースを見て驚きましたが、今大阪駅南東側では、阪神百貨店(大阪梅田ツインタワーズサウス:188m)の新築工事と地下道拡幅工事、阪神大阪梅田駅拡幅工事が行われていますが、その阪神百貨店の前に大根が生えていたそうです。なぜ大根？と言うのが本音ですが、辛抱強く頑張ったのでしょう。アスファルトの下にもn年、大根石を穿つ、大根の思いも天に届く…といったところでしょうか。いやはや、ミラクルです。

https://youtube.com/watch?v=yPE_ZRljcbk/

3.3 趣味で作りたいもの

先生のプログラムを拝読させていただいたことで、勉強も兼ねて、一人でライブラリを作りたいなと思いました。これがJavaなのかPythonなのか、はたまたTypeScriptなのかは不明ですが、自らが理解したオブジェクト指向を形にし、不足している分野を明確にしていき、就職後も周囲のエンジニアに比べオブジェクト指向について精通した人間になれるよう挑戦していきたいと思います。

3.4 参考文献

enumについて

<https://docs.oracle.com/javase/jp/8/docs/technotes/guides/language/enums.html>