わかりすぎる
Java8の教科書

参考情報

ＳＣＣ

Java８で新たに追加された機能の紹介

Java 8 は、2015 年 1 月時点で Java の最新バージョンとしてリリースされています。

「わかりすぎる Java8 の教科書」は、Java プログラムの入門書としての位置づけから、Java 8 で新たに追加された機能について必要に応じて紹介しており、新機能全てを紹介しているわけではありません。

そこで、すでに Java の基本文法について知識がある方へ向けて、Java 8 で新たに追加された機能の中から、重要と思われる機能を「参考情報」として簡単に紹介します。

また、Java の上級者には「JDK8 の新機能」というオラクルのリリースノートが以下のサイトに掲載されているので参考にしてください。

http://www.oracle.com/technetwork/jp/java/javase/overview/8-whats-new-2157071-ja.html

1. ラムダ式のサポート

「ラムダ式」については、本書の p.379 で簡単な解説があります。

ラムダ式を利用することで、これまで宣言、定義して明示的に呼び出さなければ利用できなかったメソッドを、非常に簡略化した文法で記述できるようになります。

また、ラムダ式で重要になるのが「関数型インターフェース」です。

関数型インターフェースとは､抽象メソッドを 1 つだけ持つインターフェースのことで、通常「@FunctionalInterface」を付けて宣言します。この FunctionalInterface アノテーションは、関数型インターフェースに対して必須のアノテーションではありませんが、コンパイラに関数型インターフェースであることを知らせることで、誤った記述を事前に警告することができます。

2. メソッド参照のサポート

「メソッド参照」という機能が、Java 8 ではサポートされています。メソッド参照とは、関数型インターフェース型の参照変数にメソッドそのものを代入することができる機能のことで、次の構文で代入するメソッドを指定します。

クラス名::メソッド名 インスタンス変数名::メソッド名

サンプルコード

```
@FunctionalInterface
interface Printer {
    public void myPrint(String name);
}

public class Test {
    public static void main(String[] args) {
    Printer p = Test::mainPrint; // メソッドを Printer 型変数に代入
        printName(p);           // メソッド参照を引数として渡す
    }
    static void printName(Printer p) {
        p.myPrint("参照太郎");   // メソッド参照を使い呼び出す
    }
    // メソッド参照の参照先メソッド
    static void mainPrint(String message) {
        System.out.println(message);
    }
}
```

実行結果

```
参照太郎
```

3. 仮想拡張メソッドのサポート

仮想拡張メソッドとは、インターフェースに追加できるようになった default メソッドや static メソッドのことです。本書では、p.282 に簡単な解説があります。

仮想拡張メソッドが Java に導入されたことで、オーバーライドをする必要がなくなります。その結果、インターフェースという制限こそありますが、本来 Java ではできなかった「多重継承」と似た継承関係を実現することができます。

サンプルコード

```
interface MyInterface {
    default void temp1() {
        System.out.println("default メソッド");
    }
    static void temp2() {
        System.out.println("static メソッド");
    }
}

class MyClass1 implements MyInterface { // オーバーライドなし
}

class MyClass2 implements MyInterface {// オーバーライドあり
    @Override
    public void temp1() {
        System.out.println("Override メソッド");
    }
}

public class Test {
    public static void main(String[] args) {
        MyInterface o = new MyClass1();
        o.temp1();           // オーバーライドなし
        o = new MyClass2();
        o.temp1();           // オーバーライドあり
        MyInterface.temp2();  // static メソッドの呼び出し
    }
}
```

実行結果

```
default メソッド
Override メソッド
static メソッド
```

4. Stream API

「Stream API」とは、「内部イテレータ」を実現する API です。

「イテレータ」とは、コレクションオブジェクトに安全にアクセスするためのデザインパターンのことで、Java では Iterator インターフェースにより実装され、拡張 for 文はこの Iterator インターフェースを利用してコレクションの要素にアクセスしています。

Stream API では、この機能を API としてメソッド内に閉じ込めてしまい、for 文を使わずにコレクションの要素にアクセスできるようにします。

Stream API を使うには、Stream のインスタンスを生成します。例えば、以下のコードは、配列(コレクションオブジェクト)から Stream インスタンスを生成して、for 文を使わずに forEach メソッドとラムダ式で出力するコードです。

サンプルコード

```
import java.util.Arrays;
import java.util.stream.Stream;

public class Test {
    public static void main(String[] args) {
        String[] array = { "春", "夏", "秋", "冬" };
        Stream<String> stream = Arrays.stream(array);
        stream.forEach(value -> System.out.println("array:" + value));
    }
}
```

実行結果

```
array:春
array:夏
array:秋
array:冬
```

5. 日時 API

Java 8 には、新しい日時処理の API が追加されました。
これまでの、Java の Date クラスや Time クラスは、日付や時刻を指定したインスタンスが生成できない、または計算ができないといった不便なものでした。そこで Java 8 では、新たに「java.time パッケージ」が導入されました。

java.time パッケージには、以下のクラスが用意されています。

LocalDate	日付（タイムゾーンを考慮しない）
LocalDateTime	日時（タイムゾーンを考慮しない）
LocalTime	時刻（タイムゾーンを考慮しない）
OffsetDateTime	日時（UTC からの時差）
OffsetTime	時刻（UTC からの時差）
ZonedDateTime	日時（タイムゾーンを持つ）
Duration	時間の量
Period	日付の量

6. 新しい JavaScript エンジン

そもそも、JavaScrip は Web ページに直接記述できる簡易言語として発達してきました。しかし近年では、この JavaScrip を Web ページ以外の環境から利用する仕組みが整いつつあります。このような背景から Java 8 では、新しい JavaScript エンジン「Nashorn（ナスホーン）」を搭載しました。

これまでの Java には、Mozilla ファウンデーションによる「Rhino」という JavaScript エンジンが搭載されていました。対して、Java 8 の「Nashorn」はオラクル製であり、すべてが Java で書かれています。そのため、これまでよりもさらに Java の環境から JavaScript が利用しやすくなりました。

ちなみに、「Rhino」は英語の「サイ」を意味しており、Nashorn（ドイツ語の「サイ」）の名前の由来になっています。

以　上