拡張for文

2018年12月13日 木曜日　　11:09

拡張for文が使えるケースでは、特に理由がない限り、イテレータを使う必要はなくなった。

コレクションを順に処理するforループの中で要素の追加や削除を行うことはできない。
要素を削除すると、UnsupportedOperationExceptionが発生。
for文の中では要素は原則として参照することができるだけと考える。

For(int I = 0, j = 0; i<10 ; i++, j++)
　　初期化や繰り返しのたびに評価される式は複数記述可能

For(int I = 0, long j = 0; i<10 ; i++, j++)
　　型が違うとエラー。予めフィールドで型宣言を行うとエラーにはならない

For(String a, b : dogs, cats)
　　拡張for文では、複数の値を指定できない。

コンストラクタ

2018年12月13日 木曜日　　11:44

引数のあるコンストラクタを定義すると、引数のないデフォルトコンストラクタが消滅するので、引数のないコンストラクタも必要なら、自分で定義する必要がある。

複数コンストラクタを定義する際は同じ処理を重複して記述せずに、thisを用いる。
コンストラクタをいくつ定義しても、フィールドに値をセットするコードは最後のコンストラクタ1回だけしか記述しない。

```
Hoge(int a){
    this(int a, 任意の値(初期値など));
}
Hoge(int a, intb){
    this.a = a;
    this.b = b;
}
```

アノテーション

2018年12月13日 木曜日　　12:40

アノテーションはプログラムそのもには影響を及ぼさず、コンパイラや実行環境によるプログラムの解釈に影響を及ぼすメタデータ。
コンパイラでエラーや警告メッセージを出力・抑制、外部ファイルとのリンクを確立したり、実行環境によってプログラムの動作を変更したりすることができる。
独自のアノテーションを定義することもできる。
オーバーライドするメソッドは基本的に@Overrideをつける。

null

2018年12月13日 木曜日　　14:19

CやC++ではnullの値は0として定義されているがJavaでは何も参照されていない状態を指す
そのため、オブジェクトとして評価しようとすると実行時に例外が発生する。
Java8ではnullチェックに、Objects.isNull()もしくはObjects.nonNull()を使用可能。

Javaでは、名前の最後にsが付くものは、sがつかないクラスやインタフェースに関連するユーティリティメソッドを集めたクラス。

nullになる可能性がある値を取り扱う時には、Java8で導入されたOptionalクラスを活用するのが良い

boolean型の変数は、必ずtrueかfalseです。一方、Booleanクラスはtrue、false以外にもnullになることがあるので、注意する。

BigDecimal

2018年12月13日 木曜日　　14:27

金額の計算によく使われる。注意点として、値がnullになることがある。特に初期化していない場合は、値がゼロでなくnullになる。また、BigDecimalはイミュータブルであることも注意。

あ

定数とenum

2018年12月13日 木曜日　　14:32

enumは列挙型とも呼ぶ特別なクラス。明確な理由がない限り、定数として使う値もenum型として定義する方が一貫性がある。

```
enum 名前 { メンバーリスト }
enum Hoge {
    A,
    B,
    C
}
Hoge.A

enum COLOR{
    RED(0xFF0000), GREEN(0xFF00), BLUE(0xFF), BLACK(0);
    private final int value;
    COLOR(int value) { this.value = value}
    public int value() { return value; }
}
```

enumは特殊なクラスであり、扱い次第では可読性が下がるので、基本的には定数定義に限って使うべきで、インタフェースやクラスの代用にするべきではない。
また、実行時に一つのインスタンスが作成されて共用されるので、フィールドも共用される。

enumに関して、java.util.EnumSetとjava.util.EnumMapクラスがある。

日付と時間

2018年12月13日 木曜日　　15:00

nanoTime()が追加されて、Java仮装マシンの高精度時間ソースの現在値をナノ秒の単位で取得できるようになった。

Optionalクラス

2018年12月13日 木曜日　　17:51

Optionalクラスはnullになる可能性がある値を保存するときに使う

```
Optional<String> os = optional.ofNullable(str);
int len = os.orElse(“”).length();
os.ifPresent(System.out::printing);
```

ラムダ式

2018年12月14日 金曜日　　9:43

{args} -> { prog}

args は引数、progは実行するコード。引数は複数でも良い。引数が1個の場合は()を省略可能。その場合は引数の型も省略しなければならない。
実行するコードも複数で良く、ステートメントが1つだけの時にはコードを囲む{}を省略可能。

匿名クラスは殆どの場合、容易にラムダ式に書き換えることができる。匿名クラスの中でthisを参照すると、その匿名クラスのインスタンスを指す。ラムダ式は式なので、thisを参照するとその式が含まれているクラスのインスタンスを指す。

メソッド参照

2018年12月14日 金曜日　　9:59

メソッド参照は他のメソッドに渡してメソッドを実行することができる。

class::method

instance::method

this::method

e.g. hoge.forEach(System.out::Pringle)

Stream

2018年12月14日 金曜日　　10:05

ストリームは、ある処理の出力を次の入力によって処理をつなげる技術
メソッド参照やラムダ式は、Streamと共に頻繁に使われる。

ストリーム処理は、parallelStream()を使うだけで、可能であれば各ストリーム処理が並列で実行されるようになる。そのため、各処理が終わってから次の処理を行う場合よりも早く処理全体を完了できることが期待できる。

殆どの場合、メソッド参照が使える場面ではラムダ式を使うことができる。
どちらが良いかは、可読性や慣例に従って決めるので、技術的に優劣はない。

コレクション

2018年12月12日 水曜日　10:27

複数のデータをまとめて扱うためのクラスの総称をコレクションと呼ぶ。
格納できるのは、参照型のみで基本データ型は格納できない。
基本データ型を扱うには、ラッパークラスを使用する。配列も格納可能。
要素が参照型の場合は、格納されるのはオブジェクトへの参照(アドレス)。

コレクションインタフェース

Collection-List	要素をインデックスと関連づけて管理するため、同じ要素を重複登録可能
-Set	登録されたエレメントを一意に識別するため、同じ要素の重複不可
Map	要素をキー値とペアとして管理する。キーにはオブジェクトを用いる。 要素の重複登録は可能だが、キーの重複は不許可。

List	順序	ソート状態	同期
ArrayList	インデックス	不可	
Vector	インデックス	不可	○
LinkedList	インデックス	不可	
Set			
HashSet	なし	不可	
LinkedHashSet	挿入順	不可	
TreeSet	ソート済み	自然順序/独自の比較規則	
Map			
HashMap	なし	不可	
Hashtable	なし	不可	○
TreeMap	ソート済み	自然順序/独自の比較規則	
LinkedHashMap	挿入順	不可	

コレクションで型を指定しないと、警告が出る。実行自体は可能。
@SuppressWarnings(“unchecked”)を使用するとコンパイル時のエラーを回避できる。

Collections.unmodifiableList(hoge)
他のプログラマが誤って変更しないようししたい時に役立つ

Listインタフェース

2018年12月12日 水曜日　10:37

データ間に明らかな前後関係が存在する場合はListを利用する。

ArrayList

サイズが拡張できる配列のようなもの。参照の処理速度を要求されるときに適している

ソートはインデックスによる順次付きコレクションにかかわらず実装されていない。
挿入や、削除などの変更処理が多い場合はLinkedListの方が向いている。

要素としてnullを追加できる。

マルチスレッド環境で、複数のスレッドが同時のコレクションの内容を変更した時に、

内容は保障されないので、プログラマが同期を行う必要がある。
同期を行わない点を除いてVectorとほぼ同じ機能を提供。

ArrayListは同期を行わない分高速

LinkedList

要素同士が双方向にリンク(次要素と前要素の参照を保持)した構造になっている

要素の追加や削除の処理の際に、関連する前後の要素の参照を組み替えるだけで済むの

で、追加や削除の処理が多い処理に適している。

LinkedListはスタックやキューの実装に向いている。

Vector

基本的な機能はArrayListと同じだが、Vectorの一連のメソッドはスレッドセーフ(複数の

スレッドから呼ばれても問題が生じない作りになっている)のための同期化がされている
そのため、複数のスレッドから同時にアクセスできない。同期化すると実行速度が低下するの

で、特別な理由がない限りArrayListを使用した方が良い。

Vectorは同期化されること以外はArrayListとほぼ同等。

ArrayListは最後に要素を追加したり最後の要素を削除するのは容易だが、途中に要素を挿入したり削除したりするには、それ以降の要素を全て移動してずらさなければならないので、時間がかかることがある。

LinkedListは要素間をリンクでつないでるので、途中の要素を挿入したり削除するときは、単にリンクをつなぎかえるだけなので速い速度で実行できる。

Setインタフェース

2018年12月12日 水曜日　　10:38

コレクションに重複する値を格納できないという特別な条件がある場合はSetを使用
重複不可。重複の判定にはequals()によって判定される。
null値を格納可能だが、1つのセットで1つだけnull値が強化される。

HashSet

ソートできないセット。オブジェクトの格納時にハッシュコード(オブジェクトを数値で表現したもの)を使用し、同じオブジェクト仮装でないかを判断するのでハッシュコードを生成するhashCode()の効率を高めれば、要素へのアクセス速度も向上する。要素の順序は一定には保たれず、同期化も行わない。
nullを要素として追加可能。重複を認めない場合はHashSetを使用する。

LinkedList

HashSetに双方向のリンク機能をつけたもの
要素を順序づけして、取り出したい時に適している。
この順序は要素がセットに挿入された順序になる(挿入順)

TreeSet

要素の昇順でソートされる。順序は自然順序(アルファベット、数値順など)になる
独自の比較規則を設定することも可能。同期化は行われない。

Setインタフェースは重複項目を持たないので、equals()で比較してもtrueが帰る要素ペアを持たない。

Mapインタフェース

2018年12月12日 水曜日　　10:40

任意のキー値を使って格納される要素の参照を行う場合はMapを使用
キー値は、マップに格納されている要素を探すための目印になるオブジェクト
Mapでは要素に一意にのキー値を割り当てするが、キーと値はどちらもオブジェクト。

キーがマップのどの場所に格納されているかはキー値のハッシュコードに左右されるので、ハッシュコードを取得できるhashCode()の効率を高めれば、要素へのアクセス速度も向上する。HashSetも同様。
キー値はマップの中で一意の値でなくてはならない。

HashMap

シンプルなマップで順序を保持せず、同期化も行わない。キー値としてnullをひとつだけ持つことができる。重複キーは不許可。値のnullは重複可

HashTable

HashMapの機能を同期化したもの。キー値にnullは認められていない。

LinkedHashMap

LinkedHashSetのようにHashMapに双方向のリンク機能をつけたもの。双方向リンクの処理を行うために追加や削除の速度は多少遅くなるが、その分読み取りの処理速度は向上する。

TreeMap

MapのサブインタフェースであるsortedSetの機能を実装しているので、要素の自然順序に従ってソートされる。重複したキーは不許可で、同期化も行われない。TreeSetと同じくオブジェクトの作成時にコンストラクタで絵独自の比較規則を渡すことが可能。

数とオブジェクトの比較

2018年12月14日 金曜日　　11:03

整数値を比較するときは==で大丈夫だが実数値の場合は注意する。
double x = 5.6;
double y = 7.0 * 0.8; //5.600000000005　　⇦　極小の誤差が生じる。
x == y はfalseになる。これは2進数に変換された際に極小の誤差が生じるから。
このような場合は、差の絶対値を計算して、その値が極めて小さいかどうかで判断する
e.g. Math.abs(x - y) < 0.00001

オブジェクトの内容が同一か明示するために、contentEquals()を使うと明確になる。
異なるクラスの内容が同じであることを示す時に使う。

オブジェクトを比較する時に、Javaの==はオブジェクトが同じか調べ、equals()はオブジェクトの値が同じであるかを調べる。
数値オブジェクトの大小を比較する時にはcompareTo()を使う

クラス設計

2018年12月14日 金曜日　　11:40

内部クラスはあるクラスの内部にネストしたクラス。普通のクラスとして記述可能
内部クラスのオブジェクトを直接作るとエラーになる。
内部クラスのオブジェクトを作成するには、メソッドを追加して、エンクロージングクラスのインスタンスを作成する必要がある。

インタフェース

2018年12月14日 金曜日　　11:52

defaultメソッドはキーワードdefaultを宣言しそのインタフェースに属するオブジェクトに共通する動作を記述する。
e.g. Dogインタフェースなら犬に共通する吠えるという名前のdefaultメソッドを定義する。
defaultメソッドはインスタンスフィールドを持つことができない。
defaultメソッドはインスタンスメソッドなので、他のインスタンスメソッドを呼び出すことができる。

defaultメソッドやstaticメソッドを除いた、インタフェースのメソッドは暗黙的にpublic abstractになる。
インタフェースのフィールドは暗黙でpublic static final になる。
これらは変更できないので省略可能。

正規表現

2018年12月14日 金曜日　　12:11

Javaで正規表現を利用するときはPatternとMatcherクラスを使用する
Patternは正規表現を表すオブジェクト、Matcherはパターンに一致するか検索するクラス

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

String str = “あいうえお”;
String ptn = “あ[たちつてと]”; //「あ」の次の文字が「た〜と」であるかチェック
Pattern p = Pattern.compile(pin);
Matcher m = p.matcher(str);
if(m.find){ System.out.println(“Matched”);}else{ }
```

パターンにマッチした文字列の置換は、Matcherクラスのメソッドを使って行うことができる
String replaceAll()　　パターンとマッチする部分を全て引数に置き換える
String replaceFirst()　　パターンとマッチする最初の部分を引数に置き換える

最初にマッチした部分を参照したい時にはその部分の正規表現を()で囲んで、置換の際にその部分を参照する時には$1を使用。

```
String ptn = “私([のはをに])”;
String rep = “僕$1”;
```

2番目にマッチした部分を参照したいときは＄2を使い、以降3,4,,,nと続く。
＄0はマッチした文字列全体を表す。この場合はパターン全体を表すので正規表現パターンを()で囲む必要はない。
最初にマッチしたものだけ置換したいときは、Matcher.replaceFirst()を使う。

正規表現パターンをコンパイル際に、正規表現が無効またはフラグが適切でない場合には例外が発生する。
IllegalArgumentException 定義済みマッチフラグに対応するビット値以外の値がflgに設定
PatternSyntaxException　正規表現の構文が無効

例外処理の際は各々の例外をインポートする必要がある。
また、$1とするべき箇所を$2などにすると、IndexOutOfBoundsExceptionが発生