Kotlin の概要

・Kotlin ってなんだろう？

2011 年 7 月、カリフォルニアで開かれた JVM Language Summit というイベント中で初めて登場した。

↓

発表したのは、『IntelliJ IDEA』などの IDE（Integrated Development Environment:統合開発環境）製品の開発元であった JetBrain 社

・なぜ今 Kotlin を使うのか？

現在は、プログラミング言語がたくさん！

その中でも Java は有力な選択肢となっている。

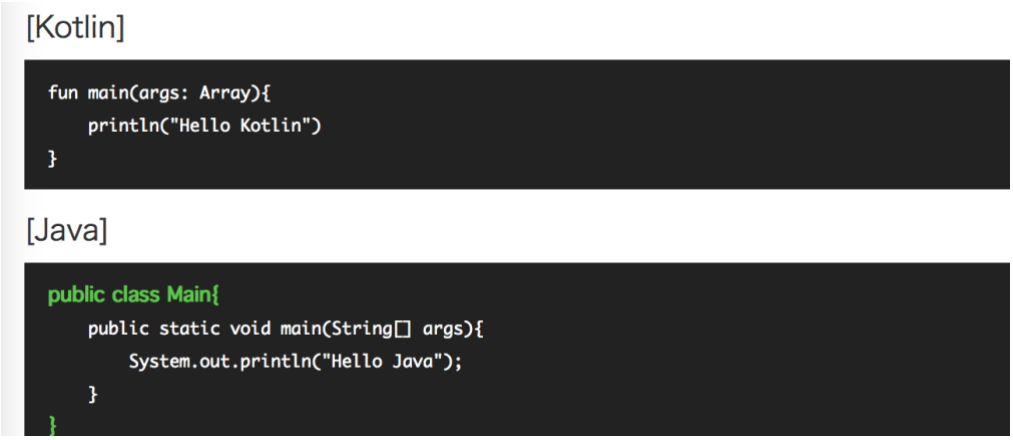
Java の問題は、記述の単調さ、型安全性の問題、後方互換の維持などがある。

Kotlin では多くの概念は Java と同じ。

Java とどう違うのか？

Main メソッドはクラスの外

Java では main メソッドは public なクラスの static メンバとして書く必要があったが、Kotlin では、いきなりメインメソッドを書くことができる。



セミコロンレス

Java では文末のセミコロンが抜けるといちいち怒られるが、Kotlin では改行するだけで文末とみなされる。一行に複数の文を書きたい場合はセミコロンがいる。

基本データ型（プリミティブ型）は無し

Java では int や float や char などの基本データ型が用意されているが Kotlin ではない。その代わりに扱えるクラス（オブジェクト）が用意されている。

型	説明
Double	64ビット浮動小数点数
Float	32ビット浮動小数点数
Long	64ビット符号付き整数
Int	32ビット符号付き整数
Short	16ビット符号付き整数
Byte	8ビット符号付き整数
Char	１文字を表す文字型
Boolean	真偽値(trueまたはfalse)
String	文字列

それぞれ頭文字は大文字なので注意。

型の宣言不要

Kotlin は型推論が備わっており、変数の宣言時に初期化（値の代入）を行う場合は型を書く必要はない。
その代わり、変数の宣言時には var か val をつける必要がある。Var は普通の変数、val は定数となり再代入不可になる。（java の final と同じ）

[Kotlin]

```
var num1 = 0 //変数
val key = "abcd" //定数

key = "efgh" //定数に再代入はコンパイルエラー
var num2 //その場で初期化しない場合は型を明示しないとコンパイルエラー
```

関数（メソッド）の定義の仕方

関数クラスのメンバである必要はなく、トップレベルで宣言することができる。関数に宣言する際には fun を頭につける。仮引数は型を明示する必要がある。戻り値がある場合は（）の後ろにコロンをつけて書き足す。戻り値がない場合は何も書かない。

[Kotlin]

```kotlin
fun plus(num1: Int, num2: Int): Int{
    return num1 + num2
}
fun main(args: Array<String>){
    println(plus(1, 2))
}

//=> 3
```

[Java]

```java
class Main{
    public static voi main(String[] args){
        System.out.println(plus(1, 2));
    }
    static int plus(int num1, int num2){
        return num1 + num2;
    }
}

//=> 3
```

引数のデフォルト値を設定できる

Kotlin では引数のデフォルト値を設定することができる。

[Kotlin]

```kotlin
fun hello(name: String, language: String = "en"){
    //引数のlanguageによって言語を変えて、挨拶する処理
}
fun main(args: Array) {
    hello("田中", "ja")  //普通に引数を2つ渡して呼ぶ
    hello("Alex")        //第二引数を省略してデフォルト引数を使う
    hello("Michel", language="fr") //引数名を指定して渡すこともできる
    hello(language="es", name="Alonso") //引数名を指定する場合、渡す順番は自由
}
```

配列は Array オブジェクト

Kotlin ではいわゆる配列はない。その代わりに Array クラスを使う。

Array オブジェクトを作る際には、arrayOf メソッドを使う。要素にアクセスする際には普通に[]でインデックスを指定することができる。

[Kotlin]

```
var array = arrayOf(1, 2, 3)
array[0] = 5
println(array[0])

//=> 5
```

[Java]

```
int[] array = new int[]{1, 2, 3};
array[0] = 5
System.out.println(array[0]);

//=> 5
```

コネクションは読み込み専用＆書き込み可

Kotlin のコネクションは List,Map,Set の 3 つクラスが用意される。それぞれ基本的には読み取り専用になっていて、要素を一度入れたら書き換えることができない。(値を変更するメソッドが用意されない）書き込み（要素の変更）もしたい場合はそれぞれ MutableList、MutableMap、MutableSet クラスを使う必要があります。

New は無し、コンストラクタは init

インスタンス化する際の new は要らない。コンストラクタは、init{}で定義す

る。コンストラクタに渡したい引数はクラス名の後ろに（）をつけてそこにかく。

[Kotlin]

```kotlin
fun main(args: Array) {
    var human = Human("Nobuo")
    println(human.name)
}
class Human(name: String){
    val name: String
    init{
        this.name = name
    }
}
```

[Java]

```java
public class Main{
    public static void main(String[] args) {
        Human human = new Human("Nobuo");
        System.out.println(human.name);
    }
}
class Human{
    final String name;
    Human(String name){
        this.name = name;
    }
}
```

プロパティは自動でセット可能

コンストラクタに渡した値をそのままプロパティに代入したいようなパターンが多くあると思う。上の書いたプログラムもそうである。コンストラクタが受け取るべき仮引数を定義する際に、var か val を書いて宣言することで、その変数をそのクラスのプロパティ（インスタンスフィールド）として宣言したこと

になり、しかも、そのプロパティへ代入する処理をコンストラクタに書かなくても良い。

[Kotlin]

```
class Human(val name: String){
    //コンストラクタ無し
}
fun main(args: Array) {
    var human = Human("Nobuo")
    println(human.name)
}

//=> Nobuo
```

文字列の中に変数を入れる

ダブルクォーテーションで囲った文字列リテラルの中に変数を放り込むことができる。変数名の頭に＄をつけて放り込むだけでできる。

[Kotlin]

```
num = 3
print("好きな数字は$num")

#=> 好きな数字は3
```

ただし、この場合、変数名の後ろに文字が続く場合境目をうまく区切ってくれないことがある。

[Kotlin]

```
var num = 3
print("好きな数字は$numです。")

#=> コンパイルエラー Unresolved reference: numです
```

$ {}を使えば明示的に変数部分を指定できる。

[Kotlin]

```
var num = 3
print("好きな数字は${num}です。")

#=> 好きな数字は3です。
```

・JVM について

Java が動作する仮想マシン JVM（**Java Virtual Machine**）他に「Java 仮想マシン

」、「JavaVM」ということもある。

JVM は Java を動かすために必要なソフトウェアであって、高性能で高信頼性を備えていて、非常に魅力的である。Java のための資産として、ライブラリーやフレームワーク、ツールなどが豊富！Kotlin プログラムは、JVM や Java 用のライブラリ、フレームワークといった資産をそのまま使える。

さらに「Kotlin と Java の相互運用性は 100%」100%ということは Java で記述されたプログラムを、Kotlin で記述したプログラムから使用することと、その逆が可能である。

Kotlin は Java よりもシンプルかつ安全に設計されている。

なぜ安全なのか？

・Kotlin には、よくあるプログラミングミスを未然に防ぐための仕組みが備わっている。

・型や null の扱いが厳格である。キャストや null のデリファレンスによる実行事例外が起こることはまれ。特に null にまつわる安全確保の仕組みを「Null 安全」という。

Null 安全とは、Java が `NullPointerException` が出た時に、kotlin の時に実行エラーになるかもしれないコードを kotlin コンパイラが検出して、コンパイル時に教えてくれる。エラーの検出は遅れれば遅れるほど対処が困難になる。実行エラーはテストで検出できなければ、潜在バグとしてプロダクトを埋め込まれてリリースされてしまうかもしれない。コンパイルエラーであれば解消されないままリリースされるということはない。

この Null 安全は実際にプログラムを書く時に理解できると思う。

参考文献：Java プログラマが Kotlin でつまづきがちなところ

https://qiita.com/koher/items/d9411a00986f14683a3f

静的型付けについて

Kotlin ソースコードは Java バイトコードに変換（コンパイル）されます。コンパイラは、コンパイル時にソースコートの誤りを発見すると、Java バイトコードを生成しない。そのため、プログラマはバグを早い段階で発見でき、安全あプログラムを作ることができる。

オブジェクト指向について

Kotlin のクラスベースはオブジェクト指向言語である。Java のように、定義されたクラスからインスタンスを生成することができる。Java とは違って、Kotlin にはプリミティブ型（int や char など）はなく、すべてがオブジェクトであり、一貫した扱い方が可能！またプロパティやオブジェクト宣言、拡張関数など、Java にはない便利な機能がある。これは実際にプログラムを書いて体験してみる。

オブジェクト指向とは

・モノとして考える。

・現実世界と一緒

・オブジェクト指向は「概念」だ！概念と言う言葉自体難しいが物事や対象を

丸ごとひっくる見たときの大まかな理解のこと

・オブジェクト指向は対象を操作するイメージ

・オブジェクト指向プログラミング（Object Oriented Programming: OOP）

とは、プログラムを手順ではなくて、モノの作成と操作として考える。オブジ

ェクトとは「モノ」と言う意味

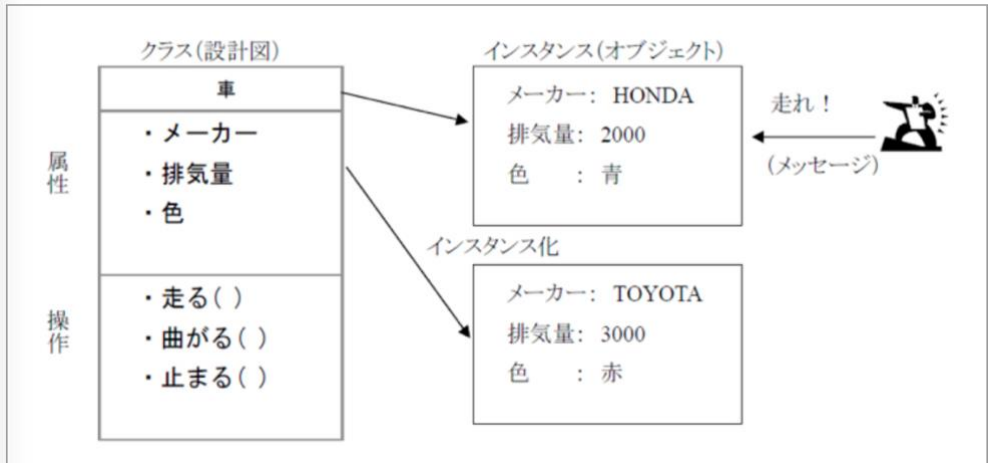
テレビはリモコンで操作するというイメージ
・大変な作業をなくすことができる。

・大人数で開発するときに便利である。モノを用意して、それを他の人が触れないようにしておけば、他の人がプログラムを壊してしまう心配がなくなる。
・同じようなものを作りやすい。

オブジェクト指向の基本用語

オブジェクト（Object）

オブジェクトは、オブジェクト指向の根本
オブジェクトとは、「対象」「物」という意味で、プログラミングにおいてはデータと処理の集まりを意味している。オブジェクト指向で現実のものを例えるとPCやスマホもオブジェクトである。



クラス（Class）

クラスとはオブジェクトの設計書のようなもの。オブジェクトの中のプロパティやメソッドをひとまとめにしたもの。

プロパティ（Property）

オブジェクトが持っているデータのことをプロパティ（属性）という。車の例えだと、車というオブジェクトは「メーカー」「排気量」「色」といったプロパティを持っていると言える。

メソッド（method）

メソッド（操作）とは、オブジェクトが持っている処理のことで、9R 魔の例だと「走る」「曲がる」など、オブジェクトから何かしらのアクションを起こす処理のこと。

インスタンス化（instance）

インスタンスとは「実体」「事例」という意味で、プログラムでオブジェクトを実際に生み出されるものである。設計図からオブジェクトを作ることをインスタンス化と呼ぶ。上が Java、下が Kotlin

```
クラス名 参照変数 = new クラス名();
```

```
c1 = new Car();
```

インスタンス生成

コンストラクタは関数を呼ぶようにして使える。*new* キーワードはいらない。

```
val invoice = Invoice()
val customer = Customer("Joe Smith")
```

カプセル化

オブジェクトが持つデータや処理のうち、別のオブジェクトから直接利用される必要のないものをいう。利用する場合は外部から操作するために作られた処理を設けることをいう。

継承

特定のオブジェクトの機能を引き継いで使うことを継承という。例、車からトラックをいう。

ポリモーフィズム

クラスによって同一のメソッドで異なる処理が行えるという性質。

コンストラクタ

コンストラクタとは、クラスからオブジェクトを作成した際に、自動的に実行されるメソッドのことで、メンバ変数の初期化などの主に行う。

オブジェクト指向の学び方

既存のコードを改善しながらオブジェクト指向設計を学ぶ
やや極端なコーディング規則を使って、オブジェクト指向らしい設計を体で覚える。
どうすればいいのか？
実際のコードで設計の違いを知る。・・・やりにくいコードを小さなメソッドや小さなクラスにまとめてみる。重複したコードをメソッドに抽出する。重複した箇所を、抽出したメソッドを呼び出すように書き換える。抽出したメソッドをどちらかのクラスだけに置き、他のクラスからそのメソッドを呼び出す。

抽出したメソッドを置くために新しいクラスを作成する。

関数型プログラミングについて

Kotlin には第 1 級オブジェクトとしての関数がある。つまり、関数を数値や文字列など他の値のように関数の引数として渡したり、戻り値として受け取ったりすることが可能である。これによって、より粒度の小さい単位で再利用が可能になり、抽象的なプログラミングが可能になる。

このことは、Kotlin の簡潔さを実現している仕組みの一つ。これは関数型プログラミングの一要素に過ぎないから、Kotlin は関数型言語ではない。

実際にどこで使われているか？

国内だとサイバーエージェント社の映像視聴 Android アプリ「FRESH!by AbemaTV」や Sansan 社の名刺アプリ「Eight」の Android 版などは Kotlin で開発されている。