最速でJavaエンジニアになって稼ぐ

2018/10/24

人生逃げ切りオンラインサロン内

目次的なやつ

- Javaの基本的な考え方とか動きとか
- オブジェクト指向って何？
- 型の種類と使い方
- クラス・フィールド・メソッド・アクセス修飾子とか。
- インターフェースと実装。抽象クラス。継承。
- オーバーライド・オーバーロード
- 例外処理
- コンパイルとビルド

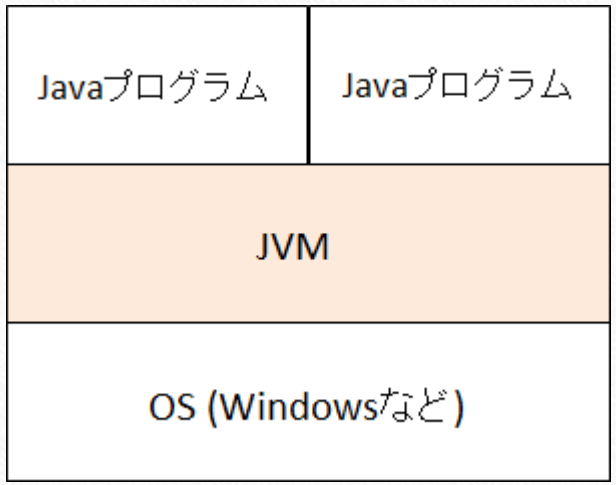
Javaの基本的な考え方とか動きとか

- JavaはJVM(JavaVirtualMachine)上で動く。
- このJVMがめちゃくちゃ賢い。凄い。なんか凄い。
- でもJVMの中身自体は知らなくていい。何してくれてるのかだけを知っていれば問題なし。

- JVMは「Javaの実行環境」。JVMなしでJavaが動くことはない。その為、前回話したJREと呼ばれるJavaの実行環境のセットの中には必ずJVMが付属している。Javaインストール時にもJVMがインストールされている。

- JVMは「Linux」「Windows」「Mac」といったほぼ全てのOS上で動作する。しかもこれらのOSは現在では携帯端末のOSにもなっていたり、レジや案内板といった様々な電子機器のOSにもなっている。
- つまり「Javaはあらゆるデバイス上で動作する」という利点があるという事。
- 最近流行りの言語の多くは「ブラウザ上で動作する言語」だったりするので、あらゆるデバイス上で動作する(ブラウザとか必要じゃない)言語というのは業務でもやはり強い。

- こんなイメージになる(ざっくり



- JVMはしかも「実行する」だけではなくて「管理」も行うので、プログラムの動作保証みたいなものも一部行っている。

- 管理対象としては「プログラムコード変換」「メモリ管理」「スレッド管理」が主になる。
- コード変換は「コンパイル」の事。Javaにおけるコンパイルは二段階あり、Java→JVM→OSといった流れを踏む。コードの変換が2回行われる。
- メモリ管理は「プログラムを実行する際のメモリ領域の自動確保、自動削除」を行ってくれる。ここが特に優秀でメモリ管理を自前でしなくて良いというのがとても楽。「GC」が優秀。
- スレッド管理は「多重実行や同時実行時の振る舞いの制御」。Javaは内部的には同じプログラムを使いまわすのではないので、これがないと死ぬ。

って感じでJVMについて説明したけど
ぶっちゃけ最初の頃は何も気にしなくていい。

- Javaは静的型付け言語。他に動的型付けというものがある。
- 端的に言うと静的は「プログラムを書いてる最中に型が確定するもの」で、動的はそうでないもの。ちなみに後者も確定、というか「限定」させる事は可能。
- 個人的にメリットは何？と言われると「プログラムの書きやすさ」と「型安全を保障してくれる」というところに尽きる。
- 正確に言うとプログラムの書きやすさはEclipseみたいなIDEが強い。というか強くできるから。コード補完がめちゃくちゃ強くなる。
- 型安全は「特定の型で定義したプログラムは不正な動きをしない」事を保証する事です。文字列型で定義してるのに何故か数値になってて例外が発生、みたいな。ちなみにこれはJava固有の特性ではなく、動的型付け言語であるRubyも型安全と言われてます。

実行するまでわかりません
よりも
実行する前からわかります
が好きってだけ。

とはいっても実行時になって初めて「型」が明確になるっていう動的な手法もメリットはあるので、一長一短としか言えない。後は好み。

オブジェクト指向って何？

正直よくわかってない気がする。
でもこんなイメージっていうのは伝える。

- オブジェクト指向とは「概念」なので、そもそも100%説明できるようなものではなく、ざっくり理解するもの。
- オブジェクト指向って言うけど、正確に言うなら「目的指向」だとかそういう感じになる。ようは「ある特定の目的を達成する」事を重視する考え方みたいな。
- 更にソフトウェアにおいては日夜更新があるのが当たり前。変更が有るのが当たり前。それに対して「変更しやすくしましょう」っていうのもオブジェクト指向の要素として考えられる。これは単純に特定の目的を達成しようとすると「構造や中身の変化を許容する」必要があったからと推察。

- オブジェクト指向には概念を実現するための概念みたいなのがある。
- 「多態性(ポリモーフィズム)」「継承」「カプセル化」
- ちなみにJavaのクラスベース言語に限定したものになっているので、クラスベースじゃなければ多分継承とかはない。多態性は絶対に必要。
- 詳細は割愛。作りながらじゃないと絶対理解できないので、徐々に説明。知識として頭にそっと入れておけばOK。

型の種類と使い方

- 「文字列型」「数値型」「日付型」「真偽型」「コレクション型」あたり。
- 文字列型は「String」と把握しておけばOK。（charとか知らない）
- 数値型も複数あるが、「int」と「BigDecimal」さえ押さえておけばまず問題なし。
- 日付型は「LocalDate」や「ZonedDateTime」など。一昔前は「Date」や「Calendar」というものがあったけどもういらない。
- 真偽型はboolean。
- コレクション型は「List」や「Map」の事。連続した要素の取扱。

実際に書いてみよう。

- コンソールに「HelloWorld」を表示してみよう。ただし、変数greetingをString型で定義して、そこに値を代入してから表示する事。
- コンソールに1～10まで足した結果を表示してみよう。ただし、for文を使って4行程度に収める事。出来る人は更に短くても可。
- コンソールに「3年後の今日が何曜日なのか」を計算して表示してみよう。使うのはLocalDateTimeとDayOfWeekは必ず使う。
- コンソールに「引数が0なら偽。引数が1なら真」と表示してみよう。if文で表現できる。ただし、引数はStringなのでboolean型にする。
- コレクション型は割愛。業務処理で確実に使うのでその時に説明。

答え合わせ

クラス・フィールド・メソッド
アクセス修飾子とか。

- クラスは「何をするものか(処理内容)を定義するもの」
- フィールドは「クラスで(が)利用する変数を定義するもの」
- メソッドは「クラスで(が)実現する事を具体化したもの」
- アクセス修飾子は「クラス(メソッド)を誰が使えるのかを定義したもの」

- ざっくりこんな具合。細かい話をやりだすとキリがないのでこんなところ。「プログラム」としての意味合いよりも「実質的な」意味合いを重要視してます。
- クラスフィールドは全インスタンスで共通的に作成されるうんたらかんたらみたいなのを省いてるって意味。

- 何らかの受付を行う機能を作るとした時。
- Receptionist というクラス。
- compatible というフィールド。
- accept というメソッド。
- public というアクセス修飾子。

- 受付をするという目的を遂行する為のクラスがあり、それに必要な「処理」や「状態」を保持している。そしてこの受付という処理は「誰でも利用」が出来る。

インターフェースと実装。
抽象クラス。継承。

- インターフェースはクラスと同様に「何をするものか(処理内容)を定義するもの」
- 違いはただ一点だけで、何をするかは明確だけど「具体的な方法」については決めていないもの。クラスはメソッドで具体的な方法まで書くけど、インターフェースは定義するだけで処理自体は書かない。
- インターフェースの定義に「具体的な方法」を記載することを「実装」と言う。実装の仕方は色々あるので、色々な実装が出来る。
- これを「多態性(ポリモーフィズム)」と呼ぶ。

- 抽象(Abstract)クラスもクラスと同じ。でもインターフェースほど「定義だけ」ってわけでもない。具体的な処理も書く。でも定義だけのものもある。そんなちょうど間ぐらいのやつ。
- 具体化されたものが「クラス」とするならこいつは中途半端なので「抽象クラス」と呼ばれる。安易。
- インターフェースも同じだけど、定義しただけのものがあるとJavaは実行できないので、ちゃんと「実装」しないといけない。

- 継承は知識だけ押さえておいて、なるべく使わないことをオススメ。
- 継承は最終的にがんじがらめになり、保守コストを上げるので、使わざるを得ない限りは使わない。
- 継承(extends)はクラスを引き継ぐこと。つまり目的は同じ。
- 但し、メンテナンスや機能追加の関係で「元のクラスを変更できない」ようなケースにおいて、継承することで元のクラスに変更を加えずに新たに機能追加を行うような時に使う。
- と思っているけど、まともに使っている人もいるかもしれない。少なくとも私は使ったことがないし、なるべく使わないようにしている。

実際に書いてみよう。

- Communicationというインターフェースを作ってみよう。
- greetというメソッドを定義してみよう。
- インターフェースに対して実装を作ってみよう。WorkplaceCommunicationとか。
- greetというメソッド内でHelloという言葉を受けたら、Helloと返す処理を実装してみよう。
- 職場かどうかを判断するようなフィールドを作ってみよう。
- 職場かどうかを判断するメソッドを作って、職場じゃなければ挨拶しないようにしてみよう(酷い)

greetはどうやって実行するのか？

ヒント：new演算子とmainメソッド

職場かどうかの判断は
メソッドの引数でやるの？

ヒント：コンストラクタに引数を付ける

答え合わせ

オーバーライド・オーバーロード

- オーバーライドは「既存の処理の上書き」
- オーバーロードは「既存の処理と同じ名前で違う事をする」

- 名前は似てるけど、中身は全然違うので注意。
- オーバーライドはインターフェースで定義されたものを実装する時にも使う(省略してもOK)
- オーバーロードは引数に応じて振る舞いを変えたい場合などに使う。数値型や真偽型を文字列型にしたい場合など。

例外処理

- 例外処理は「意図しない動作に対するメッセージ」を表現するもの。
- その為、正確に言うと「異常」ではなく「正常」という扱い。
- でも想定外の動きに対するメッセージではあるので「異常」という表現も変ではない。そんな感じ。

- JavaにおいてはExceptionとErrorの2種類がある。
- Exceptionは制御可能で、例外処理と言うとこちらを扱う。
- Errorは制御不可能でJVM自身が出力するもの。単純に言えば「これ以上処理を継続できないような状態」の時に出すものなので、こちらで何らかの制御を入れたところでJVMはその動きを保証できない。

実際に書いてみよう。

- Communicationクラスに新しく話しかけてきた人が誰か(同僚とか上司とか)で応答を返すか、例外を返すような処理を作ってみよう。
- 引数はStringでもbooleanでも可。
- 返り値はStringにでもしておいて、適当に返すかどうかはお任せ。
- 例外にする対象もお任せ。
- 例外クラスは自作も可能。何か面白い例外を勝手に自作してもいいし、既存の例外クラスを使ってもいい。とにかく例外をthrowしてみよう。

答え合わせ

コンパイルとビルド

- コンパイルはJavaのクラスをIDEを使って保存すれば、基本的に毎回行われている。そうでなければJavaクラスの実行をJVMが行えない。たまにコンパイルされてなくてエラーが出る事もある。
- コンパイルに関しては意識することはあまりない。というのも、IDEが勝手にやってくれるのと、コンパイルエラーが出ればIDEが検出して教えてくれる為。IDEが出しているエラーの殆どが実はコンパイルエラーだったりする。
- 逐次コンパイルしないような言語や、動的に処理する言語の場合はここらへんがどうしても難しいので、IDEのサポートも必然的に少し弱くなる。

- ビルドは一般的に「作ったプログラムを実行できる状態にする事」を指す。
- 今はIDE上で動かしているけど、実際のプログラムは任意の環境(サーバー等)で動くことが殆どなので、そこで動くような形式にする一連の流れ全てを「ビルド」と言う。
- ちなみにWebアプリとして動かす形式は大体が「war」と呼ばれるが、さらに大きな単位として「ear」があり、小さな単位として「jar」がある。複数の「jar」の集合が「war」であり、複数の「war」の集合が「ear」と取ってもらってOK。もちろん、jar単体でも動作させられる。
- このビルドに当たってはフレームワークなどを利用していると、様々な設定ファイルが存在する事になるので、ビルド手順や設定が必要になる事が多い。
- こうしたビルドをサポートするフレームワークやライブラリも存在し、業務においては必須技術(MavenやGradleとか)

次回までの宿題

今日全部できなかった人はできるまで。

今日全部出来た人は次のアプリ開発の要件を出すので、それについて調べたり着手する。

- 作成するアプリケーションは「日付情報を取り扱う機能」
- 要件としては下記。

- 業務日付(翌稼働日や翌月末、締め処理日)のようなものを管理したい。
- 業務日付の計算式を登録して、計算基準日を元に計算したい。
- 計算は画面上から手動でできればとりあえず良い。
- 手動で行う計算はシミュレーションできるようにしてほしい。
- シミュレーションした結果を実際の計算結果として反映したい。
- 業務日付は他の機能でも使いたい為、任意の日付コードみたいなもので検索して取得できるようにしてほしい。
- 検索で取得する際にはサービス経由での検索か、WebAPIのような形で検索したい。通信フォーマットはJSONでやり取りしたい。

次回からはもうアプリ開発に入ります。
頑張りましょう。

おしまい