Challenging Tomorrow's Changes

初めてやってみたシリーズ
デザインパターン編(Singleton)

技術戦略室
先端技術開発部
三井　省

伊藤忠テクノソリューションズ株式会社

初めてやってみらシリーズ(デザインパターン編)の展望

目的

マイクロサービスで活用されるDDDを理解するには、「オブジェクト指向」の理解が必要。
このシリーズを通して「オブジェクト指向」の使い方を学ぶ。

施策

シリーズでのコード展開、及び、説明補足。また不明点の共有と解消。

展望

マイクロサービスを活用したアジャイル開発等に興味を持って頂く。
各自、能動的にDojoでの発言やコード開発を実践して頂ければ幸いです。

アジェンダ

1) 前提/目的
2) オブジェクト指向って？
3) デザインパターンって？
4) Singletonパターンって？
5) 余談(メモリ)

1. 前提/目的

前提/目的

前提

・選定言語はJava
・デザインパターンを全て学ぶわけではなく、代表的なものを数個学ぶこととする
--Singleton, Template Method, Factory, Strategy, Composite等

背景

現在、マイクロサービスの活用に伴い、DDD(ドメイン駆動設計)が活用されることが多い。
DDDではDIを活用したオブジェクト指向の利用が前提（ヘキサゴナルアーキテクチャ等）となっており、DDD自体理解が難しいので、基本となるオブジェクト指向の理解を目指す。

目的

各種デザインパターンを通して、「オブジェクト指向」の使い方を学ぶ。
今回はシングルトンパターンを通してメモリ管理にも若干触れながら理解を進める。

複数のデザインパターンを通してオブジェクト指向の活用方法を学ぶため、
今回の資料のみで理解するものではないことご了承ください。

2. オブジェクト指向って？

オブジェクト指向って？

皆さん、「オブジェクト指向」って現場で使えてますか？

オブジェクト指向の設計者は以下のように定義してるようです。

- 1, *Everything Is An Object.*
- 2, *Objects communicate by sending and receiving messages (in terms of objects).*
- 3, *Objects have their own memory (in terms of objects).*
- 4, *Every object is an instance of a class (which must be an object).*
- 5, *The class holds the shared behavior for its instances (in the form of objects in a program list).*
- 6, *To eval a program list, control is passed to the first object and the remainder is treated as its message.*

— Alan Kay

どれも重要そうですが、実際現場で「使う」という観点ではそんなに多くを気にしてません。

オブジェクト指向って？

現場で「使う」という観点に絞ると、実はとてもシンプル。DDDで特に重要なのはオブジェクト指向の内、ポリモーフィズムです。以下の一言が言える方は問題なく使えています。

「インタフェース」を使う

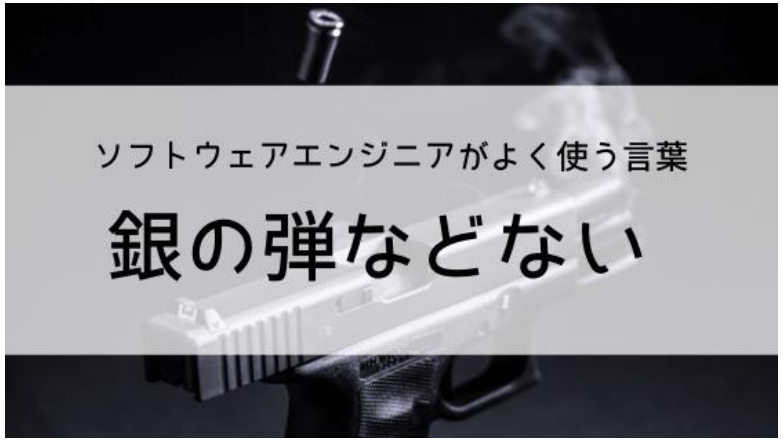
3. デザインパターンって？

デザインパターンって？

Wikiでは以下となってます。
共通的な問題を解決するパターン手法と簡単に理解して頂ければ。

Design patterns are formalized **best practices** that the programmer can use to **solve common problems** when designing an application or system.

【注意】
デザインパターンはよく使われる設計を一般化してまとめたものに過ぎず、現場の問題を全て解決するものではないです。
※IT業界に「銀の弾丸」はないのでアリマス！



4. Singletonパターンって？

Singletonパターンって？

SingletonパターンのコードをGitHubに格納しております。

https://github.com/sho-kenny/hogehoge/tree/master/Java/DP/Singleton

【Singletonパターンの有用性】

・指定クラスのインスタンスが1つに限定できる

※今回は海外の人がよく使うSingletonを実装例に挙げてます

【Singletonパターンの条件】

・privateコンストラクタ＝外部からの生成を禁止

※厳密にはリフレクションでコンストラクタアクセスできるが、普通の開発現場では、テストクラスにしかリフレクションは許容しない。

・SingletonのインナークラスにSingletonインスタンス保持

※JVM起動時にクラスローダがSingletonクラスを読み込まないのでメモリリソースの有効活用が可能（これが海外でよく使う手法）

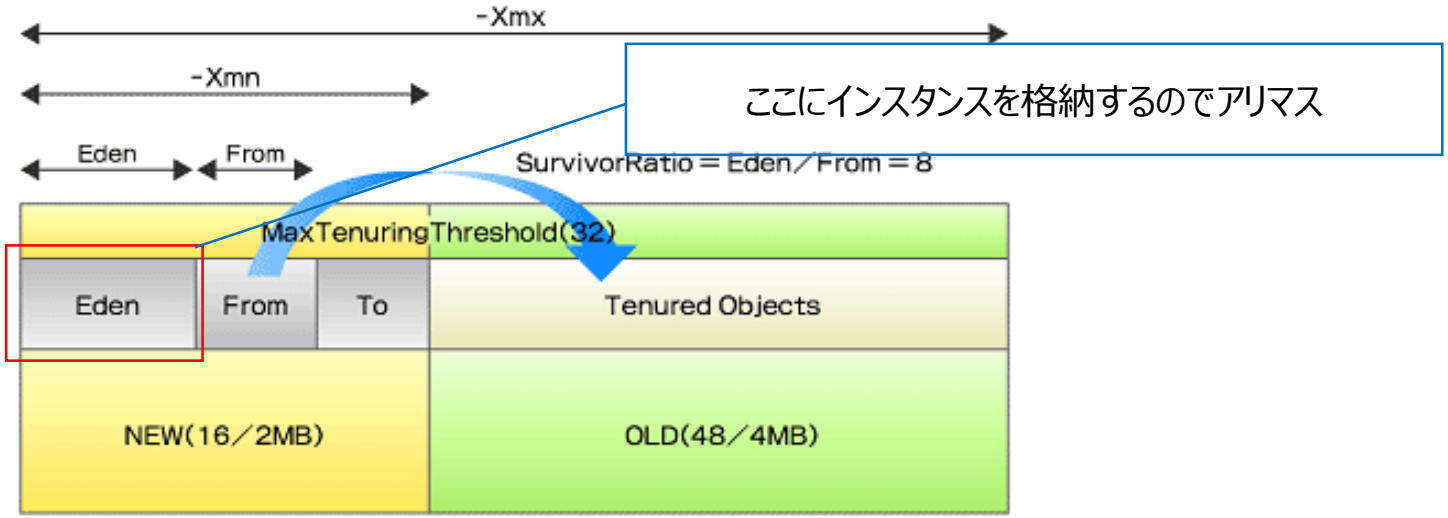
5. 余談(メモリ)

余談(メモリ)

プログラムでインスタンスの生成はよくやりますよね。

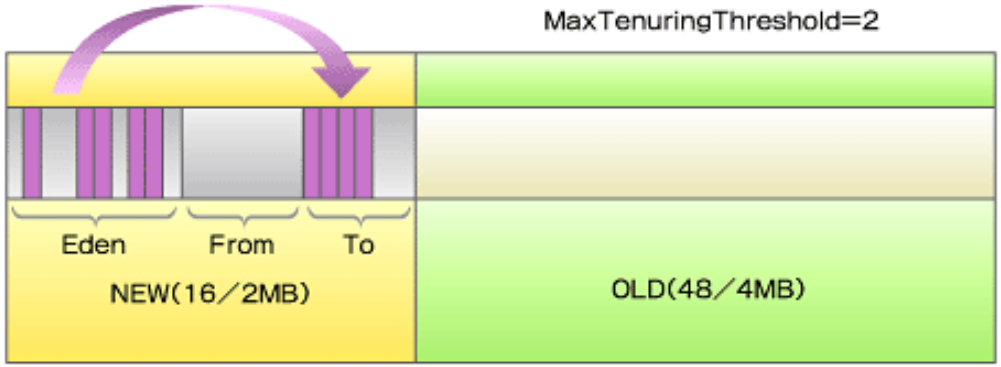
```
Hoge hoge = new HogeImple();
```

JavaやC#等の場合、newでインスタンスを生成するとメモリのヒープ領域(Eden領域)にインスタンスが格納されます。このメモリ領域に格納されたものを解放することをGC(ガベージコレクション)といいます。GCには、スカベンジGCとフルGCがあります。



余談(メモリ)

Eden領域が満杯になると、スカベンジGCが実行され、未使用インスタンスは削除されます。まだ使用中のインスタンス（他オブジェクトから参照されているオブジェクト）についてはEden領域からTo領域へ移動します。



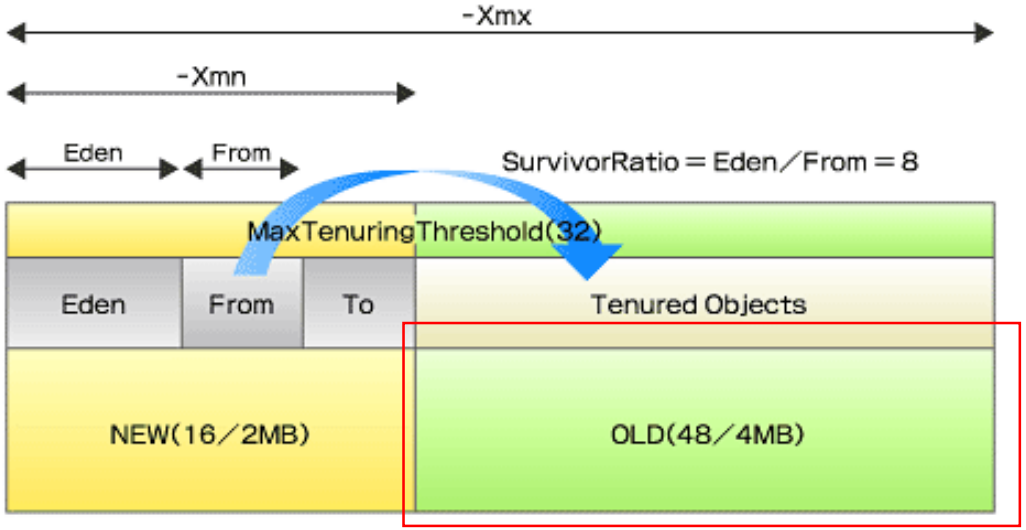
再びEden領域が満杯になると、スカベンジGCが実行され、To領域で使用中のオブジェクトはTo領域からFrom領域へ移動します。

※この繰り返しを一定数(デフォルトでは32回)以上繰り返して未だ使用中のインスタンスはOld領域へ移行させることになります。

余談(メモリ)

Old領域が満杯になると、フルGCが実行されます。ただしフルGCはリソースを大量に消費するので、アプリが停止状態となります。

慣用句でこれを**「Stop the World」**と言ったりします。



こういう経緯があり、リソースを意識して、new部分のリファクタリングも注意深くやっていたのが慣例でした。

※ただし現在はマシンリソースが発達し特に注意しなくてもよくなってます。