Future Office
Coordinate Service

オブジェクトを使った
分析から実装まで　概説

初級者向け、Javaを使った実装に至るまでの手順

はじめに

Ｊ２ＥＥ出現当時「Webとオブジェクト指向で開発ができる要員を揃える」というのが顧客の発注条件になりましたが、最近は“オブジェクト指向”という言葉を聞くことがなくなりました。

改めて“オブジェクト指向”をネットで検索して（2021年4月）みると、いくつかのサイトでは説明の最後に「設計には長い経験が必要で云々」と書いています。『経験に頼る…構造化以前の時代 に戻ってるじゃねーかコノバカチンガー！』という状況のようです。

オブジェクトを前提に作られた言語／環境をオブジェクト（インスタンス）の意識なしに使うのは危険なコードを混入させる要因になり危険です。

この資料では、構造化と対比してオブジェクト指向ではどう考えればよいか整理します。実際にシステム開発の役に立つと考えた場面だけに限定しているため、専門書に著された理論的・哲学的オブジェクト指向とは異なっている可能性があります。注意してください。

確認-1

■構造化とは？

- 構造化とは、対象の構成要素と組立方を定義すること
- 構造化分析、構造化設計、構造化プログラミングの工程がある

■オブジェクト指向とは？

- 対象を概念*1（＝オブジェクト）として把握し、概念間の関係をモデル化する
- オブジェクトはデータと動作（＝コード）を包含する

*1：“概念(原語 concept)” はそれの**名前があり、名前だけで協働者が同一認識を持つことができるもの**で、〔“取引先” ⊃ “仕入先”、“顧客”〕のように包含関係を持つ場合があります。ある程度具体化するとエンティティ(entity)になります。

※ 概念＝“名前が付いた”＝“定義できる／された” ものということです

確認-2

■そもそもの目的

- 法律／社会情勢／利害関係は常に変化する

  ∴システムもまた、**必要なタイミングで変化**できなければならない

＜その為には＞

👉 対象を、関係者が適否を判断できる粒度（実装：データ、コード）になる迄分解する

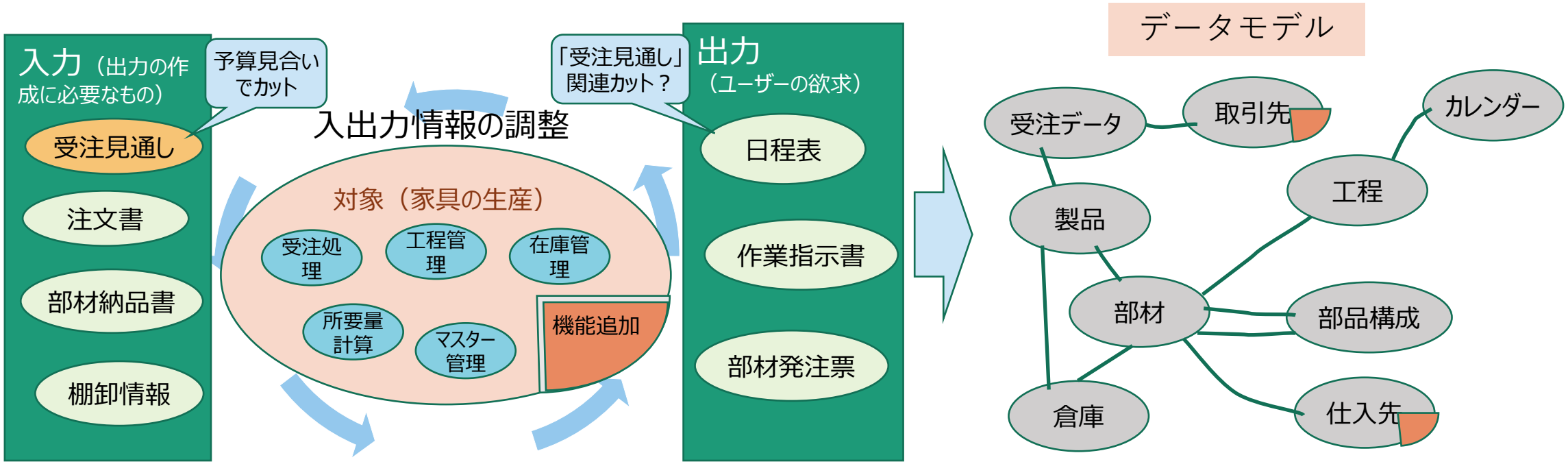
👉 構成要素（＝概念）間の関係を疎にして**影響を局所化**する（モジュール**結合度を下げる**）

構造化もオブジェクト指向も目的は
変化＝改修　の波及を防いで保守性を上げる

分析

■対象を把握し、対象の範囲を決める

＜例＞対象：家具の生産管理　［機能追加］下請けに渡す部材を売上扱いにしたい等



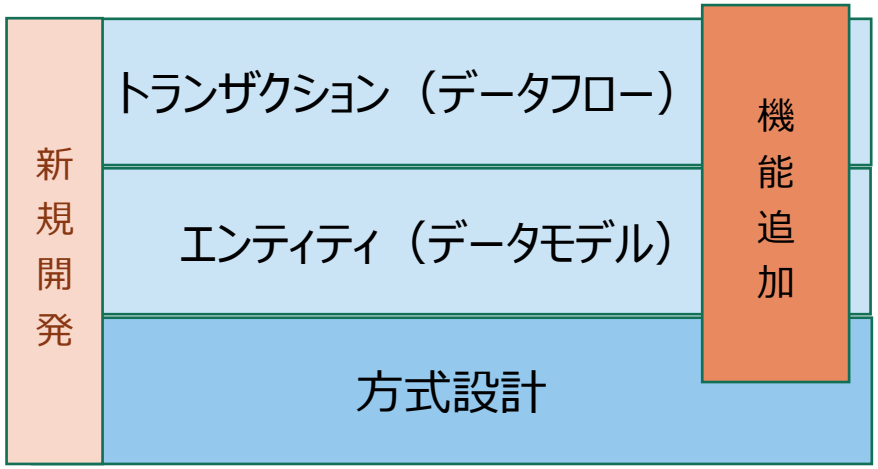
※重複や類似しているデータはないか、更新の時期が全く異なるデータグループが一つになっていないか確認

設計

■分析モデルを実装単位に

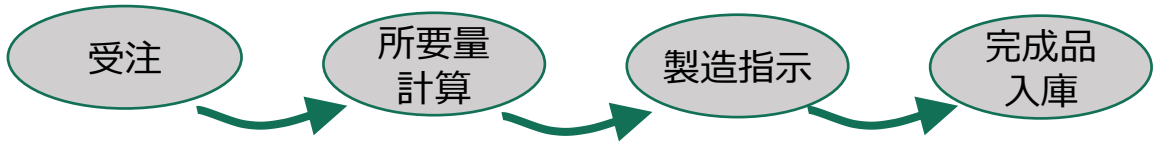
＜観点①＞方式：（例）Webアプリ＋ストアドプロシージャ

＜観点②＞機能：CRUD(Create,Read[Refer],Update,Delete)図、トランザクション・フロー



※機能追加・変更が発生したとき影響する範囲が小さくなるようにエンティティ／トランザクションから実装単位を決める
（方式はほぼ変更できない）

【トランザクション】



【ＣＲＵＤ図】

組織／業務プロセス ＼ エンティティ		受注データ	製品	部材	部品構成	倉庫（部品）	倉庫（製品）
営業	受注	C	R				
設計	所要量計算		R	R	R		
製造	製造指示					U	
	完成品入庫						C

実装

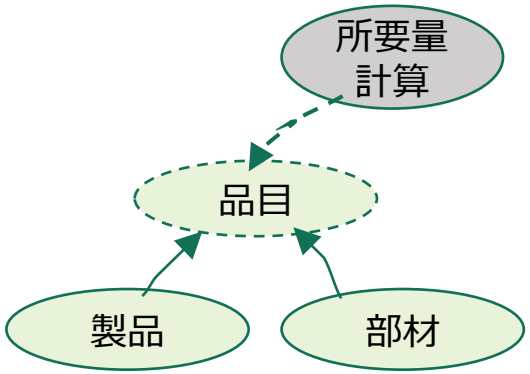
■Java言語のオブジェクト特性、使う利点

継　承　：類似クラスの拡張・再利用、基盤機能の取り込み、一部機能の置換

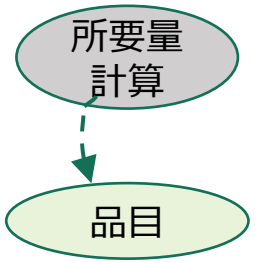
多　態　性：同一機能の複数実装を使い分け

カプセル化：不変性の保証

継承



多態性



BigDecimal 費用計算() {}
BigDecimal 費用計算(Int 消費税率) {}

※多態性は、条件／状態により異なる処理が応答すること。この例のシグネチャの違いによるもの（オーバーロード）の他に、オーバーライドによる実装もある

カプセル化

外部に情報を提供する／しないための手段

<フィールド>

public …外部から更新可能で、どこで変化したか追跡が困難になる

public static final …コンパイル時に参照先にインライン化（値のコピー）され、仕様変更が反映されなくなる

getter経由 …データを保護できる