Javaプログラミング実習

19. オブジェクト指向入門

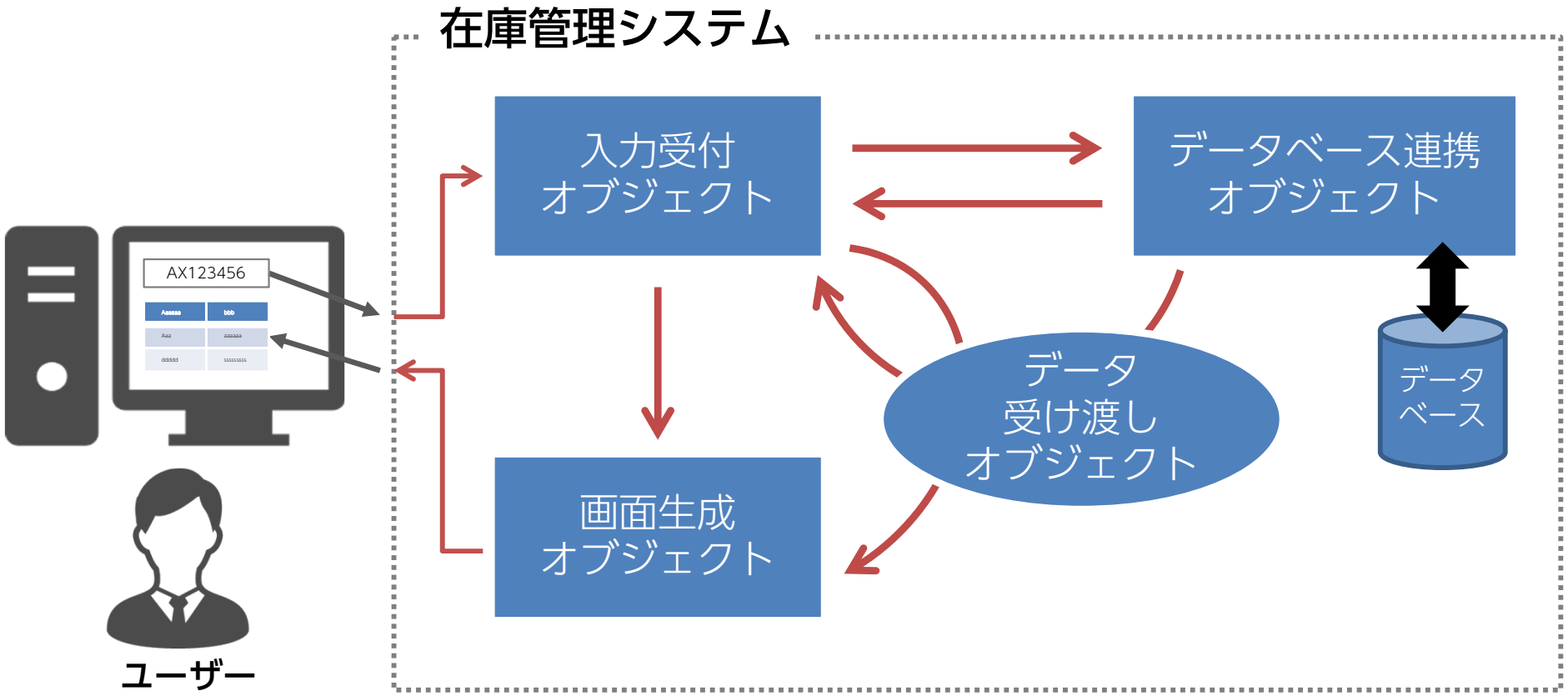
株式会社ジードライブ

今回学ぶこと

- オブジェクト指向の基本的な概念
- オブジェクト指向の３大要素
- UMLの概要

オブジェクト指向とは

- システムの振る舞いをオブジェクト同士の相互作用として捉える**考え方／概念**



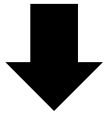
オブジェクト指向の適用

- オブジェクト指向はプログラミングだけのものではなく、分析や設計にも利用される
  - オブジェクト指向分析
  - オブジェクト指向設計
  - オブジェクト指向プログラミング

- オブジェクト指向という考え方に基づいて、プログラムを構築する技法をオブジェクト指向プログラミングと呼ぶ
  - OOP（Object Oriented Programming）

オブジェクトとは

- この考え方において、各オブジェクトはそれぞれ**何らかの役割／責務**を持っている
  - その役割／責務を果たすために必要な情報を保持することができる
  - その役割／責務を果たすために機能を有している



- ある役割／責務を果たすための情報や機能の集合がオブジェクト
  - Javaでは、クラスやクラスを元に生成されるインスタンスがオブジェクトに相当する

オブジェクト指向 ≒ 適切なクラス分けを行い、システム全体を構築する

オブジェクトとは

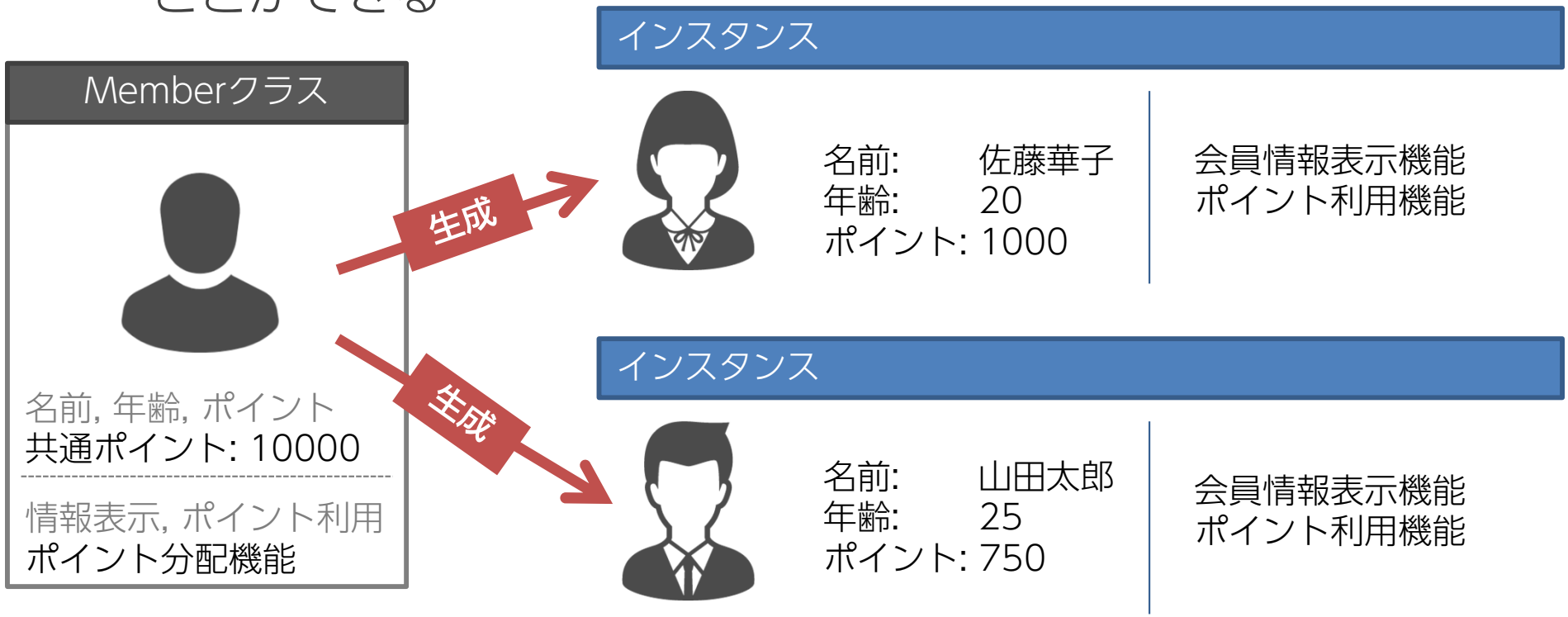
- オブジェクトが保持する情報は、**属性**と呼ばれる
  - Javaでは、フィールド／プロパティに相当する
- オブジェクトがもつ機能は、**操作／振る舞い**と呼ばれる
  - Javaでは、メソッドに相当する

会員情報を扱う役割を担うMemberオブジェクト(Memberクラス)

```
public class Member {
    // 属性(フィールド／プロパティ)
    private String name; // 会員の名前
    private int age;     // 会員の年齢
    private int personalPoint // 会員個人に帰属するポイント
    private static int sharedPoint = 10000; // 全会員で共有するポイント

    // 操作／振る舞い(メソッド)
    public void showInfo() { … } // 会員情報表示機能
    public void usePoint(int pointToUse) { … } // ポイント利用機能
    public static void share(int point, Member... m) { … } // ポイント分配機能
    …
```

クラスとインスタンス

- クラスはインスタンスの雛形
  - インスタンスは雛形であるクラスを元に生成される
  - インスタンスはそれぞれ固有の属性値を保持することができる
  - インスタンスは、クラスで定義されているメソッドを使用することができる



クラスとインスタンス

- クラスはそのまま利用することも、インスタンスを生成してから利用することも可能 (15.クラスの利用 p6～9)
  - staticが付いているフィールドやメソッドは、クラスから直接利用できる
  - staticが付いていないフィールドやメソッドは、インスタンスから利用できる

staticについては次章で学習

例：Memberクラスの利用

```
// クラスからインスタンスを生成し、操作
Member m1 = new Member("佐藤華子", 20, 1000);
m1.showInfo();
Member m2 = new Member("山田太郎", 25, 750);
m2.usePoint(300);
m2.showInfo();

// クラスを直接利用
Member.share(3000, m1, m2);
```

練習

- 練習19-1

オブジェクト指向の3大要素

- カプセル化、継承、ポリモーフィズム(多態性)は、オブジェクト指向の3大要素と呼ばれる機能
- Javaを含むオブジェクト指向プログラミング言語では、これら3大要素を実装するための仕様・文法が用意されている

① カプセル化
  - オブジェクトのもつ内部情報を隠蔽する機能

② 継承
  - あるクラスをベースとして、新しいクラスを作成する機能

③ ポリモーフィズム(多態性)
  - 同名のメソッドであっても、オブジェクトごとに固有の処理を実装するための機能

カプセル化

- オブジェクト外部からオブジェクト内のデータに直接アクセスできないようにする仕組み
  - フィールドに対し、直接アクセスを許可するのではなく、メソッドを通じて、データにアクセスを許可する仕組み

カプセル化の例

privateなので、外部から直接アクセスできない
(age = -25;
のような記述はできない)

メソッドを通じて、ageにアクセスすることができる

```
public class Member {
    private String name; // 会員の名前
    private int age;     // 会員の年齢
    …
    public void setAge(int age) {
        //引数として渡された年齢の絶対値を算出
        this.age = Math.abs(age);
    }
    …
```

継承

- あるクラスの定義を受け継ぎつつ、フィールドやメソッドを追加・変更した新たなクラスを定義できる仕組み
  - 容易に機能を拡張することができる

継承の例：Memberクラスを継承したPremiumMemberクラス

Memberクラスがもつフィールドやメソッドに加えて、使用することができる

```
public class PremiumMember extends Member {
   // silver会員, gold会員などのステージ
   private String stage;
   …
   // 会員特典のアイテム取得メソッド
   public void getSpecialItem() {
       switch(stage) {
         case "silver": … break;
         case "gold": … break;
   …
```

ポリモーフィズム(多態性)

- ある名前のメソッドを呼び出した時に実行される処理は、そのメソッドを持つオブジェクトの種類によって決まるという仕組み
  - 同名のメソッドであっても、オブジェクトごとに異なる処理を定義することができる

ポリモーフィズムの例

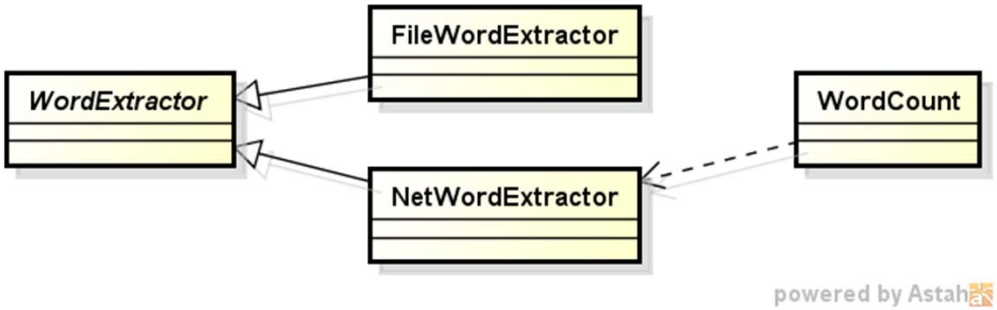
```
String message = "こんにちは";

// PrintStreamオブジェクトのprintln()メソッド
// => コンソール画面にメッセージを表示させる
System.out.println(message);

// PrintWriterオブジェクトのprintln()メソッド
// => ブラウザ上にメッセージを表示させる
response.getWriter().println(message);
```

同名のメソッドだが、異なる処理内容が定義されている

両方とも同じ「表示させる」という処理なので、別々のメソッド名ではなく、同じメソッド名が付いている方がわかりやすく、利用しやすい

モデリングとUML

- 開発対象となるシステムやプログラムを図や文章などで表現することを**モデリング**と呼ぶ
- オブジェクト指向の考え方で分析や設計を行う際には**UML**と呼ばれる表記方法を用いてモデリングを行う
  - UML = Unified Modeling Language



UML誕生の経緯

- 1980年代後半から、オブジェクト指向技術を用いた様々な開発手法が登場した
  - その中で使われる図の記法もそれぞれ異なっていた
- そのような中、オブジェクト指向分析設計手法を標準化する動きが登場し、1997年にUML1.1がオブジェクト指向モデリング言語の標準規格としてOMGに正式に承認された
  - 元々は分析設計手法まで含めて標準化しようという試みだったが、最終的には**図の表記方法のみ**の規格となった

※OMG：Object Management Group(非営利の標準化事業体)

UMLの利用場面

- ソフトウェア分析での利用
  - 作ろうとしているソフトウェアがどのようなものかを検討したり説明する際にUMLを使う
- ソフトウェア設計での利用
  - ソフトウェアをどのように実現するかを検討したり説明するためにUMLを使う

自分の考えをまとめる、他の開発者とアイデアを共有する、
仕様書に記載する、など様々な場面で利用される

UMLの特徴

- 図の記法と意味を定義している
  - 会社やチームで独自に使用している図法よりも汎用性がある
- 13種類の図がある
  - ソフトウェア開発の様々な場面で使用できるように既存の開発方法論で使用されてきた様々な図をベースにして作られている
- 開発プロセスとは独立した**記法の部分のみ**を定義している
  - 13種類のうちどの図をどのように使うかは自由

UMLで使用する図の種類(1)

- クラス図
- オブジェクト図
- パッケージ図
- コンポーネント図
- 配置図
- コンポジット構造図

構造図：
ソフトウェアの静的な
側面を表現する図

UMLで使用する図の種類(2)

- ユースケース図
- アクティビティ図
- シーケンス図
- コミュニケーション図
- ステートマシン図
- 相互作用概要図
- タイミング図

振る舞い図：
ソフトウェアの動的な
側面を表現する図

UMLの使い方

- 全ての図を使って分析や設計を行う必要は無い
  - 必要な図のみを使えばよい
- ソフトウェアの全てを詳細にUMLで記述する必要はない
  - 重要な部分、図にした方が理解しやすい部分のみ図にすればよい