プログラミング言語設計論

2014年度

第5回: mixins&trains /
デザインパターン/ フレームワーク

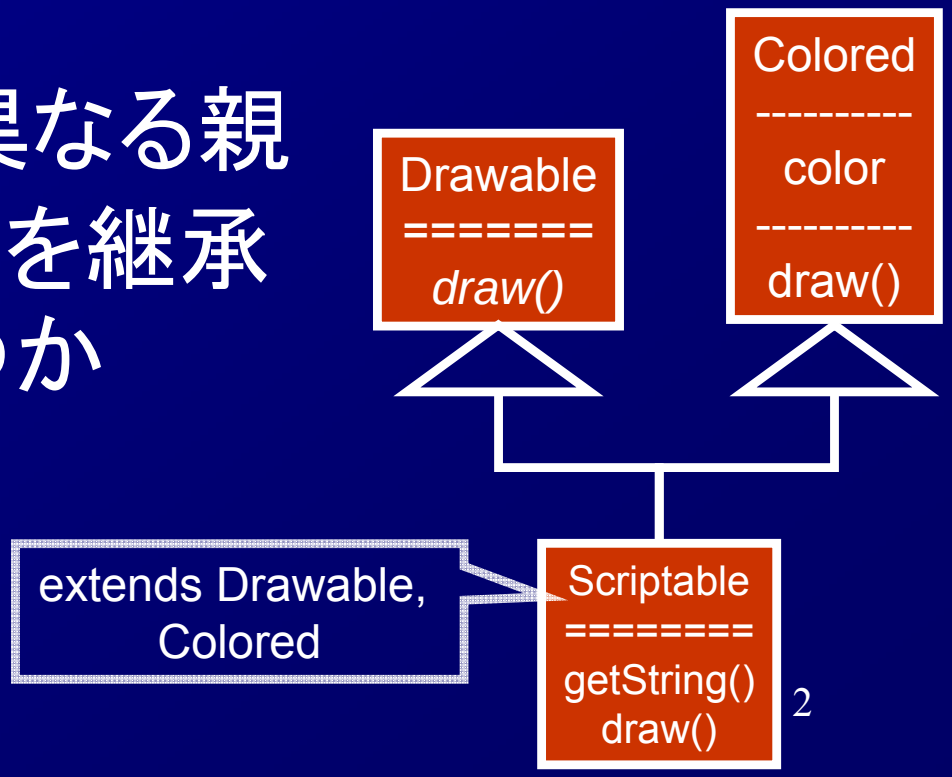
担当: 増原英彦

(復習) 多重継承の功罪

功) 追加的な機能を再利用できる

罪) 曖昧性: 両親が同名のメソッドを定義した場合、何が起きる?

罪) ダイヤモンド継承: 異なる親を経由して同じ祖先を継承したとき、それは1つか複数か?



traitsとmixins

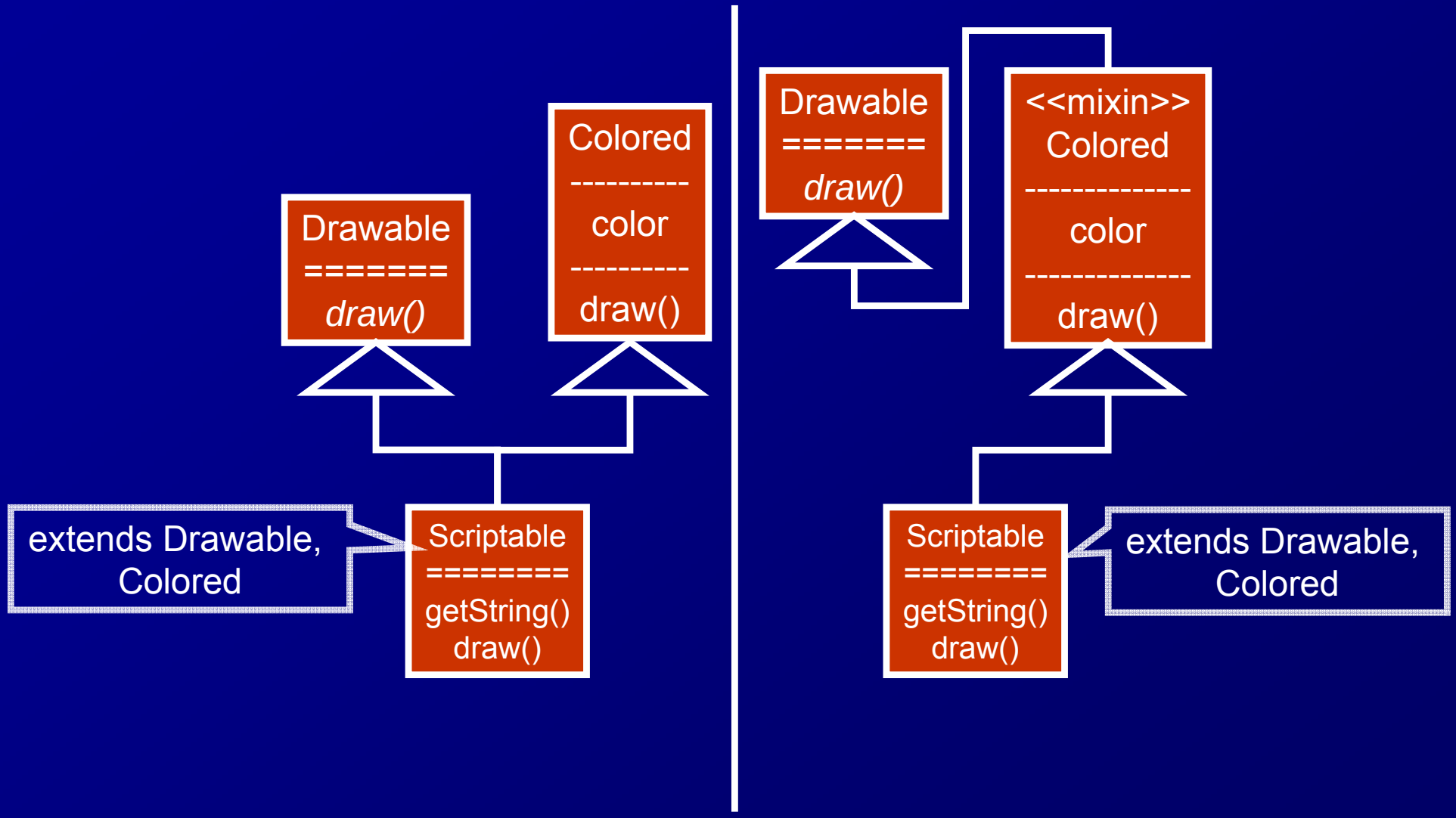
- 追加的な機能を提供するために特化した機構
  - 継承は単一 (親は1クラスだけ)
- 区別: 多重継承で起きる問題を
  - mixins [BC90] — 直線化(linearize)で解決
  - traits [SDNP03] — 平坦化(flattening)で解決

  呼び名に混乱あり (Scalaのtraitはここでのmixin)
- (別の区別: インスタンス変数を持つ / 持たない)

mixins [BC90]

- 親クラスを後から決められるクラス
- 親クラスと一緒に指定すると、親クラスと子クラスの間に入る (直線化)
  - 曖昧性の解決: mixinsが親クラスより優先
  - ダイヤモンド継承の解決: mixinsは「別の物」
- mixinを継承したmixinを作ってもよい
- mixin単独でオブジェクトは作れない
- 代表例: Scalaのtraits

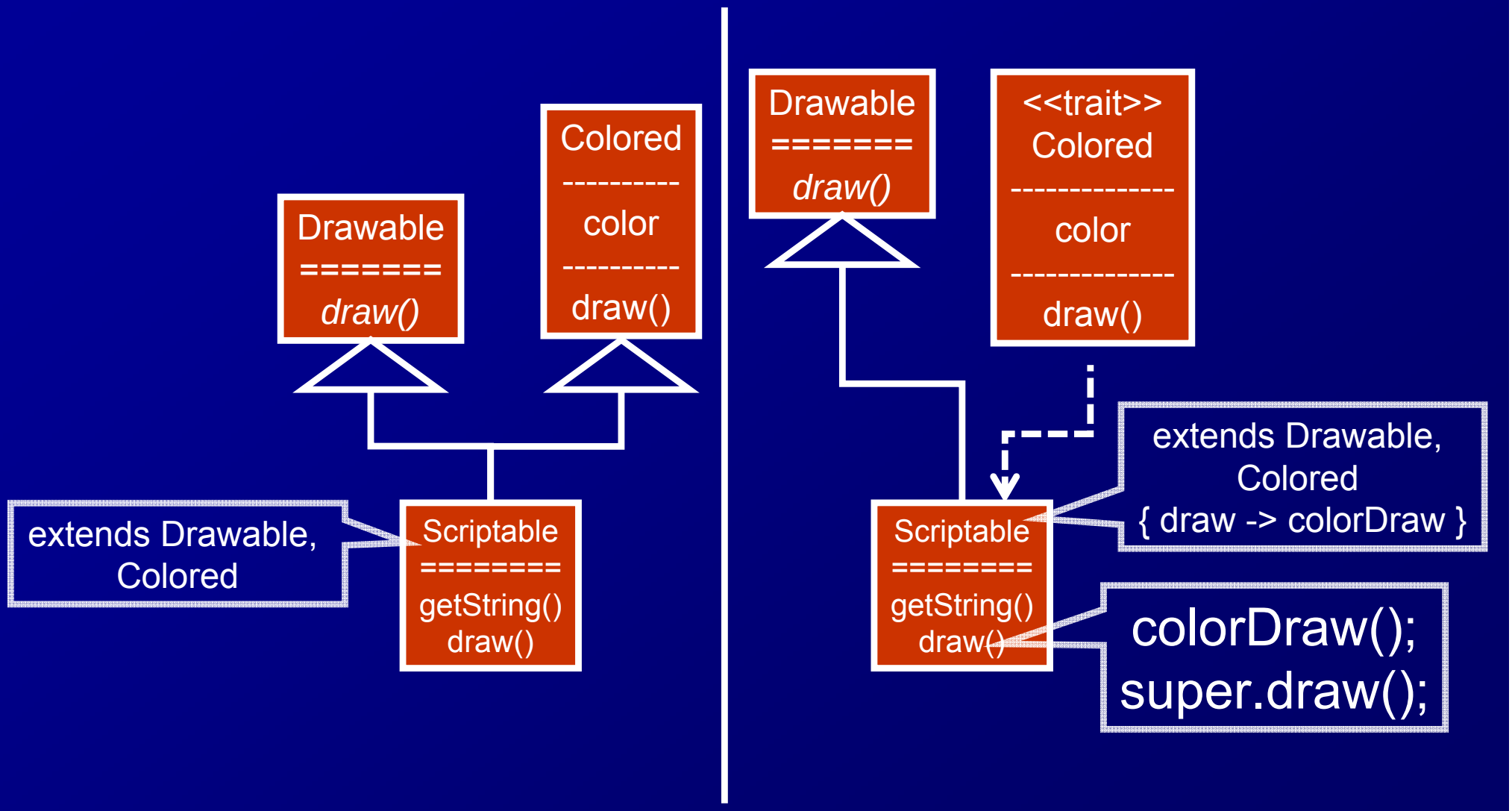
多重継承 vs mixins



traits [SDNP03]

- 子クラスに定義を挿入する親クラス
- 名前が重複するときは明示的に付け替える (平坦化)
  - 曖昧性とダイヤモンド継承の解決

多重継承 vs traits



問題1/2 (10分)

- 3次元CADプログラムの設計をせよ
- 登場するモノ
  - 物体(Volume) — 空間中にある部品それぞれ
  - 表示画面(View) — 正面図と側面図それぞれ
- 使われ方
  - 利用者が物体を操作すると対応する物体のmove()が呼ばれ、位置が変更される
  - アニメーション命令によっても物体のmove()が呼ばれ、位置が変更される
  - 物体の位置が変更されたら正面図、側面図のdraw()が呼び出され、新しい状態を表示する
- 注意: 下線部は設計外とする。メソッドの中身は書かなくてよい

問題2/2 (10分)

ゲームプログラムの共通部分を部品として提供せよ

- 右の黄枠外が共通部分
- 言語は何でもよい (OOP)
- 提供された部品を使う人が何をすればよいのかが分かるように

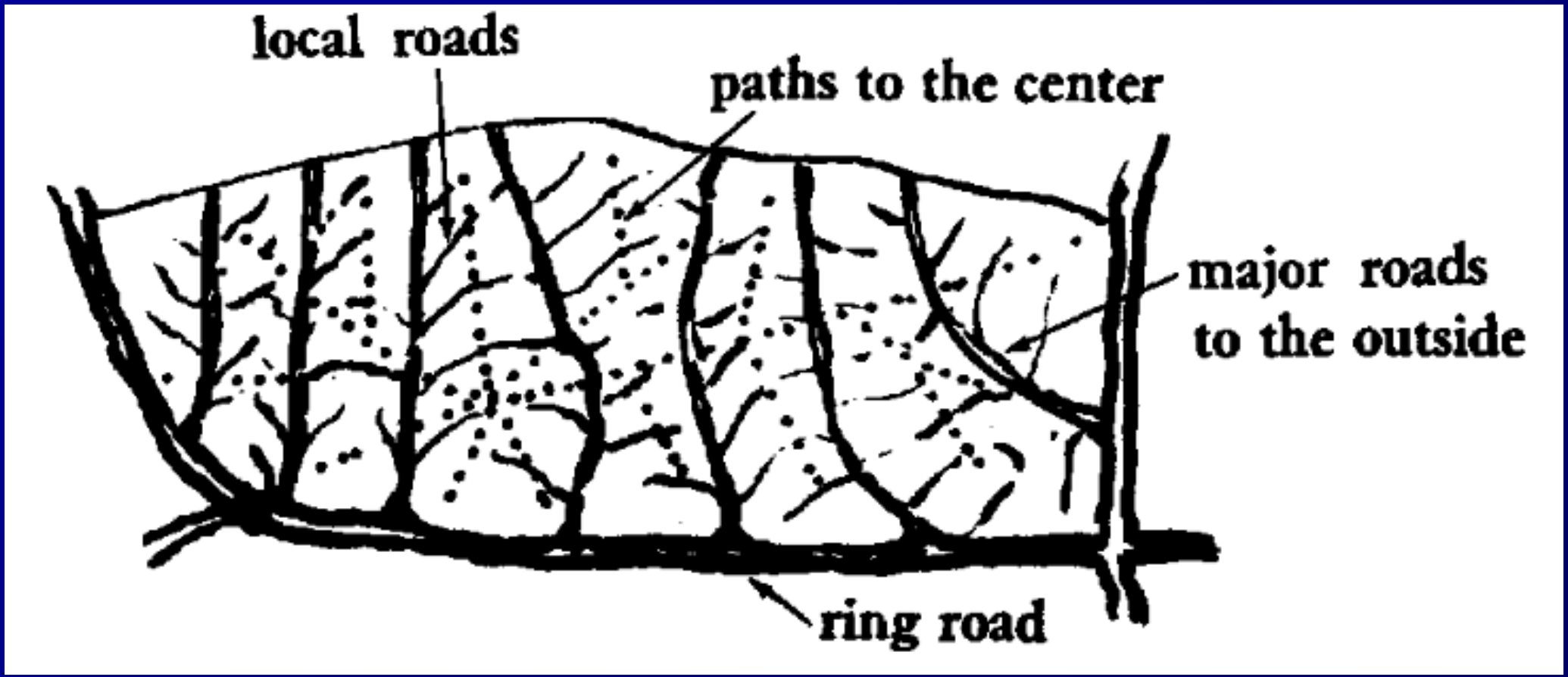
```
byte[] imageBuf1 = ..., iageBuf2 = ...;
int previousKey = 0;
while (true) {
    int c = キーボード状態の読み取り;
    int key = cとpreviousKeyを使って
                「現在押されているキー」を決定;
    previousKey=key;
    switch (key) {
    case Left: 左に動かす; break;
    case Right: 右に動かす; break;
            ...;
    }
    敵を動かす;
    clear(imageBuf1);
    キャラクタをimageBuf1に描く;
    transferToDisplay(imageBuf1);
    imageBuf1とimageBuf2を入れ替え;
    一定時間待つ;
}
```

デザインパターン

再利用のための デザインパターン: 由来

- Alexanderのデザインパターン [Alexander77]
  - 領域: 建築内部設計～都市設計
  - 規模: 色々なスケール
  - 対象: 少数の部品間 (例: 階段、扉)
  - 目的: 経験的に知られている部品間の関係を言語化すること
  - 注意: 個々の部品のカタログではない

Alexanderのパターン例

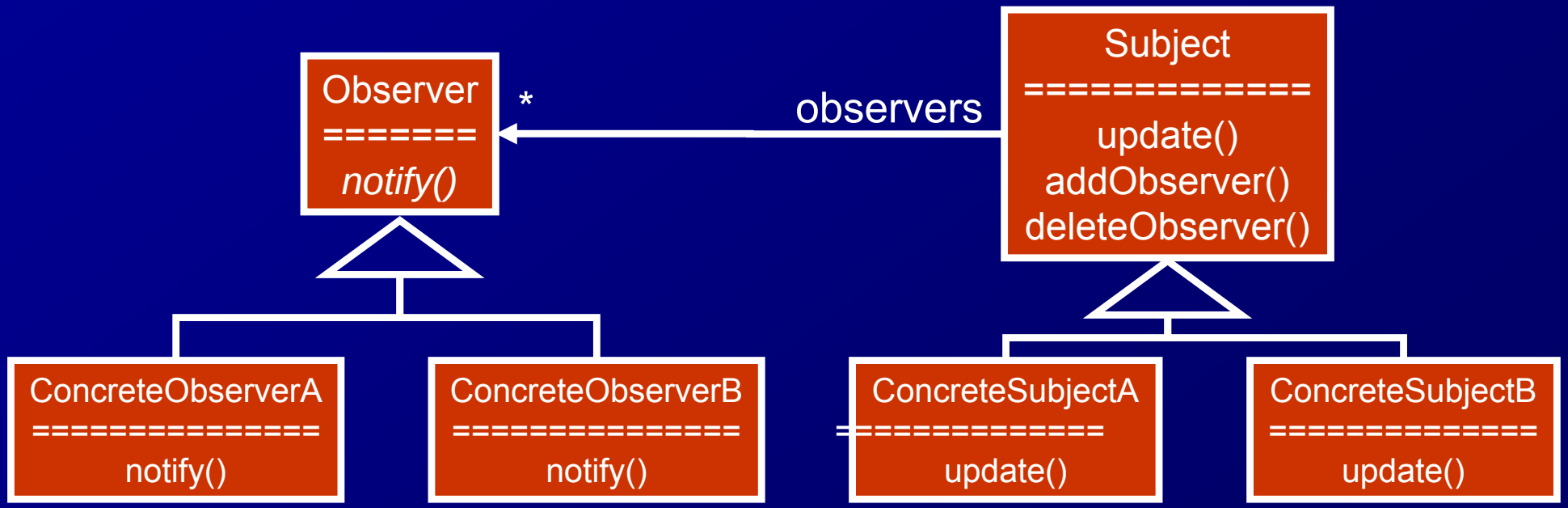


再利用のための
デザインパターン: GoFパターン [GoF94]

- 領域: クラス設計
- 規模: 数個のクラス
- 注意:「再利用」は、パターンに従うソフトウェア設計におけるクラスの再利用性
  - 仕様が変更したときに既存の定義を変更せずに対応できること
  - パターンを再利用しているわけではない

デザインパターンの例: Subject-Observer

- 名前: Observer Pattern
- 意図: オブジェクトの間に一対多の関係を定義し、1つのオブジェクトの状態が変化したら、それに依存する全オブジェクトが通知を受け自動的に更新する

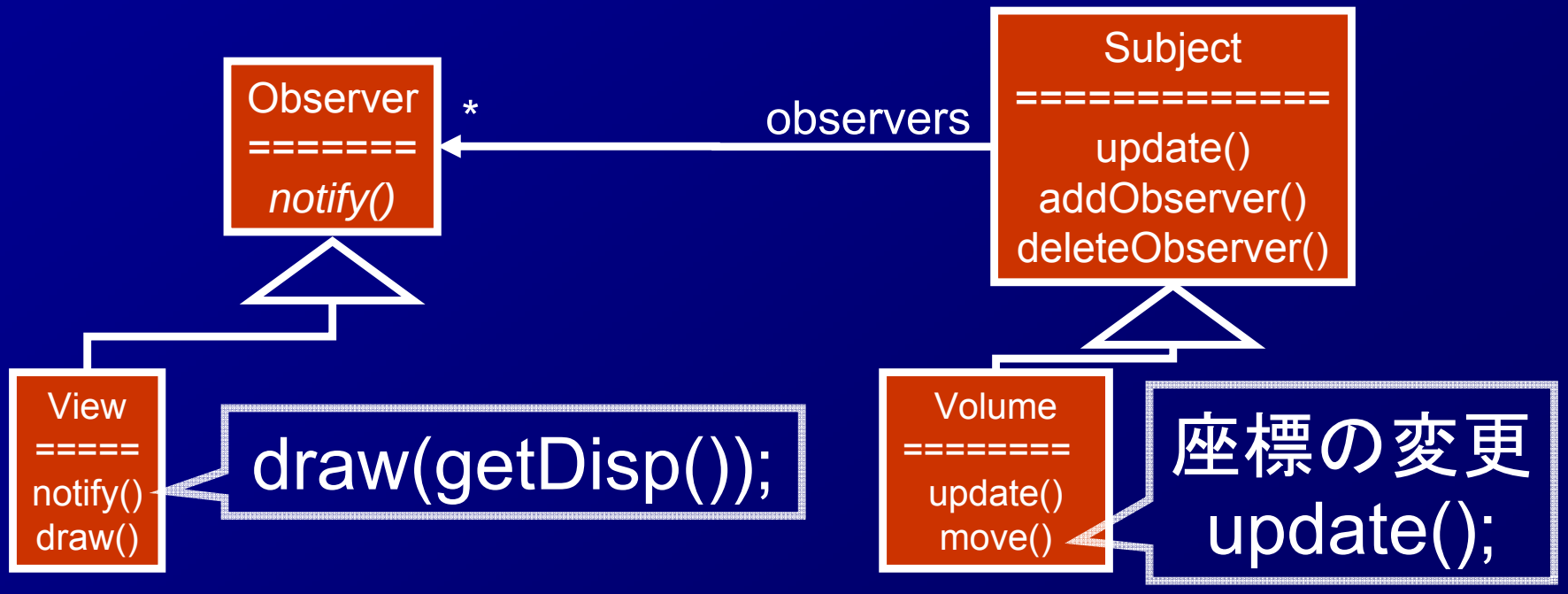


問題1/2 (10分)

- 3次元CADプログラムの設計をせよ
- 登場するモノ
  - 物体(Volume) — 空間中にある部品それぞれ
  - 表示画面(View) — 正面図と側面図それぞれ
- 使われ方
  - 利用者が物体を操作すると対応する物体のmove()が呼ばれ、位置が変更される
  - アニメーション命令によっても物体のmove()が呼ばれ、位置が変更される
  - 物体の位置が変更されたら正面図、側面図のdraw()が呼び出され、新しい状態を表示する
- 注意: 下線部は設計外とする。メソッドの中身は書かなくてよい

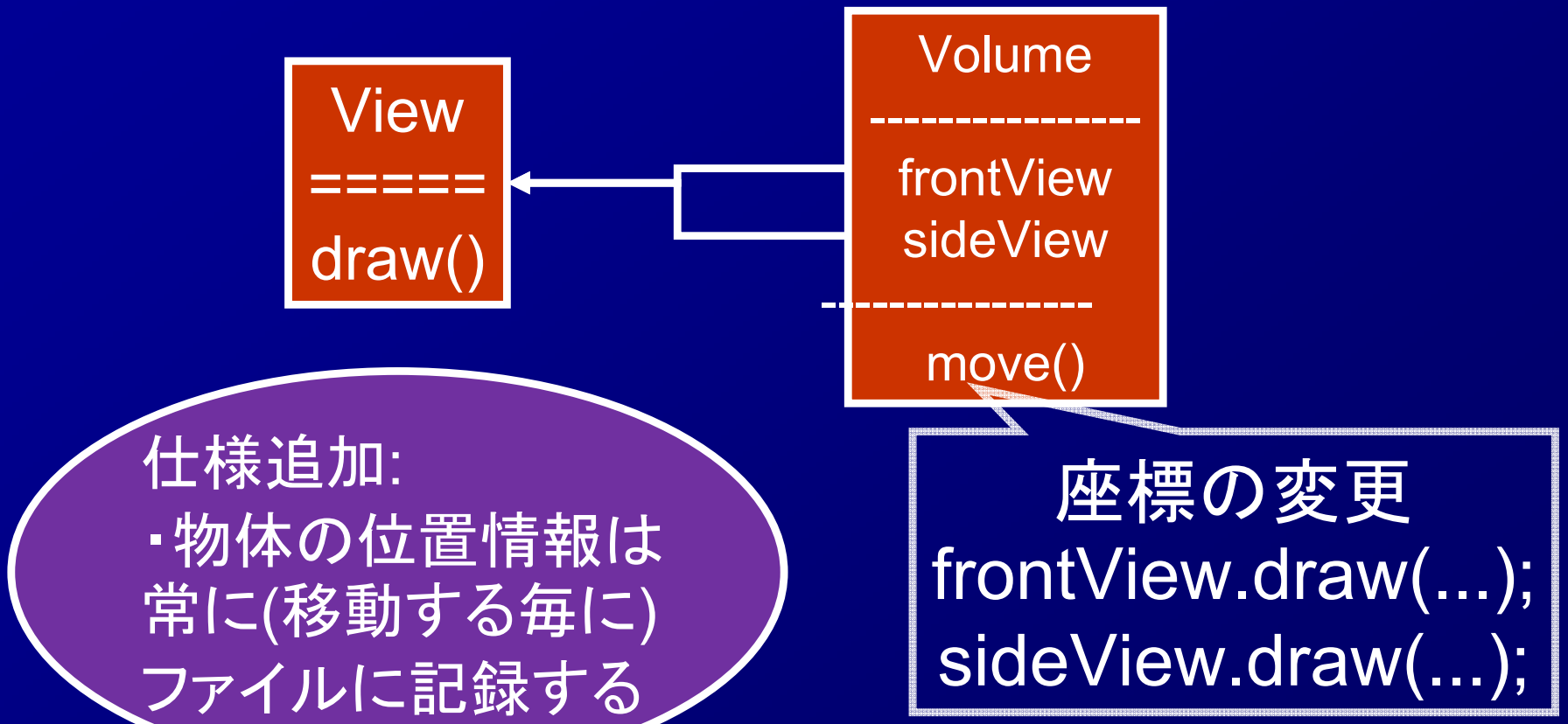
デザインパターンを利用した例: Subject-Observer

- 3次元CAD
- 空間に配置された物体を、正面と側面の2つの視点から表示している。物体の配置が変化したら全ての表示を更新する。



デザインパターンによる再利用性: Subject-Observer

- デザインパターンに従わない場合



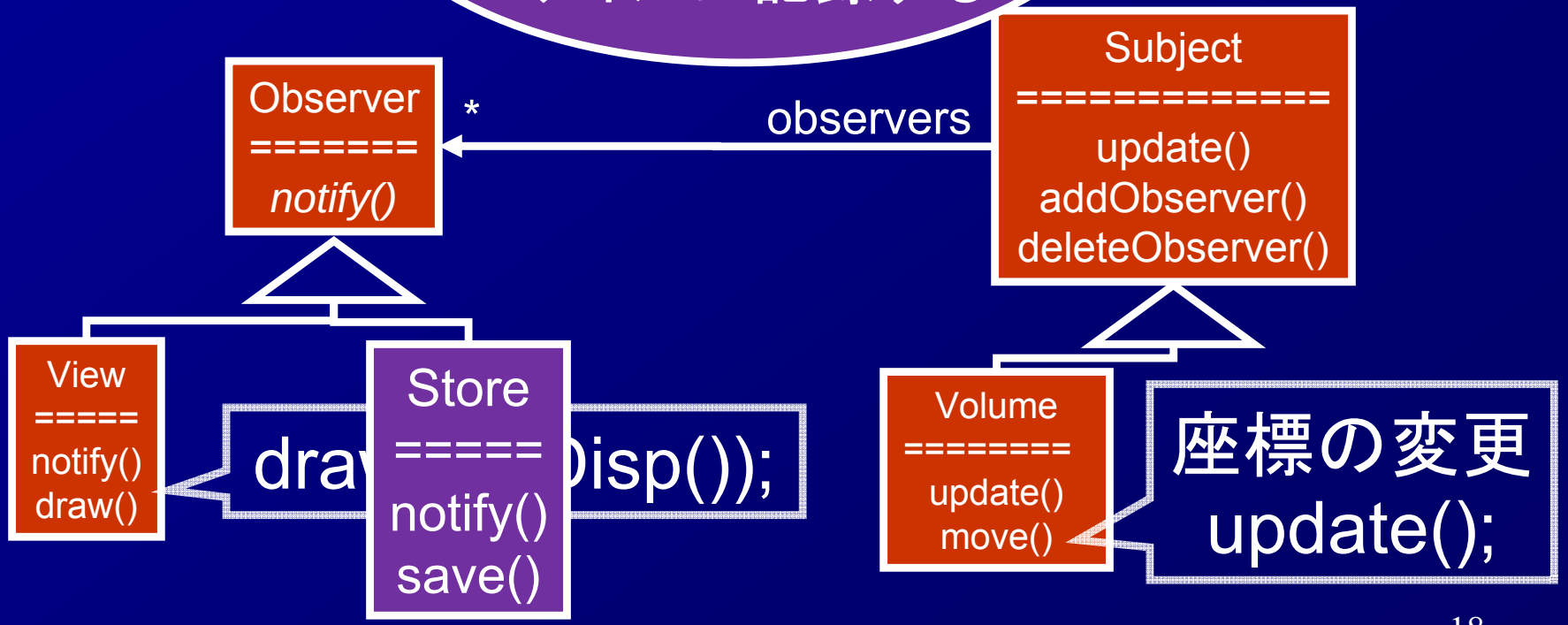
デザインパターンによる再利用性: Subject-Observer

- 3次元CAD
- 空間に配置され
- る。物体の配置か

ら表示してい

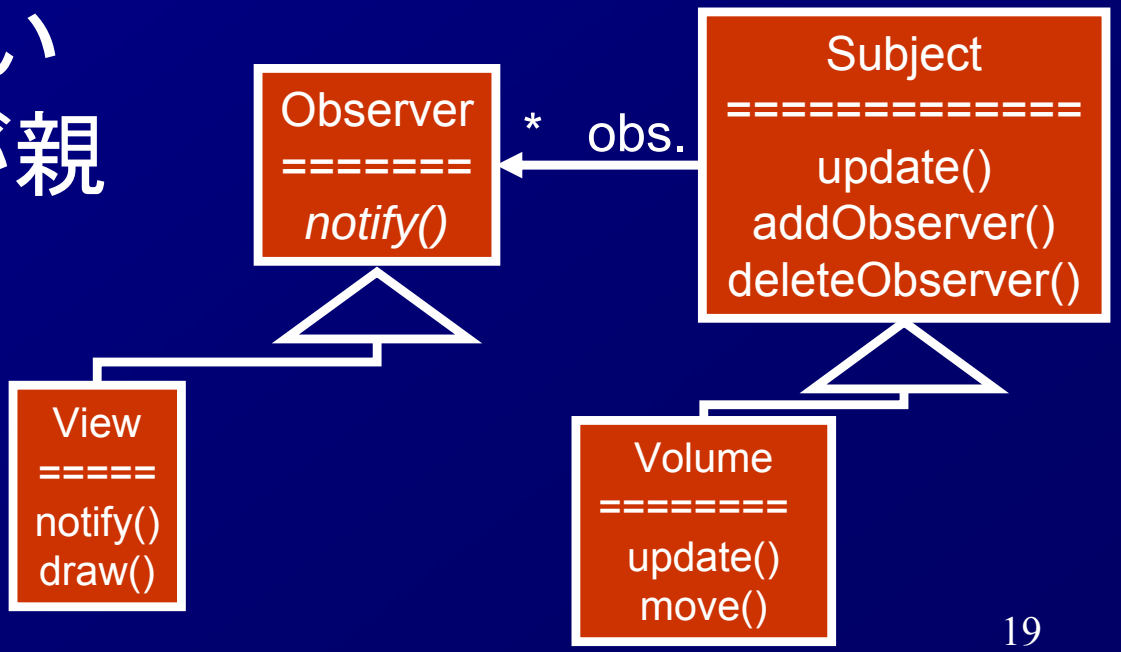
。

仕様追加:
・物体の位置情報は
常に(移動する毎に)
ファイルに記録する



脇道: traits/mixinsの使われ所

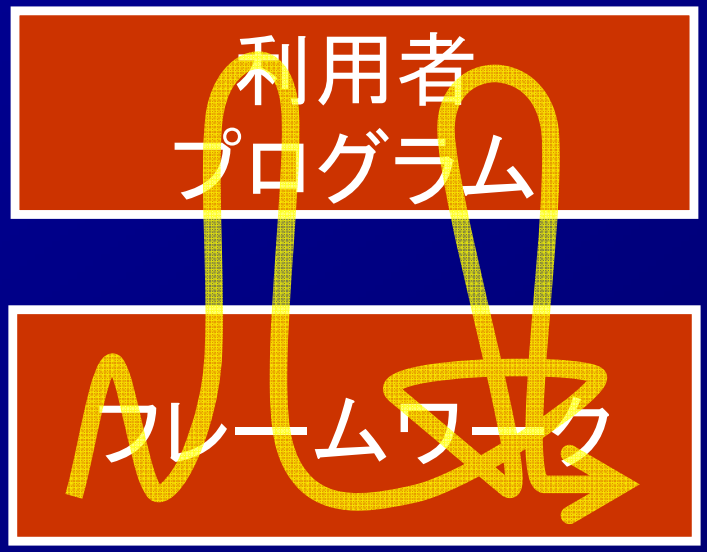
- Subjectはobeversの追加・削除・update という機能を提供
- Volumeの親をSubjectにしたくない 例えば「表示しない 抽象幾何物体」が親
- 既存のImageも 表示対象

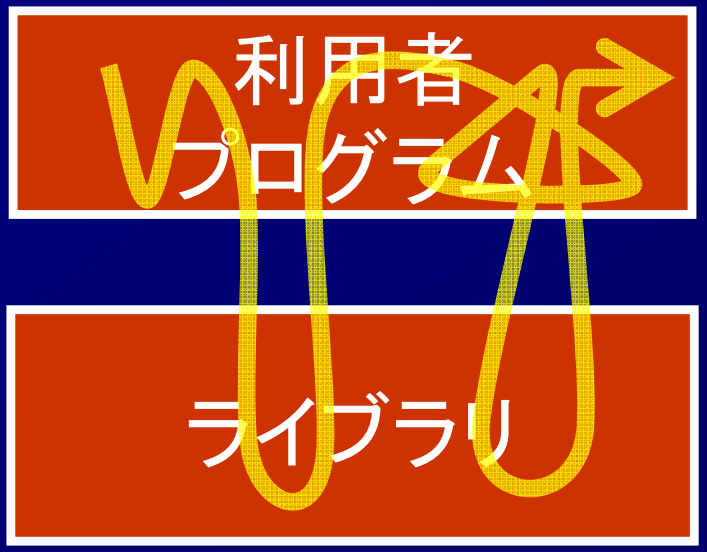


フレームワーク

フレームワーク vs ライブラリ
プログラム再利用の手法

- フレームワーク: 制御の流れを含む再利用(能動的)
- ライブラリ(狭義): サービスの受動的提供





フレームワークの提供方法

- 手続き型言語・関数型言語: 利用者がコールバック関数を作成してフレームワークが提供する関数に渡す (eg. X11ウインドウシステム上のGUIフレームワーク)
- OOP言語: フレームワークは「基本クラス」を提供、利用者は子クラスを作成して穴(各種の動作を行うメソッド)をオーバーライドする
  - 基本クラスに既定の動作を定義できる (オーバーライドしなければよい)

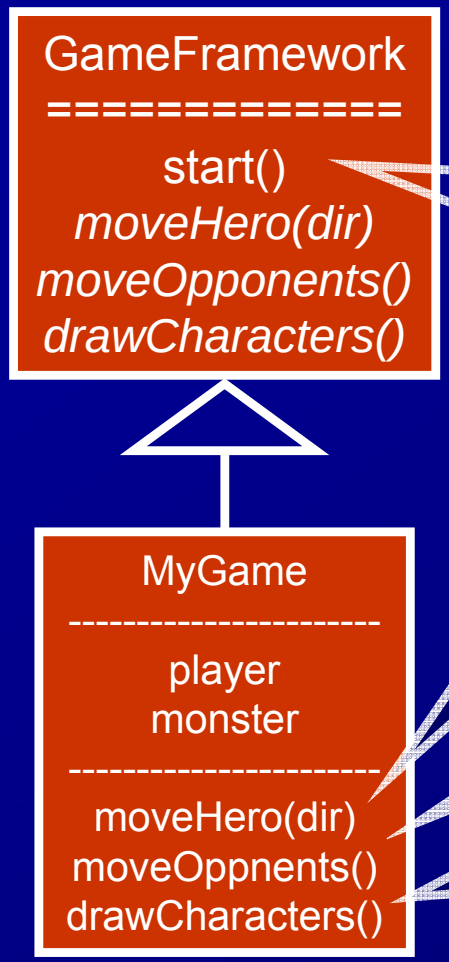
問題2/2 (10分)

ゲームプログラムの共通部分を部品として提供せよ

- 右の黄枠外が共通部分
- 言語は何でもよい (OOP)
- 提供された部品を使う人が何をすればよいのかが分かるように

```
byte[] imageBuf1 = ..., iageBuf2 = ...;
int previousKey = 0;
while (true) {
    int c = キーボード状態の読み取り;
    int key = cとpreviousKeyを使って
                「現在押されているキー」を決定;
    previousKey=key;
    switch (key) {
    case Left: 左に動かす; break;
    case Right: 右に動かす; break;
            ...;
    }
    敵を動かす;
    clear(imageBuf1);
    キャラクタをimageBuf1に描く;
    transferToDisplay(imageBuf1);
    imageBuf1とimageBuf2を入れ替え;
    一定時間待つ;
}
```

フレームワークの例



主人公の位置をdirに従って更新

化物の位置をランダムに更新

主人公と化物を描画

```
byte[] imageBuf1 = ..., iageBuf2 = ...;
int previousKey = 0;
while (true) {
    int c = キーボード状態の読み取り;
    int key = cとpreviousKeyを使って
                「現在押されているキー」を決定;
    previousKey=key;
    this.moveHero(key);
    this.moveOpponents();
    clear(imageBuf1);
    drawCharacters(imageBuf1);
    transferToDisplay(imageBuf1);
    imageBuf1とimageBuf2を入れ替え;
    一定時間待つ;
}
```

参考文献

[SDNP03] Schärli, Nathanael, Stéphane Ducasse, Oscar Nierstrasz, and Andrew P. Black. "Traits: Composable units of behaviour." In ECOOP 2003–Object-Oriented Programming, pp. 248-274, 2003.

[BC90] Bracha, Gilad, and William Cook. "Mixin-based inheritance." In *Proceedings of OOPSLA/ECOOP,* pp.303-311, 1990.

[Alexander77] Alexander, Christopher, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977

[GoF94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994