第7章
コレクション

この章の目的

◆コレクションとは何か
◆コレクションの正しい扱い方を知る

内容

内容とキーワード

- ◆車輪の再開発
- ◆ループ処理中の条件分岐ネスト
- ◆低凝集なコレクション処理
- ◆ファーストクラスコレクション
  - ■変更できないコレクション

車輪の再開発

車輪の再開発

既に存在する技術などを，知らないか，もしくは意図的に無視して，新たに同じようなものを作り出してしまうことを車輪の再開発と呼ぶ.

◆わざわざ自前でコレクション処理を実装してしまう

■以下は所持品の中に牢獄の鍵があるか調べるコード

```
boolean hasPrisonKey = false;

for (Item each : item) {
    if (each.name.equals(“牢獄の鍵”)) {
        hasPrisonKey = true;
        break;
    }
}
```

既にライブラリで実装されているような内容をわざわざ書いてしまう

まずはライブラリを探してみよう

実は一行で書けてしまう

```
boolean hasPrisonKey =
       item.stream() .anyMatch(item -> item.name.equal(“牢獄の鍵”));
```

ループ処理中の条件分岐ネスト

ループ処理中の条件分岐ネスト

◆メンバーの状態を調べ，毒状態の場合にヒットポイントを減少させるロジックは，，，

```
for (Member member : members) {
    if (0 < member.hitPoint) {
        if (0 < member.containsState(StateType.poison)) {
            member.hitPoint -= 10;
            if (member.hitPoint <= 0) {
                member.hitPoint = 0;
                member.addState(StateType.dead);
                member.removeState(StateType.poison);
            }
        }
    }
}
```

なんかごちゃごちゃしている

ループ処理中の条件分岐ネスト

早期continueによって条件分岐のネストをほどこう．また早期breakとうまく使い分けよう．

◆メンバーの状態を調べ，毒状態の場合にヒットポイントを減少させるロジックはこう書く．

```
for (Member member : members) {
    if (member.hitPoint == 0) continue;
    if (!member.containsState(StateType.poison)) continue;

    member.hitPoint -= 10;

    if (0 < member.hitPoint) continue;

    member.hitPoint = 0;
    member.addState(StateType.dead);
    member.removeState(StateType.poison);
}
```

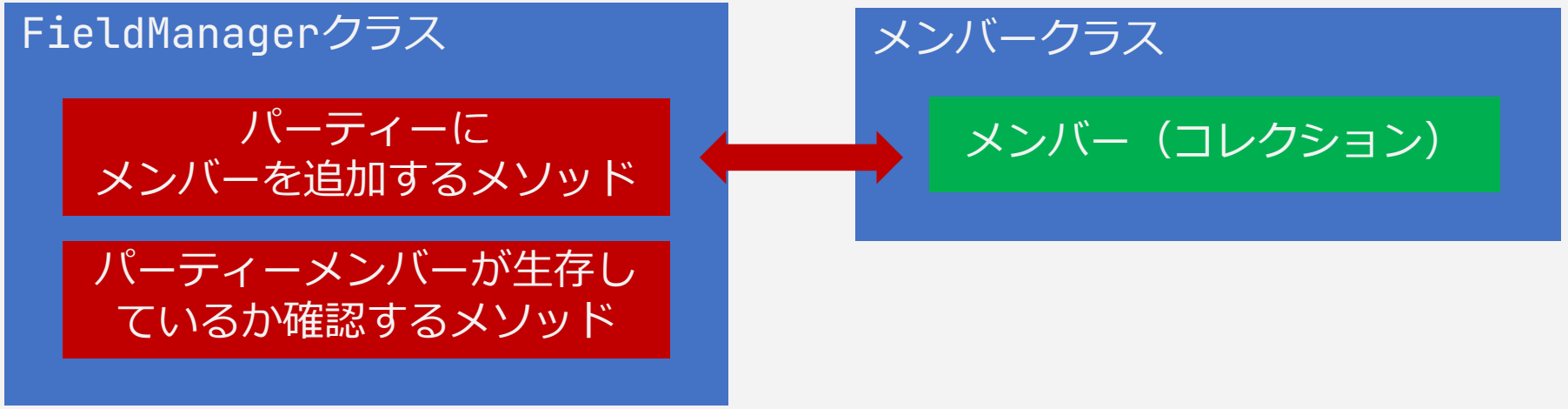
早期continue

ネストを解消できましたね

低凝集なコレクション処理

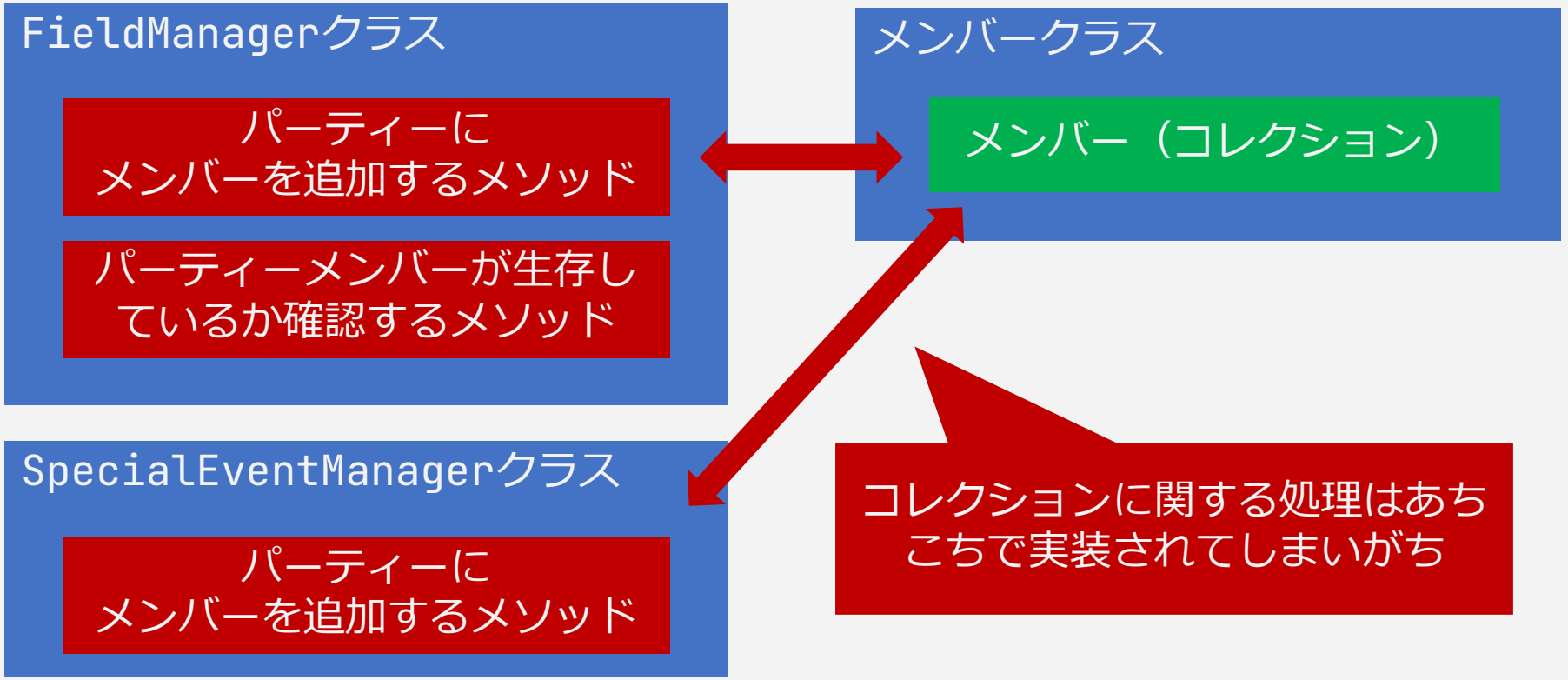
低凝集なコレクション処理

◆例えばパーティーのメンバーについて，以下の実装を考えてみる



低凝集なコレクション処理

◆例えばパーティーのメンバーについて，以下の実装を考えてみる



ファーストクラスコレクション

ファーストクラスコレクション

ファーストクラスコレクションの考え方を実践

第3章で学習　：　良いクラスの構成要素

- インスタンス変数
- インスタンス変数を不正状態から防御し，正常に操作するメソッド

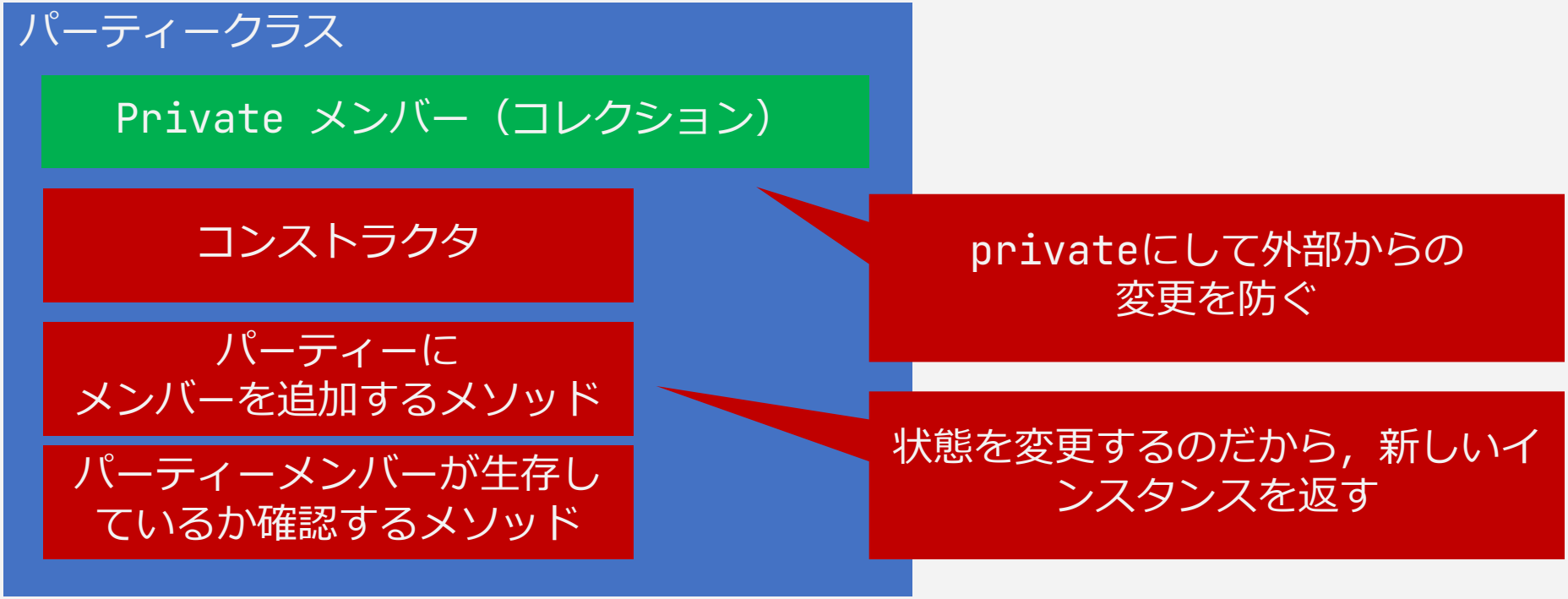
＋

新たに以下を加える

- コレクション型インスタンス変数
- コレクション型変数を不正状態から防御し，正常に操作するメソッド

ファーストクラスコレクション

◆コレクション型インスタンス変数ももれなく関連するクラスに所属させよう.



変更できないコレクション

◆以下のようなコードを書くと，一体どうなるでしょうか？

```
class Party {
    private List<Member> members;
    // 省略
    List<Member> members() {
        return members;
    }
}
```

privateなので外部からはアクセスできないはず？

変更できないコレクション

◆以下のようなコードを書くと，一体どうなるでしょうか？

```
class Party {
    private List<Member> members;
    // 省略
    List<Member> members() {
        return members;
    }
}
```

```
members = party.members();
members.add(newMember);

…

members.clear();
```

あれ，変更できてしまいますね？

変更できないコレクション

◆以下のようなコードを書くと，一体どうなるでしょうか？

```
class Party {
    private List<Member> members;
    // 省略
    List<Member> members() {
        return members;
    }
}
```

```
members = party.members();
members.add(newMember);

…

members.clear();
```

このメソッドがインスタンス変数をそのまま外部へ渡しているため，実質公開されていると同じになってしまっている．

変更できないコレクション

◆外部にコレクションを渡すときは不変にしよう.

```
class Party {
    private List<Member> members;
    // 省略
    List<Member> members() {
        return members.unmodifiableList();
    }
}
```

unmodifiableList()で得たコレクションは要素の追加や削除ができない.

```
members = party.members();
members.add(newMember);

…

members.clear();
```

コンパイルエラー！

次の章に向けて

◆コレクションの扱い方について学んだ。
◆車輪の再開発をしていなか？

◆次章では密結合と責務について学ぶ