Programming II

第6回 カプセル化（第13章）

塩澤秀和

6.1 カプセル化

- カプセル化とは（p.492）
  - 別名「情報隠蔽（いんぺい）」＝ 情報を隠す機能
  - クラスの内部処理の詳細を外部から隠して，カプセルに入れるように保護し，使用方法だけを外部に公開する
  - Javaではアクセス修飾子（private等）で制御する
- カプセル化によるメリット
  - 大規模または複数人による開発で特に有効
  - 公開して共有する部分を決めることで，分担が明確化し，自由に修正できる部分とできない部分が切り分けられる
  - クラスの内部状態を参照・変更する経路を制限し，そこでミスや不正がないか妥当性をチェックできる

6.2 アクセス修飾子

- クラスメンバのアクセス制御（p.498）
  - クラスは，各フィールド/メソッドの公開範囲を指定できる
  - たいてい，フィールドはprivate，メソッドはpublicにする

修飾子	意味	公開範囲	UML
public	公開	全てのクラス（プログラム全体）	＋
protected	限定公開	自分の子クラスか，同じパッケージのクラス	＃
（なし）		同じパッケージのクラス（Java特有）	
private	非公開	自分のクラス内	ー

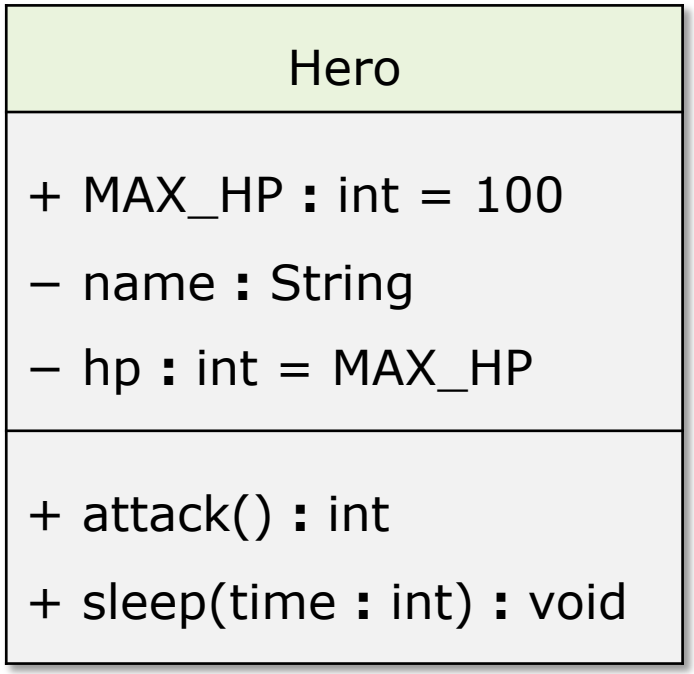
- クラス自体のアクセス制御（p.512）
  - publicをつけると…　プログラム全体に公開
  - publicをつけないと…　同じパッケージにだけ公開

6.3 クラス定義とクラス図

- アクセス制御を指定

Hero.java

```
public class Hero {

    public final int MAX_HP = 100;

    private String name;
    private int hp = MAX_HP;

    public int attack() {
        // 省略
    }

    public void sleep(int time) {
        // 省略
    }
}
```



6.4 アクセス修飾子の使用例

```
public class Robot {
    private String name;

    public Robot(String name) {
        this.name = name;
    }

    public void hello() {
        System.out.println("コンニチハ " + this.name + " デス");
    }
}

public class Main {
    public static void main(String[] args) {
        Robot robo1 = new Robot("ロボタ");
        // robo1.name = "ロボコ";  // できない！
        robo1.hello();
    }
}
```

6.5 ゲッター/セッターメソッド

- メソッドを経由したフィールド操作（p.503）
  - フィールドをprivateにし，専用メソッドからアクセスさせる
  - 読み出し専用や書き込み専用の情報を実現できる
- ゲッター（getter）（p.506）
  - 外部から，フィールドの値を取得（get）するためのメソッド
  - 慣習で，「データ型 getフィールド名()」と定義する
- セッター（setter）（p.506）
  - 外部から，フィールドに値を設定（set）するためのメソッド
  - 慣習で，「void setフィールド名(データ型 引数)」と定義
  - その際，設定値の妥当性をチェックできる

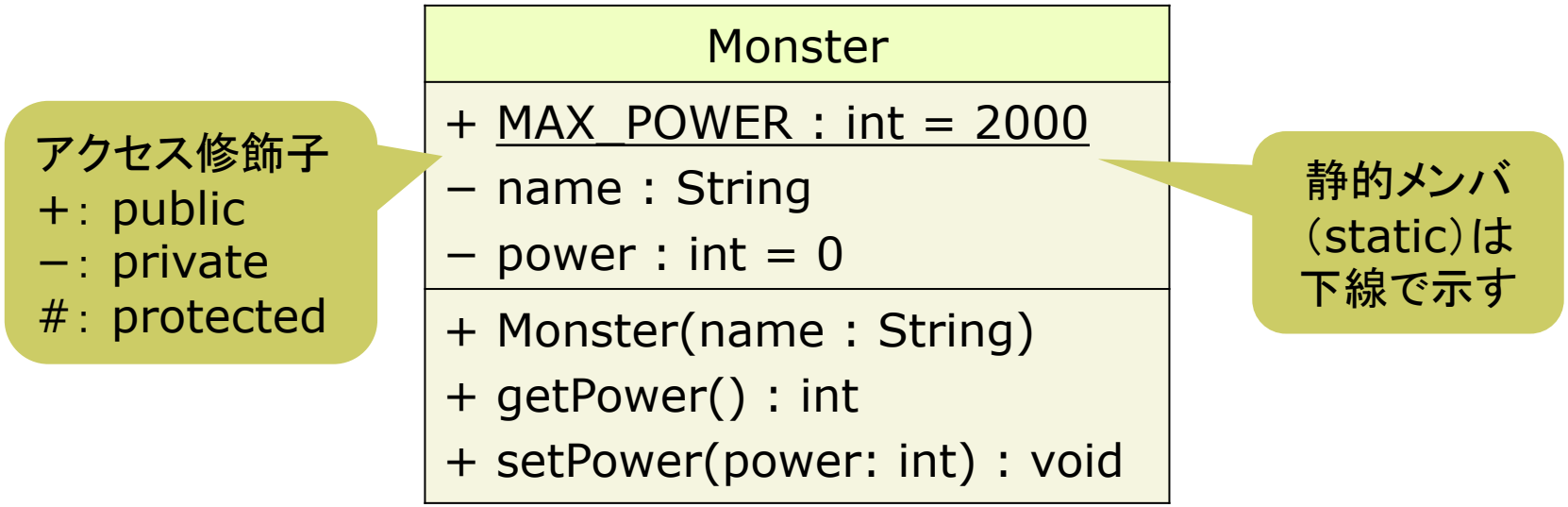
6.6 ゲッターとセッターの例

```
public class Monster {
    public static final int MAX_POWER = 2000;  // Javaの定数の作り方
    private String name;
    private int power = 0;

    public Monster(String name) { this.name = name; }

    public int getPower() { return this.power; }  // ゲッター

    public void  setPower(int power) {  // セッターによる妥当性チェック
        if (power < 0 || power > Monster.MAX_POWER) {
            throw new IllegalArgumentException("引数が不正です");
        }
        this.power = power;
    }
}
```

「例外」事象に対して，プログラムを終了させる方法
（Illegal＝不正な，Argument＝引数，Exception＝例外）
第13回の授業で説明するので今回はこのまま入力

6.7 ゲッターとセッターの例（続き）

```
public class Main {
    public static void main(String [] args) {
        Monster dragon = new Monster("ドラゴン");
        dragon.setPower(1500);  // ここを7000や-100に変えて実行してみよ
        System.out.println(dragon.getPower());
    }
}
```



演習

- 6.6～6.7
  - 最終的なソースコードと実行結果を提出

6.8 静的メンバ

- staticフィールド/メソッド（p.540）
  - 個別のインスタンスではなく，クラスが持つメンバ
  - インスタンスが1つも生成されていなくても，存在する
  - 定数の定義によく使われる（public static final）（p.546）
  - 例： Math.max(), Math.PI, public static void main()
- 静的メンバの利用（p.543）
  - 通常は，「クラス名.メンバ名」で参照する
  - 「インスタンス変数.メンバ名」でも，同じものを指す
  - 静的メソッドの中では，「this」（暗黙を含む）は使えない
- クラス図における静的メンバ
  - 下線を引いて示す

第6回 まとめ

- カプセル化
- アクセス修飾子
  - public 公開
  - private 非公開
- ゲッターとセッター
  - データ型 getフィールド名()
  - void setフィールド名(データ型 引数)
- 静的メンバ
  - static
  - 定数定義(public static final)
  - クラス図の描き方