Java講習

後期第1回

何やるの？

- 簡単なRPGを作りながらオブジェクト指向を勉強します
- 1年生にとっては, かなり飛躍した内容で難しいと感じるかも...
- 完全に理解する必要は無いです(というか僕もよく分からん)
  頑張って「なんとなく」使えるようになりましょうー

今日やること

1. ポエム(良いコードとは)
2. 今の知識でRPGを作ってみよう！
3. オブジェクト指向プログラミングの発想

良いコード？？

リーダブルコードだの, UNIX哲学だのを見ると

良いコード？？

リーダブルコードだの, UNIX哲学だのを見ると

1. 他の人が最短時間で理解できる
2. プログラミング初学者でも理解できるくらい単純
3. 将来的に苦労しない

の3つが良いコードの条件なのかなあと個人的には思います

※僕が要約してまとめただけです

※僕はこの条件を満たすコードを書ける気がしません
　これが満たせるコードが書ける人は , 超超スーパープログラマーだと思います

オブジェクト指向とは？

オブジェクト指向は「将来的に苦労しないため」のアプローチの1つ

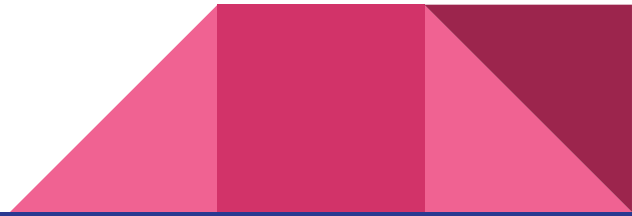
言い換えれば保守・拡張が容易になる

ゲーム作りたいなら絶対理解しないと無理！！(Unityでも同じ)

今の知識でRPG作れるんじゃね？

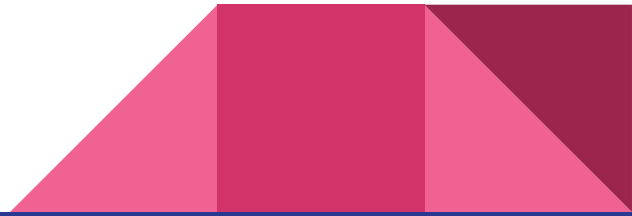
…でもそんなの使わなくても簡単なゲームなら作れるのでは？

⇒実際にやってみた



想定

- ターミナル上で動かす
- ターン制バトル
- 味方3人, 敵2人
- それぞれHP, MP, 攻撃力(ATK), 防御力(DEF), 速さ(SPEED)を持つ



まずは味方1を定義

```
int mikata1Hp = 150;
int mikata1Mp = 15;
int mikata1Atk = 20;
int mikata1Def = 10;
int mikata1Speed = 5;
```

全員分

```
int mikata1Hp = 150;
int mikata1Mp = 15;
int mikata1Atk = 20;
int mikata1Def = 10;
int mikata1Speed = 5;

int mikata2Hp = 100;
int mikata2Mp = 20;
int mikata2Atk = 5;
int mikata2Def = 12;
int mikata2Speed = 10;

int mikata3Hp = 80;
int mikata3Mp = 80;
int mikata3Atk = 80;
int mikata3Def = 80;
int mikata3Speed = 80;

int enemy1Hp = 200;
int enemy1Mp = 200;
int enemy1Atk = 200;
int enemy1Def = 200;
int enemy1Speed = 200;

int enemy2Hp = 250;
int enemy2Mp = 250;
int enemy2Atk = 250;
int enemy2Def = 250;
int enemy2Speed = 250;
```

味方1が敵に攻撃するときの処理

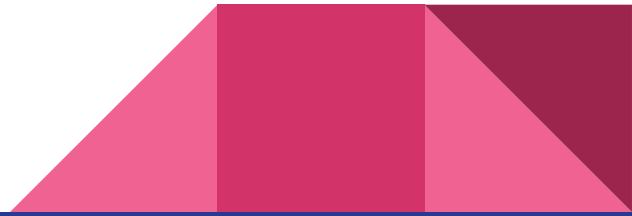
```
/* 味方1の攻撃処理 */
int target = sc.nextInt();  // どの敵に攻撃するか
if (target == 1) {
    enemy1Hp = mikata1Atk - enemy1Def;
} else if (target == 2) {
    enemy2Hp = mikata1Atk - enemy2Def;
}
```

全員分

```
/* 味方1の攻撃処理 */
target = sc.nextInt();  // どの敵に攻撃するか
if (target == 1) {
    enemy1Hp = mikata1Atk - enemy1Def;
} else if (target == 2) {
    enemy2Hp = mikata1Atk - enemy2Def;
}

/* 味方2の攻撃処理 */
target = sc.nextInt();  // どの敵に攻撃するか
if (target == 1) {
    enemy1Hp = mikata2Atk - enemy1Def;
} else if (target == 2) {
    enemy2Hp = mikata2Atk - enemy2Def;
}

/* 味方3の攻撃処理 */
target = sc.nextInt();  // どの敵に攻撃するか
if (target == 1) {
    enemy1Hp = mikata3Atk - enemy1Def;
} else if (target == 2) {
    enemy2Hp = mikata3Atk - enemy2Def;
}

/* 敵1の攻撃処理 */
target = sc.nextInt();  // どの味方に攻撃するか
if (target == 1) {
    mikata1Hp = enemy1Atk - mikata1Def;
} else if (target == 2) {
    mikata2Hp = enemy1Atk - mikata2Def;
} else if (target == 3) {
    mikata3Hp = enemy1Atk - mikata3Def;
}

/* 敵2の攻撃処理 */
target = sc.nextInt();  // どの味方に攻撃するか
if (target == 1) {
    mikata1Hp = enemy1Atk - mikata1Def;
} else if (target == 2) {
    mikata2Hp = enemy1Atk - mikata2Def;
} else if (target == 3) {
    mikata3Hp = enemy1Atk - mikata3Def;
}
```

無理

これだけで心が折れてきたのが分かるように, 無理です
なぜ無理なのか？



原因1：関連したデータが独立している

下の画像のmikata1___という変数は全て味方1に関するデータ
…だけど変数がそれぞれ独立しているので，関連したものだと分かりにくい！

それに味方1も2も同じ種類のデータを持ってる…

```
// 味方1の情報
int mikata1Hp = 150;
int mikata1Mp = 15;
int mikata1Atk = 20;
int mikata1Def = 10;
int mikata1Speed = 5;

// 味方2の情報
int mikata2Hp = 100;
int mikata2Mp = 20;
int mikata2Atk = 5;
int mikata2Def = 12;
int mikata2Speed = 10;
```

原因2 : 見えてる変数が多い

ぱっと見で, 変数が非常に多い
これは**精神衛生上良くない！！！！！！！**

しかも, これらのパラメータが自由にいじれてしまう！

atk, def, speedはどこでも使うわけではない
　　⇒これらを使うメソッドからのみ見えればいい！

```
int mikata1Hp = 150;
int mikata2Hp = 100;
int mikata3Hp = 80;

int mikata1Mp = 150;
int mikata2Mp = 100;
int mikata3Mp = 80;

int mikata1Atk = 150;
int mikata2Atk = 100;
int mikata3Atk = 80;

int mikata1Def = 150;
int mikata2Def = 100;
int mikata3Def = 80;

int mikata1Speed = 150;
int mikata2Speed = 100;
int mikata3Speed = 80;

int enemy1Hp = 200;
int enemy2Hp = 250;

int enemy1Mp = 200;
int enemy2Mp = 250;

int enemy1Atk = 200;
int enemy2Atk = 250;

int enemy1Def = 200;
int enemy2Def = 250;

int enemy1Speed = 200;
int enemy2Speed = 250;
```

原因3：似たロジックを沢山書く必要がある

「変数名が違うだけで処理は同じ」という箇所が多い

メソッドにまとめればいいんだけど、パラメータの変数が多いので難しい。

```
/* 味方3の攻撃処理 */
target = sc.nextInt();  // どの敵に攻撃するか
if (target == 1) {
    enemy1Hp = mikata3Atk - enemy1Def;
} else if (target == 2) {
    enemy2Hp = mikata3Atk - enemy2Def;
}

/* 敵1の攻撃処理 */
target = sc.nextInt();  // どの味方に攻撃するか
if (target == 1) {
    mikata1Hp = enemy1Atk - mikata1Def;
} else if (target == 2) {
    mikata2Hp = enemy1Atk - mikata2Def;
} else if (target == 3) {
    mikata3Hp = enemy1Atk - mikata3Def;
}
```

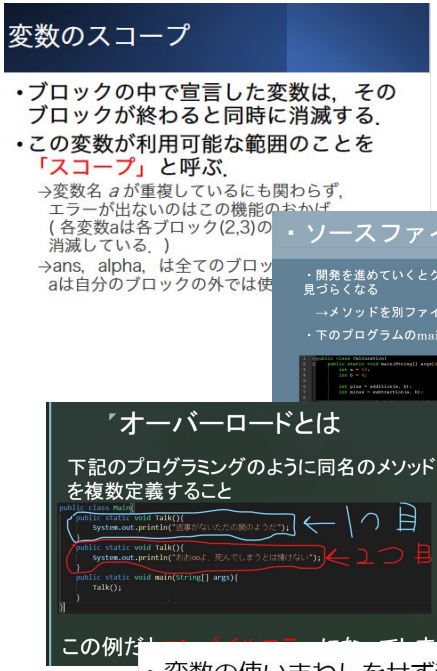
解決したいけど…

従来の方式でこれらの問題を解決するために、

- スコープを上手いこと制限する
- ソースファイルを上手いこと分割する
- 処理を独立したメソッドに上手く抽出する
- 一部の変数をimmutable(変更不能)にする

などの手を尽くしてもカバーしきれない部分がある

- HPなどのデータは結局外に置く必要がある
- 似たような処理やデータ群をまとめる術が無い



オブジェクト指向プログラミングの発想

結局, オブジェクト指向プログラミング(OOP)とは何なのかというと

データを「まとめて, 隠して, たくさん作る」

ことで、先ほどのような問題を解決することです

※これだけでなく, 他にも沢山の機能があります .

「まとめる」

関連したデータやメソッドをまとめて…

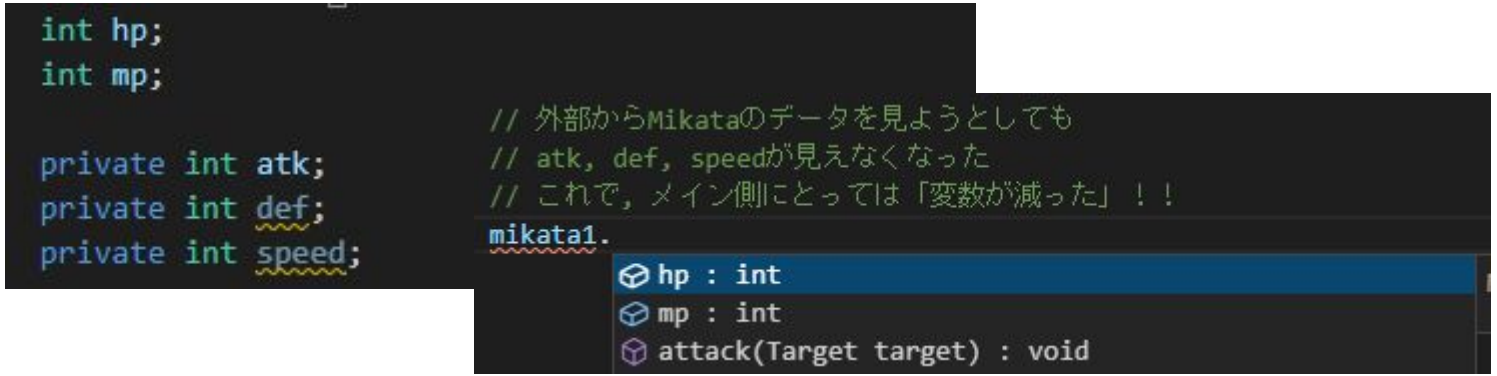
```
public class Mikata {
    // 今までのパラメータや攻撃処理を
    // Mikataというものにまとめた！！

    int hp;
    int mp;
    int atk;
    int def;
    int speed;

    void attack(Target target) {
        target.hp = this.atk - target.def;
    }
}
```

```
mikata1.hp;     // mikata1のHP
mikata1.mp;     // mikata1のMP
mikata1.atk;    // mikata1のATK
mikata1.def;    // mikata1のDEF
mikata1.speed;  // mikata1のSPEED
```

※classとかTargetとかは気にしないでください

「隠す」

外部から不必要なデータを隠して…



「たくさん作る」

「まとめて, 隠した」ものを複製する

```
// Mikataをもとにして3人の味方を作成
// この3つそれぞれが独自のHP, MP, ATK, DEF, SPEEDを持つ！
Mikata mikata1 = new Mikata();
Mikata mikata2 = new Mikata();
Mikata mikata3 = new Mikata();

mikata1.hp;     // mikata1のHP
mikata2.hp;     // mikata2のHP
mikata3.hp;     // mikata3のHP
```

恐るべしOOP

これによって,

1. 関連したデータをまとめて扱える
2. メイン側から見て, 変数の数が減る
3. ロジックの共通化ができる

という効用があります。すごい！

具体的なことは次回以降やります

今後の方針

第2回 : クラスの定義(勇者, 魔法使い, … , 敵などを作る)
第3回 : インスタンスメソッド(攻撃処理を作る)
第4回 : ライブラリとクラスメソッド(メイン画面・戦闘画面を作る)
第5回 : 継承(良い例がない)
第6回 : ポリモーフィズム(メイン処理を作る)
第7回 : カプセル化(色々隠す)
第8回 : 総仕上げ

ソースコードはGitHubに上げます

参考文献

「オブジェクト指向でなぜ作るのか」平澤 章
「リーダブルコード」Dustin Boswell, Trevor Foucher, 角 征典(訳)
「Java本格入門」谷本 心, 阪本 雄一郎, 岡田 拓也, 秋葉 誠, 村田 賢一郎
「Linux kernel coding style」Linus Torvalds(原著作者)
http://archive.linux.or.jp/JF/JFdocs/kernel-docs-2.6/CodingStyle.html