

Java講習

後期第2回

前回やったこと

RPGを作ろうとしたけど無理だった

⇒ オブジェクト指向の導入によって解決しよう！

オブジェクト指向のポイントの1つは

データを「まとめて、隠して、たくさん作る」



今回やること

1. クラスとインスタスの紹介・使い方
2. なんとなくのイメージ



まずはこの惨状を解決したい

```
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 = enemy2Atk - mikata1Def;
} else if (target == 2) {
    mikata2Hp = enemy2Atk - mikata2Def;
} else if (target == 3) {
    mikata3Hp = enemy2Atk - mikata3Def;
}
```

原因1：関連したデータが独立している(再掲)

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;
```

「まとめる」(再掲)

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

「クラス」の雑な説明

データ(変数やメソッド)をまとめるためには「クラス」を使う

```
public class Mikata {  
    String name;  
  
    int hp;  
    int mp;  
    int atk;  
    int def;  
    int speed;  
  
    void printName() {  
        System.out.println("name : " + name);  
    }  
}
```

詳しい話は後でやります

クラス宣言方法

```
(public) class <クラス名> {  
    .....  
}
```

- ・クラスの中には

データ(属性)やメソッド
を書く

- ・先頭のpublicはあってもなくてもいい(重要ではある)
- ・クラス名の先頭は**必ず大文字**

```
public class Mikata {  
    String name;  
  
    int hp;  
    int mp;  
    int atk;  
    int def;  
    int speed;  
  
    void printName() {  
        System.out.println("name : " + name);  
    }  
}
```

データ(属性)

メソッド

クラスの使い方

基本的にjavaでは1つのclassに1つのファイルが対応する
なので複数のファイルを作ることになる



ディレクトリ構造

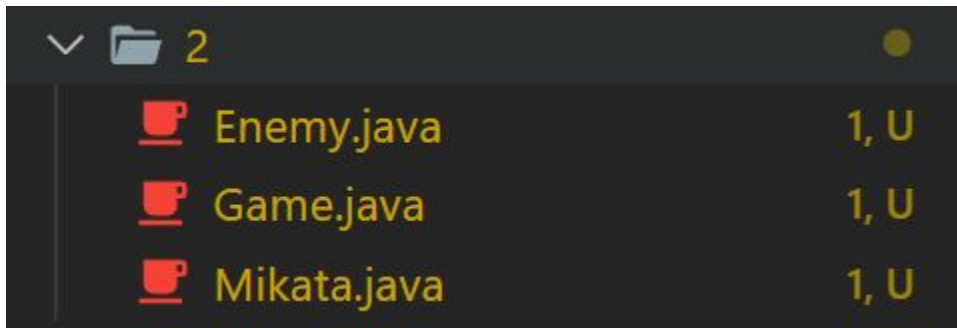
- ・基本的に使うファイルは同じディレクトリに入れる

Game.java - mainメソッドがあるクラス

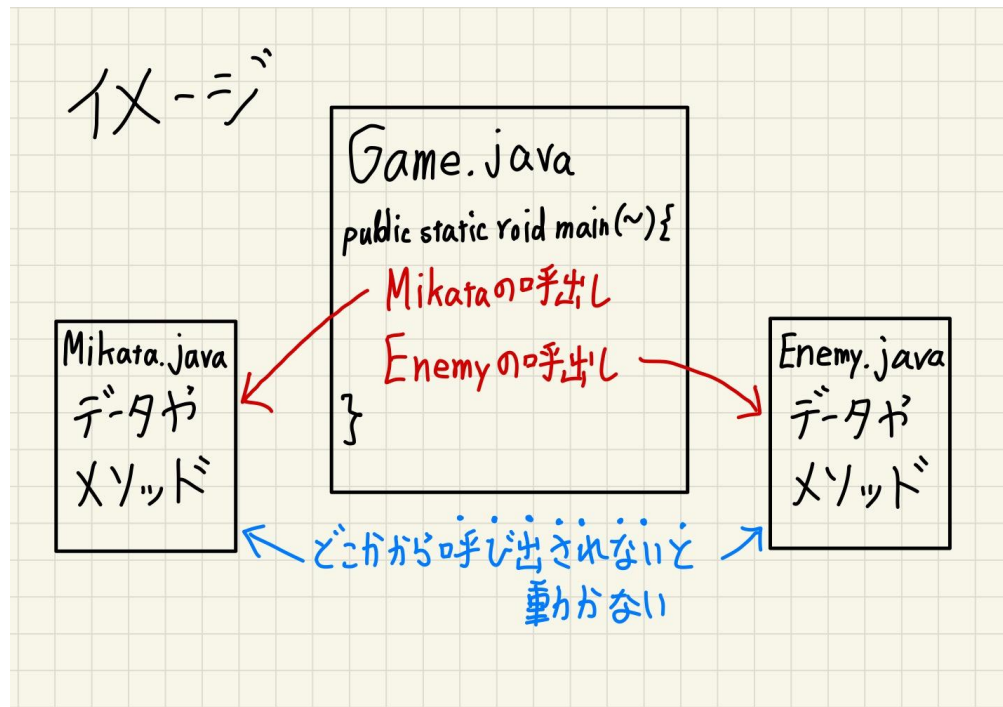
Mikata.java - 味方を表すクラス

Enemy.java - 敵を表すクラス

⇒ Mikata, EnemyはGame(main)から呼び出して使う



mainのイメージ



Mikataの定義

今回はMikataクラスだけ作ってみる

Mikataが持つべきデータは

- 名前(String)
- HP(int)
- MP(int)
- 攻撃力(int)
- 防御力(int)
- 速さ(int)

と考えると、クラスを定義

```
public class Mikata {  
    String name;  
    int hp;  
    int mp;  
    int atk;  
    int def;  
    int speed;  
}
```

MainからMikataを呼び出そう

MainメソッドのあるGame.javaで実際に呼び出してみる

<クラス名> <変数名> = **new** <クラス名>();

下のm1のような変数を**インスタンス**という.

```
public static void main(String[] args) {  
    Mikata m1 = new Mikata();  
}
```

データの呼び出し

データを呼び出すときは「.」を使う

例) m1のHPは, m1.hpと指定することで取得できる

```
Mikata m1 = new Mikata();

m1.name = "AAAA";
m1.hp    = 100;
m1.mp    = 20;
m1.atk   = 10;
m1.def   = 15;
m1.speed = 5;

System.out.println("name : " + m1.name);
System.out.println("hp : "   + m1.hp);
System.out.println("mp : "   + m1.mp);
System.out.println("atk : "  + m1.atk);
System.out.println("def : "  + m1.def);
System.out.println("speed : " + m1.speed);
```

```
name : AAAA
hp : 100
mp : 20
atk : 10
def : 15
speed : 5
```

クラス・インスタンスのイメージ

クラスとインスタンスって結局何？



「クラスの定義」に注目してみる

Mikata.javaを例に考えてみると, このMikataは

- String型の変数name
- int型の変数hp
- int型の変数mp
- int型の変数atk
- int型の変数def
- int型の変数speed

を持っているものと見るができる.

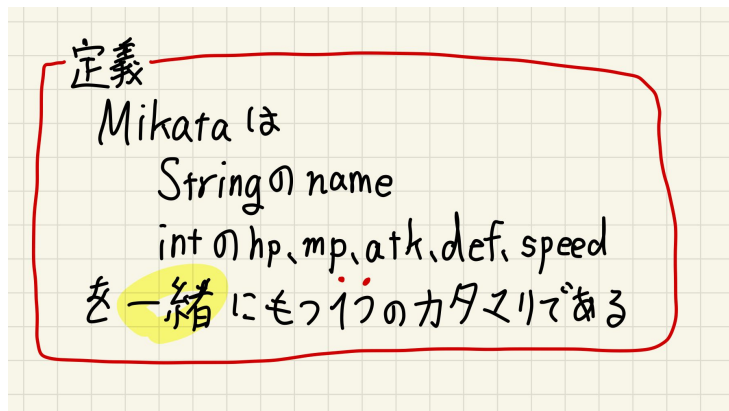
```
public class Mikata {  
    String name;  
    int hp;  
    int mp;  
    int atk;  
    int def;  
    int speed;  
}
```


クラス = 設計図

Mikataクラスの定義は,

Mikataが「**どのようなデータを持ち, どのようなメソッドを持つか**」
を決めている.

Mikataという物(概念)の設計図と考えてもいい.



定義
Mikata は
String の name
int の hp, mp, atk, def, speed
を一緒にもつ1つのカタマリである

なんでそんな考え方するの？

なんで？回りくどくない？

いきなりMikataのような物体を考えればいいじゃん？

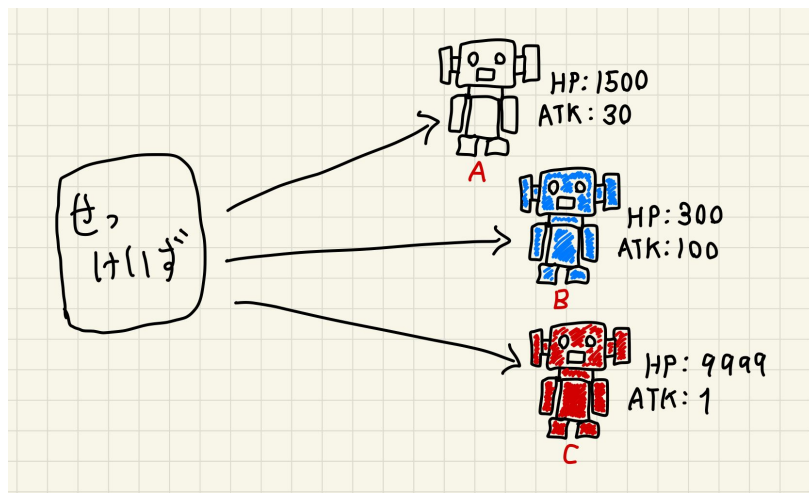
なんで設計図を作ってるの？



設計図であることのうれしみ

設計図であるということは...

これを基に**たくさんのMikataを作ることができる**ということ



「抽象的」であるからこそ

しかも, クラスの定義ではname, ... , speedを持っていると**だけ**定義されている.

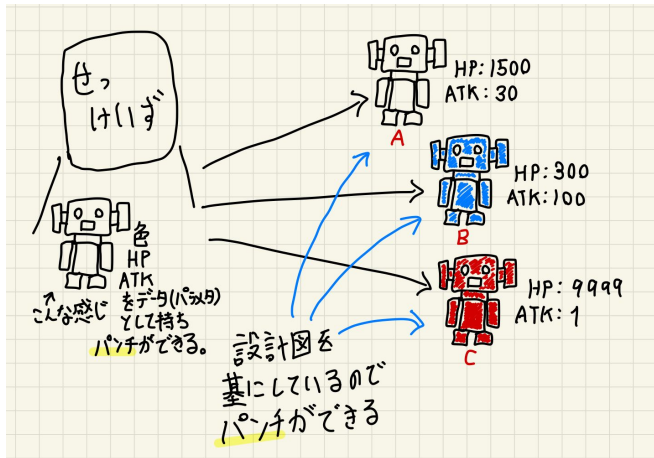
値が厳密に決められているわけではない！



「抽象的」であるからこそ

しかし, 設計図から生成したものが**それぞれ**が値を持つことができる!

例) 下の図の場合, A.HPは1500, B.HPは300, C.HPは9999



こうしてクラス(設計図)から作ったものを**インスタンス**という

「抽象的」であるからこそ

```
Mikata m1 = new Mikata();  
m1.hp     = 100;
```

```
Mikata m2 = new Mikata();  
m2.hp     = 90;
```

```
Mikata m3 = new Mikata();  
m3.hp     = 230;
```

```
System.out.println("m1.hp : " + m1.hp);  
System.out.println("m2.hp : " + m2.hp);  
System.out.println("m3.hp : " + m3.hp);
```

m1.hp : 100

m2.hp : 90

m3.hp : 230

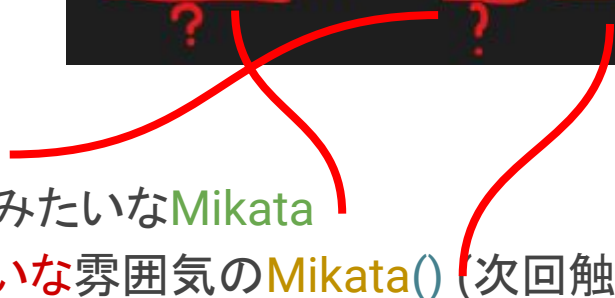
ここでコードを振り返ってみる

```
public class Mikata {  
    String name;  
    int hp;  
    int mp;  
    int atk;  
    int def;  
    int speed;  
}  
  
Mikata m1 = new Mikata();  
  
m1.name = "AAAA";  
m1.hp = 100;  
m1.mp = 20;  
m1.atk = 10;  
m1.def = 15;  
m1.speed = 5;  
  
System.out.println("name : " + m1.name);  
System.out.println("hp : " + m1.hp);  
System.out.println("mp : " + m1.mp);  
System.out.println("atk : " + m1.atk);  
System.out.println("def : " + m1.def);  
System.out.println("speed : " + m1.speed);
```

ここでコードを振り返ってみる

クラスの定義はなんとなく分かったけど、下の文は何？

```
Mikata m1 = new Mikata();
```



1. 謎のnew
2. 変数の型みたいなMikata
3. 関数みたいな雰囲気のMikata() (次回触れます)

new演算子

newはインスタンスを新しく生成することを表す演算子

```
m1 = new Mikata();
```

この場合, Mikataクラスを基にしてm1というインスタンスを新しく作ることを示している

まあ, おまじないと思ってもいいかも

クラスは変数の型？

あたかもMikata型の変数mikataを定義しているみたいですが...

```
Mikata m1 = new Mikata();
```

言ってしまうとその通りなんです

これはMikata型の変数mikataを宣言しています
(まるでint型の変数aを"int a = 1"と宣言するかのよう)



そもそもの話

int型を例に考えてみる

```
int a = 10;
```

これは, int型の変数aを宣言している(初期値が10)

当たり前ですが, int型の変数は**整数の値を1つ持つ**という特徴がある



そもそもの話

じゃあさっきの宣言を振り返ると

```
Mikata m1 = new Mikata();
```

これは, Mikata型の変数m1を宣言している(初期値が**新しい**インスタンス)

そしてMikata型は**Mikataのインスタンスを1つ持つ**という特徴がある
(インスタンスはname, hp, mp, ... , speedの6つの値を持つ)

→**int型**のときとほとんど同じ！



結局何が言いたいのか

クラスを定義すること ⇔ 型を定義すること

Mikataクラスは, 複数の型の変数やメソッドを束にして新しい型にしたものと捉えることができる

Mikata型の変数には, Mikata型のインスタンスが入る(本当は違う)



まとめ：クラスの定義

- クラスは**設計図**みたいなもの
- クラスの定義は**型の定義**
- どのようなデータを持つか
どのようなメソッドを持つかを定義する
- クラスを基にして, 異なるデータを持つ
インスタンスを**たくさん作れる**

```
public class Mikata {  
    String name;  
    int hp;  
    int mp;  
    int atk;  
    int def;  
    int speed;  
}
```

```
Mikata m1 = new Mikata();
```

演習課題

- ・Enemyクラスを作成しなさい。Enemyは以下のようなデータを持つ。
 - 名前, HP, MP, 攻撃力, 防御力, 速さ
- ・Enemyのインスタンスをmainで生成し, HPを表示させなさい.



参考文献

「オブジェクト指向でなぜ作るのか」平澤 章

「新分かりやすいJava入門編」川場 隆

「スッキリわかるJava入門」中山 清喬, 国本 大悟

「リーダブルコード」Dustin Boswell, Trevor Foucher, 角 征典(訳)

「New演算子 - Wikipedia」

<https://ja.wikipedia.org/wiki/New%E6%BC%94%E7%AE%97%E5%AD%90>

