

Java講習

後期第3回

前回やったこと

- ・クラスは設計図
- ・クラスの定義は型の定義
- ・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



前回の演習課題の回答

- ・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

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

今回やること

1. インスタンスメソッド
2. コンストラクタ
3. 変数の中身
4. 攻撃メソッドを作る



メソッド

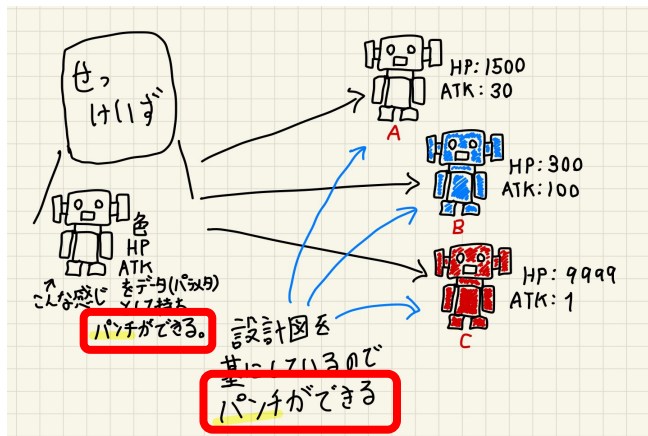
前回ちらっと話しましたが、
クラスの定義にはメソッドを含めることができます

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

設計図には「機能」も書ける

イメージは, 設計図に**機能(メソッド)**を定義しておけば,
それを基にして作った**実体(インスタンス)**もその機能が使える...という感じ

このようにインスタンスが持つメソッドを**インスタンスメソッド**という



インスタンスメソッドの定義方法

```
<返り値の型> <メソッドの名前>() {  
    処理...  
}
```

staticを付けないことに注意！

```
void printStatus() {  
    System.out.printf("%7s ", name);  
    System.out.printf("HP : %4d, ", hp);  
    System.out.printf("MP : %4d, ", mp);  
    System.out.printf("ATK : %4d, ", atk);  
    System.out.printf("DEF : %4d, ", def);  
    System.out.printf("SPEED : %4d\n", speed);  
}
```

インスタンスメソッドの呼び出し方

データを呼び出すのと同じように、「.」を使う

例) m1のステータス表示メソッドは, m1.printStatus()と指定して呼び出せる

```
Mikata m1 = new Mikata(  
    "KNIGHT", 100, 20, 10, 15, 12  
);  
m1.printStatus();
```

```
Java\Java-lecture\3_method>java Game  
KNIGHT HP : 100, MP : 20, ATK : 10, DEF : 15, SPEED : 12
```


ここまでのコード

そういえばRPGを作っているのですた！

第1回で説明したように, オブジェクト指向を用いて

データを「まとめて, 隠して, たくさん作る」

ことができているのでしょうか！？



Before -> After

```
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;
```



```
Mikata m1 = new Mikata();
m1.name = "KNIGHT";
m1.hp = 100;
m1.mp = 20;
m1.atk = 10;
m1.def = 15;
m1.speed = 12;

Mikata m2 = new Mikata();
m1.name = "WIZARD";
m1.hp = 85;
m1.mp = 35;
m1.atk = 5;
m1.def = 15;
m1.speed = 20;

Mikata m3 = new Mikata();
m1.name = "TANK";
m1.hp = 240;
m1.mp = 0;
m1.atk = 20;
m1.def = 40;
m1.speed = 5;

Enemy e1 = new Enemy();
m1.name = "ENEMY1";
m1.hp = 400;
m1.mp = 30;
m1.atk = 10;
m1.def = 0;
m1.speed = 15;

Enemy e2 = new Enemy();
m1.name = "ENEMY2";
m1.hp = 1200;
m1.mp = 0;
m1.atk = 50;
m1.def = 5;
m1.speed = 1;

m1.printStatus();
m2.printStatus();
m3.printStatus();
e1.printStatus();
e2.printStatus();
```

逆に行数が増えてるんですが...

ある部分に注目してみると

結局値を代入するために行数を使っている気がする...

```
Mikata m2 = new Mikata();  
m1.name = "WIZARD";  
m1.hp = 85;  
m1.mp = 35;  
m1.atk = 5;  
m1.def = 15;  
m1.speed = 20;
```

変数の初期化は大事！ だけど

変数に何も入っていないという状態は**非常にまずい**ので、
値の初期化をするのが当たり前

```
Mikata m1 = new Mikata();  
  
System.out.printf("m1.name : %s\n", m1.name);  
System.out.printf("m1.hp   : %f\n", m1.hp);
```

```
\Java\Java-lecture\3_method>java OldGame  
m1.name : null  
m1.hp   : 0
```

だけでもう少し何とかならんのか
⇒ **コンストラクタ**の利用

余談：Java公式のお言葉

It's not always necessary to assign a value when a field is declared. Fields that are declared but not initialized will be set to a reasonable default by the compiler. Generally speaking, this default will be zero or null, depending on the data type. Relying on such default values, however, is generally considered bad programming style.

→「変数は初期化しなくても0かnullが入るけど、それに頼るなよ～」

```
\Java\Java-lecture\3_method>java OldGame  
m1.name : null  
m1.hp   : 0
```

引用元：Oracle Java Documentation "Primitive Data Types"

コンストラクタの定義

インスタンスのデータの初期化を一気に行うために, **コンストラクタ**を使う
コンストラクタは**クラス内で**以下のように定義する

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

メソッドとの違いは**返り値の型が無い**こと



コンストラクタの定義

Mikataクラスの定義の中で、コンストラクタを定義する

基本的には初期化したい「クラスの属性」を引数として受け取り、
コンストラクタ内で初期化処理を行う。

```
Mikata(String name, int hp, int mp, int atk, int def, int speed) {  
    this.name = name;  
    this.hp   = hp;  
    this.mp   = mp;  
    this.atk  = atk;  
    this.def  = def;  
    this.speed = speed;  
}
```

this

「インスタンスのデータ」はthisを使って呼び出す(引数との区別)

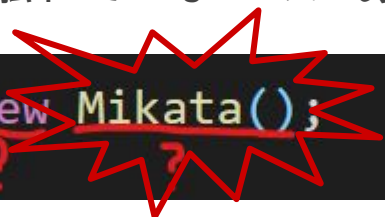
```
public class Mikata {  
    String name;  
  
    int hp;  
    int mp;  
    int atk;  
    int def;  
    int speed;  
  
    Mikata(String name, int hp, int mp, int atk, int def, int speed) {  
        this.name = name;  
        this.hp = hp;  
        this.mp = mp;  
        this.atk = atk;  
        this.def = def;  
        this.speed = speed;  
    }  
}
```

thisはインスタンス
自身を表す

コンストラクタの使い方

そういえば, インスタンス生成の部分で触れていないのがありました

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



The image shows a code snippet on a dark background. The text is 'Mikata m1 = new Mikata();'. The word 'Mikata' is green, 'm1' is blue, 'new' is purple, and 'Mikata()' is yellow. There are red underlines under 'Mikata', 'new', and 'Mikata()'. A red starburst graphic is drawn over the 'new' and 'Mikata()' parts. Below each underline, there is a red question mark.

これがコンストラクタの呼び出しみたいなものです

コンストラクタの呼び出し

Main側ではこのように呼び出す.

引数には初期値を指定する(関数呼び出しと同じ)

```
Mikata m1 = new Mikata("KNIGHT", 100, 20, 10, 15, 12);  
Mikata m2 = new Mikata("WIZARD", 85, 35, 5, 15, 20);  
Mikata m3 = new Mikata("TANK", 240, 0, 20, 40, 5);  
  
m2.printStatus();
```

```
\\Java\\Java-lecture\\3_method>java Game  
WIZARD HP : 85, MP : 35, ATK : 5, DEF : 15, SPEED : 20
```

ここまでのコード

試しに見てみると... すごいスッキリした！

```
public class Game {  
    public static void main(String[] args) {  
  
        Mikata m1 = new Mikata("KNIGHT", 100, 20, 10, 15, 12);  
        Mikata m2 = new Mikata("WIZARD", 85, 35, 5, 15, 20);  
        Mikata m3 = new Mikata("TANK", 240, 0, 20, 40, 5);  
  
        Enemy e1 = new Enemy("ENEMY1", 400, 30, 10, 0, 15);  
        Enemy e2 = new Enemy("ENEMY2", 1200, 0, 50, 5, 1);  
  
        m1.printStatus();  
        m2.printStatus();  
        m3.printStatus();  
        e1.printStatus();  
        e2.printStatus();  
        System.out.println();  
    }  
}
```

視聴者参加型企画

下のようなコードがあります.

これを実行すると, どのように出力されるでしょう! ?

```
/*  
    MusicianクラスはString型の属性nameを持ち,  
    nameを出力するprintName()メソッドを持つ  
*/  
Musician musician = new Musician();  
  
Musician john = musician;  
john.name = "John";  
Musician paul = musician;  
paul.name = "Paul";  
  
john.printName();  
paul.printName();
```

1. John
Paul

2. John
John

3. Paul
Paul

4. Ringo
George

視聴者参加型企画

下のようなコードがあります.

これを実行すると, どのように出力されるでしょう! ?

```
/*
    MusicianクラスはString型の属性nameを持ち,
    nameを出力するprintName()メソッドを持つ
*/
Musician musician = new Musician();

Musician john = musician;
john.name = "John";
Musician paul = musician;
paul.name = "Paul";

john.printName();
paul.printName();
```

Paul
Paul

1. John
Paul

2. John
John

3. Paul
Paul

4. Ringo
George

視聴者参加型企画

「当たり前だろ！(怒)」と思う人もいて当然な内容ですが、
分からなかった人もいるはずなので、理由を今から解説していきます



変数の中身

理解するためには,

musician, john, paulには何が入っているのかを知る必要があります.

```
Musician musician = new Musician();  
  
Musician john = musician;  
john.name = "John";  
Musician paul = musician;  
paul.name = "Paul";
```

「前回, 変数にはインスタンスが入ると説明したはずでは？」

前回のスライド再掲

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型のインスタスが入る(本当は違う)

変数にはインスタンスの「場所」が入る

```
Musician musician = new Musician();
```

まずはこの1行ですが,

`new Musician()`はインスタンスを新しく作ることだと説明しました.

ですが, `musician`に代入しているのはインスタンス本体ではなく,
「メモリのどこかに作られたインスタンスの"場所"」です.

メモリのどこかに ... ヒープ領域に作られます (ggって)

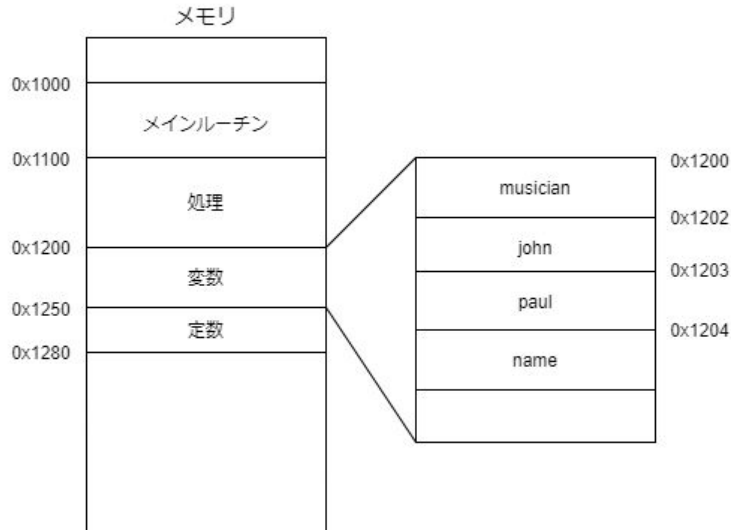


補足：メモリとは

プログラムはコンピュータの中の「メモリ」に書き込まれて実行される
処理はもちろん, 変数やメソッドも全部

メモリにはアドレスが割り振られており,
それで場所を指定することができる.
逆に言えば, **場所が分かれば変数の値を
取得することも可能.**

作ったインスタンスもメモリのどこかにあり,
その**アドレス**がmikataに代入される.

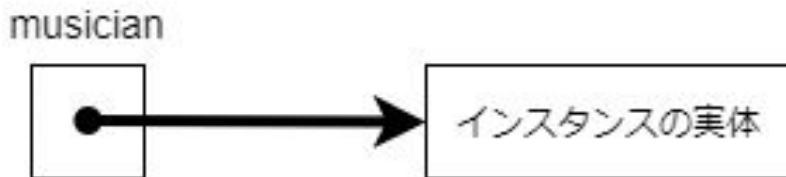


なぜ「場所」なのか

インスタンスの実体は結構大きく

これを変数にまるごと入れるのは少し都合が悪い

そこで「場所」を入れることで間接的に参照できるようにした



それを踏まえて先ほどのコードを試してみる

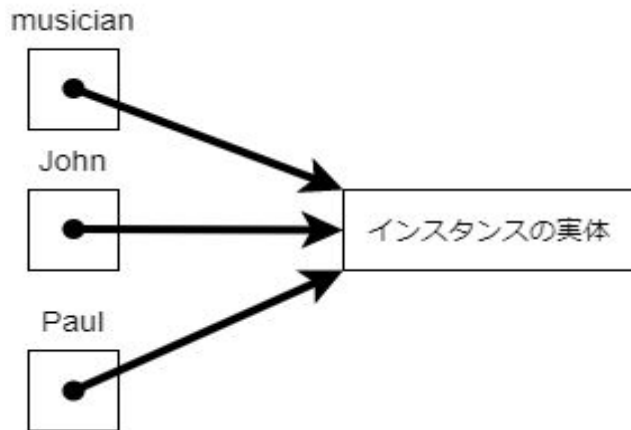
```
Musician musician = new Musician();  
  
Musician john = musician;  
john.name = "John";  
Musician paul = musician;  
paul.name = "Paul";
```

musicianには、新しく作ったMusicianのインスタンスの場所が入る。

john = musician, paul = musicianは、
Musicianのインスタンスの場所をコピーしていると考えれば...？

それを踏まえて先ほどのコードを試してみる

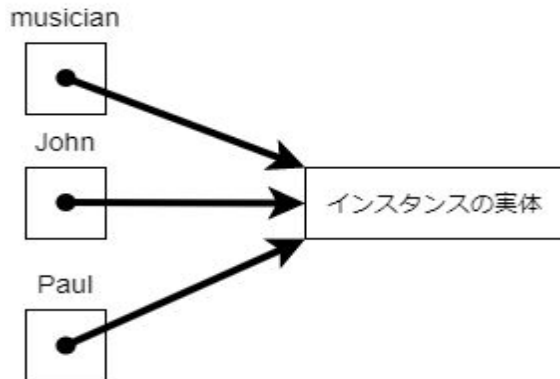
こんな感じ！



それを踏まえて先ほどのコードを見てみる

john.nameもpaul.nameも, 同じインスタンスのnameを指しているので,
一番最後に変更したpaul.name = "Paul"によって,
実行すると「Paul\nPaul\n」と出力される.

```
Musician musician = new Musician();  
  
Musician john = musician;  
john.name = "John";  
Musician paul = musician;  
paul.name = "Paul";  
  
john.printName();  
paul.printName();
```



今からやりたいこと

実は第1回から実装していたものがあります...

それは**攻撃処理**です！！！！

```
/* 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;
}
```

攻撃処理の要件

1. 味方は敵に, 敵は味方にしか攻撃できないとする
2. ダメージ計算は, **相手のHP** -= **自分の攻撃力** - **相手の防御力**
3. **相手のHP**がマイナスになってしまったら0に補正
4. **自分の攻撃力** - **相手の防御力**がマイナスになってしまったら0に補正
5. どれくらいダメージを与えたかを表示する
例) AAAはBBBに○○のダメージを与えた！

結構ややこしいですね...



何が必要か？

例えばMikataクラスにAttackメソッドを実装しようとするれば,
ダメージ計算と攻撃の対象の指定のために, 相手の情報が必要になる！

Mikataにとっては相手の情報はEnemy(のインスタンス)！！



インスタンスを渡す

相手の情報が必要なから,

```
void attack(Enemy enemy) { ... }
```

みたいな感じになる！(攻撃対象を引数に渡す)



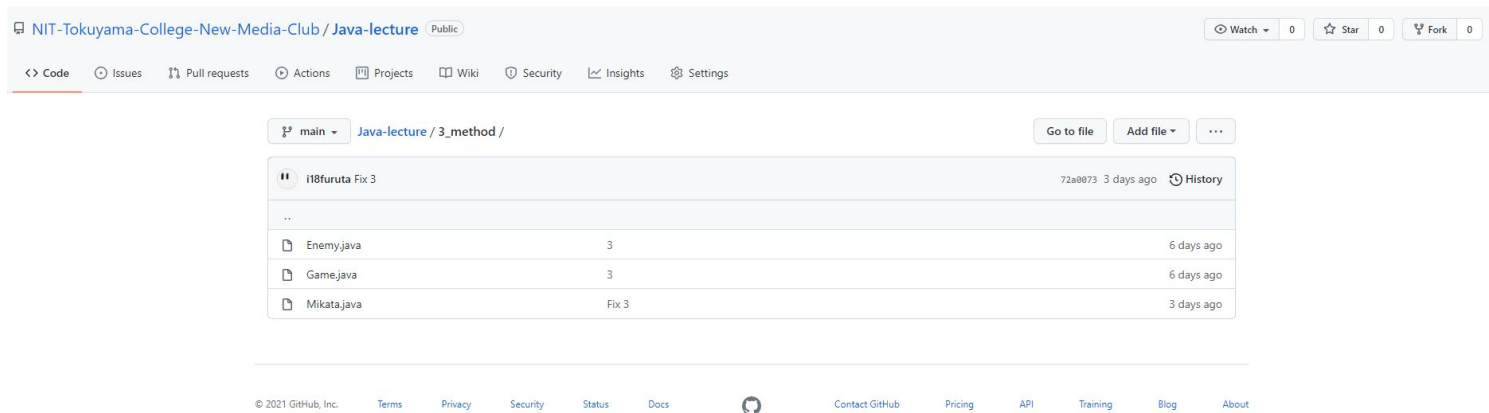
実装

引数にEnemyのインスタンスを渡し、
自身のステータスとenemyのステータスを使ってダメージ計算をしています。

```
void attack(Enemy enemy) {  
    int damage = this.atk - enemy.def;  
    System.out.println(  
        this.name + "は" + enemy.name + "に" + damage + "ダメージを与えた"  
    );  
    if (damage <= 0) {  
        damage = 0;  
    }  
  
    enemy.hp = enemy.hp - damage;  
    if (enemy.hp <= 0) {  
        enemy.hp = 0;  
    }  
    System.out.println(enemy.name + "のHPは残り" + enemy.hp);  
}
```

演習課題

実はGitHubにattack()メソッドを使う例も含めて上げてます.



The screenshot shows a GitHub repository page for 'NIT-Tokuyama-College-New-Media-Club / Java-lecture'. The repository is public. The navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The repository has 0 Watchers, 0 Stars, and 0 Forks. The main content area shows the file structure for the 'main' branch, specifically the 'Java-lecture / 3_method /' directory. The files listed are 'Enemy.java', 'Game.java', and 'Mikata.java', each with a commit hash and a timestamp. The commit history for the 'main' branch is also visible, showing a commit by 'i18furuta' with the message 'Fix 3'.

NIT-Tokuyama-College-New-Media-Club / Java-lecture Public

Watch 0 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main Java-lecture / 3_method /

Go to file Add file ...

i18furuta Fix 3 72a0073 3 days ago History

..

Enemy.java	3	6 days ago
Game.java	3	6 days ago
Mikata.java	Fix 3	3 days ago

© 2021 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

演習課題

このコードをcloneするなりコピーするなりして色々触ってみてください。

- attackメソッドやprintStatusメソッドの表示をいじる
- インスタンスの初期値をいじる
- 新しいメソッドを追加する

色々あって力尽きたので今日はこんな感じのゆる課題です



参考文献

「オブジェクト指向でなぜ作るのか」平澤 章

「Oracle Java Documentation」

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

