Java講習

後期第5回

前回やったこと

・APIリファレンスの見方

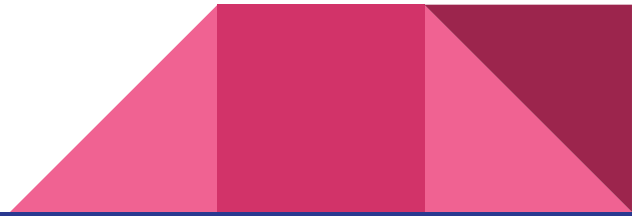
・クラスメソッド(色指定printfメソッド)の自作

・コマンド入力の実装

・ライブラリの作者はすごい

今回やること

1. 画面の整理(前回の復習)
2. 現在の実装の問題点
3. 継承



クラスメソッド復習

staticが付いたメソッドをクラスメソッドと呼んだりします

復習代わりにこんなメソッドを作りました

```
static void clearTerminal() {
    System.out.print("\033[H\033[2J");
    System.out.flush();
}
```

"\033[H\033[2J"は前回の色指定同様
ターミナルの表示に影響を与える特殊なコードです

呼び出しと実演

while文の中の, コマンド受付の次の行で呼び出してみました

sc.nextLine()を二つ書き, ボタンが押されるまで待機するようにします

```
} else if (cmd == COMMAND_ATTACK) {
    m1.attack(e1);
    printStatus(m1, m2, m3, e1, e2);
}

// 何か押されるまで待機
sc.nextLine();
sc.nextLine();

clearTerminal();
```

今日やりたいこと

HPとMPの最大値を作りたい

ここに最大値を表す変数maxHpとmaxMpを追加しよう！

```
String name;

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

maxHpとmaxMpの追加

Mikataに追加しました
コンストラクタやprintStatus()も修正する必要があります

```
int hp;
int maxHp;
int mp;
int maxMp;
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.maxHp = hp;
    this.mp    = mp;
    this.maxMp = mp;
    this.atk   = atk;
    this.def   = def;
    this.speed = speed;
}
```

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

maxHpとmaxMpの追加

Enemyも...

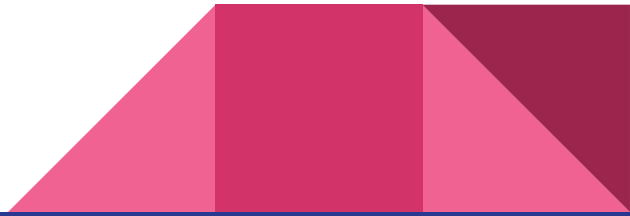
```
int hp;
int maxHp;
int mp;
int maxMp;
int atk;
int def;
int speed;

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

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

ちょっとだるい

ちょっと仕様を変更しただけで, 結構な手間...

仕方のないことだけど, もう少し楽にならないかなと思う



よく考えてみれば

MikataもEnemyも, 同じデータ・メソッドを持ち, 同じように初期化するのが分かります

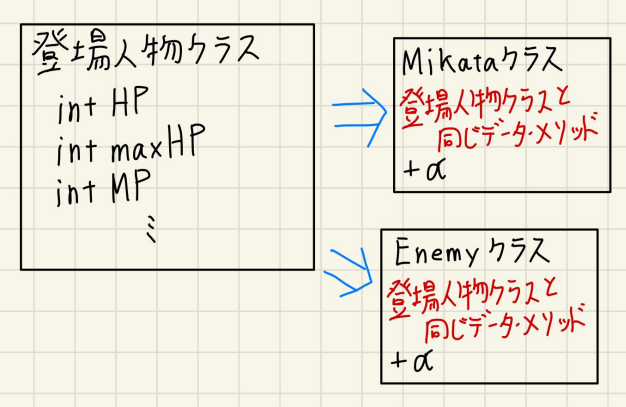
だから両方にHPとMPの最大値のデータを定義しました

```
public class Enemy {
    String name;

    int hp;
    int maxHp;
    int mp;
    int maxMp;
    int atk;
    int def;
    int speed;
```

```
public class Mikata {
    String name;

    int hp;
    int maxHp;
    int mp;
    int maxMp;
    int atk;
    int def;
    int speed;
```

こういうものを共通化したい！

イメージとしてはこんな感じ



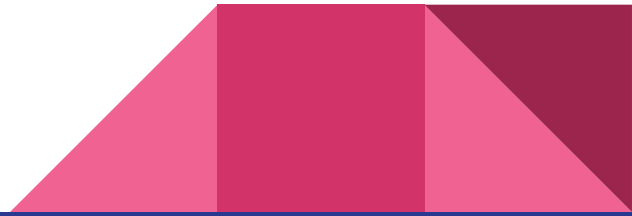
こういうことを実現する機能がオブジェクト指向にはあります

継承

このように

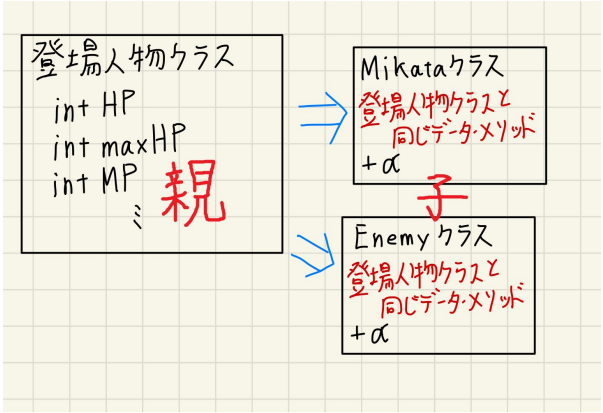
「共通化できる部分をまとめて, 別のクラスに受け継がせる」

ことを継承といいます



継承

継承元のクラス(登場人物クラス)を親クラス(スーパークラス)と呼び
継承先のクラス(Mikata, Enemyクラス)を子クラス(サブクラス)と呼びます



子クラスは親クラスのデータとメソッドを持ちます

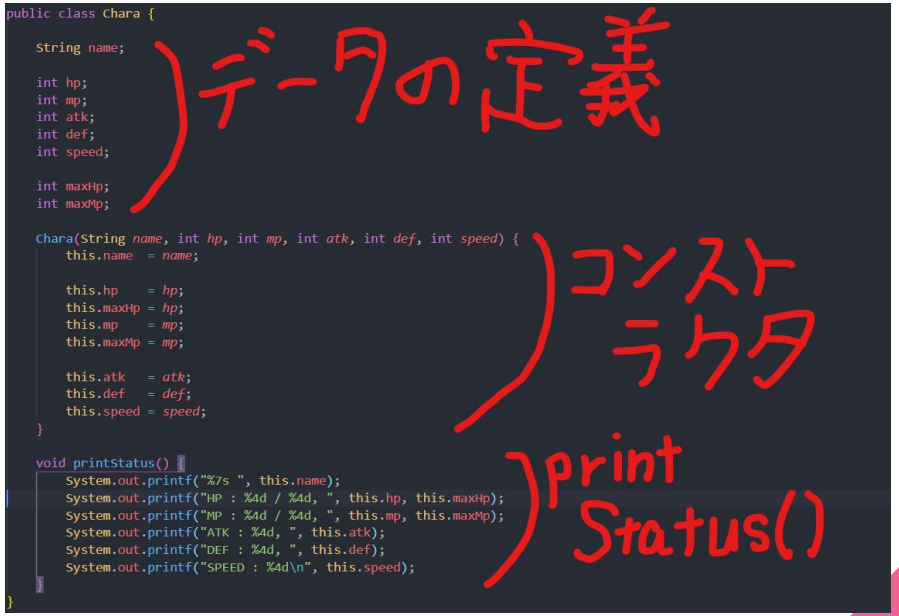
継承

今回はMikataとEnemyの親クラスを作ります

名前はCharaとしました(Characterのつもりです... あまり良い名前ではない)

親クラスの記述

親クラスには, MikataとEnemyに共通してあったものを記述しています




```java
public class Chara {

    String name;

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

    int maxHp;
    int maxMp;

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

        this.hp    = hp;
        this.maxHp = hp;
        this.mp    = mp;
        this.maxMp = mp;

        this.atk   = atk;
        this.def   = def;
        this.speed = speed;
    }

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

継承の記述方法

子クラス側でどのクラスを継承するのかを指定できます

```
public class Mikata extends Chara {
```

(public) class <クラス名> extends <親クラス名> {

これにより, MikataはCharaクラスを継承したことになります
つまり, Charaのデータとメソッドを持つようになります

注意点

ただし, 注意としてはコンストラクタは継承されないということです

書いてあげましょう

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

        this.hp    = hp;
        this.maxHp = hp;
        this.mp    = mp;
        this.maxMp = mp;

        this.atk   = atk;
        this.def   = def;
        this.speed = speed;
    }
}
```

super()

しかし, コンストラクタを書くのも楽ができます

super()を使えば, 親のコンストラクタを呼び出すことができます

```
public class Mikata extends Chara {
    Mikata(String name, int hp, int mp, int atk, int def, int speed) {
        super(name, hp, mp, atk, def, speed);
    }
}
```

継承

attack()メソッドはMikataとEnemyで微妙に違うので, 書いてあげます

```java
public class Mikata extends Chara {
    Mikata(String name, int hp, int mp, int atk, int def, int speed) {
        super(name, hp, mp, atk, def, speed);
    }

    void attack(Enemy enemy) {
        int damage = this.atk - enemy.def;

        System.out.println("--------------------");
        System.out.print(this.name + "は" + enemy.name + "に");
        ColorPrint.redPrintf("%d", damage);
        System.out.println("のダメージを与えた");

        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);
        System.out.println("--------------------");
    }
}
```

```java
public class Enemy extends Chara {
    Enemy(String name, int hp, int mp, int atk, int def, int speed) {
        super(name, hp, mp, atk, def, speed);
    }

    void attack(Mikata mikata) {
        int damage = this.atk - mikata.def;

        System.out.println("--------------------");
        System.out.print(this.name + "は" + mikata.name + "に");
        ColorPrint.redPrintf("%d", damage);
        System.out.println("のダメージを与えた");

        if (damage <= 0) {
            damage = 0;
        }

        mikata.hp = mikata.hp - damage;
        if (mikata.hp <= 0) {
            mikata.hp = 0;
        }
        System.out.println(mikata.name + "のHPは残り" + mikata.hp);
        System.out.println("--------------------");
    }
}
```

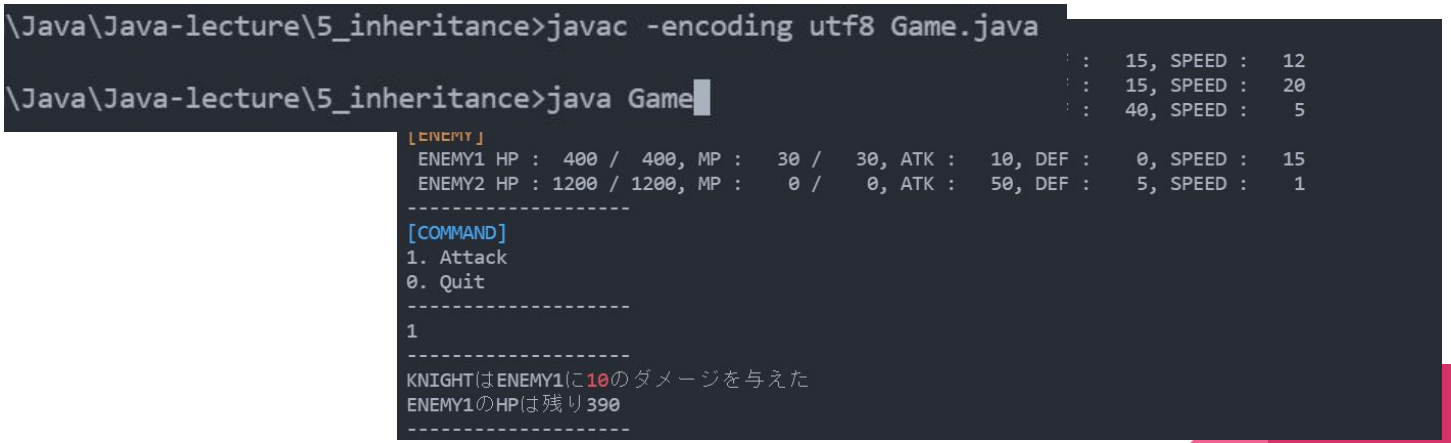
このように, 違う箇所だけ書けばいいので非常に楽です

本当はattack()も共通化することができる(次回やる)

動作確認

コンパイルもちゃんと通りました

ちゃんと動いてそうです



継承のありがたみ

printStatus()メソッドで, HPが0なら赤で表示するようにしましょう

親クラスに共通化しているので, Chara.javaだけいじればいいですね！

```
void printStatus() {
    System.out.printf("%7s ", this.name);

    if (this.hp <= 0) {
        ColorPrint.redPrintf("HP : %4d / %4d, ", this.hp, this.maxHp);
    } else {
        System.out.printf("HP : %4d / %4d, ", this.hp, this.maxHp);
    }
```

継承のありがたみ

ちゃんとMikataもEnemyも変更が反映されてます

```
--------------------
KNIGHTはENEMY1に100のダメージを与えた
ENEMY1のHPは残り0
--------------------
[PARTY]
 KNIGHT HP :  100 /  100, MP :  100 /  100, ATK :  100, DEF :   15, SPEED :   12
 WIZARD HP :   85 /   85, MP :   35 /   35, ATK :    5, DEF :   15, SPEED :   20
   TANK HP :    0 /    0, MP :    0 /    0, ATK :   20, DEF :   40, SPEED :    5
[ENEMY]
 ENEMY1 HP :    0 /  400, MP :   30 /   30, ATK :   10, DEF :    0, SPEED :   15
 ENEMY2 HP : 1200 / 1200, MP :    0 /    0, ATK :   50, DEF :    5, SPEED :    1
--------------------
```

補足

Javaでは, 複数のクラスを継承すること(多重継承)はできません

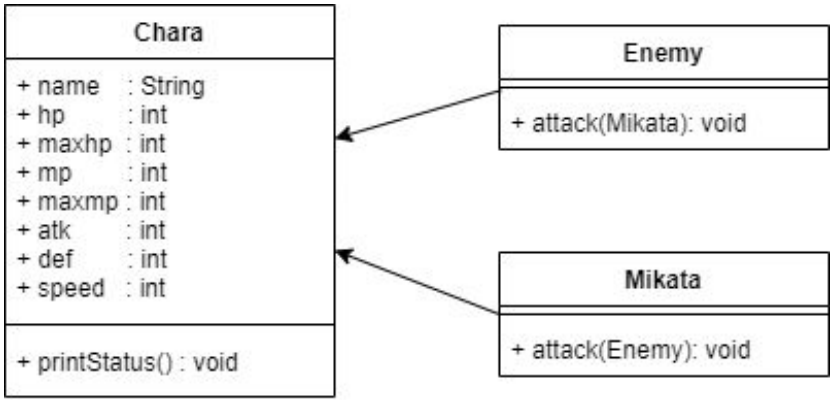
できる言語もあります (Pythonとか)

インターフェースは複数実装できる

まとめ

現在のクラスの関係をまとめました(クラス図というやつです)

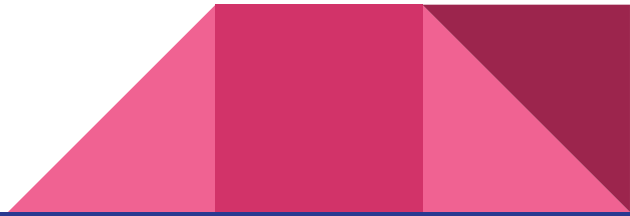
矢印の先の方向が親クラスです
下の図だと, MikataとEnemyがCharaを継承しているのを表しています



まとめ

・継承は, 同じメソッドとデータを共通化できる

・コード量が減るだけでなく, 仕様変更や修正にも対応しやすくなる



参考文献

「オブジェクト指向でなぜ作るのか」平澤 章