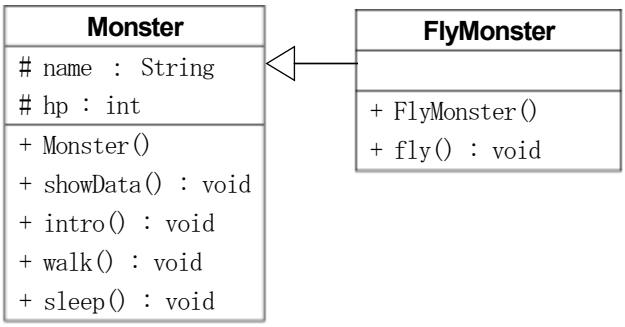
● J2Kad05D「クラスの継承」

Monster クラス（ムックル）を使った処理が作成されている。ムックルを FlyMonster に変更し、「3：飛ぶ」処理を実装せよ。

```
Monster m = new Monster();
    ↓
FlyMonster m = new FlyMonster();
```



FlyMonster クラスの仕様（Monster クラスを継承する）

メソッド	仕様
コンストラクタ	「FlyMonster クラスのコンストラクタが呼び出されました！」と表示する。
void fly()	hp <= 0 のとき、「疲れて飛べないよ～」と表示する。 そうでないとき、「～が飛ぶよ！びゅ～ん!!」（～は名前）と表示し、hp を 1 減らす。

課題完成時の画面

```
Monster クラスのコンストラクタが呼び出されました！
FlyMonster のコンストラクタが呼び出されました！
ムックル（体力：4）
何をしますか？（0：自己紹介、1：散歩、2：眠る、3：飛ぶ、-1：終了）＞0
おいらの名前はムックル。
趣味は散歩。特技はどこでも眠れることだよ。

ムックル（体力：4）
何をしますか？（0：自己紹介、1：散歩、2：眠る、3：飛ぶ、-1：終了）＞1
てくてく・・・

ムックル（体力：3）
何をしますか？（0：自己紹介、1：散歩、2：眠る、3：飛ぶ、-1：終了）＞2
ぐうぐう・・・
体力が 1 ポイント回復した！

ムックル（体力：4）
何をしますか？（0：自己紹介、1：散歩、2：眠る、3：飛ぶ、-1：終了）＞3
ムックルが飛ぶよ！びゅ～ん！

ムックル（体力：3）
何をしますか？（0：自己紹介、1：散歩、2：眠る、3：飛ぶ、-1：終了）＞-1
```

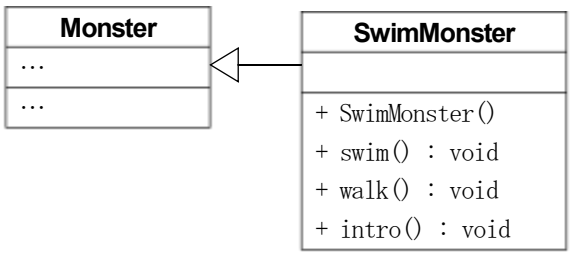
コンストラクタの呼び出し順に注意すること。

追加メソッド

● J2Kad05C「オーバーライド」

main 関数に Monster クラスの実体（コイキング）を使った処理が作成されている。コイキングを SwimMonster に変更し、「3：泳ぐ」処理を実装せよ。

```
Monster m = new Monster();
    ↓
SwimMonster m = new SwimMonster();
```



SwimMonster クラスの仕様（Monster クラスを継承する）

メソッド	仕様
コンストラクタ	「FlyMonster クラスのコンストラクタが呼び出されました！」と表示する。
void swim()	hp <= 0 のとき、「疲れて泳げないよ～」と表示する。 そうでないとき、「～が泳ぐよ！ぶくぶく･･･」（～は名前）と表示し、hp を 1 減らす。
void walk()	「尾びれだと歩けないよ～」と表示する。
void intro()	Monster クラスの intro メソッドを呼び出した後、「泳ぎも得意さ！」と表示する。

課題完成時の画面

```
Monster クラスのコンストラクタが呼び出されました！
SwimMonster のコンストラクタが呼び出されました！
コイキング（体力：10）
何をしますか？（0：自己紹介、1：散歩、2：眠る、3：泳ぐ、-1：終了）＞3
コイキングが泳ぐよ！ぶくぶく・・・っ！？

コイキング（体力：9）
何をしますか？（0：自己紹介、1：散歩、2：眠る、3：泳ぐ、-1：終了）＞1
尾びれだと歩けないよ～

コイキング（体力：9）
何をしますか？（0：自己紹介、1：散歩、2：眠る、3：泳ぐ、-1：終了）＞0
おいらの名前はコイキング。
趣味は散歩。特技はどこでも眠れることだよ。
泳ぎも得意さ！

コイキング（体力：9）
何をしますか？（0：自己紹介、1：散歩、2：眠る、3：泳ぐ、-1：終了）＞-1
```

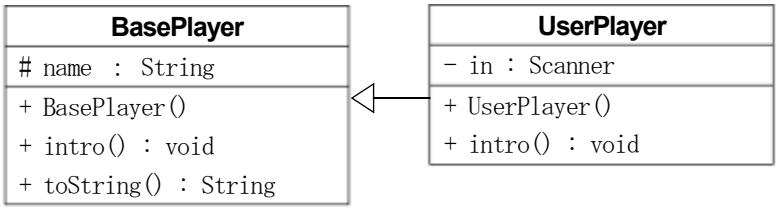
追加メソッド

オーバーライド

Monster クラスの intro の呼び出し

● J2Kad05B「石取りゲームの準備」

「石取りゲーム」(JKad19S) のプレイヤーを表す BasePlayer クラスと UserPlayer クラスを作成、自己紹介する処理を作成せよ。



BasePlayer クラスの仕様

メソッド	仕様
コンストラクタ	「BasePlayer のコンストラクタが呼び出されました！」と表示し、name に「CPU」を代入する
void intro()	「名前：～・・・取る石の数を乱数で決めます。」(～は name) と表示する。
String toString()	name を返す（このメソッドがあるとインスタンス名を戻り値の文字列として扱える)。

UserPlayer クラスの仕様

メンバ関数	仕様
コンストラクタ	「UserPlayer のコンストラクタが呼び出されました！」と表示する。 「あなたの名前を入力してください>」と表示して、入力された名前を name に代入する。
void intro()	「名前：～・・・あなたが操作するプレイヤーです。」(～は name) と表示する。

main メソッドの仕様

① BasePlayer のインスタンスを生成し、自己紹介させる。
② UserPlayer のインスタンスを生成し、自己紹介させる。
③ ①と②のインスタンス名を使って「～と～が戦います！」(～は BasePlayer の名前と UserPlayer の名前) と表示する。

課題完成時の画面

```
石取りゲームの準備をします！

BasePlayer のコンストラクタが呼び出されました！
名前：CPU・・・取る石の数を乱数で決めます。

BasePlayer のコンストラクタが呼び出されました！
UserPlayer のコンストラクタが呼び出されました！
あなたの名前を入力してください>ECC
名前：ECC・・・あなたが操作するプレイヤーです。

CPU と ECC が戦います！
```

● J2Kad05A「石取りゲーム再び！」

J2Kad05A クラスに石取りゲーム（JKad19S）完成版相当のプログラムが準備されている。

ルール

・20 個ある石を交互に取っていき、最後のひとつを取った方が負け。
・1 回で取れる石の数は 1～3 個。
・先手は UserPlayer、後手は BasePlayer。

① BasePlayer と UserPlayer に以下のメソッドを追加し、これらのクラスを使った処理に置き換えよ。

追加するメソッド（引数 stone は現在の石の数）

メソッド	仕様
int takeStone(int stone)	・BasePlayer：乱数で 1～3 の値を返す。 ・UserPlayer：1～3 の値を入力して返す。

② BasePlayer を継承して CompPlayer を作成し、BasePlayer を CompPlayer に変更せよ（少し手強くなる）。

CompPlayer クラスの仕様

メソッド	仕様
コンストラクタ	「CompPlayer のコンストラクタが呼び出されました！」と表示し、name に「HAL」を代入する。
void intro()	「名前：～・・・少し強いです。」（～は name）と表示する。
int takeStone(int stone)	stoneNum が 1：　1 を返す。 stoneNum が 2～4：　stoneNum-1 を返す。 stoneNum が 6～8：　stoneNum-5 を返す。 それ以外：　乱数で 1～3 を返す。

①UserPlayer VS BasePlayer

```
名前：ECC・・・あなたが操作するプレイヤーです。
名前：CPU・・・取る石の数を乱数で決めます。

残り 20 個：●●●●●●●●●●●●●●●●●●●●
ECC の番です。
何個取りますか？（1-3）＞3
3 個取りました！
　　⋮
残り 2 個：●●
ECC の番です。
何個取りますか？（1-3）＞1
1 個取りました！

残り 1 個：●
CPU の番です。
3 個取りました！
CPU の負けです！
```

②UserPlayer VS CompPlayer

```
名前：ECC・・・あなたが操作するプレイヤーです。
名前：HAL・・・少し強いです。

残り 20 個：●●●●●●●●●●●●●●●●●●●●
ECC の番です。
何個取りますか？（1-3）＞3
3 個取りました！
　　⋮
残り 4 個：●●●●
HAL の番です。
3 個取りました！

残り 1 個：●
ECC の番です。
何個取りますか？（1-3）＞1
1 個取りました！
ECC の負けです！
```

● J2Kad05S「最強！MasterPlayer！！」※J2Kad05A の main メソッドをコピーして作成

CompPlayer より強い MasterPlayer（名前：ECC）を作成せよ。MasterPlayer の takeStone メソッドのアルゴリズムは各自で考えること。

・先手：UserPlayer　→　MasterPlayer に変更

ヒント：CompPlayer のアルゴリズムを参考にすること。

相手の番のときの石の数	相手が取れる石の数	勝つために自分が取る石の数	残った石の数（相手の番）
⋮	⋮	⋮	⋮
9	1	3	5
	2	2	5
	3	1	5
5	1	3	1
	2	2	1
	3	1	1
1	1	←自分の勝ち	

課題完成時の画面

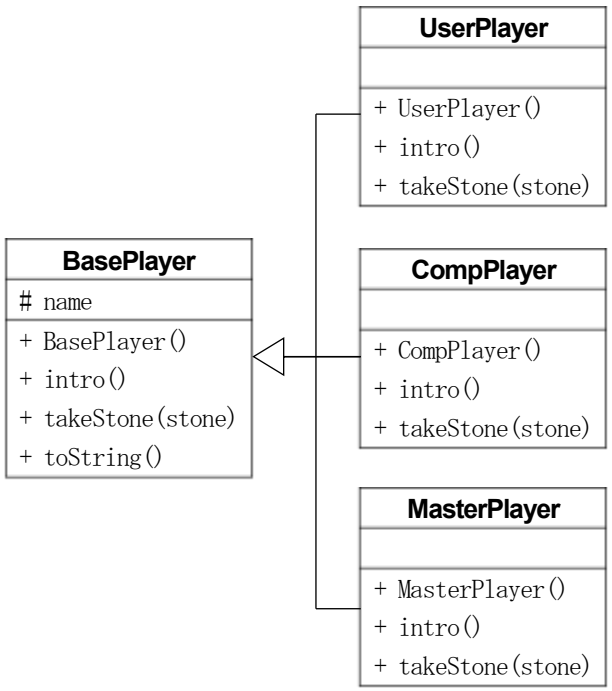
```
名前：ECC・・・最強です！
名前：HAL・・・少し強いです。

残り 20 個：●●●●●●●●●●●●●●●●●●●●
ECC の番です。
3 個取りました！

残り 17 個：●●●●●●●●●●●●●●●●●
HAL の番です。
3 個取りました！
      ⋮
残り 5 個：●●●●●
HAL の番です。
1 個取りました！

残り 4 個：●●●●
ECC の番です。
3 個取りました！

残り 1 個：●
HAL の番です。
1 個取りました！
HAL の負けです！
```



● J2Kad05X「ポリモーフィズムに挑戦！」※J2Kad05A の main メソッドをコピーして作成

ポリモーフィズム（教科書 P.209）を使うと先手の処理と後手の処理をひとつにまとめることができる。main メソッドの処理を簡略化せよ。

J2Kad05A の main メソッド（Before）

```
public static void main(String[] args) {
        ⋮
    while(true) {
        // あなたの手番
        先手の処理
        // CPU の手番
        後手の処理
    }
        ⋮
}
```

課題完成時の main メソッド（After）

```
public static void main(String[] args) {
        ⋮
    while(true) {
        現在の手番のプレイヤーの処理
        プレイヤーの交代
    }
        ⋮
}
```

課題完成時の画面

（J2Kad05A または J2Kado5S と同じ）