● J2Kad04D「クラスメンバとインスタンスメンバ」

以下の仕様でし処理を作成せよ。

仕様 1. Student クラスを作成し、のび太・しずか・ジャイアンの情報を表示する処理を作成せよ。
仕様 2. コース名（course）に static を付けて動作確認せよ。

Student クラス（新規作成）の仕様

書式	説明
private String name	名前
private String course	コース名
public Student(String name, String course)	コンストラクタ。引数の値を対応するフィールドに設定する。
public void showData()	名前とコースを表示する。

のび太・しずか・ジャイアンの初期値

インスタンス名	初期値（名前、コース）
nobita	のび太、IE
sizuka	しずか、SK
suneo	スネ夫、SE
jaian	ジャイアン、オレさまのコース

main メソッドの仕様

① Student クラスのインスタンス（nobita、sizuka、suneo、jaian）を生成する。
② 各インスタンスのデータを表示する。

仕様 1 まで完成時の画面

```
のび太がやってきた！
しずかがやってきた！
スネ夫がやってきた！
ジャイアンがやってきた！
それぞれのデータを表示します！
名前：のび太 コース：IE
名前：しずか コース：SK
名前：スネ夫 コース：SE
名前：ジャイアン コース：オレさまのコース
```

仕様 2 まで完成時の画面

```
のび太がやってきた！
しずかがやってきた！
スネ夫がやってきた！
ジャイアンがやってきた！
それぞれのデータを表示します！
名前：のび太 コース：オレさまのコース
名前：しずか コース：オレさまのコース
名前：スネ夫 コース：オレさまのコース
名前：ジャイアン コース：オレさまのコース
```

仕様 2 まで作成すると全員が
「オレさまのコース」になる。

課題完成時の Student クラス

Student
- name : String - course : String
+ Student(name:String, course:String) + showData() : void

● J2Kad04C「インスタンス配列と参照」

羊を表すSheepクラスを作成し、羊を数える処理を作成せよ（なお、課題作成途中に寝てしまわないこと）。

Sheepクラス（新規作成）の仕様（クラス図も参考にすること）

書式	説明
int count	羊の数。初期値0。
void countSheep()	「羊を数えます！」と表示し、羊の数を数える。 羊が0匹のとき「まだ羊はいません！」と表示する。 羊がいるとき「羊が1匹」「羊が2匹」と羊の数だけ表示する。
コンストラクタ	「羊がやってきた！」と表示し、羊の数を1増やす。

Sheep
- count : int
+ countSheep() : void + Sheep()

mainメソッドの仕様

① 羊の数を数える。
② Sheep型の変数を1つ宣言し、羊の数を数える。
③ Sheepのインスタンスを1つ生成し、②の変数に設定、羊の数を数える。
④ Sheepの配列（要素数3）を生成し、羊の数を数える。
⑤ 配列の各要素にSheepのインスタンスを設定し、羊の数を数える。

課題完成時の画面

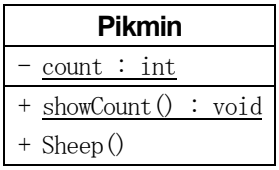
```
①
羊を数えます！
まだ羊はいません！
②
羊を2匹連れてきます！
羊を数えます！
まだ羊はいません！
③
羊がやってきた！
羊を数えます！
羊が1匹・・・
④
羊を3匹（配列で）連れてきます！
羊を数えます！
羊が1匹・・・
⑤
羊がやってきた！
羊がやってきた！
羊がやってきた！
羊を数えます！
羊が1匹・・・
羊が2匹・・・
羊が3匹・・・
羊が4匹・・・
```

● J2Kad04B「ピクミンを探せ！」

ピクミンを表す Pikmin クラスを作成し、ピクミンを探す処理を作成せよ。なお、1 回あたりに見つかるピクミンは 0～3 匹とし、0 匹のときは見つからなかったと表示する。

Pikmin クラス（新規作成）の仕様（クラス図も参考にすること）

メソッドの書式	説明
void showCount()	「これまでに見つけたピクミンは～です！」と表示する。
コンストラクタ	「ピクミンがやってきた！」と表示し、ピクミンの数を 1 増やす。



課題完成時の画面

```
ピクミンを探します！
これまでに見つけたピクミンは0匹です！
どうしますか？（0：探す、-1：やめる）>0
ピクミンがやってきた！
ピクミンがやってきた！
ピクミンがやってきた！
3匹見つけた！

これまでに見つけたピクミンは3匹です！
どうしますか？（0：探す、-1：やめる）>0
見つからなかった！

これまでに見つけたピクミンは3匹です！
どうしますか？（0：探す、-1：やめる）>0
ピクミンがやってきた！
1匹見つけた！

これまでに見つけたピクミンは4匹です！
どうしますか？（0：探す、-1：やめる）>0
ピクミンがやってきた！
ピクミンがやってきた！
2匹見つけた！

これまでに見つけたピクミンは6匹です！
どうしますか？（0：探す、-1：やめる）>-1
```

● J2Kad04A「カードシャッフル再び！」

0～9 までの番号が書かれたカードを表す Card クラス（最大 10 枚まで）を作成し、カードを 10 枚引いて表示する処理を作成せよ。

ヒント：カードシャッフルは JKad21D の**リスト 1** を参考にすること（もちろん他の方法でも OK）。

Card クラス（新規作成）の仕様　※必要なフィールドは各自で考えること

書式	説明
コンストラクタ	0～9 までの番号を決める。 ただし他のインスタンスと番号が重ならないようにすること。
int getNumber()	自分の番号を返す。

Card
…
+ Card() + getNumber() : int

main メソッドの仕様

① Card クラスのインスタンスを生成し、番号を表示する。
② ①を 10 回繰り返す。

課題完成時の画面（10 枚すべて異なる番号になる）

```
10 枚カードを引きます！
カードの番号は 5 です！
カードの番号は 8 です！
カードの番号は 0 です！
カードの番号は 6 です！
カードの番号は 7 です！
カードの番号は 9 です！
カードの番号は 2 です！
カードの番号は 4 です！
カードの番号は 3 です！
カードの番号は 1 です！
```

● J2Kad04S「レジ待ち行列①」

ECC が激安スーパーを開業した！その名も「激安スーパーECC」。客の呼び込みとレジ打ちを行う処理が準備されている（J2Kad04S クラス）。ところがレジ待ち行列をスタック形式にしたため、後から並んだ人が先に処理されてしまう！

1．先に並んだ人から先に処理されるようにキュー（Queue）クラスを作成せよ。インターフェイス（フィールド・メソッドの書式）は Stack クラスと同じ。
2．Stack クラスの行列を Queue クラスに置き換えて動作確認せよ。

課題作成前の画面（Stack クラスの場合、行列の最後の人から処理される）

```
いらっしゃい！激安スーパーECC です！！
何をしますか？（0：客を呼び込む、1：レジを打つ、-1：店をたたむ）＞0
ギャラドスがやってきた！
ミュウがやってきた！
カビゴンがやってきた！

現在のレジ待ち行列です！
0：ギャラドス
1：ミュウ
2：カビゴン

何をしますか？（0：客を呼び込む、1：レジを打つ、-1：店をたたむ）＞0
スピアーがやってきた！
ピジョンがやってきた！
ムックルがやってきた！もう店に入れない！残念！！

現在のレジ待ち行列です！
0：ギャラドス
1：ミュウ
2：カビゴン
3：スピアー
4：ピジョン

何をしますか？（0：客を呼び込む、1：レジを打つ、-1：店をたたむ）＞1
ピジョンは帰っていった！！

現在のレジ待ち行列です！
0：ギャラドス
1：ミュウ
2：カビゴン
3：スピアー

何をしますか？（0：客を呼び込む、1：レジを打つ、-1：店をたたむ）＞1
スピアーは帰っていった！！
      ：
```

Stack
- container : Monster[]
- sp : int
+ Stack(size : int)
+ push(data : Monster) : void
+ pop() : Monster
+ getData(i : int) : Monster
+ size() : int
+ empty() : boolean
+ full() : boolean

Queue
- container : Monster[]
- sp : int
+ Queue(size : int)
+ push(data : Monster) : void
+ pop() : Monster
+ getData(i : int) : Monster
+ size() : int
+ empty() : boolean
+ full() : boolean

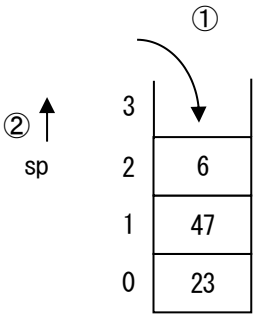
Queue のインターフェイス（フィールド・メソッドの書式）は Stack と同じ

Stack の場合、最後尾のスピアーが処理される。Queue にすると先頭のギャラドスから処理されるようになる。

参考：キュー（Queue クラス、データシフト方式）

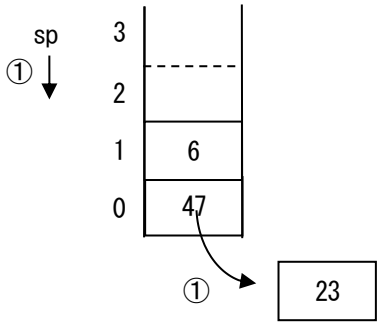
キューは、最初に格納したデータが最初に取り出されるというデータ構造です（First In, First Out、略して FIFO)。非同期処理をするプログラム間でデータを受け渡す場合などで使われます。

データ格納（プッシュ）



データ格納時は
①sp の指す場所にデータを格納し、
②sp を+1 する。

データ読み出し（ポップ）



データ読み出し時は
①0 番からデータを取り出し、
②sp を-1 すると同時に残っているデータをずらす。

● J2Kad04X2「レジ待ち行列②（リングバッファ）」

ECC が激安スーパーの 2 号店を開業した！ところがレジ待ち行列をまたもやスタック形式にしてしまった！！

1．リングバッファ形式のキュー（Queue2）クラスを作成せよ。
2．Stack クラスの行列を Queue2 クラスに置き換えて動作確認せよ。

Queue2
- container : Monster[] - start : int - used : int
+ Queue2(size : int) + push(data : Monster) : void + pop() : Monster + getData(i : int) : Monster + size() : int + empty() : boolean + full() : boolean

メソッドの書式は Stack と同じ

参考：リングバッファ（Queue2 クラス）

行列の先頭位置を表す start と並んでいる人数を表す used を使って作成します。データの書き込みは start＋used の場所に行います。書き込みを行うと used を 1 増やします。読み出しは start の場所を読み出し、start を 1 進め used を 1 減らします。読み出す場所や書き込む場所が配列の最後尾を越えたら先頭へ戻ります（最後尾と先頭がリング状につながっているイメージ)。