● J2Kad04D「クラス変数」（入門編 P.139「インスタンス変数」、P.176「クラス変数」）

以下の仕様で処理を作成せよ。

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（IE、SK、SE など）※仕様②で static を付ける
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「クラスメソッド」（P.171「インスタンスメソッドとは」、P.180「クラスメソッド」）

羊を表す Sheep クラスが準備されている。コンストラクタで名前（name）を設定し、intro メソッドで自己紹介する。

仕様① 羊を 3 匹生成し、自己紹介する処理を作成せよ。
仕様② Sheep クラスに以下のメンバを追加し、羊の生成前と生成後で羊の数を表示する処理を追加せよ。

Sheep クラスに追加するメンバ

	書式	説明
フィールド	`private static int n = 0;`	生成した羊の数、初期値は 0
メソッド	`public static void countSheep()`	生成した羊の数を表示する。「羊は全部で xx 匹です！」

リスト 1：J2Kad04C クラス

```
public class J2Kad04C {
    public static void main(String[] args) {
        羊の数を表示する
        Sheep s1 = new Sheep();
        Sheep s2 = new Sheep();
        Sheep s3 = new Sheep();
        羊の数を表示する
        s1.intro();
        s2.intro();
        s3.intro();
    }
}
```

課題完成時の画面（羊の数の表示は仕様②）

```
羊は全部で 0 匹です！
ホイヘンスがやってきた！
ニュートンがやってきた！
ジュールがやってきた！
羊は全部で 3 匹です！
「わたしはホイヘンス、よろしくメェ～！」
「わたしはニュートン、よろしくメェ～！」
「わたしはジュール、よろしくメェ～！」
```

課題完成時の Sheep クラス

Sheep
− name : String − n : int
+ Sheep() + intro() : void + countSheep() : void

static メンバ（下線）は仕様②で追加する。

● J2Kad04B「インスタンスの配列」（入門編 P.145「インスタンスの配列」）

J2Kad04C と同じ処理を Sheep クラスの配列で作成せよ。

課題完成時の画面

（J2Kad04C と同じ）

● J2Kad04A「学籍番号を割り当てろ！」※J2Kad04D をコピーして作成

課題完成時の画面と同じようになるように、Student クラスを修正せよ。main メソッドは J2Kad04D と同じで OK。

① ジャイアンが勝手に「オレさまのコース」に変更しないようにする。
② 学籍番号を 2230001 から連番で割り当てるようにする（showData メソッドに学籍番号の表示も追加すること）。

課題完成時の画面

```
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　学籍番号：2230001
名前：しずか コース：SK　学籍番号：2230002
名前：スネ夫 コース：SE　学籍番号：2230003
名前：ジャイアン コース：オレさまのコース 学籍番号：2230004
```

課題完成時の Student クラス

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

● J2Kad04S「カードを並べよう！」

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。

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	「カードを引いた！」と表示し、0～9 までの番号を割り当てる。 ただし他のカード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

課題完成時の画面（10 枚すべて異なる番号になる）

```
0 から 9 までのカードをランダムに並べます！
カードを引いた！
カードを引いた！
カードを引いた！
カードを引いた！
カードを引いた！
カードを引いた！
カードを引いた！
カードを引いた！
カードを引いた！
カードを引いた！
順番にカードを表示します！
カードの番号は 4 です！
カードの番号は 1 です！
カードの番号は 8 です！
カードの番号は 3 です！
カードの番号は 2 です！
カードの番号は 9 です！
カードの番号は 5 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！
```

● J2Kad04X1「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」。J2Kad04X1 に客の呼び込みとレジ打ちを行う処理が準備されている。ところがレジ待ち行列をスタック（Stack）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！ 先に並んだ人から先に処理されるキュー（Queue）クラスを作成し、Stack を Queue に置き換えよ。なお、Queue のインターフェイス（public なフィールド・メソッドの書式）は Stack クラスと同じで OK。

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！

現在のレジ待ち行列です！
0：ギャラドス

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
ミュウがやってきた！

現在のレジ待ち行列です！
0：ギャラドス
1：ミュウ
    ⋮
  （中略）
    ⋮
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
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
```

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

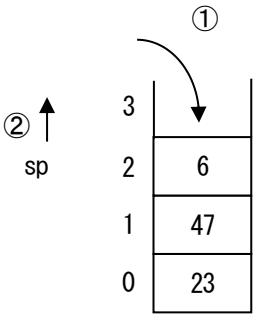
Queue のインターフェイス（フィールド・メソッドの書式）は Stack と同じ

Stack の場合、最後尾のピジョンが処理される。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）

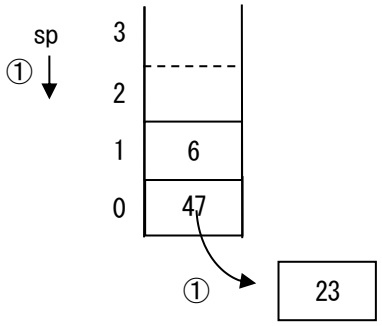
キューは、最初に格納したデータが最初に取り出されるというデータ構造です（First In, First Out、略して FIFO)。非同期処理をするプログラム間でデータを受け渡す場合などで使われます。

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！！ リングバッファ形式のキュー（Queue2）クラスを作成し、Stack と置き換えよ。

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)。

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

メソッドの書式は Stack と同じ