● J2Kad20D「スタックオーバーフロー」（実践編 P.79「プログラムの実行とメモリの管理」）

J2Kad20D クラスを作成し、スタックオーバーフローが発生することを確認せよ。

J2Kad20D クラス（ファイル「J2Kad20D.java」）

メソッド	仕様
public static void overflow(int n)	引数 n をインクリメントしたのち、n の値を表示する。 表示後、更新された n を引数に overFlow を呼び出す（再帰呼び出し）。 例外 StackOverFlowError が発生するので、main メソッドへ「throws」すること。
main メソッド	overflow メソッドを呼び出す（引数の値は 0）。 例外 StackOverFlowError をキャッチしたら、キャッチした例外を表示する。

課題完成時の画面

```
1
2
3
4
5
6
7
8
9
10
 ⋮
20437
20438
20439
20440
20441
20442
20443
20444
java.lang.StackOverflowError
```

スタックオーバーフローが発生したら終了する。

● J2Kad20C「ヒープとガーベッジコレクション」

（実践編 P.81「空きメモリサイズの確認」、P.84「ガーベッジコレクションとは」）

main メソッド（**リスト 1**）に以下の処理を作成せよ。

① 現在の空きメモリサイズを表示する。
② 配列 dataSet の全要素に対して以下の③④の処理を行う。
③ DataSet クラスのインスタンスを生成する。
④ インスタンスを 100 個生成するごとに、生成済みインスタンス数とメモリの空きサイズを表示する。
⑤ 実行して空きサイズを確認する。

上記ができたら③と④の間に以下の処理を追加し、再度、実行して空きサイズを確認する。

```
dataSet[i] = null;   // インスタンス解放（実際に解放されるのはガーベッジコレクション実行時）
```

リスト 1：ヒープとガーベッジコレクション（ファイル「J2Kad20C.java」）

```
public class J2Kad20C {
    public static void main(String[] args) {
        class DataSet {
            int[] data = new int[1000];
        }
        DataSet[] dataSet = new DataSet[10000];
        作成すること
    }
}
```

課題完成時の画面（①～④まで作成）

```
現在のメモリの空きサイズ：265413472
生成済みインスタンス数：100　現在のメモリの空きサイズ：264486696
生成済みインスタンス数：200　現在のメモリの空きサイズ：264241152
      ⋮
生成済みインスタンス数：10000　現在のメモリの空きサイズ：226709536 ←― 確認する
```

課題完成時の画面（インスタンス解放あり）

```
現在のメモリの空きサイズ：265413472
生成済みインスタンス数：100　現在のメモリの空きサイズ：264486696
生成済みインスタンス数：200　現在のメモリの空きサイズ：264241152
      ⋮
生成済みインスタンス数：4800　現在のメモリの空きサイズ：245602384 ↘
生成済みインスタンス数：4900　現在のメモリの空きサイズ：266905608 ←― ガーベッジコレクション
      ⋮                                                              （メモリが増えている）
生成済みインスタンス数：10000　現在のメモリの空きサイズ：246390760 ←― 確認する
```

● J2Kad20B「コレクションとラムダ式」（実践編 P.138「forEach メソッドとラムダ式」）

座標を表す Point クラスを ArrayList に格納する処理が準備されている（**リスト 1**）。①更新前のデータを表示する処理、②各 Point クラスの x と y を 2 倍にする処理、③更新後のデータを表示する処理を以下の 2 パターンで作成せよ。

パターンＡ　拡張 for 文を使って作成する。
パターンＢ　forEach にラムダ式を渡して作成する。

Point クラスの仕様（作成済み）

メンバ	説明
public int x, y	X 座標、Y 座標
public Point()	x と y に 0～49 の値を設定する。
public void printInfo()	データ（座標および x と y を加算した値）を表示する。

リスト 1：コレクションとラムダ式（ファイル「J2Kad20B.java」）

```
public class J2Kad20B {
    public static void main(String[] args) {
        ArrayList<Point> pointList = new ArrayList<>();
        pointList.add(new Point());
        pointList.add(new Point());
        pointList.add(new Point());
        pointList.add(new Point());

        System.out.println("更新前のデータを表示します！");
        ①更新前のデータを表示する処理
        System.out.println("2 倍します！");
        ②各 Point クラスの x と y を 2 倍にする処理
        System.out.println("更新後のデータを表示します！");
        ③更新後のデータを表示する処理
    }
}
```

課題完成時の画面

```
更新前のデータを表示します！
(17, 30)      x + y = 47
(45,  8)      x + y = 53
( 9, 42)      x + y = 51
( 0, 13)      x + y = 13
2 倍します！
更新後のデータを表示します！
(34, 60)      x + y = 94
(90, 16)      x + y = 106
(18, 84)      x + y = 102
( 0, 26)      x + y = 26
```

● J2Kad20A「コレクションと並べ替え」※J2Kad20B をコピーして作成

（実践編 P.141「ラムダ式を用いた並べ替え」、P.118「sort メソッドによる並べ替え」）

J2Kad20B の「②各 Point クラスの x と y を 2 倍にする処理」を「②各 Point クラスを並べ替える処理」に変更せよ。並べ替えは X 座標と Y 座標を加算した値で昇順とし、以下の 2 パターン作成すること。

パターン A（実践編 P.141、もしくは検索）

ArrayList の sort メソッドに大小比較のラムダ式を渡して並べ替える。大小比較する 2 つのポイントを p0、p1 とすると大小比較（ラムダ式の処理内容部分）は以下の通り。

```
(p0.x + p0.y) - (p1.x + p1.y)
```

パターン B（実践編 P.118、もしくは検索）

① Point クラスに Comparable インターフェイスを実装して CompareTo メソッドをオーバーライドする。
② Collections.sort メソッドで並べ替える。

課題完成時の画面

```
更新前のデータを表示します！
(46, 13)     x + y = 59
(30, 30)     x + y = 60
(27,  6)     x + y = 33
(17, 35)     x + y = 52
並べ替えます！
更新後のデータを表示します！
(27,  6)     x + y = 33
(17, 35)     x + y = 52
(46, 13)     x + y = 59
(30, 30)     x + y = 60
```

● J2Kad20S「シネコン ECC①（ダブルブッキング！）」※パッケージ pac20s に作成

世界に羽ばたく ECC がシネコンへの進出を決めた！ただし最初は様子を見るため、スクリーンは１つ、チケット売り場の窓口は３つからのスタートとなった。ところが、業者に発券システムのプログラムを頼んだところ、同じ座席番号のチケットが発券されるという不具合が発生した。どの窓口で購入しても座席番号が重ならないようにプログラムを修正せよ。

ヒント：J2Kad16X2 で行った処置と基本的に同じ。

課題作成前の画面（業者のプログラム）

```
ようこそ！シネコン ECC へ！
何番の窓口で購入しますか？（0～2、-1：終了）＞0
のび太が担当します！
あなたの座席は１番です！

何番の窓口で購入しますか？（0～2、-1：終了）＞1
スネ夫が担当します！
あなたの座席は１番です！

何番の窓口で購入しますか？（0～2、-1：終了）＞2
ジャイアンが担当します！
あなたの座席は１番です！

何番の窓口で購入しますか？（0～2、-1：終了）＞-1
```

のび太もスネ夫もジャイアンも
同じ番号のチケットを発券する。

課題完成時の画面

```
ようこそ！シネコン ECC へ！
何番の窓口で購入しますか？（0～2、-1：終了）＞0
のび太が担当します！
あなたの座席は１番です！

何番の窓口で購入しますか？（0～2、-1：終了）＞1
スネ夫が担当します！
あなたの座席は２番です！

何番の窓口で購入しますか？（0～2、-1：終了）＞2
ジャイアンが担当します！
あなたの座席は３番です！

何番の窓口で購入しますか？（0～2、-1：終了）＞-1
```

窓口が変わってもチケット番号は続きからになる。

パッケージについて（実践編 P.25「1-4 パッケージの作成」、実践編 P.30「1-5 クラスのアクセス制御」を参照）

ファイル「J2Kad20S.java」はパッケージ「pac20s」の中に作っています。public のついたクラス（J2Kad20S）はパッケージ外からでも参照できますが、何もついていないクラス（TicketMaker クラスと SalesPerson クラス）はパッケージ内でしか使えません。したがって TicketMaker と SalesPerson は pac20s 専用クラスとなります。J2Kad20X でも若干仕様の異なる TicketMaker と SalesPerson が出てくるので、こちらもパッケージ pac20x 専用にしてお互いがバッティングしないようにしています。

● J2Kad20X「シネコン ECC②（スクリーン増設！）」※パッケージ pac20x を新規作成

シネコン事業が思いもよらず好調なため、スクリーン数を増やすことになった！以下の処理を追加せよ。

① スクリーン数が 3 つに対応できるようにプログラムを修正せよ（TicketMaker のインスタンスを 3 つまで生成できるように改造すること）。
② 3 つある窓口のうち、窓口 2 番のジャイアンの売り上げがどうも芳しくない。これに怒った支配人がジャイアンを解雇して自動券売機（SalesMachine クラス）を導入した！窓口 2 番を自動券売機に置き換えよ。

なお、パッケージ「pac20x」を新規に作成、ファイル「J2Kad20S.java」をコピーして（ファイル名は「J2Kad20X.java」）作成すること。

課題完成時の画面

```
ようこそ！シネコン ECC へ！
何番の窓口で購入しますか？（0～2、-1：終了）＞0
のび太が担当します！
何の映画を見ますか？（0：ポケットモンスター、1：ドラえもん、2：アンパンマン、-1：終了）＞0
あなたの座席はスクリーン 0 の 1 番です！

何番の窓口で購入しますか？（0～2、-1：終了）＞1
スネ夫が担当します！
何の映画を見ますか？（0：ポケットモンスター、1：ドラえもん、2：アンパンマン、-1：終了）＞0
あなたの座席はスクリーン 0 の 2 番です！

何番の窓口で購入しますか？（0～2、-1：終了）＞0
のび太が担当します！
何の映画を見ますか？（0：ポケットモンスター、1：ドラえもん、2：アンパンマン、-1：終了）＞1
あなたの座席はスクリーン 1 の 1 番です！

何番の窓口で購入しますか？（0～2、-1：終了）＞2
ワタシハ自動券売機ノ ECC2000 ダ
何の映画を見ますか？（0：ポケットモンスター、1：ドラえもん、2：アンパンマン、-1：終了）＞1
ホレ、スクリーン 1 ノ 2 番ダ！ヨロコベ！！

何番の窓口で購入しますか？（0～2、-1：終了）＞-1
```

スクリーンごとに番号が割り振られる。
またジャイアンは自動券売機に置き換わる。
（チケット発券時のメッセージも変える）