# **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

…

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

ガーベッジコレクション

（メモリが増えている）

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）**

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。

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**

**課題完成時の画面**

窓口が変わってもチケット番号は続きからになる。

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**

**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）