● J2Kad15D「例外処理①（try～catch)」

2 つの整数を入力すると割り算を行って結果を表示する処理が準備されている（**リスト 1**)。ただし入力時に整数ではなく文字列を入力したり、2 つ目の整数（割る数）に 0 を入力したりすると例外が発生する。

① 例外が発生したときに例外の名前と「例外が発生しました！」という文字列を表示するようにせよ。
② **リスト 1** の破線部分を以下のメソッドに置き換えても例外をキャッチできることを確認せよ。

J2Kad15D クラスに追加するメソッド

メソッド	仕様
public static void div(int n1, int n2)	リスト 1 の破線部分と同じコード

リスト 1：割り算（J2Kad15D クラス）

```
public class J2Kad15D {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("整数 1 を入力してください>");
        int n1 = in.nextInt();
        System.out.print("整数 2 を入力してください>");
        int n2 = in.nextInt();
        System.out.println(n1 + "÷" + n2 +"を計算します！");
        int ans = n1 / n2;
        System.out.println("計算結果は" + ans + "です！");
    }
}
```

← div メソッドに置き換える部分

課題作成前の画面①（整数 1 に文字列を入力したとき）

```
整数 1 を入力してください>ecc
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:943)
    at java.base/java.util.Scanner.next(Scanner.java:1598)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2263)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2217)
    at J2Kad15D.main(J2Kad15D.java:9)
```

← InputMismatchException が発生

課題作成前の画面②（整数 2 に 0 を入力したとき）

```
整数 1 を入力してください>10
整数 2 を入力してください>0
10÷0 を計算します！
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at J2Kad15D.main(J2Kad15D.java:14)
```

← ArithmeticException が発生

課題完成時の画面①（整数１に文字列を入力したとき）

```
整数1を入力してください>ecc
java.util.InputMismatchException
例外が発生しました！
```

課題完成時の画面②（整数２に０を入力したとき）

```
整数1を入力してください>10
整数2を入力してください>0
10÷0を計算します！
java.lang.ArithmeticException: / by zero
何か例外が発生しました！
```

● J2Kad15C「例外処理②（try～catch～finally)」

J2Kad15D において発生した例外によって表示する文字列を変更せよ。また finally ブロックを追加し動作確認せよ。

・InputMismatchException が発生したとき　…例外の名前と「int 型でない値が入力されました！」を表示する
・ArithmeticException が発生したとき　…例外の名前と「0 除算が発生しました！」を表示する
・finally ブロックの処理　…「finally ブロックの処理です！」と表示する

課題完成時の画面①（整数１に文字列を入力したとき）

```
整数1を入力してください>ecc
java.util.InputMismatchException
int型でない値が入力されました！
finallyブロックの処理です！
```

課題完成時の画面②（整数２に０を入力したとき）

```
整数1を入力してください>10
整数2を入力してください>0
10÷0を計算します！
java.lang.ArithmeticException: / by zero
0除算が発生しました！
finallyブロックの処理です！
```

課題完成時の画面③（正常終了のとき）

```
整数1を入力してください>10
整数2を入力してください>5
10÷5を計算します！
計算結果は2です！
finallyブロックの処理です！
```

● J2Kad15B「ECC 長屋の 3 号室！」

ECC 長屋（部屋数 3）に入居している羊たちに自己紹介させる処理が準備されている。

① 存在しない部屋番号を入力して例外が発生したときに「そんな部屋番号はありません！」と表示せよ。
② 部屋番号に文字列など整数でない値が入力されたとき「int 型でない値が入力されました！」と表示せよ。

課題作成前の画面①（2 より大きい値を指定すると例外が発生する）

```
ポアンカレがやってきた！
ピタゴラスがやってきた！
ホイヘンスがやってきた！
何号室を見ますか？（-1：興味なし）＞3
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
    at J2Kad15B.main(J2Kad15B.java:12)
```

課題完成時の画面

```
ラグランジュがやってきた！
ガリレオがやってきた！
エラストテネスがやってきた！
何号室を見ますか？（-1：興味なし）＞0
「わたしはラグランジュ、よろしくメェ～！」

何号室を見ますか？（-1：興味なし）＞2
「わたしはエラストテネス、よろしくメェ～！」

何号室を見ますか？（-1：興味なし）＞3
そんな部屋番号はありません！
何号室を見ますか？（-1：興味なし）＞ecc
int 型でない値が入力されました！
何号室を見ますか？（-1：興味なし）＞-1
```

※②について
nextInt メソッドのときに文字列など異なるデータ型が入力されると InputMismatchException が発生しますが、Scanner クラスのバッファ内に不正な値が残り続けてしまいます。そのためループで再度入力する場合に InputMismatchException が自動的に発生し無限ループになってしまいます（↓のようなことになります）。

```
何号室を見ますか？（-1：興味なし）＞int 型でない値が入力されました！
何号室を見ますか？（-1：興味なし）＞int 型でない値が入力されました！
（延々続く）
```

よって以下のように対処します。

```
int n = Integer.parseInt(in.next());        // 文字列として入力して int 型に変換する
```

変換できないときは NumberFormatException が発生するのでこれを catch します。

● J2Kad15A「例外処理まとめ」

以下の仕様で割り算をして結果を配列に格納する処理を作成せよ。また**課題完成時の画面**を参考に例外発生時の処理も作成せよ。

・結果を格納する配列の要素数は 5 とする。
・割られる数はキーボードから入力する。
・割る数は 0～9（乱数で決定）とする。
・計算結果は配列に格納する。このとき格納先のインデックスは 0～9（これも乱数で決定）とする。
・これらの処理を無限ループで繰り返す。なお 0 除算の例外が発生したときにループから抜けるものとする。

課題完成時の画面

```
割られる数を入力してください＞10
9 で割ります！
計算結果は 1 です！
配列の 4 番目に格納します！
処理が正常に行われました！
finally ブロックの処理です！

割られる数を入力してください＞ecc
int 型でない値が入力されました！
finally ブロックの処理です！

割られる数を入力してください＞10
7 で割ります！
計算結果は 1 です！
配列の 8 番目に格納します！
配列に格納できません！
finally ブロックの処理です！

割られる数を入力してください＞10
0 で割ります！
0 除算が発生しました！
finally ブロックの処理です！
終了します！
```

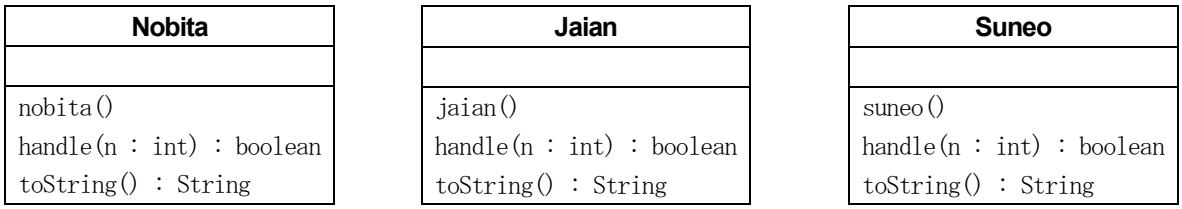
← 0 除算が発生したらループを抜けて終了する

● J2Kad15S「ECC 苦情処理センター（開設準備室)」

ECC 苦情処理センターでは受け付けた苦情を 0 から 99 までの番号に分類して処理を行う。ここで働くのは優秀なスタッフばかりだ。センターを開設する前にこのスタッフたちがどれぐらい優秀かを調べるため、各スタッフを表すクラスを業者に作成してもらってシミュレートすることになった。ところが業者の作成したクラスには無駄なコードが多く、新しいスタッフを追加したり処理を簡略化したりするのが難しい。スーパークラス（Handler クラス）を導入し、各クラスの処理を簡略化せよ。

優秀なスタッフたち

スタッフ	対応できる番号
のび太	20 未満のとき対応可能
ジャイアン	苦情番号とは関係なく、乱数（0～3）で 0 が出たら対応可能
スネ夫	3 の倍数のとき対応可能



リスト 1：Nobita クラス

```
public class Nobita extends Handler {
    public Nobita() { System.out.println("のび太がスタンバイしました！"); }
    public boolean handle(int n) {
        if (n < 20) {
            System.out.println("のび太：私が対応します！");
            return true;
        } else {
            System.out.println("のび太：専門外です・・・");
            return false;
        }
    }
    public String toString() { return "のび太"; }
}
```

Jaian、Suneo も名前と if 文の条件式が異なるだけでほぼ同じ処理。業者は Jaian、Suneo にも同じようなコードを記述している。

ヒント：メソッドの一部だけをオーバーライドするには？→J2Kad10S

課題実行時の画面

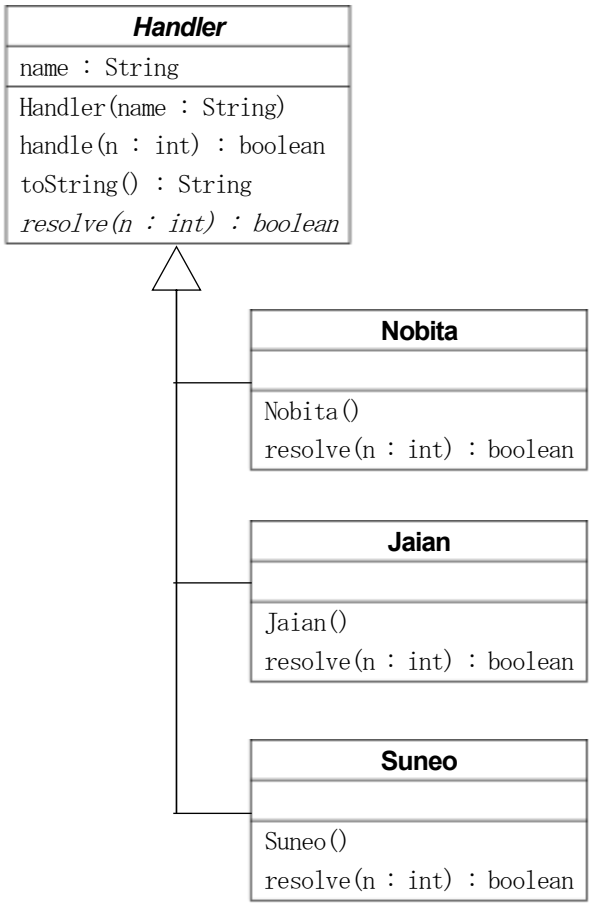
```
ECC 苦情処理センター開設準備室です！
苦情処理のシミュレーションを行います！
のび太がスタンバイした！
ジャイアンがスタンバイした！
スネ夫がスタンバイした！
苦情番号：0 を受け付けた！
のび太：私が対応します！
ジャイアン：私が対応します！
スネ夫：私が対応します！

苦情番号：1 を受け付けた！
のび太：私が対応します！
ジャイアン：専門外です・・・
スネ夫：専門外です・・・

苦情番号：2 を受け付けた！
のび太：私が対応します！
ジャイアン：専門外です・・・
スネ夫：専門外です・・・
      ⋮
（中略）
      ⋮
苦情番号：99 を受け付けた！
のび太：専門外です・・・
ジャイアン：専門外です・・・
スネ夫：私が対応します！

それぞれが対応した回数は
のび太：20 回
ジャイアン：31 回
スネ夫：34 回
```

課題完成時のクラス図



● J2Kad15X「ECC 苦情処理センター（Chain of Resposibility)」※検索すること

ECC 苦情処理センターでは「優秀なスタッフたち」に若干の不安を覚えたため、必要であれば「さらに優秀なスタッフたち」を応援として呼ぶことができるようにした。ところが応援の呼び出しも含めた苦情処理を業者に作ってもらったところ、**見るのもおぞましいとんでもないコード**を作ってきた（次ページ**リスト１**、心して見るように)。「さらに優秀なスタッフたち」のクラス（Sizuka、Dekisugi）を追加し、**リスト１**のコードを簡略化せよ。なお、苦情対応の順は以下の通り。

のび太 → ジャイアン → スネ夫 → （応援を呼んでいたら）しずか → 出木杉

リスト１：見るのもおぞましいとんでもないコード（J2Kad15X クラス）

```
public class J2Kad15X {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("ここはECC苦情処理センターです！");
        System.out.println("優秀なスタッフが対応します！");

        Nobita nobita = new Nobita();
        Jaian jaian = new Jaian();
        Suneo suneo = new Suneo();
        Sizuka sizuka = null;               // まだいない
        Dekisugi dekisugi = null;           // まだいない

        boolean helper = false;           // false：応援なし、true：応援あり

        while(true) {
            System.out.println();
            System.out.print("どうしますか？ (0：苦情を受け取る、1：応援を頼む、-1：もうやだ) ＞");
            int cmd = in.nextInt();
            if (cmd < 0) break;
            // 応援を頼む
            if (cmd == 1) {
                sizuka = new Sizuka();
                dekisugi = new Dekisugi();
                helper = true;
                continue;
            }
            // 苦情処理
            int n = (int)(Math.random() * 100);
            System.out.println("苦情番号：" + n + "を受け付けた！");
            if (!nobita.handle(n)) {
                if (!jaian.handle(n)) {
                    if (!suneo.handle(n)) {
                        if (helper) {
                            if (!sizuka.handle(n)) {
                                dekisugi.handle(n);
                            }
                        }
                    }
                }
            }
        }
        System.out.println("おつかれさまでした！");
    }
}
```

さらに優秀なスタッフたち

スタッフ	対応できる番号
しずか	2 の倍数のとき対応可能
出木杉	7 の倍数以外のとき対応可能

課題完成時の画面

```
ここは ECC 苦情処理センターです！
優秀なスタッフが対応します！
のび太がスタンバイした！
ジャイアンがスタンバイした！
スネ夫がスタンバイした！

どうしますか？（0：苦情を受け取る、1：応援を頼む、-1：もうやだ）＞0
苦情番号：72 を受け付けた！
のび太：専門外です・・・
ジャイアン：私が対応します！

どうしますか？（0：苦情を受け取る、1：応援を頼む、-1：もうやだ）＞0
苦情番号：40 を受け付けた！
のび太：専門外です・・・
ジャイアン：専門外です・・・
スネ夫：専門外です・・・

どうしますか？（0：苦情を受け取る、1：応援を頼む、-1：もうやだ）＞1
しずかがスタンバイした！
出木杉がスタンバイした！

どうしますか？（0：苦情を受け取る、1：応援を頼む、-1：もうやだ）＞0
苦情番号：85 を受け付けた！
のび太：専門外です・・・
ジャイアン：専門外です・・・
スネ夫：専門外です・・・
しずか：専門外です・・・
出木杉：私が対応します！

どうしますか？（0：苦情を受け取る、1：応援を頼む、-1：もうやだ）＞-1
おつかれさまでした！
```

課題完成時の Handler クラス

Handler
…
next ： Handler
…
setNext(next ： Handler) ： Handler

ヒント：のび太が対応できないときはジャイアンへ、ジャイアンが対応できないときはスネ夫へ処理を回すしくみを考える。
（「Chain of Responsibility」←検索、または前期課題 JKad14A を参照←ただしこちらはメソッド版）