情報処理技法
(Javaプログラミング)1

第11回
エラーに対してどう対応する?

人間科学科コミュニケーション専攻
白銀 純子

第11回の内容

- 例外処理

前回の復習問題の解答

- 下記のプログラムで、実行時にエラーが出る箇所と、その理由を説明しなさい。

```
int i, sum = 0;
int score[] = {55, 90, 79, 82, 88};
for (i = 1; i <= 5; i = i + 1) {
  sum = sum + score[i];
}
```

配列の要素数は5つなので、添え字は0～4しか使えないが、
for文で添え字が1～5を使うようになっている。

例外への対処

プログラムで発生するエラー(p. 230)

- プログラムの実行時に発生する可能性のあるエラー
  - 配列で、利用可能な範囲外の添え字を使おうとしたとき
  - String型の値をint型に変換できないとき
    - Ex. 入力された文字列をint型に変換したいときに、「abc」という文字列が入力される、など
  - 入力しようとしたファイルが存在しないとき
  - 数を0で割ろうとしたとき
  - etc.

プログラムを実行してみなければ、発生するかどうかわからないエラー
＝コンパイル時には発見できないエラー

「例外」と呼ぶ

例外に対処するには?(p. 232)

- 例外が発生すると...
  - プログラムの実行がその時点で終了してしまう

- 例外を発生させないためには...?
  1. 例外が発生しないよう、プログラムを書いておく
     - 完全には難しい(入力データなどは実行時でないと判断不可)
  2. 例外に対処するための処理をプログラムに書いておく
     - 例外が発生しても、それなりの処理を行う

例外処理の書き方(基本形)(p. 232)

```
try {
    例外が発生する可能性のある処理
}
catch (例外の種類を表すクラス名　変数名) {
    例外が発生したときに行う処理
}
```

- tryを書いたら、必ずcatchも書かなければならない
- try文の中にcatchを書いてはならない
- tryの「}」の後、catchの前には何も書いてはならない

try～catch(p. 232)

- try
  - 例外が発生する可能性のある処理を、「try{～}」の間に書く
- catch
  - tryの中の処理で例外が発生したときに、行われる処理を書く

tryの処理(1)(p. 232)

- 例外が発生する可能性のある処理
  - 標準入力の処理
  - ファイル入出力の処理

    Javaの文法上の規則として、例外処理を書かなければならないもの
    (書かなければコンパイルエラー)

  - 配列を扱う処理
  - 文字列をint型に変換する処理
  - 割り算の処理

    文法上の規則としては、例外処理を書く必要はないが、必要に応じて
    自分の判断で例外処理を書くもの

  - etc.

tryの処理(2)(p. 232)

- 例外が発生する可能性のあるポイント
  - tryで、例外が発生する可能性のあるポイントをきちんと囲む必要
    - このポイントを囲んでいなければ、例外処理の意味はなし
    - 標準入力やファイル入出力では、このポイントを囲んでいなければ、コンパイルエラー

例外処理の書き方(基本形)(p. 232)

```
try {
    例外が発生する可能性のある処理
}
catch ( 例外の種類を表すクラス名 変数名) {
    例外が発生したときに行う処理
}
```

例外にも様々な種類

catchの処理(例外の種類)(p. 233)

- 例外の種類を表すクラス名
  - 例外には、様々な種類が存在
    - 入出力に関係する例外(入出力ができなかった場合に例外が発生)
    - 配列の添え字に関する例外(利用可能な範囲外の添え字を使おうとしたときに例外が発生)
    - 割り算に関する例外(数を0で割ろうとしたときに例外が発生)
  - tryで発生する可能性のある例外の種類を指定
    - 適切な種類を指定しておかないと、例外処理ができない

例外の種類(IOException)(1)(p. 238)

- IOException
  - 入出力に関する例外
    - 標準入力・ファイル入力で、入力ができない場合
      - 標準入力: プログラムをターミナルから起動していない場合などは、入力不可能
      - ファイル入力: 読み込もうとしたファイルが、「読み込み」のアクセス権がない場合などは入力不可能
    - ファイル出力で、出力ができない場合
      - 書き込もうとしたファイルが、「書き込み」のアクセス権がない場合などは出力不可能

例外の種類(IOException)(2)(p. 238)

- IOException
  - 分類されているパッケージ: java.io
    - 「import java.io.IOException」または「import java.io.*;」がなければコンパイルエラー
  - 例外が発生する可能性の　あるポイントが、tryの中に書かれていなければ、コンパイルエラー
    - ポイント: readLine()メソッド, ファイルを開く処理など

例外の種類(IOException)(3)(p. 238)

標準入力

```
String str;
BufferedReader br = new BufferedReader(new   InputStreamReader(System.in));
try {
    str = br.readLine();
}
catch (IOException e) {
}
```

ファイル入力

```
String str;
try {
   FileReader  fr = new  FileReader("入力するファイルの名前");
   BufferedReader  br = new  BufferedReader(fr);

   str = br.readLine();
   br.close();
}
catch (IOException e) {
}
```

例外が発生する可能性のあるポイント
(実際に入力をしているポイント)

発生する例外は入出力関係、と指定

例外の種類(ArithmeticException)(p. 240)

- ArithmeticException
  - 計算に関する例外
    - 整数の割り算で、数を0で割ろうとした場合(小数の割り算でこの例外の発生はなし)
  - 分類されているパッケージ: java.lang

```
int num1, num2, division;
String str1, str2;
str1 = br.readLine();
str2 = br.readLine();
num1 = Integer.parseInt(str1);
num2 = Integer.parseInt(str2);
try {
    division = num1 / num2;
}
catch (ArithmeticException e) {
}
```

例外が発生する可能性のあるポイント
(割り算をしているポイント)

発生する例外は計算関係、と指定

例外の種類(StringIndexOutOfBoundsException)(p. 241)

- StringIndexOutOfBoundsException
  - 文字列における、文字の位置(インデックス)に関する例外
    - 文字列において、実際には存在しない位置を指定した場合
  - 分類されているパッケージ: java.lang

```
String sub, original = "abcdef";
try {
    sub = original.substring(3, 10);
}
catch(StringIndexOutOfBoundsException e) {
}
```

例外が発生する可能性のあるポイント
(文字列での、文字のインデックスを指定しているポイント)

発生する例外は文字列のインデックス関係、と指定

例外の種類(NumberFormatException)(p. 242)

- NumberFormatException
  - 文字列の数値変換に関する例外
    - 数値に変換することができない文字列を、変換しようとした場合
  - 分類されているパッケージ: java.lang

```
String str;
int num;
BufferedReader br = new BufferedReader(new   InputStreamReader(System.in));
try {
    str = br.readLine();
    num = Integer.parseInt(str);
}
catch (NumberFormatException e) {
}
```

例外が発生する可能性のあるポイント
(文字列を数値に変換しようとしているポイント)

発生する例外は文字列の数値変換の関係、と指定

例外の種類(ArrayIndexOutOfBoundsException)(p. 243)

- ArrayIndexOutOfBoundsException
  - 配列の添え字に関する例外
    - 利用可能な範囲外の添え字を使おうとした場合
  - 分類されているパッケージ: java.lang

```
int[] num = {10, 20, 30, 40, 50};
int i, sum = 0;
try {
    for (i = 0; i < 10; i++) {
        sum = sum + num[i];
    }
}
catch (ArrayIndexOutOfBoundsException e) {
}
```

例外が発生する可能性のあるポイント
(i番目の配列、と配列に添え字をあてはめて使っているポイント)

発生する例外は配列の添え字関係、と指定

例外の種類(IndexOutOfBoundsException)

- IndexOutOfBoundsException
  - ArrayListのインデックスに関する例外
    - 利用可能な範囲外のインデックスを使おうとした場合
  - 分類されているパッケージ: java.lang

```
ArayList<Integer> numList = new ArrayList<Integer>();
numList.add(1);
numList.add(2);
numList.add(3);
try {
    int i, sum = 0;
    for (i = 0; i <= 10; i = i + 1) {
    sum = sum + numList.get(i);
}
catch(IndexOutOfBoundsException e) {
}
```

例外が発生する可能性のあるポイント
(i番目のインデックスの要素を取り出そうとしているポイント)

発生する例外はArrayListのインデックス関係、と指定

例外の種類(FileNotFoundException)(1)(p. 244)

- FileNotFoundException
  - ファイルに関する例外
    - 読み込もうとしたファイルが存在しない場合
  - 分類されているパッケージ: java.io
    - 「import java.io.FileNotFoundException」または「import java.io.*;」がなければコンパイルエラー
  - ただし、IOExceptionを使っていれば、FileNotFoundExceptionは不要
    - IOExceptionは、FileNotFoundExceptionも兼ねている
    - ファイルは存在しても読み書きできないのか、ファイルが存在自体しないのか、を区別したいなどのときには両方利用する

例外の種類(FileNotFoundException)(2)(p. 244)

```
String str;
try {
  FileReader  fr = new  FileReader("入力するファイルの名前");
  BufferedReader  br = new  BufferedReader(fr);

  str = br.readLine();
  br.close();
}
catch (FileNotFoundException e) {
}
```

例外が発生する可能性のあるポイント
(ファイルの読み込みを決めているポイント)

発生する例外はファイル関係、と指定

例外が発生すると...

- 例外が発生した以降の処理が実行されない

```
String str;
int num1, num2, sum;
BufferedReader br = new BufferedReader(new   InputStreamReader(System.in));
try {
    str = br.readLine();
    num1 = Integer.parseInt(str);

    str = br.readLine();
    num2 = Integer.parseInt(str);

    sum = num1 + num2;
}
catch (NumberFormatException e) {
}
```

Ex. ここで例外が発生する

この部分の処理が実行されない

実行されない部分の代わりになる処理をcatchに書く

例外処理の書き方(基本形)(p. 232)

```
try {
    例外が発生する可能性のある処理
}
catch (例外の種類を表すクラス名  変数名) {
    例外が発生したときに行う処理
}
```

発生した例外についての
詳細な情報が格納される

catchの処理(変数)(p. 236)

- 変数名
  - 発生した例外についての詳細な情報が格納される
    - 「例外の種類を表すクラス名」がデータ型

catchの処理(内容)(p. 236)

- 例外の内容を出力することが多い
  - 標準出力で出力することが多い
  - 出力をすることで、プログラムの利用者が、なぜプログラムを実行できないかを知ることができる
- 入力間違いを防ぐ目的で利用されることもある

catchの処理(内容)(例1)(p. 236)

```
String str;
try {
   FileReader  fr = new  FileReader("sample.txt");
   BufferedReader  br = new  BufferedReader(fr);

   str = br.readLine();
   br.close();
}
catch (FileNotFoundException e) {
    System.out.println("sample.txtというファイルは存在しないので、読み込めません。");
}
```

catchの処理(内容)(例2)(p. 236)

```
try {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    str = br.readLine();
    int num = Integer.parseInt(str);
}
catch(NumberFormatException e) {
    System.out.println("入力されたデータは数値ではないため、処理できません。");
}
```

catchの処理(内容)(例3)

```
try {
    String str;
    int num, code = 1;
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("数を1つ入力してください。");
    str = br.readLine();

    while (code == 1) {
     try {
      num = Integer.parseInt(str);
      code = 0;
     }
     catch(NumberFormatException e) {
      System.out.println("入力された文字列は数に変換できません。入力しなおしてください。");
      str = br.readLine();
     }
    }
   }
catch(IOException e) {
    System.out.println("標準入力の処理ができませんでした。");
}
```

while文が終了した時には、必ず変数numに数が入っている

複数種類の例外に対する処理(p. 246)

- 1つのtryの中に複数種類の例外が発生することも

```
String str;
int num;
try {
  FileReader  fr = new  FileReader("sample.txt");
  BufferedReader  br = new  BufferedReader(fr);

  str = br.readLine();
  num = Integer.parseInt(str);
  br.close();
}
```

FileNotFoundExceptionの可能性

IOExceptionの可能性

NumberFormatExceptionの可能性

例外処理の書き方(応用)(p. 246)

```
try {
    例外が発生する可能性のある処理
}
catch (例外の種類を表すクラス名1  変数名) {
    1の例外が発生したときに行う処理
}
catch (例外の種類を表すクラス名2  変数名) {
    2の例外が発生したときに行う処理
}
```

catchはいくつ分書いてもOK

例外処理の書き方(応用)(例)(p. 246)

```
String str;
int num;
try {
   FileReader  fr = new  FileReader("sample.txt");
   BufferedReader  br = new  BufferedReader(fr);

   str = br.readLine();
   num = Integer.parseInt(str);
   br.close();
}
catch(FileNotFoundException e) {
   System.out.println("このファイルは存在しません。");
}
catch(IOException e) {
   System.out.println("このファイルからデータを読み込むことはできません。");
}
catch(NumberFormatException e) {
   System.out.println("読み込んだデータを数値に変換することができません。");
}
```

catchを必要なだけ並べる

catchを複数並べる場合(1)

- 上に書かれたcatchから順にチェックされ、該当したcatchで例外処理
  - if文と同様
    - 複数のcatchに該当する例外であっても、先に書かれているところで例外処理
      (その後のcatchはチェックしない)

catchを複数並べる場合(2)

"sample.txt"ファイルが存在しない
＝ FileNotFoundExceptionもIOExceptionも発生する可能性

```
String str;
int num;
try {
  FileReader  fr = new  FileReader("sample.txt");
  BufferedReader  br = new  BufferedReader(fr);

  str = br.readLine();
  num = Integer.parseInt(str);
  br.close();
}
catch(FileNotFoundException e) {
   System.out.println("このファイルは存在しません。");
}
catch(IOException e) {
   System.out.println("このファイルからデータを読み込むことはできません。");
}
```

FileNotFoundExceptionだけ発生
- IOExceptionは発生しない(2つ目のcatchは処理されない)

catchを複数並べる場合(3)

- catchでの例外処理を書く順序は、原則何でもOK
  - Ex. StringIndexOutOfBoundsExceptionとNumberFormatException
    - StringIndexOutOfBoundsExceptionを先に書いてもOK
    - NumberFormatExceptionを先に書いてもOK
- ただし、IOExceptionとFileNotFoundExceptionは別
  - IOExceptionはFileNotFoundExceptionを兼ねている
    - IOExceptionをFileNotFoundExceptionの前に書くと、FileNotFoundExceptionの例外が発生することはない

コンパイルエラー

FileNotFoundExceptionは、IOExceptionの前に書く必要

やってみよう!

- 教科書p. 254の例題01-06をやってみよう
- 追加
  - 標準入力から1つ文字列を入力し、その文字列のインデックス3から10の部分文字列を取り出すプログラム
    - どんな文字列が入力されても、部分文字列を取り出せるプログラムにすること
      - Ex.「abc」のようにインデックスが10まで存在しない文字列が入力されれば、入力しなおしを求める

期末試験

- 期末試験
  - 7月31日(火) 2限 24102教室
  - 解答時間: 80分
  - 持ち込みすべて可の実技メインの試験
    - 筆記も少しあり