Java プログラミング入門

— 例外処理・ファイル操作 —

早稲田大学

今日の内容

- **コマンドライン引数**
  プログラムの実行時に，コマンドラインから引数としてデータを渡す方法

- **例外処理**
  プログラムの実行時に例外が発生した時に適切に処理を行うための機能

- **ファイル入出力**
  データの入出力をファイルに対して行う

コマンドライン引数

コマンドライン引数

プログラムの実行時にコマンドラインから引数を渡すことができる

例

```
$ java␣SampleArgs␣aaa␣bbb␣ccc ↵
```

```
public static void main(String[] args) {
   ....
}
```

- **コマンドライン引数** (aaa bbb ccc) **は** `String[] args` **に格納されている**
- `args` **は** `String` **型の配列**

コマンドライン引数の使用例

SampleArgs.java

```
public class SampleArgs{
  public static void main(String[] args) {
    for(int i=0;i<args.length;i++){
      System.out.println("args[" + i + "] : " + args[i]);
    }
  }
}
```

```
$ java␣SampleArgs␣abc␣123␣4.5 ↵
args[0] : abc
args[1] : 123
args[2] : 4.5
```

- スペースによりコマンドライン引数を複数入力できる
- コマンドライン引数の個数は `args.length` 個である

文字列から数値への変換（1）

文字列を数値に変換するメソッド

メソッド	説明
`Integer.parseInt(String s)`	文字列 `s` を `int` 型に変換
`Double.parseDouble(String s)`	文字列 `s` を `double` 型に変換

```
String s1 = "123", s2 = "4.5";
int x1;
double x2;

x1 = Integer.parseInt(s1);          — s1 を int 型に変換
x2 = Double.parseDouble(s2);        — s2 を double 型に変換
System.out.println(x1);             — x1 を表示
System.out.println(x2);             — x2 を表示
```

実行例

```
123
4.5
```

文字列から数値への変換（2）

例題

コマンドラインから 2 つの実数を引数として与え，それらの和を計算しなさい．

SumOfDouble.java

```
public class SumOfDouble {
  public static void main(String[] args) {
    double x1, x2;

    if (args.length == 2) {
      x1 = Double.parseDouble(args[0]);
      x2 = Double.parseDouble(args[1]);
      System.out.println(x1 + x2);
    } else {
      System.out.println("error:two arguments are required.");
    }
  }
}
```

例外処理（1）

配列の長さを超えた範囲の配列要素にアクセスしようとすると....

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: **
        at ****.main(sample.java:**)
```

- **プログラムの実行時に例外が発生し，強制終了**

try ブロックと catch ブロック

プログラムの実行時に例外が try ブロックの中で発生したときに、catch ブロックで例外処理を行うことができる

例外処理（2）

try-catch 文

```
try {
    例外が発生する可能性がある文
}
catch (Exception e) {
    例外処理
}
```

- try 節が実行され，何も例外が発生しなければ catch 以降は省略され，例外処理は終了
- 例外が発生すると，そこで処理が中断され，発生した例外が catch に渡される
- その後，発生した例外の内容に応じて例外処理を行う

例外処理（3）

inputInt メソッド

```
public static int inputInt() {
  int a = 0;

  try{
    Scanner sc = new Scanner(System.in);
    a = sc.nextInt();
  }
  catch (Exception e) {
    e.printStackTrace();
    System.exit(1);
  }

  return a;
}
```

ファイルからのデータ入力（1）

SampleFile.java

```
import java.io.File;
import java.util.Scanner;

public class SampleFile{
  public static void main(String[] args) {
    try {
        File file = new File(args[0]);
        Scanner sc = new Scanner(file);
        while( sc.hasNextLine() ){
            System.out.println(sc.nextLine());
        }
        sc.close();
    }
    catch (Exception e) {
        e.printStackTrace();
        System.exit(1);
    }
}
```

ファイルからのデータ入力（2）

sc.nextLine()

- この命令を繰り返すことでファイルのデータを 1 行ずつ読み込むことができる

while(sc.hasNextLine()){

- hasNextLine() は次の行があるとき True を返す。
- False を返すまで while ループする。従って、ファイルのすべてを読み込むことが出来る。

ファイルへのデータ出力（1）

SampleFileOut.java

```
import java.io.*;

public class SampleFileOut {
  public static void main(String[] args) {
    String filename = "sample_number.txt";
    String line;
    try {
      FileWriter fw = new FileWriter(filename);
      BufferedWriter bw = new BufferedWriter(fw);
      PrintWriter pw = new PrintWriter(bw);

      for(int i=1;i<=10;i++){
        System.out.println(i*100);
        pw.println(i*100);
      }
      pw.close();
    }
    catch (Exception e) {
      System.out.println(e);
      System.exit(1);
    }
  }
}
```

ファイルからのデータ出力（2）

String filename = "sample_number.txt";

- **入力されたファイル名**”sample_number.txt”**を** String **型の変数** filename **に設定**

```
FileWriter fw = new FileWriter(filename);
BufferedWriter bw = new BufferedWriter(fw);
PrintWriter pw = new PrintWriter(bw);
```

- **ファイルからデータを読み込むための準備.**
- **この段階で，変数** pw **がファイル**”filename”**に対応する**

pw.println(データ);

- **データを対象となるファイル（**pw**）に** 1 **行で書き込む**