Java講座

第5回

情報科学部コンピュータ科学科
2年　竹中　優

今回の内容

- インナークラス(内部クラス)
- 無名サブクラス
- try-catch文
  - Exceptionクラス
  - Errorクラス
  - throws文
  - throw文
- おまけ

インタークラス(内部クラス)

インナークラス(内部クラス)とは

- Javaでは、クラス内だけで使うような簡単なクラスをそのクラス内に定義することが出来る。

- 次頁に例を示す。

インナークラス(内部クラス)について

```
public class OuterClass{
        //インナークラス
        class InnerClass{

        }
}
```

- インナークラスもクラスのメンバーである。
- インナークラスはフィールドやメソッドと同じように定義する。
(修飾子も同様)

- インナークラスにアクセスするにはアウタークラスのインスタンスが必要。
ただし、修飾子によってはアクセスできない。

- サンプルコード:
java_lec05.samples.Sample_InnerClass.java

インナークラスのサンプルコード

```java
/**
 * アウタークラス<br/>
 * 今回はインナークラスに対してアウタークラスと呼ぶだけで，通常は呼ばない．
 * @author Masaru TAKENAKA
 *
 */
class OuterClass {
    /** インナークラスのインスタンス */
    InnerClass inner;

    public OuterClass(){
        System.out.println("OuterClass.OuterClass()");
        inner = new InnerClass();
    }

    /**
     * インナークラス<br/>
     * 内部クラスとも言う．
     * @author Masaru TAKENAKA
     *
     */
    private class InnerClass {

        public InnerClass(){
            System.out.println("OuterClass.InnerClass.InnerClass()");
        }

    }

}
```

無名サブクラス

無名サブクラス

```
3  public class Sample_NoNameSubClass {
4
5      public static void main(String[] args) {
6          new Sample_NoNameSubClass().start();
7      }
8      private void start(){
9          Test t1 = new Test();
10
11         //無名サブクラス
12         Test t2 = new Test(){
13             @Override
14             public void echo(){
15                 System.out.println("Test.echo()をオーバーライドしました.");
16             }
17         };
18
19         System.out.print("t1 echo -> ");
20         t1.echo();
21         System.out.print("t2 echo -> ");
22         t2.echo();
23     }
24 }
25 /**
26  * Sample_NoNameSubClassクラスで利用するテスト用のクラス
27  * @author Masaru TAKENAKA
28  */
29 class Test {
30     public void echo(){
31         System.out.println("Test.echo()");
32     }
33 }
```

- 20行目t1.echo()と22行目t2.echo()の出力はどうなるだろうか？

実行結果

出力結果

```
t1 echo : Test.echo()
t2 echo : Test.echo()をオーバーライドしました.
```

- t2はTestクラスのインスタンスだが、Testクラスのサブクラスである。

無名サブクラスとは

- Javaでは、
  new クラス名(コンストラクターへの引数){　};
  とすると、{}内でメソッドのオーバーライドのみを行える。
  すなわち、そのクラスに無い、新しいメソッドを定義することは出来ない。

- このように宣言されたクラスを無名サブクラス、または単に無名クラスという。

- 無名サブクラスは、ある箇所でのみメソッドを手軽にオーバーライドしたい場合などに利用する。

try-catch文

try-catch文とは

- エラー処理を行うための構文
- try-catch文を使うことで、
  「○○のエラーが発生した時、××の処理を行う」ということが可能になる。

- Javaでは、
  発生したエラーのことを「例外(Exception)」という。

try-catch文のフォーマット

```
try{
        //例外が発生し得る処理を記述
}
catch(例外のクラス名1　発生した例外の変数名){
        //例外が発生した時の処理を記述
}
catch(例外のクラス名2　発生した例外の変数名){
        //例外が発生した時の処理を記述
}
……
```

- 例外(エラー)が発生し得る箇所をtry{　}で囲む。
- tryのブロック内で例外が発生した場合、対応する例外のcatch(~){　}に処理が移る。
- catch(~){　}はいくつでも記述でき、例外が発生した場合の処理を記述する。

try-catch文の例

```
int [] array = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
try{
    array[10] = 10;//array[10]でエラーが起こるはず
    System.out.println(“catched no exception.”);
}
catch(ArrayIndexOutOfBoundsException e){
    System.out.println(“catched exception.”);
}
```

- 上記の例のようにtry-catch文を記述する。
- catchし得る例外が他にもある場合、catch文を増やしても良い。
- 変数eはcatchした例外についての情報が格納されているので、それらを扱う場合に使用する。
- 例外をcatchした場合、再度try文に戻ることはない。したがって、上記例では
  System.out.println(“catched no exception.”);
  は処理されない。

finally文について

- finally文とはtry-catch文と一緒に使用する構文

- 次項に例を示す

finally文の例

```
int [] array = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
try{
    array[10] = 10;//array[10]でエラーが起こるはず
    System.out.println("catched no exception.");
}
catch(ArrayIndexOutOfBoundsException e){
    System.out.println("catched exception.");
}
finally{
    System.out.println("finally statement");
}
```

- finally文は例のようにcatch文の後に追加する
- finally文は省略可
- finally文の処理は、例外(エラー)が起こっても起こらなくても、最後に処理される。
- サンプルコード:java_lec05.samples.Sample_try_catch.java

throws文とは

- メソッドに対して付加することで、メソッド内である例外が発生し得る、ということを示すことが出来る。

- これを行うことで、メソッド内で例外を処理するのではなく、
メソッドの呼び出し側で例外を処理することが出来る。

throws文の記述

```
戻り値の型　メソッド名(引数) throws 例外クラス名{
        処理;
}
```

- 例外クラス名には、ExceptionまたはErrorクラスのサブクラスであれば何でも良い。
- 複数指定したい場合は、「,」で区切る。

- サンプルコード：
  java_lec05.samples.Sample_throws.java

throw文とは

- これまで「発生した例外」を処理してきた。
- しかし、例外を意図的に発生させたい場合がある。
  そんなときに「throw」を使う。

throw文の記述例

```
void throwTest(){
    ~
    throw new NullPointerException();
    ~
}
```

- 上記のように例外を発生させたい箇所で、「throw　例外のインスタンス;」と記述すれば良い。
- このままだと、プログラムがエラー終了するだけなので、特に理由がない限りtry-catch文で例外処理を行おう。

Exceptionクラスのサブクラス

- NullPointerException
  オブジェクトが生成されていない状態(nullのとき)で、メンバーにアクセスした時に発生する。
- ArrayIndexOutOfBoundsException
  配列の長さを範囲を超えてアクセスした時に発生する。
- NumberFormatException
  Integer.parseInt, Double.parseDoubleなど「文字列→数字」の変換に失敗した時に発生する。
- RuntimeException
  このクラスのサブクラスは例外処理をしなくても良い。ただし、該当する例外が発生した場合は以上終了する。
- IOException
  入出力に関する例外。この例外は滅多に発生しないですが、発生してはいけない例外(エラー)だと思っておけば良いでしょう。

etc.

Exceptionクラス

- ArrayIndexOutOfBoundsExceptionや NullPointerExceptionなど、～Exceptionというクラスは全てExceptionクラスを継承している。
- そのため、catch(Exception e){～} などとすれば全ての例外を処理できる。
  (第4回より、型の同一視)

- ただし、型をExceptionとすると予期しない例外も受け取れてしまうため、特に理由がない場合はオススメしない。

- サンプルコード:
  java_lec05.samples.Sample_Exception.java

Errorクラスとは

- Exceptionクラスのようなものでerrorクラスがある。
- Errorクラスは、Exceptionとは異なり、普通にコーディングしている分には発生しないようなエラーを表すものと覚えておけばよい。
- ErrorもExceptionと同様にcatchブロックの引数に指定できる。

Errorクラスのサブクラス

- OutOfMemoryError
  Javaの仮想メモリ領域が足りなくなったときに発生する(普通はできない)。

- StackOverflowError
  メソッドの再帰呼び出しなどでJavaのスタック領域がオーバーフローしたときに発生する。

- 他は知りません。。。

Errorクラス

- Exceptionクラスと同様に、~Errorと付くクラスはErrorクラスを継承している。
- そのため、catch(Error e){ ~ } などとすれば全てのエラーを処理できる。(オブジェクトの同一視)
- Errorについては、発生すること自体がプログラムの誤りで可能性が高いので、try-catch文で処理しない方がいいかもしれない。
- サンプルコード:
  java_lec05.samples.Sample_Error.java

問題1

- try-catch文を用いて、
  数字が入力された時はコンソールに出力、
  それ以外が入力された時は(数字が入力されるまで)再度入力を促すプログラムを作成せよ。

- 数字とは、整数でも小数でも良い。

Appendix

可変長配列と連想配列

- これ以降のスライドでは、
  より便利な内容について説明していく。

- 知らなくてもまったく問題はないが、知っておくととても便利な機能、クラスである。

可変長配列について

- 可変長配列とは、長さが変わる配列のこと。
- 今までは、
  int[] array = new int[5];
  array = new int[10];
  というように、newで新しく領域を確保しなければ長さは変わらず、また「new」すると各要素の値は0になってしまう。

- 一つの要素だけを末尾に追加したい！
  そんな時に可変長配列を利用する。

可変長配列の例

- Javaに用意されている可変長配列を表すクラスとして、以下のクラスがある。
  - ArrayList
  - LinkedList
  - Vector
  - Stack　(少し違う)

- とりあえず、ArrayListクラスを使えばよい。

ArrayListの使い方

```
ArrayList list = new ArrayList();
list.add(1);
list.add(2);
list.add(3);
list.add(4);
list.add(5);
```

- まず、ArrayListのインスタンスを生成する。
- このとき長さは5、要素は{1, 2, 3, 4, 5}が格納されている。

- 追加、削除、挿入などのメソッドについては次項へ

ArrayListのメソッド

- int size();
  - 配列の大きさ(長さ)を取得する
- boolean add(T), void add(int, T);
  - オブジェクトを末尾に追加する、指定した場所にオブジェクトを挿入する
- T remove(int), boolean remove(T);
  - 指定した場所の要素、またはオブジェクトを削除する
- T get(int);
  - 指定した場所の要素を取得する
- T set(int, T);
  - 指定した場所の要素を引数のオブジェクトで置き換える
- boolean contains(T);
  - 指定したオブジェクトがこの配列に含まれているかどうか
- サンプルコード:java_lec05.samples.Sample_ArrayList.java

テンプレートについて

- 先ほどまでの
  ArrayList list = new ArrayList();
  だと警告(黄色線)が出てしまう。
  このArrayListのオブジェクト(list)が何の配列であるか指定されていないためである。

- これを解消するには、
  ArrayList<Integer> list =
  　　new ArrayList<Integer>();
  と指定することが出来る。

- int型の配列だったら、Integer
  double型の配列だったら、Double
  String型の配列だったら、String
  というようにクラスを指定する。

連想配列について

- まず配列とは、
  添え字という数字一つに対して、値が一つあり、添え字：値 ＝ 1：1である。

- 連想配列とは、
  その名の通り配列のようなものである。
  連想配列では、添え字を表すものが数字でなくても良い。
  つまり、
  文字列：値 ＝ 1：1　など。

- ここで、添え字の役割を持つものをキーという。

連想配列の例

- Javaに用意されている連想配列を表すクラスとして、以下のクラスがある。
  - HashMap
  - LinkedHashMap
  - TreeMap

- とりあえず、HashMapクラスを使えばよい。

HashMapの使い方

```
HashMap<String, Integer> map =
                new HashMap<String, Integer>();
map.put(“A”, 0);
map.put(“B”, 1);
map.put(“C”, 2);
map.put(“D”, 3);
map.put(“E”, 4);
```

- まず、ArrayListと同様に型を指定して、HashMapのインスタンスを生成する。
- このとき長さは5、要素は{“A”=1, “B”=2, “C”=3, “D”=4, “E”=5}が格納されている。
  ただし、HashMapは順不同となる。

- 追加、削除などのメソッドについては次項へ

HashMapのメソッド

- int size();
  - 連想配列の大きさを取得する
- V put(K, V);
  - 指定したキーと値でマップに追加する
- V remove(K);
  - 指定したキーの要素を削除する
- V get(K);
  - 指定したキーを持つ値を取得する
- containsKey(K), containsValue(V);
  - 指定したキー、または値がマップに存在するかどうか
- サンプルコード:
  java_lec05.samples.Sample_HashMap.java

HashMapについて

- 要素は追加した順番に限らず、順不同である。

- 追加した順に並べたい場合は、
  LinkedHashMapクラスを利用しよう。
  使い方はほとんど同じ。

終わり