2016 年度 JAVA 講座第六週目

目次

パッケージ

パッケージとは、Java のクラスやインターフェースをひとまとめにしたもののことである。
第三週 JAVA 講座で使用した動的配列などのコレクションを提供する util パッケージや、数学的な関数を提供する math パッケージ等が標準で用意されている。

パッケージの作成

1. プロジェクトを右クリック
2. 「新規」→「パッケージ」をクリック
3. 「package-info.java を作成する」のチェックを外す
4. 「packageA」という名前を付けて「完了」をクリック

パッケージエクスプローラーの中に、パッケージが追加されていれば、作成されている。
今、作成したパッケージに以下のクラスを追加してみる。

ClassA.java

```
package packageA;

public class ClassA {
	public void print(){
		System.out.println("This is classA in packageA.");
	}
}
```

さらに、メイン関数に以下の記述を追加する。

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

		ClassA classA = new ClassA();
		classA.print();
	}
```

このようにソースコードを記述すると、エラーが発生する。何故なら、メイン関数を実行するクラスのパッケージと ClassA のパッケージが異なるからだ。異なるパッケージのクラスは、特殊な宣言方法でないと使用することができない。

パッケージの使用方法

異なるパッケージを使用するには、import キーワードを使う必要がある。

```
package week6;
import packageA.ClassA;

public class week6 {

	public static void main(String[] args) {

		ClassA classA = new ClassA();
		classA.print();
	}
}
```

そこで、メイン関数を以下の様に変更する。

こうすることで、packageA 内の ClassA をメイン関数のクラスでも使用することが可能になった。

と宣言することで、packageA 内の全てのクラスを使用できるようになる。

```
Import packageA.*;
```

異なるパッケージ同名クラスの宣言

別々のパッケージで、クラス名が重なってしまっても、使用することができる。

そこで、packageB を作成し、その中に以下のクラスを作成する。

classB.java

```
package packageB;

public class ClassA {
	public void print(){
		System.out.println("This is ClassA in packageB");
	}
}
```

さらに、メイン関数を以下の様に記述する。

```
package week6;

public class week6 {

	public static void main(String[] args) {

		packageA.ClassA classA = new packageA.ClassA();
		packageB.ClassA classB = new packageB.ClassA();
		classA.print();
		classB.print();
	}
}
```

こうすることで、異なるパッケージの同じクラスを使用することができる。

パッケージの側面から見たアクセス修飾子

アクセス修飾子	機能
private	同じクラスからのみアクセス可能
なし	同じパッケージからアクセス可能
protected	同じパッケージ、継承先からアクセス可能
public	全クラスからアクセス可能

ラッパークラス

プリミティブ型をオブジェクトに変更する時に、ラッパークラスを使用する。

プリミティブ型のラッパークラスの対応を以下に載せる。

プリミティブ型	ラッパークラス
int	Interger
char	Character
byte	Byte
short	Short
boolean	Boolean
long	Long
float	Float
double	Double

以下に、ラッパークラスの２つ挙げておく。

```
// 文字列の数字を整数値に変換
int num = Integer.parseInt("1000");
// 実数値を文字列に変換
String num = Double.toString(123.456);
```

ラッパークラス利用法:キャスト

ラッパークラス利用法:ArrayList

```
package week6;
import java.util.ArrayList;

public class week6 {
	public static void main(String[] args) {
		ArrayList<Integer> arraylist = new ArrayList<Integer>();
		for(int i=0;i<10;i++)arraylist.add(i);
		for(int i :arraylist)System.out.println(i);
	}
}
```

え ｒ m 例外:Exception

プログラムの実行中にエラーが起きることを例外という。

頻繁に発生する例外としては、

ゼロで割り算を使用した時に発生する「ArithmeticException 」

存在しない null の値を参照しようとした時に発生する「NullPointerException」（通称:ヌルポ）

配列の範囲外に参照しようとすると発生する「ArrayIndexOutOfBoundsException」

などがある。

ぬるぽ

```
package week6;

public class week6 {
	public static void main(String[] args) {
		String str = null;
		System.out.println(str.length());
	}
}
```

上記のソースを実行すると、「NullPointerException」が発生する。
例外への対策方法を大きく分けてふたつある。
まず、例外が発生しないようソースを書くこと。
そして、もうひとつは、try 文を使用すること。

例外処理:try-catch-finaly

```
try{
        例外が発生しそうなプログラムの処理を記述する

}catch(発生した時に対策したい例外を指定する){
        例外は発生した時の処理を記述する

}finally{
        例外の発生に関係なく行う処理を記述する
        省略可能
}
```

try-catch-finally を使用して、例外が起きた時に行うべき処理を書くことができる。

例外処理のサンプルソース

```
package week6;

public class week6 {
      public static void main(String[] args) {
            String str = null;
            //String str = "ぬるぽ";
            //例外が発生しそうなとこでtryを囲む
            try{
            System.out.println(str.length());
            }//catchで指定した例外が発生した時の処理を書いておく
            catch(NullPointerException E){
                  System.out.println("ぬるぽ:nullの値には参照できない");
            }//例外が発生しなくてもしても行う処理を記述するfinally
            //省略可
            finally{
                  System.out.println("プログラムを終了");
            }
      }
}
```

throws キーワードを使用すると例外が発生させる関数を定義することも可能である。
また、例外処理をしないとコンパイルすらしてくれないクラスや関数がある。

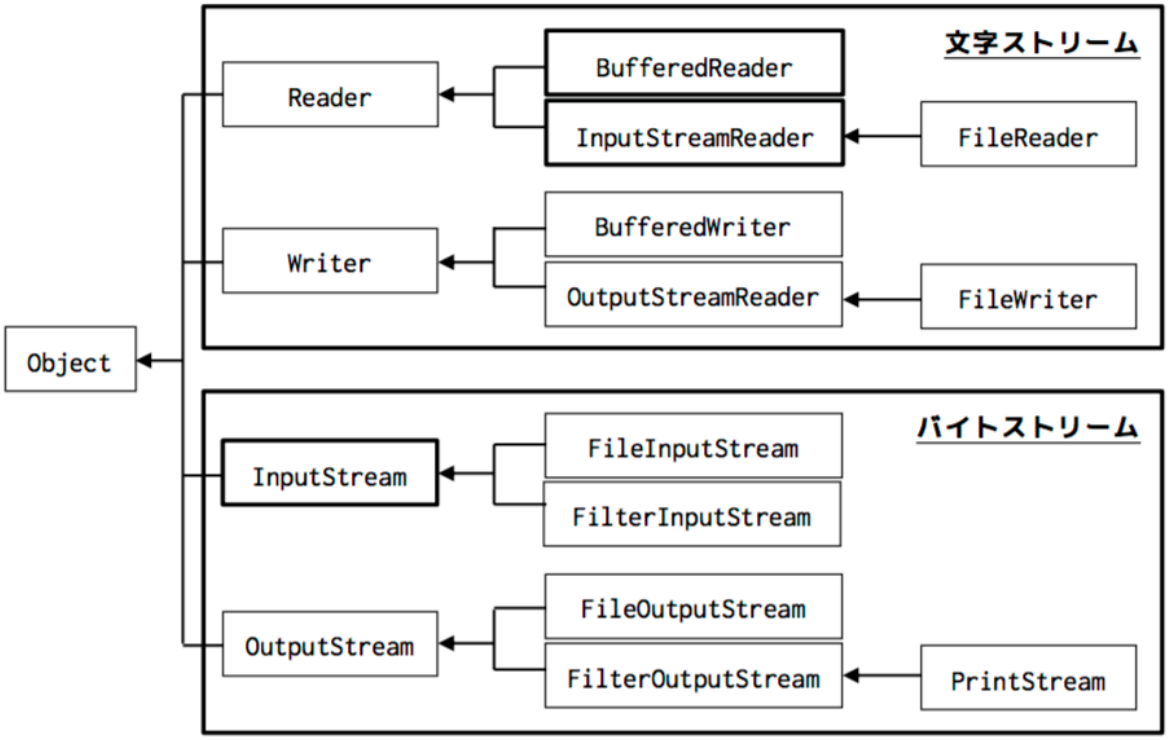
その一つが、標準入力である。

標準入力

Java の標準入力は、様々な知識を必要とする。
Java では、入出力を行うには、ストリームというものを使用する。
元来、stream とは、流れを意味する英語で、Java ではデータの流れを表す抽象的な概念として使われている。これを使うことで、様々な環境で同様なデータの入出力が行える。
文字を出力する際に使用していた System.out は PrintStream というクラスのインスタンスである。PrintStream は文字を出力するストリームである。
キーボードからの入力を受け取る際には、InputStream,　InputStreamReader, BufferedReader が必要になる。

- InputStream : 入力をバイトデータで受け取るストリーム
- InputStreamReader : バイトストリームから文字ストリームに変換するストリーム
- BufferedReader : 文字をバッファリングしてテキストを効率よく読み込むストリーム

ストリームの関係は以下のようになっている。



```java
package w6;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Week6 {

	public static void main(String[] args) {
		//System.in:キーボードからの入力をバイトで受け取る
		//InputStreamReader:バイトデータを文字に変換する
		InputStreamReader isr = new InputStreamReader(System.in);
		//BufferedReader:文字をバッファリングして文字列として読み込む
		BufferedReader br = new BufferedReader(isr);
		try {
		System.out.print("input:");
		String str = br.readLine();
		System.out.println("output:" + str);
		}
		catch(IOException e) {
		System.out.println(e.getMessage());
		}
		}

}
```

※標準入力をするときは、例外処理も同時に記述しないとコンパイルが通らない。

スレッド

スレッドとは、プログラムの処理の単位のことであり、普通のプログラムは一つのスレッドで実行するシングルスレッドで、実行されている。
複数の処理を平行して行うことをマルチスレッドという。
例えば、ゲームなどはマルチスレッドで動いている。Loading 画面などがマルチスレッドで実行されており、ゲームに必要なデータ（音楽、CG 等を）メモリに格納する処理をしながら、画面上には現在ロード中であることを表す Loading 画面を表示している。
ただ、一つの CPU につき、一つの命令しか実行できないので、短時間で複数の命令を切り替えて実行する時分割処理（タイムシェアリング）でマルチスレッドを実現している。
Java では、Thread クラスを継承することで、マルチスレッド処理を行うことができる。
まず、マルチスレッドをするクラスを作成してみる。
Week6.java

```
package w6;

public class Process extends Thread{
	//マルチスレッド処理をさせたい部分をrunメソッドにオーバライド（上書き）する
	public void run(){
		for(int i =0;i<10;i++){
			System.out.println(i);
			try {
				Thread.sleep(200);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
	}
}
```

次に、main メソッドで実行させてみる
出力結果から、２つの処理が同時に行われていることがわかる。

演習問題

1.キーボードから、数字を入力し、その数字の二倍を出力するプログラムを作成せよ(double 型で)。

Sample Input and Output

```
Input: 2.3
Output: 4.6
```

2.キーボードから、２つの数字と四則演算の符号を入力し、その入力をもとに四則演算を行うプログラムを作成せよ（double 型）。例外処理はしなくてもいい。

Sample Input and Output

```
Input:1 + 1
Output:2
```

```
(String[] args) {
ess1 = new Process();
Process process2 = new Process();
//startメソッドで、処理を開始する
process1.start();
process2.start();
}
```

Sample Input and Output2

```
Input: 2 * 3
Output:6
```

ヒント

String 型の split メソッドやキャストを使用するとできるかも