Java講座2017

～第2回～

めんどくさいやつら

1. 継承

- 継承とは前回説明したクラスを違うクラスから楽に扱うこと
- 言葉で伝える力が無いので実際に見ていきます

- 2種類の車を表すクラスCarAとCarBを以下のように定義

```
class CarA{
        public void accele(){
                処理
        }

        public void  brake(){
                処理
        }

        public void hybrid(){
                処理
        }
```

```
class CarB{
        public void accele(){
                処理
        }

        public void  brake(){
                処理
        }

        public void gasoline(){
                処理
        }
```

- acceleとbrakeは同じなのに2回書くのバカらしくない？
- 片方を修正した時もう片方も同様の修正をしないといけない

→同じ部分はまとめて別のクラスに書いちゃえばいい

- 同じ部分をまとめたクラスCar

```
class Car{
        public void accele(){
                処理
        }

        public void  brake(){
                処理
        }
```

- を用意
- このCarを継承したCarAとCarBを以下のように書く

```
class CarA extends Car{
        public void hybrid(){
                処理
        }
```

```
class CarB extends Car{
        public void gasoline(){
                処理
        }
```

- このようにすることでCarAとCarBはaccele、brakeを使うことができ

修正が必要なときはCar内のacceleとbrakeを書き直すだけで済む

- 継承の書き方

- class サブクラス名 extends スーパークラス名{

  ～

  }

- 先ほどの例のCarクラスを親クラス、スーパークラスと呼び
  CarA、CarBクラスを子クラス、サブクラスと呼びます

- Javaでは1つの子クラスに対して1つの親クラスしか継承できません

- プログラムには直接関係ありませんが
  継承の本質は「子クラス is 親クラス」の関係が成り立つということにあります

- 例で言うとCarA is Carです。CarAという製品は車なので当たり前です

- たとえばAirplaneAクラスを考える時に発進はaccele、停止はbrake使えばいいから
  fly()ってメソッドだけCarに追加して継承しちゃおう
  みたいにしてAirplane extends Carとすると「飛行機A is 車」となりおかしいです

- この場合はちゃんと「Airplane」クラスを作成して継承しましょう

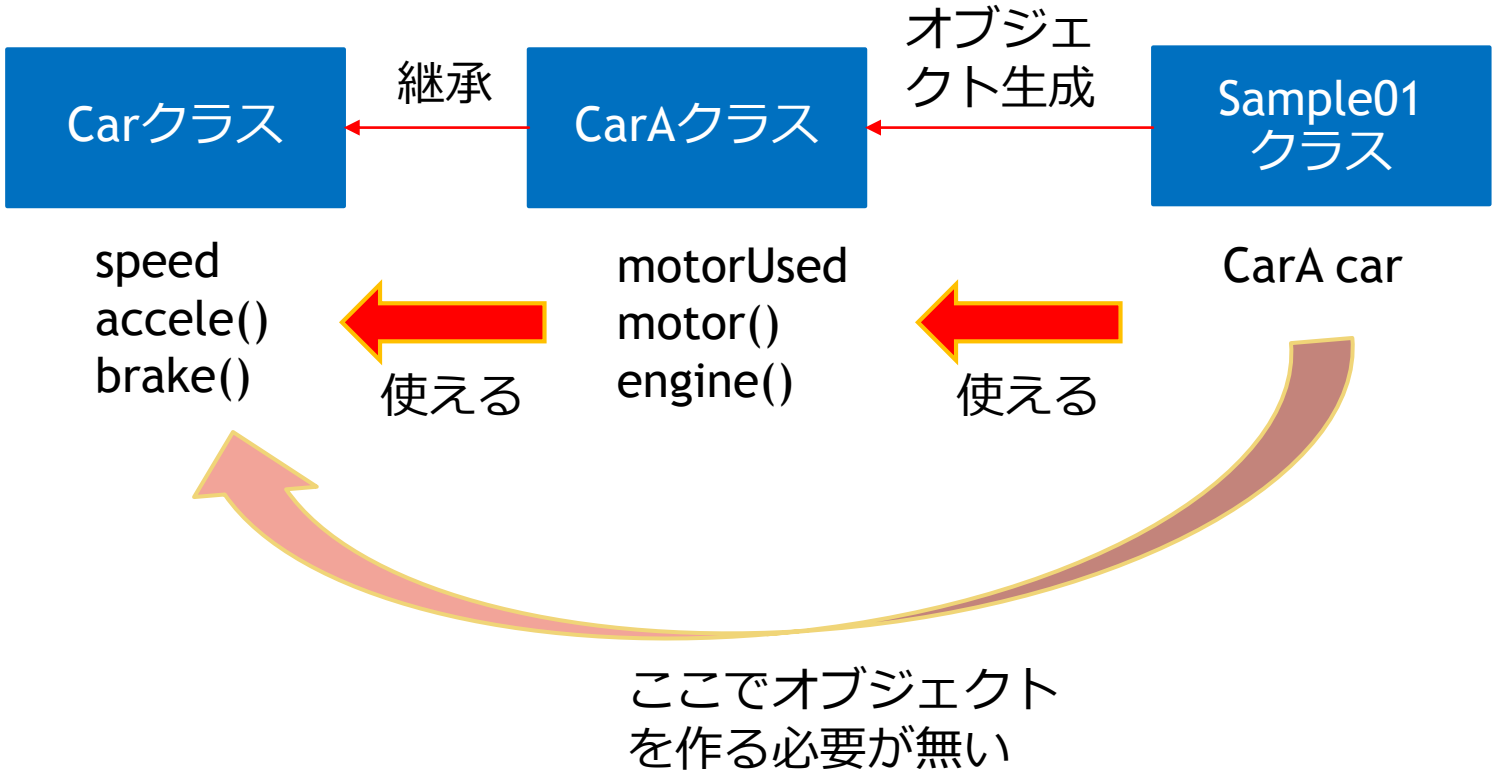
- これを理解して継承を扱ってこそのJavaです
- Carクラスにfly()なんてメソッドがはいっていたらダメ！

- 前回作ったプロジェクト「java01」にパッケージ「lesson02」を追加
- Sample01とCarクラスを以下のように書いてSample01を実行しましょう

```java
package lesson02;

public class Sample01 {

    public static void main(String[] args) {
        CarA car = new CarA();
        car.accele();
        System.out.println("現在" + car.speed + "キロ");

        car.motor();
        if (car.motorUsed == true) {
            System.out.println("モーターで走行中");
        } else if (car.motorUsed == false) {
            System.out.println("エンジンで走行中");
        }
    }
}
```

```java
package lesson02;

class Car {
    int speed = 0;

    public void accele() {
        speed = 10;
    }

    public void brake() {
        speed = 0;
    }
}

class CarA extends Car {
    boolean motorUsed = false;

    public void motor() {
        motorUsed = true;
    }

    public void engine() {
        motorUsed = false;
    }
}
```

- このように子クラスは継承した親クラスの変数や関数を使用可能になる

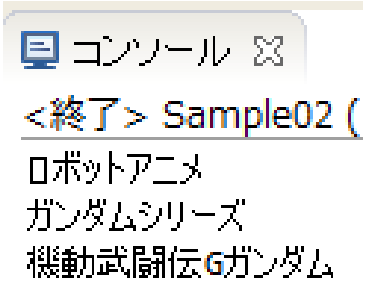


- 続いてコンストラクタを書くとどうなるか
- Sample02とRobotクラスを作る　→　Sample02を実行

```java
package lesson02;

public class Sample02 {

    public static void main(String[] args) {
        Ggundom domon = new Ggundom();
    }
}
```

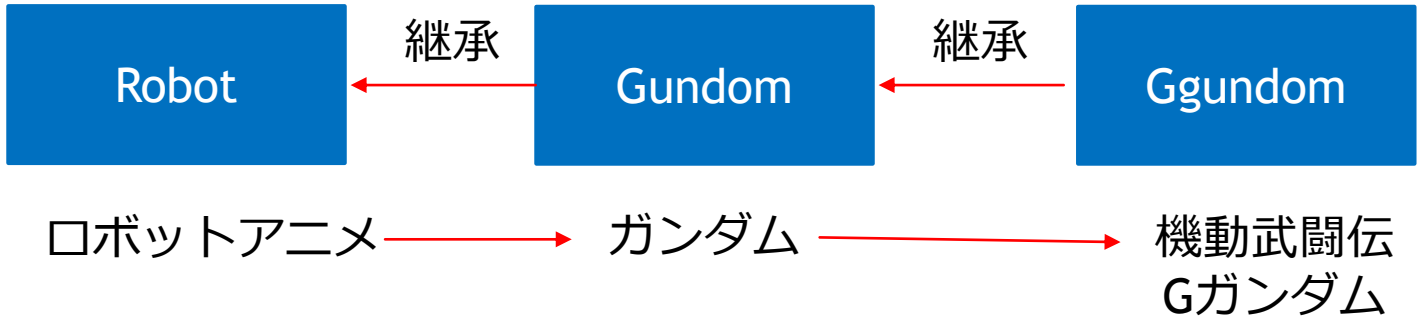
```java
package lesson02;

public class Robot {
    Robot() {
        System.out.println("ロボットアニメ");
    }
}

class Gundom extends Robot {
    Gundom() {
        System.out.println("ガンダムシリーズ");
    }
}

class Ggundom extends Gundom {
    Ggundom() {
        System.out.println("機動武闘伝Gガンダム");
    }
}
```

- 実行すると↓こうなる



- 呼んだコンストラクタはGgundom()のはず？

- 子クラスのコンストラクタは親クラスのコンストラクタを先に実行する



- 子クラスのコンストラクタで明示的に親クラスのコンストラクタを呼ぶとき
- super();
- と書く

  ※super.～と書くと

  親クラスにアクセス可能

```
public class Robot {
    Robot() {
        super();
        System.out.println("ロボットアニメ");
    }
}

class Gundom extends Robot {
    Gundom() {
        super();
        System.out.println("ガンダムシリーズ");
    }
}

class Ggundom extends Gundom {
    Ggundom() {
        super();
        System.out.println("機動武闘伝Gガンダム");
    }
}
```

- 実はさっきの例は→

  ということ

- 親の親であるRobotは

  何をsuperしてる？

- そもそも「クラス」は

  「java.lang.Object」のサブクラスとして作られているので

  ↑のコンストラクタが内部的に呼ばれています

- 次はオーバーライド

- 子クラスのメソッド名と親クラスのメソッド名が被った時
  子クラスのメソッド内容に書き換えることが出来る
  これをメソッドのオーバーライドという

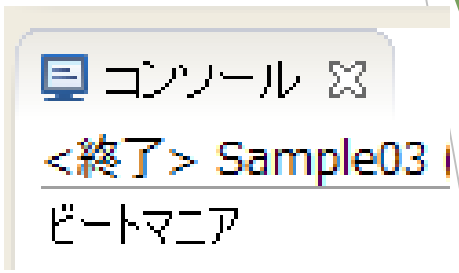
- 例えば
- Otogeというクラスで表示する関数Outを用意
- Otogeを継承したBeatmaniaでもOutを用意

- ※@Overrideはアノテーションといい
  特に書かずともプログラムは動きますが
  このメソッドはオーバーライドしている
  ことを明示的に表すものなので
  書いた方が分かりやすくなります

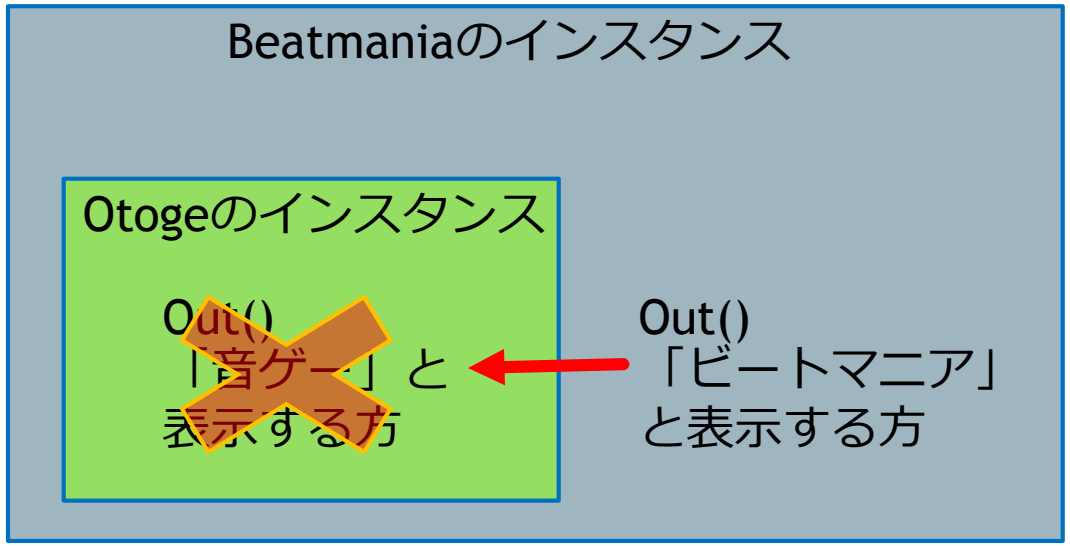
```
public class Otoge {
    public void out() {
        System.out.println("音ゲー");
    }
}

class Beatmania extends Otoge {
    @Override
    public void out() {
        System.out.println("ビートマニア");
    }
}
```

- Sample03というクラスを作り実行してみる

```
public class Sample03 {

    public static void main(String[] args) {
        Beatmania bemani = new Beatmania();
        bemani.out();
    }
}
```



- Otoge内のOutよりBeatmaniaのOutが優先されるため
  「音ゲー」ではなく「ビートマニア」と表示される



2. 抽象クラス

- 継承と関係のとても深いもの
- 継承が分からないと？？？なので今わからない場合、まずは継承を理解しましょう

- とりあえずコードを見る
- →が抽象(abstract)クラス

- 抽象クラスは他クラスから継承して使用します

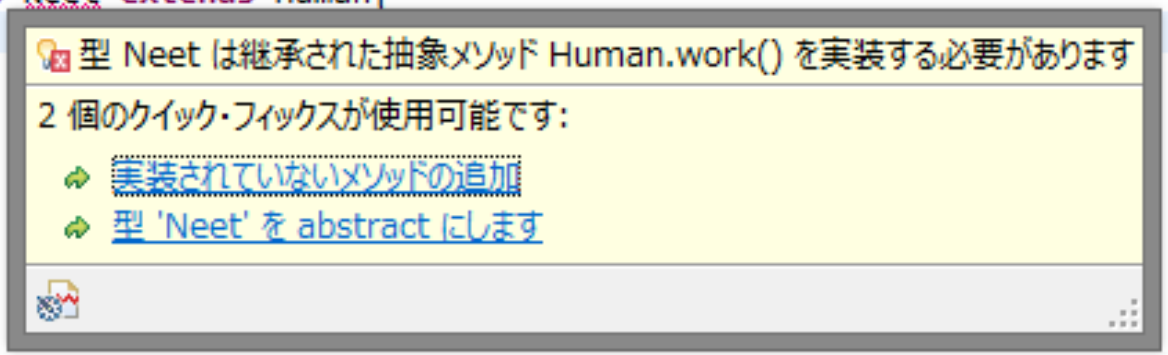
```
public abstract class Human {
    void wakeup() {
        System.out.println("起きました");
    }

    void sleep() {
        System.out.println("寝ました");
    }

    abstract void work();
}
```

- wakeupとsleepは今まで通りの普通のメソッドです
- work();ってメソッド定義の書き方に違反してね？

  →これは抽象(abstarct)クラスであり、抽象クラスの定義の仕方です

- 実際にNeetというクラスから継承しましょう

```
package lesson02;

public class Neet extends Human{

}
```



- Humanを継承するとエラーが出る

  →workというメソッドを実装しろ！というエラー

- 抽象クラスで定義した抽象メソッド→未定義
- 抽象メソッドは継承先でオーバーライドさせ定義を記述させる力を持つ

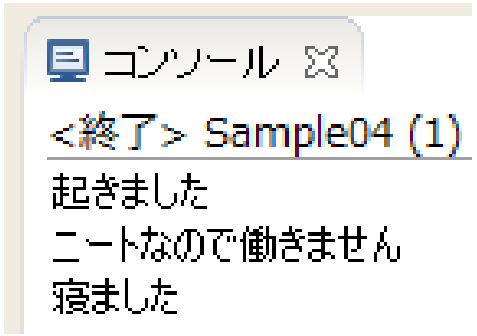
- 今、抽象メソッドwork()を「Human」を継承した「Neet」内で書かなければならない

```
public class Neet extends Human {
    @Override
    void work() {
        System.out.println("ニートなので働きません");
    }
}
```

- ↑のようにオーバーライドしたwork()を書くとエラーが消える

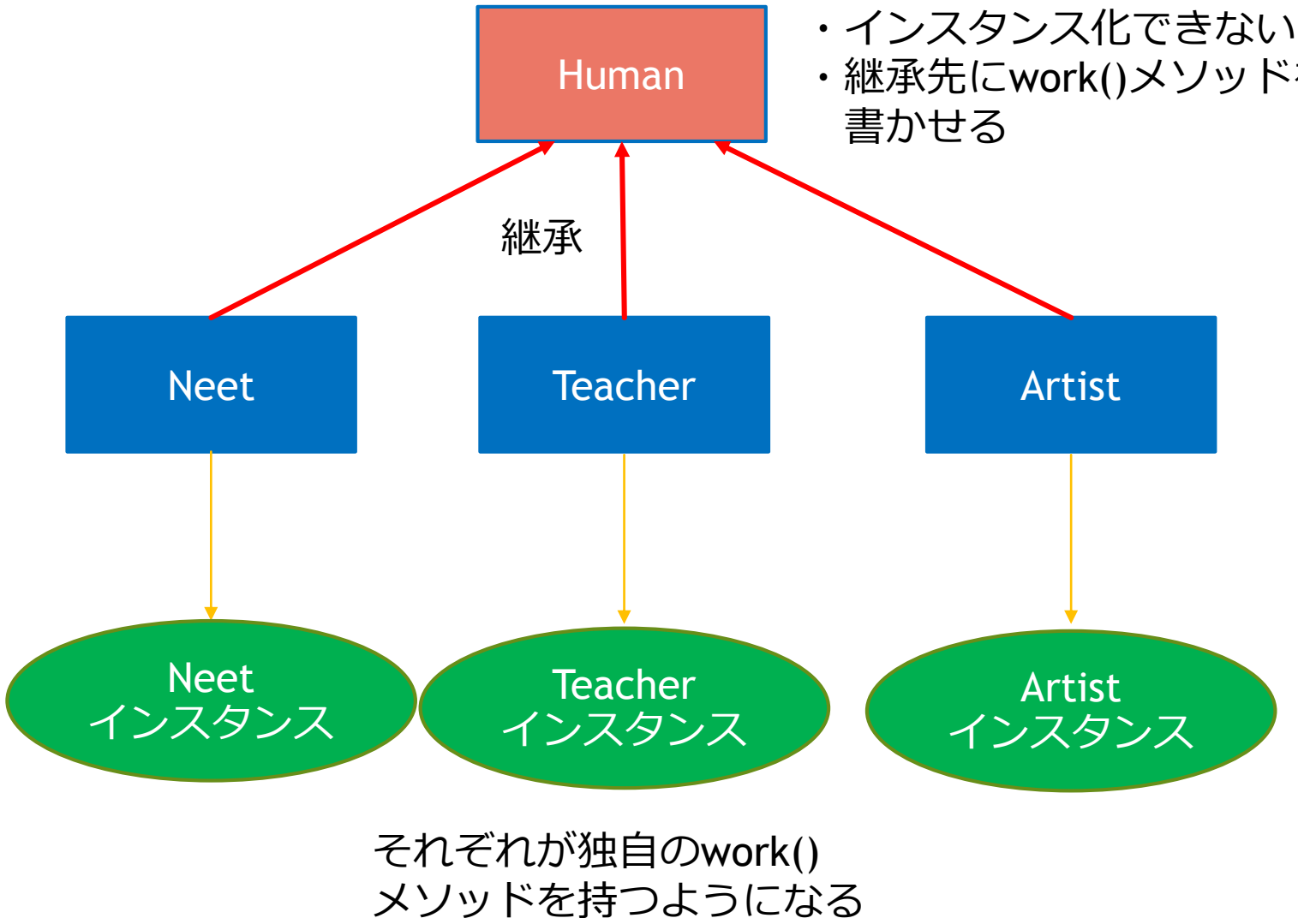
- Sample04というクラスを作成し↓を記述

```
public class Sample04 {
    public static void main(String[] args) {
        Neet neet = new Neet();
        neet.wakeup();
        neet.work();
        neet.sleep();
    }
}
```



- 実行するとオーバーライドしたworkが呼ばれ↑のように表示される

- イメージ図


Human
抽象クラス
・インスタンス化できない
・継承先にwork()メソッドを
書かせる
継承
Neet
Teacher
Artist
Neet
インスタンス
Teacher
インスタンス
Artist
インスタンス
それぞれが独自のwork()
メソッドを持つようになる

- ここで「あれ？抽象クラスなくてもプログラム動くよね？」
  と思う人がいるかもしれません

- その通り！！！抽象クラスなんてなくても普通に動きます
- じゃあ何故書くのか？

- クラス設計を行う際により正確に機能を実装するために書きます
- 今回のような小規模プログラムでは良さが分かりませんが
  規模が大きくなると力を発揮します

- 例えば100個の職業についてworkを書くときHumanを継承すれば
  しまった！ラストの10個にwork書き忘れた！なんて起きなくなります

3. 例外処理

- プログラムを書いていてエラーが起きたことのない人はいないでしょう
  大きく分けて2通り
  - ・コンパイル時のエラー　→　文法ミスがほとんどで書き直せる
  - ・コンパイル通ったのに起きるエラー　→　うわぁぁぁぁぁぁぁ

- というわけで例外処理はプログラムの処理中に何かしらのエラーが発生した場合にエラーで終了。ではなく指定した動作を行ってもらえるように書くものです

- 書き方

- try{
  行いたい処理
  ～
  ～
  }
  catch(例外クラス　変数名){
  例外発生時の処理
  ～
  }
  catch(例外クラス　変数名){
  例外発生時の処理
  ～
  }
  ……

- まずはSample05クラスを作成し↓を書き実行
- 次に→に書き直して実行

```java
package lesson02;

public class Sample05 {
    public static void main(String[] args) {
        int[] num = { 123, 456, 789 };

        for (int i = 0; i < 4; i++) {
            System.out.println(num[i]);
        }
    }
}
```

```
コンソール
<終了> Sample05 [Java アプリケーション] C:¥Software¥Java¥jdk¥bin¥javaw.exe (2017/07
123
456
789
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at lesson02.Sample05.main(Sample05.java:8)
```

```java
package lesson02;

public class Sample05 {
    public static void main(String[] args) {
        int[] num = { 123, 456, 789 };

        try {
            for (int i = 0; i < 4; i++) {
                System.out.println(num[i]);
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("numの配列の範囲を越えました");
        }
    }
}
```

```
コンソール
<終了> Sample05 [Java ア
123
456
789
numの配列の範囲を越えました
```

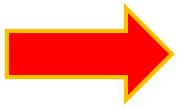
- 例外処理を書いた方ではエラーで終了せず表示したい文章を表示して終了した
- try内でnumの範囲が0~2なのにi=3で例外(配列の範囲を超えた場合)が発生しcatchに進み処理を行う

- try内での例外発生の有無に関係なく絶対にこの処理は行いたい！という場合catch(){}に続けてfinally{}と書く

- 例えば、さっきの例で最後に「プログラムは終了しました」という一文を絶対に表示させたいとき

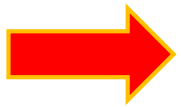
- ↓のように書くとコンソール表示にたどり着く前のfor文i=3でcatchに進む

```
public class Sample05 {
    public static void main(String[] args) {
        int[] num = { 123, 456, 789 };

        try {
            for (int i = 0; i < 4; i++) {
                System.out.println(num[i]);
            }
            System.out.println("プログラムは終了しました");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("numの配列の範囲を越えました");
        }
    }
}
```



- finallyを加え、その中に書くと

```
public class Sample05 {
    public static void main(String[] args) {
        int[] num = { 123, 456, 789 };

        try {
            for (int i = 0; i < 4; i++) {
                System.out.println(num[i]);
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("numの配列の範囲を越えました");
        } finally {
            System.out.println("プログラムは終了しました");
        }
    }
```



- finallyはtryを抜ける瞬間に呼ばれるので絶対に呼ばれる
- 例ではtry(エラー発生、まだtryは抜けない)　→　catch(エラー処理)

→　finally(catchの処理を終え一度tryに戻りすぐ抜ける)

- finallyは主にファイル操作を行う際に役立ちます

4. ファイル操作

- 今までプログラム内に直打ちしていたデータ
  データの量が増えるとテキストなどの外部ファイルから取ってきたくなる

- ファイル操作により外部ファイルの読み書きができるようになります

- ファイルの指定方法
- import java.io.File;　でインポートして使えるようになる

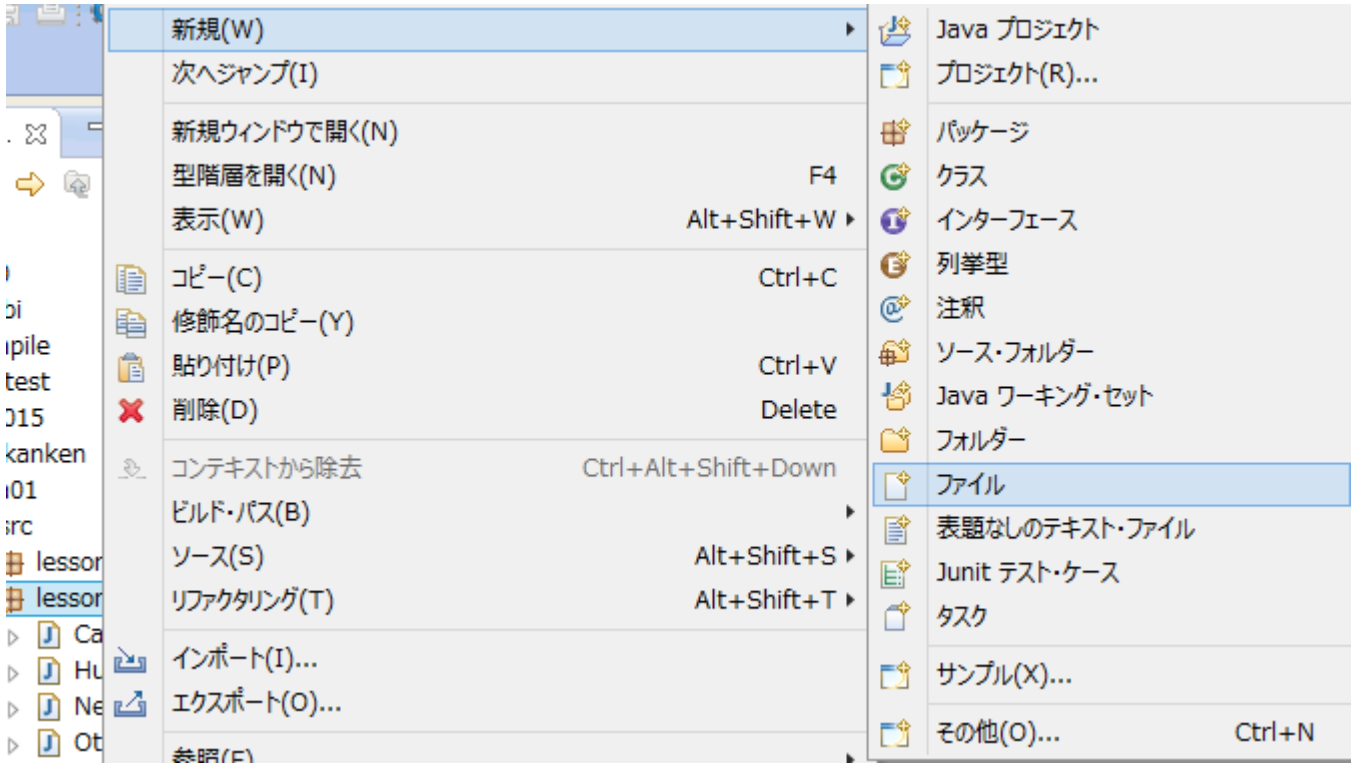
- File(ファイルのパス名)　でファイルを指定
- Fileの持つ関数

  boolean exsits()　→　ファイルorディレクトリがあるときtrue

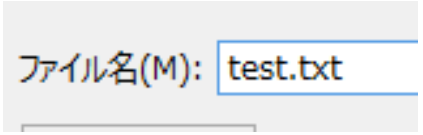
  boolean isDirectory()　→　ディレクトリを指すときtrue

  boolean isFile()　→　ファイルを指すときtrue

- 読み込み用ファイルを用意
- パッケージlesson02を右クリック、新規からファイルを選択



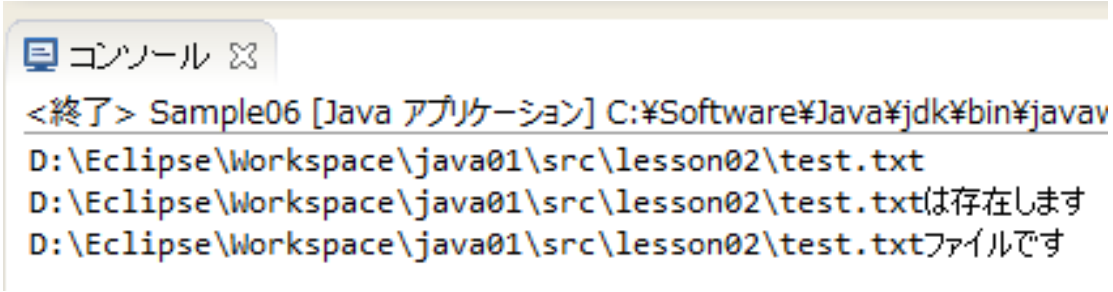
- 「test.txt」と入力し完了をクリック



- lesson02内に「test.txt」があることを確認
- 続けてSample06クラスを作成し↓のように書く

```
package lesson02;

import java.io.File;

public class Sample06 {
    public static void main(String[] args) {
        String path = System.getProperty("user.dir");// 「java01」というプロジェクトのパスまで入る
        path += "\\src\\lesson02\\test.txt"; // 用意した「test.txt」までのパスを追加
        System.out.println(path);
        File file = new File(path);
        if (file.exists()) {
            System.out.println(file + "は存在します");
        }
        if (file.isDirectory()) {
            System.out.println(file + "はディレクトリです");
        }
        if (file.isFile()) {
            System.out.println(file + "ファイルです");
        }
    }

}
```

- test.txtはテキストファイルなので↓のように表示される



- 「D:¥Eclipse¥Workspace¥java01¥src¥lesson02¥test.txt」でtest.txtの場所を示す¥とバックスラッシュは環境によって変わります(意味は同じと考える)
- 続けてこのtest.txtを使用してファイルの読み込みを行う
- test.txtを開き適当に文章を打つ
- 書いたら上書き保存し、Sample06を開く

- →のように書く
- まずは開くファイルのパスを指定、Fileに入れる
- Scanner型変数を用意
- 実際に開く部分は
  例外処理で書きます
- ファイルのオープン・
  クローズを行うとき
  例外処理を書くように

```java
package lesson02;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class Sample06 {
    public static void main(String[] args) {
        String path = System.getProperty("user.dir");// 「java01」というプロジェクトのパスまで入る
        path += "\\src\\lesson02\\test.txt"; // 用意した「test.txt」までのパスを追加
        File file = new File(path);
        Scanner scanner = null;
        try {
            scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                System.out.println(line);
            }
        } catch (FileNotFoundException e) {
            System.out.println(e);
        } finally {
            if (scanner != null) {
                scanner.close();
            }
        }
    }
```

- tryの中でファイルをScanner変数に入れる
- Scanner(File)でFileに入っているファイルをオープンします
- boolean hasNextLine()
  オープンしたファイルに次の行がある場合trueを返す
- String nextLine()
  次の行に進み、進んだ先の行をString型に入れる

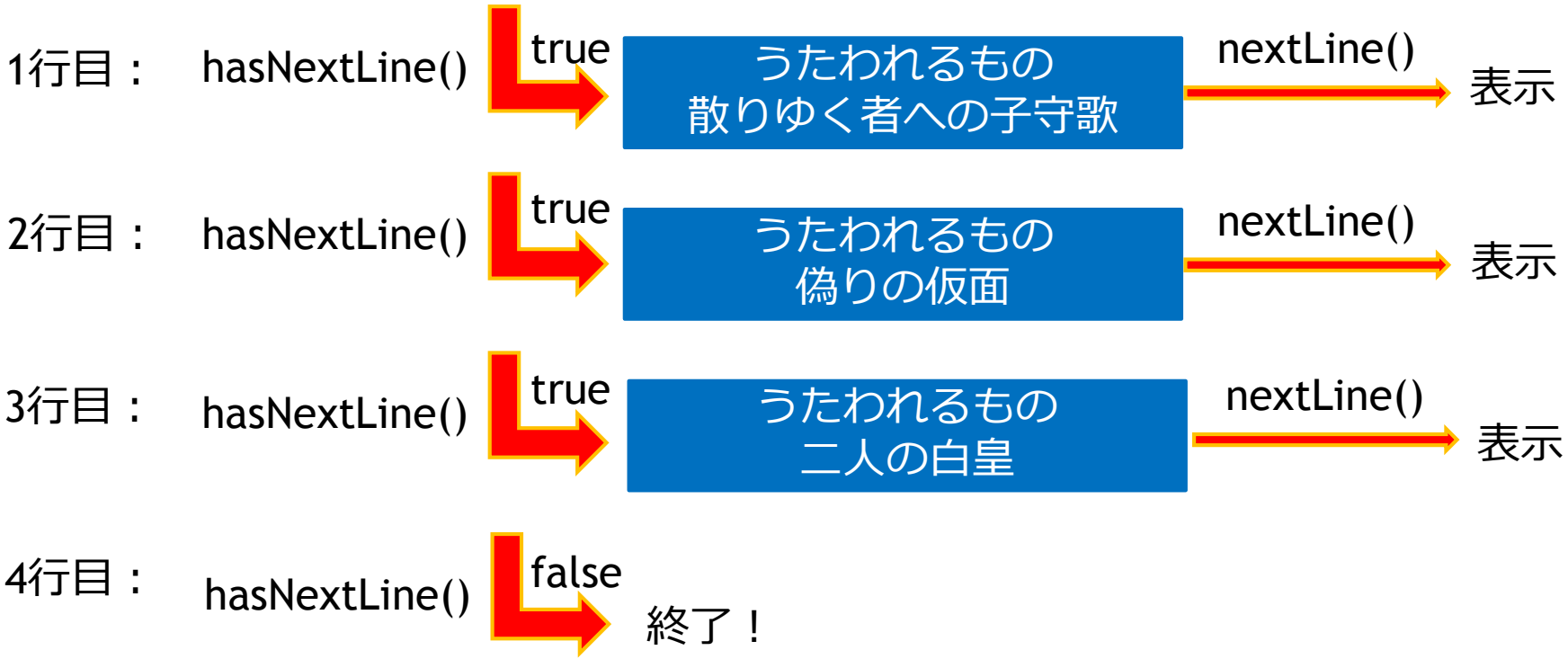
- catch(FileNotFoundException e)
  ファイルオープンで開くファイルが見つからないとき、上記エラーを受け取る

- finallyの中身は必ず呼ばれる
  →オープンしたファイルは必ずクローズする必要があります
  エラーが出て終了してしまう場合でもクローズしたい
  のでfinally内にscanner.close()でファイルをクローズする

- test.txtに→のように書く場合

うたわれるもの　散りゆく者への子守歌
うたわれるもの　偽りの仮面
うたわれるもの　二人の白皇

- File fileにtest.txtが入る
- Scanner(file)で上記テキストファイルをオープン



- 最後finallyでファイルクローズ

- ファイルのパス名を～test.txtから
- ～tests.txtとかちょっと変えて存在しないファイルにしてから実行してみてください

- すると例外処理されてcatch内の処理が行われてファイルの中身は何も表示されない

- 最後にファイルの書き込み

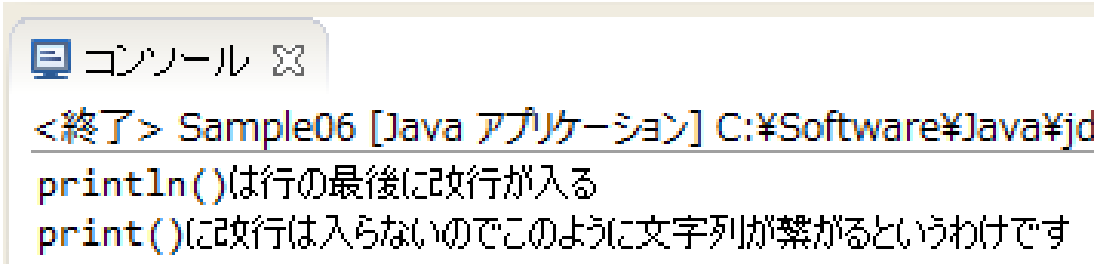
- Sample07を作成し↓を書く

- 今回はtest.txtを再度指定する
- PrintWriter(File)にFileが既に
  存在する場合上書き
  存在しない場合ファイルを新規作成

- print()
  今の行に内容追加
- println()
  今の行に内容追加+行末で改行

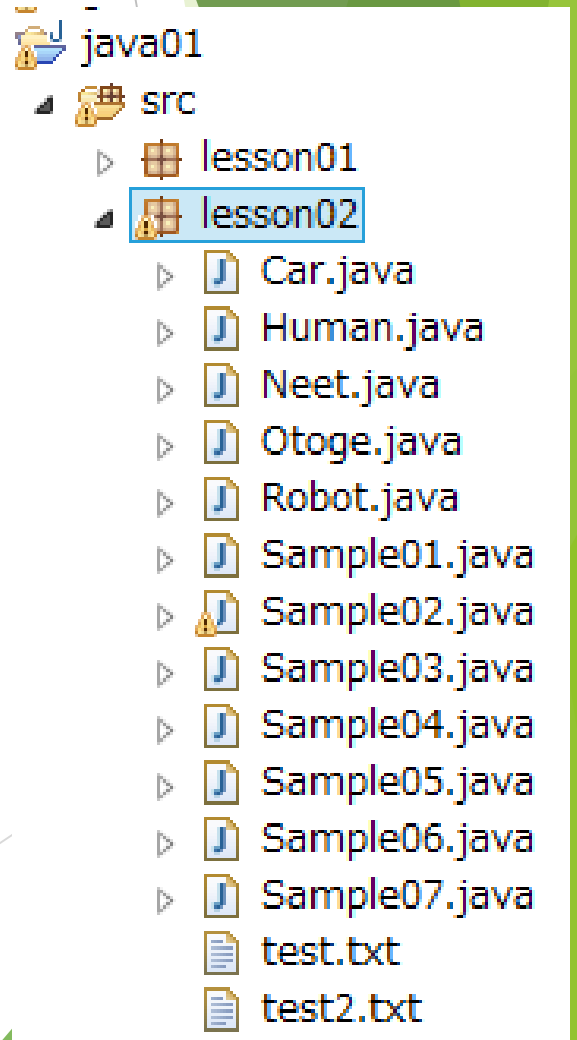
```
package lesson02;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class Sample07 {
    public static void main(String[] args) {
        String path = System.getProperty("user.dir");
        path += "\\src\\lesson02\\test.txt";
        File file = new File(path);
        PrintWriter writer = null;
        try {
            writer = new PrintWriter(file);
            writer.println("println()は行の最後に改行が入る");
            writer.print("print()に改行は入らない");
            writer.print("のでこのように文字列が繋がる");
            writer.println("というわけです");
        } catch (FileNotFoundException e) {
            System.out.println(e);
        } finally {
            if (writer != null) {
                writer.close();
            }
        }
    }
}
```

- 上手く動くとtest.txtに上書きできているので
- 先ほどのSample06でtest.txtを読み込みましょう



- 自分で直接書いた内容が上書きされ↑の文章が表示される

- Sample07のpathを～test.txtから～test2.txtに変えて実行してみましょう
- 上手く動いた人は→の画面のlesson02を選択した状態でF5を押す
- するとtest2.txtが作成されていることが分かる
  （内容はtest.txtと同じ）



- 本日はここまで
- とりあえず一通りやりました

- あとスレッドについてが残っているのですが
- 講座自体はやらないつもりです
- が、後日資料だけ作成してk3HPに上げるかもしれません

- ファイル読み書きとか違う書き方があるので調べてみてください
- 継承とか抽象クラスとかを理解してもらえるとjavaの良さが分かるんじゃないでしょうか？

みなさんお疲れさまでした！