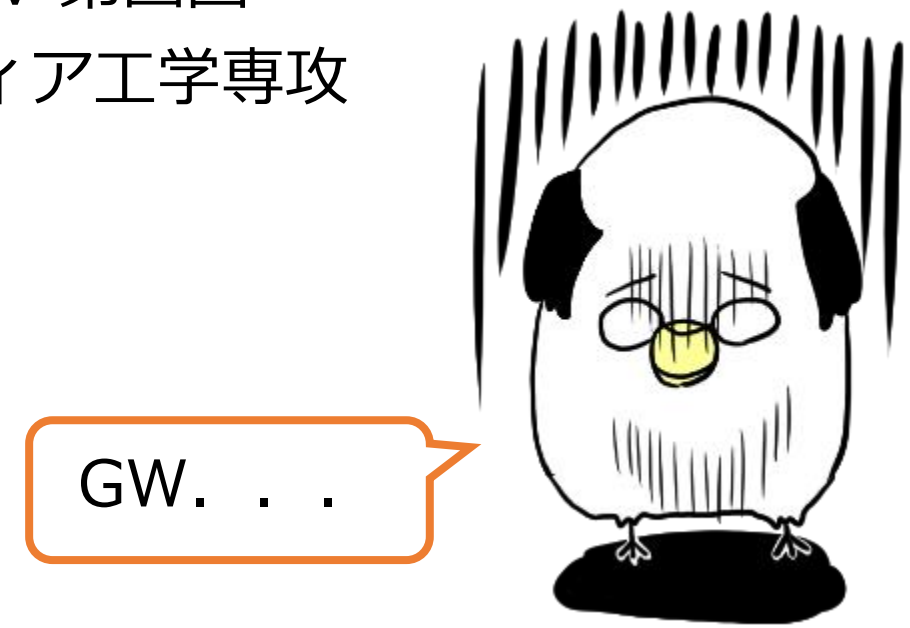
ポリモーフィズム

別名　多態性

2018/5/1(火)　プログラミングIV 第四回

福井大学　工学研究科　情報・メディア工学専攻

長谷川達人



演習問題の解説

- HP上にアップロードした.
- 難所？はコメントしてあるので各自確認すること.

- 学籍番号を入力すると，成績を表示する仕組みを開発してみましたので使ってみてください.

連絡事項

- 当初PDFにコメントを打てること，多分穴埋めが出てくるので各所メモすることと伝えていたかもしれないが，プロIVでは基本穴埋めはなくした.（お互いに面倒だし，意味なさそうなので）
- ただし，練習問題の解答などは，画面にのみ表示したり，手書きしたりするので注意すること.
- 書ききれなくて進んでしまうときには，挙手して言ってくれれば少し表示を延長する.

質問に対する回答
抽象クラスとインタフェース

- 抽象クラスとインタフェースって何
  - どちらも抽象メソッド（詳細未定メソッド）を持つ
  - **抽象メソッドのみを持つ**のがインタフェース
  - **加えてフィールドや普通のメソッドを持つ**のが抽象クラス
- 抽象クラスやインタフェースのメリットは？
  - 共通部分を事前に定義しておくことで，後世の開発者が共通見解を持って開発できる（開発ルールを定義できる）.
  - 本日学習するポリモーフィズムが使える.
- なぜabstractなのか
  - abstract: 【直訳】抽象的なもの，摘要

質問に対する回答
抽象クラス，インタフェースの書き方

- abstract，interfaceの定義
  - 抽象クラスを定義する **abstract class**
  - インタフェースを定義する **interface**

```
public abstract class Monster {
  protected int hp = 0;
  public abstract void attack();
}
```

```
public interface Magicable{
  void magic();
}
```

public abstract扱いとなる
明示してもよいが意味はない

- extends，implementesの順序
  - **public class [クラス] extends [クラスor抽象クラス]**
    implements [インタフェース]{ }

```
public class Witch extends Monster implements Magicable{
  @Override
  public void attack() { }
  @Override
  public void magic() { }
}
```

interfaceは動作だけを切り離して各所で
共同利用したいときに定義することが多い

質問に対する回答

implementsした際の動作

- 継承先（extends）と実装先（implements）で同じメソッドがあった場合どうなるのか
  - なんら問題はない.
  - メソッドの処理が書かれている可能性があるのは継承先のみであるため，オーバーライドされなければ継承先メソッドが呼ばれるだけである.
- 複数のimplements先が同じメソッド名attack()を持っていた場合どのattack()がオーバーライドされるのか.
  - まとめてattack()としてオーバーライドされる.
  - Monster implements Attack1, Attack2の場合，Monsterにattack()があることが重要であり，Attack1のattack()をオーバーライドしたのかAttack2なのかはどうでもよい.

質問に対する回答

抽象化の利便性

- 抽象化は多用しすぎると逆にごちゃごちゃしそう
  - その通り．なんでもかんでも抽象化するというよりは，
    1. オブジェクトを設計する際に必要なものを洗い出して
    2. 今後開発され得るものを洗い出して
    3. 共通にしておく方が良いものを抽象化する　方が良いだろう．
- 抽象化したらヒューマンエラーはなくなるのか．
  - なくならない．が，やらないよりは減らせる可能性は高まる．
  - 今回のようにJavaではヒューマンエラーを減らせる仕組み自体は多く準備されているため，それらを使いこなせる人と知らない人では技術力が圧倒的に差が出る．

質問に対する回答
newとオーバーライド

- newされるされないがよくわからなかった.
  されると困る状況もわからなかった.
  - 抽象クラスとして定義されたクラスはnewできない．例えば
    abstract class Monsterはmain()で
    Monster m = new Monster()はコンパイルエラーとなる.
  - 抽象クラスはまだ実装できていない抽象メソッドを持つので
    インスタンス化されると実行できないメソッドができて困る.
- オーバーライドを忘れたらどうなるのか.
  - 親クラスがメソッドを実装している場合→それが実行される.
  - 親クラスが中小メソッドの場合→オーバーライドしないとコンパイルエラーになる.

質問に対する回答
インタフェースの定数とデフォルト

- インタフェースが持つフィールドが良くわからない
  - インタフェースは基本的にはフィールドを持てない.
  - あえてフィールドを明記すると自動的にpublic static final 扱い（すなわち，定数）となる.

```
public interface WeatherListener {
  int WEATHER_SUNNY = 1;
  int WEATHER_CLOUDY = 2;
  int WEATHER_RAINY = 3;
  void getWeather(int weather);
}
```

自動でstatic定数扱いとなる

メソッドで使う定数などを
定義するとき等によく使われる

- インタフェースのdefaultメソッドが良くわからない
  - そんなに使われないと思われるので，積極的に覚える必要はない．一応，メソッドの実装も可能だよというだけである.

質問に対する回答
インタフェースの継承

- interfaceを継承しinterfaceを作る時，なぜextends
  - class <- class や interface <- interface は extends
  - interface <- class は implements と覚えるしかない.
- interfaceを継承しinterfaceを作る意味が分からない
  - インタフェースは動作の設計図を作りたい時などに作る.
  - 魔法を扱えるMagicableを定義したが，すごい魔法を扱えるmagicDx()のインタフェースを考えたときに，Magicableを継承したMagicableDxを定義すると，後者をimplementsするだけでmagic()もmagicDx()も実装できる.

質問に対する回答

修飾子

- 修飾子が良くわからなくなった.
  - アクセス修飾子
    - **public** : すべてのクラスからアクセスできる.
    - **private** : 自クラスからのみアクセスできる.
    - **protected** : 自分を継承したクラスからor 同じパッケージに属するクラスからアクセスできる.
    - 何も書かない : 同じパッケージに属するクラスからアクセスできる.
  - **final**修飾子 : 以降変更を許可しないことを示す（＝定数）
  - **static**修飾子 : 静的フィールドや静的メソッドを定義する.
    ＞静的フィールドはクラス1つにつき唯一の値を保有することができる.
    ＞静的メソッド，静的フィールドともにインスタンス化せずに扱える.

質問に対する回答

その他

- 練習問題のプログラムを打ち終わる前に進んでしまう
  - すみません善処します．全体の雰囲気である程度の人が入力を終えられるであろう時間は確保しているつもりでした.
- 小規模開発だとオブジェクト指向は逆に面倒
  - 現状のだとそう感じる面もあるが，今後先人の開発したクラスを使用できることを体験すると気持ちが変わるはず.
- ジェネリクスを学ぶ前に復習しておくとよいところは
  - 強いてあげるならば本日のポリモーフィズムである.

質問に対する回答

評価に関して

- プロIIを受講していないので試験方法がわからない
  - WebClassというシステム上で，選択式や穴埋め式の知識を問う問題を出題する予定である.
  - 前回，eclipseで開発＋提出予定とも伝えたが，よく考えると毎回演習で評価しているので，WebClassだけにするかも.
  - それに伴い点数配分も，毎週演習（4割），自由制作（3割），期末試験（3割）にするかもしれないが，これも未定.
- 演習の配点は？
  - 毎週100点満点で，要件を満たしている比率で採点している.
- 自由開発は自宅でも開発しUSBで持ってきてもよいか.
  - 全然OK．面白い作品に期待している.

抽象クラス，インタフェース

復習問題１：抽象化する練習

具体例やうまい使い方に関する質問が多かったので練習

以下のクラスを開発する予定である．今後のことも含めると抽象クラスとインタフェースを準備しておいた方が良いだろう．抽象クラスとインタフェースを設計せよ．

今後設計予定

家庭用掃除機	ポータブル掃除機	携帯電話	家庭用電話	スマホ
String 型番;	String 型番;	String 型番;	String 型番;	・・・
型番を表示する(); ごみを吸引する(); 給電する();	型番を表示する(); ごみを吸引する(); 給電する(); 充電する();	型番を表示する(); 通話先を選択する(); 通話する(); 充電する();	型番を表示する(); 通話先を選択する(); 通話する();	・・・

※ごみを吸引する();と通話する();以外は内部動作は異なるものとする．

＞すなわち，各々が異なる内部動作を実装する必要があるものとする．

＞例えば，家庭用掃除機の給電する()はコードから，ポータブルはバッテリーからである．

抽象クラス，インタフェース

復習問題１：抽象化する練習（解答欄）

本講義の概要

前半		後半	
第1回	基本文法の復習	第9回	標準ライブラリ
第2回	クラス～カプセル化の復習	第10回	ファイル入出力
第3回	抽象クラス，インタフェース	第11回	デバッグ，インポート，高速化
第4回	**ポリモーフィズム**	第12回	オブジェクト指向
第5回	GUI：Canvas	第13回	自由開発演習 1
第6回	GUI：Layout，イベントリスナ	第14回	自由開発演習 2
第7回	スレッド，例外処理	第15回	自由開発演習発表会
第8回	ジェネリクス，コレクション	第16回	期末試験

何を開発するか，少しずつ考えておくこと

本日の目標

概要

Javaの三大要素の最後の砦，ポリモーフィズム
とは何かについて学習する.

目標

ポリモーフィズムをなんとなく説明できる.

本日の提出課題

講義パート

課題を意識しながら
講義を聞くと良い.

課題 1

本日の授業を聞いて,
よくわかったと思う内容を2点簡潔に述べよ.

課題 2

本日の授業を聞いて,
質問事項または**気になった点**を1点以上簡潔に述べよ.

課題 3

感想（あれば）

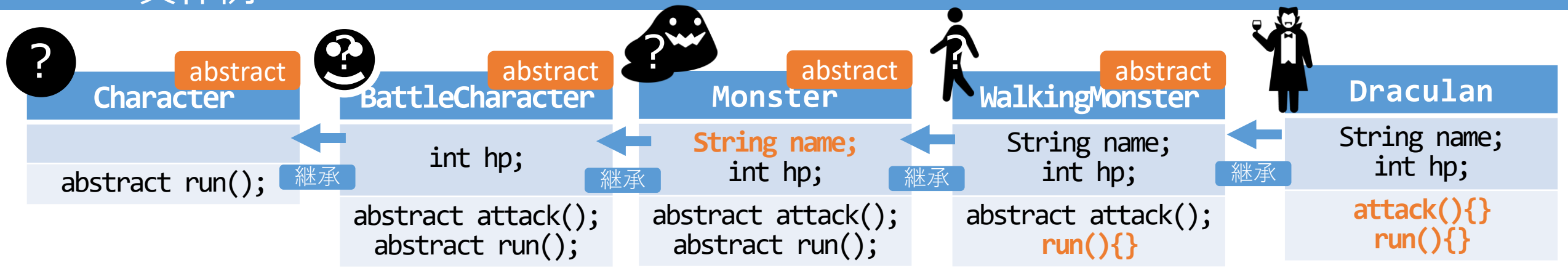
ポリモーフィズムとは

別名：多態性

型を**あえてザックリと**
捉える方法である

ポリモーフィズム

具体例



- Doraculanってザックリ捉えるとWalkingMonsterだよね．もっとザックリ捉えるとMonsterだよね.
- Slimiってザックリ捉えるとMonsterだよね.
- プリウスってザックリ捉えると車だよね.

ポリモーフィズム

ザックリ捉えるメリット

- プリウスってザックリ捉えると車だよね.

なので，プリウスは運転したことないけれど，
アクセル踏めば進むし，ハンドル回せば曲がるよね.

同様に，来年出るらしいアテンザのMCも，
アクセル踏めば進むし，ハンドル回せば曲がるよね.

厳密に言えば，アテンザには自動ブレーキONOFFのボタンがついていたり，
　・使える機能（メソッド）の種類が違う
　・アクセルを踏み，タイヤを動かすまでの内部での処理工程も違う
けれど，車を前に進ませるという目的だけで見ると車は大体同じだよねと捉える

ポリモーフィズム

ザックリ捉えるメリット

- DoraculanってザックリMonsterだよね.

なので，Doraculanインスタンスは使ったことないけれど，damage(int)でダメージを与えられるよね.

同様に，誰かが将来開発するLastBossnも，damage(int)でダメージを与えられるよね.

厳密に言えば，LastBossnにはラスボス独自のエフェクトがあったり，
・使える機能（メソッド）の種類が違う
・damage(int)でもバリアで値を軽減するなど，ダメージを受ける処理も違うけれど，Monsterに攻撃する点だけで見るとMonsterは大体同じだよねと捉える

ポリモーフィズム

ザックリ捉えるメリット

- 敵のAIを開発する側から見ると，Doraculanの使える機能とLastBossnの機能がちゃんとわかっていないと，強い敵を作る事ができない.
- 一方，勇者軍の攻撃処理等を開発する側から見ると，DoraculanでもLastBossnでもMonsterであることがわかれば何でも良い.

Draculan
String name; int hp;
attack(){} damage(int){}

LastBossn
String name; int hp;
attack(){} damage(int){} bigbang(){}

Monsterなら
誰でもいい

Knight
String name; int hp; int mp;
attack(){} damage(int){}

ポリモーフィズム

使い方

継承（extends）やカプセル化（private）のような

特別な書き方はない

ザックリ捉えると**Monster**だよね，**Slimi**って

```
Monster m = new Slimi();
```

箱の型　　**中身の型**

ポリモーフィズム

諸注意

Knight（戦士）はMonster（敵）の箱には入れない

誰でもどの箱にでも入れる<u>わけではない</u>

ザックリ捉えると**A**だよね，**B**って

```
A a = new B();
```

箱の型　　**中身の型**

が成立するとき**だけ**代入が可能

→ `extends` `implements` **継承**，**実装**関係があるときだけ

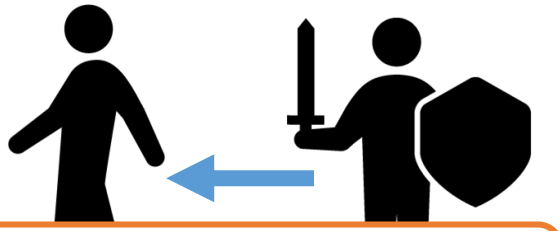
ポリモーフィズム

諸注意

逆の視点で見ると．．．

継承や実装はポリモーフィズムな扱いができるとJavaに宣言する役割も兼ねる

HyperMarikoはザックリ捉えるとMariko→○
Knightはザックリ捉えるとHuman　→○
Marikoはザックリ捉えるとHuman　→×

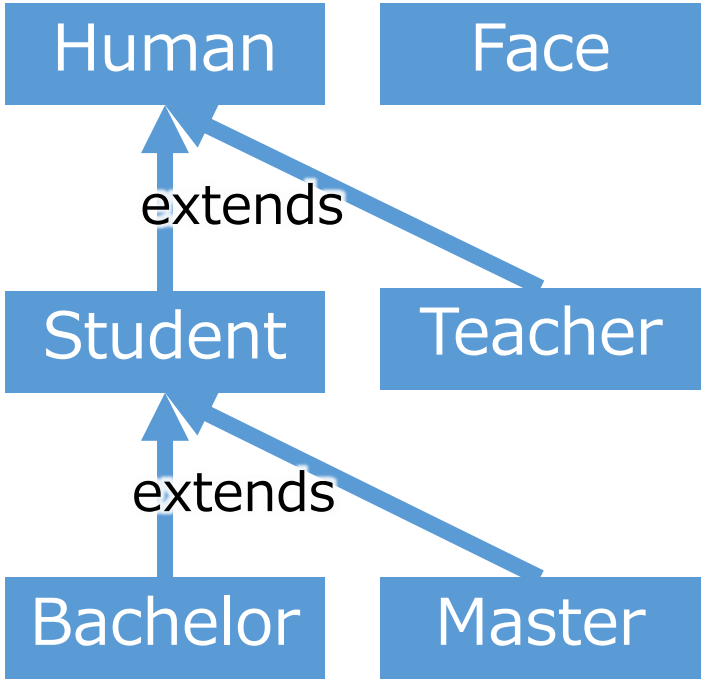


現実世界ではMarikoは恐らくHumanなのだが，
implements等Java的に宣言されていない場合は
ポリモーフィズムな扱いができない．

ポリモーフィズム

練習問題１：ザックリ捉えられるのか

左図の継承/実装関係を前提とした時，右のコードのエラー可否を判定せよ.




```
Teacher t1 = new Teacher();      // (1)
Teacher t2 = new Bachelor();     // (2)

Student s1 = new Master();       // (3)
Student s2 = new Teacher();      // (4)

Human h1 = new Teacher();        // (5)
Human h2 = new Master();         // (6)

Face f1 = new Teacher();         // (7)
Master m1 = new Student();       // (8)
```

ポリモーフィズム

練習問題２：試してみるポリモーフィズム

先週の練習問題で作成した，Monster，Magicable，Wizardnクラスでポリモーフィズムを試してみよ．

さらに，箱の型ごとに，各メソッドの使用可否を確認せよ．すなわち，箱ごとに全てのメソッドを試せ．

ポリモーフィズム

抽象クラスやインタフェースのインスタンス

抽象クラスやインタフェースはnewでインスタンス化できないと説明していたが，
箱の型としては使用することができる.

```
Wizardn w      = new Wizardn("はせがわん1", 100, 8);
Monster m      = new Wizardn("はせがわん2", 100, 8);
Magicable ma   = new Wizardn("はせがわん3", 100, 8);
Monster mo     = new Monster("はせがわんNG", 100, 8);
Magicable mag  = new Magicable("はせがわんNG", 100, 8);
```

abstract
interface
abstract
abstract
interface
interface

ポリモーフィズム

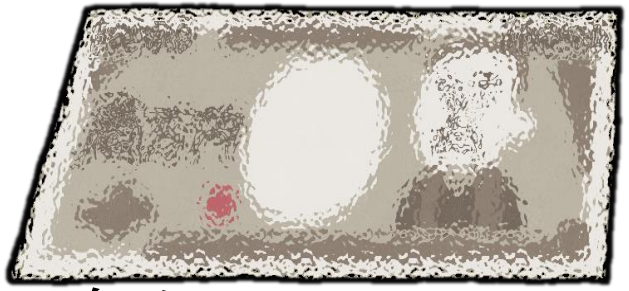
捉え方→使え方

抽象的 ← 具体的

Paper

鼻をかめる()

Illustration



鼻をかめる()
絵を楽しめる()

Money

鼻をかめる()
絵を楽しめる()
物を買える()

同じMoneyでも捉え方によって使え方が変わる

ポリモーフィズム

捉え方→使え方

.鼻をかめる()→OK

.物を買える()→OK

.物を買える()→NG

- 中身はMoneyだが，Paperとして捉えているので，Paperとしてしか振る舞うことができない.
- 箱に入れたことで，呼び出せるメソッドの種類は**箱の型で判断される**ようになる.

ポリモーフィズム

練習問題３：どうやって逃げるのか

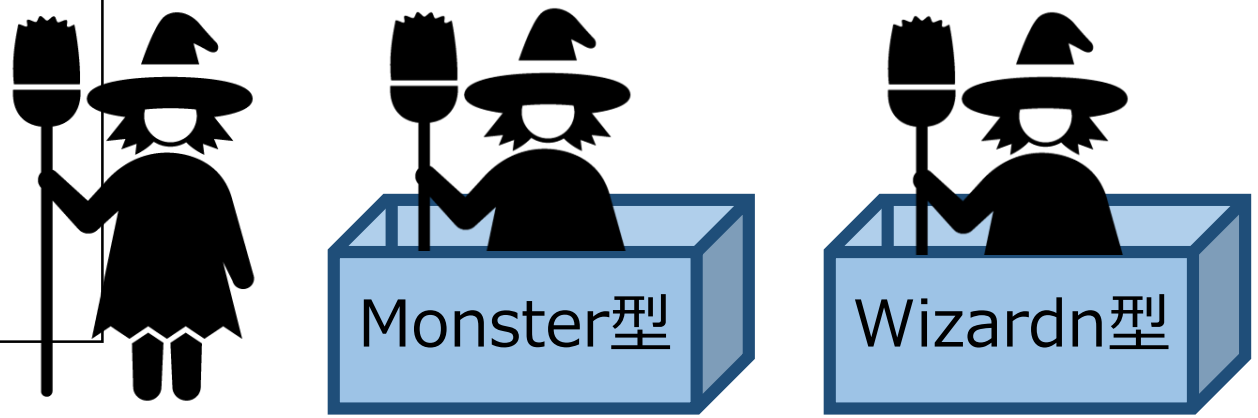
次のプログラムの出力を予測せよ.

```java
public class Monster {

  public void run(){
    System.out.println("逃げ出した");
  }
}
```

```java
public class Wizardn extends Monster{
  @Override
  public void run(){
    System.out.println("箒で飛んで逃げた");
  }
}
```

```java
public class Main {
  public static void main(String[] args){
    Wizardn w =  new Wizardn();
    Monster m =  new Wizardn();
    w.run();  // １．ここと
    m.run();  // ２．ここを予測せよ
  }
}
```



ポリモーフィズム

捉え方→使え方

箱の型　→どのメソッドが使えるのかを決める

中身の型→メソッドが呼ばれた際の動作を決める

ただのMonsterだと思いました？
箱はMonster型でも宇宙の帝王としての
能力は失っていないのですよ.

ポリモーフィズム

箱を変える，捉え方を変える

この箱に入れられてしまった私は，
宇宙の帝王としての能力を持ちつつも，
本領を発揮できないままなのでしょうか．．．

ご安心下さい！キャストできます！

```
Monster m = new Maou(“MAOU様", 53, 53);
Maou ma = m;                  // エラーになる

Maou ma = (Maou)m;            // Maou型でキャスト
ma.bigbang();                 // Maou型独自メソッドが使える
((Maou)m).bigbang();          // Maou型独自メソッドが使える
```

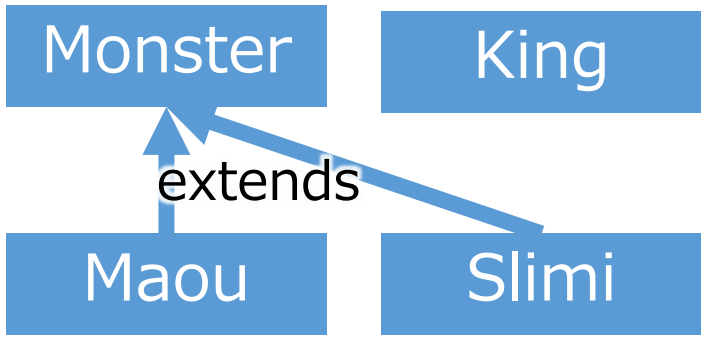
Maouと捉え直して

Maou独自メソッドを使用

ポリモーフィズム

キャストする際の注意

抽象的な型に入っている中身を，具体的な型として捉えるキャストは**ダウンキャスト**と呼ばれ，
失敗のリスクを伴うことに注意が必要である.




```
King k = new King();
Maou ma1 = (Maou) k;  // 流石にコンパイルエラーになる

Monster s = new Slimi("スライミ",1,1);
Maou ma2 = (Maou)s;   // Monster型はMaouの可能性がある
                      // のでコンパイルエラーにはならない
```

関係ない型へのキャストはコンパイルエラーになるが，
Monster型はMaou型になり得るのでコンパイルでき，
実行時にClassCastExceptionでエラー落ちする.

ポリモーフィズム

安全にキャストするための確認

安全にキャストできるか確認するため，中身の型をチェックするinstanceof演算子がある.

boolean型で返ってくる

```
[ 変数名 ] instanceof [ 型名 ]
```

```
Monster m = new Maou(“MAOU様",53,53);
System.out.println(m instanceof Maou);

Monster s = new Slimi("スライミ",1,1);
System.out.println(s instanceof Maou);

if( m instanceof Maou ){
  Maou ma = (Maou)m;
  ma.bigbang();
}
```

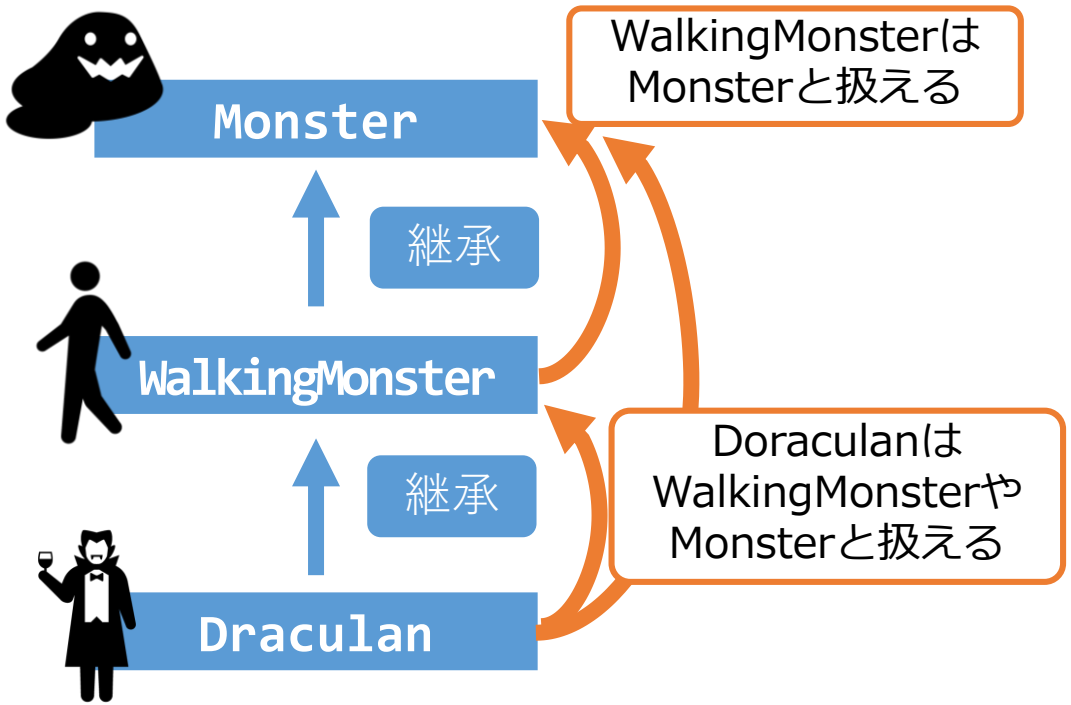
→trueを出力

→flaseを出力

→安全確認をした上でメソッドを実行

ポリモーフィズム
メリット

ポリモーフィズムの最大の特徴は，**継承関係にある複数のクラスをまとめて同じように扱うことができる**ということである．



とりあえず全員Monsterとして扱うことで，**配列化して処理をまとめたり，メソッドに汎用的な引数を実現できる**というメリットが生まれる．

ポリモーフィズム
メリット

ポリモーフィズムの1つ目のメリットは,
配列化してまとめて処理を実施できるということである.

```
// ポリモーフィズムのない世界

Maou maou = new Maou(“MAOU様",53,53);
Slimi slimi = new Slimi("スライミ", 1, 1);
Wizardn wiz = new Wizardn(“魔法", 10, 50);
// ・・・・（以下たくさんいるとする）

// 全員で攻撃したい時
maou.attack();
slimi.attack();
wiz.attack();
// ・・・・（以下略）
```

やりたいことは全員attack()なのに，一人ずつ実装しなければならない.

```
// ポリモーフィズムのある世界
Monster[] marr = new Monster[3];
marr[0] = new Maou("MAOU様",53,53);
marr[1] = new Slimi("スライミ", 1, 1);
marr[2] = new Wizardn("魔法", 10, 50);
// ・・・・（以下たくさんいるとする）

// 全員で攻撃したい時
for(Monster m : marr){
  // marrの要素が増えてもこのまま
  m.attack();
}
```

とりあえずMonster型で配列に代入し

ループでまとめて攻撃を実施する

ポリモーフィズム
メリット

ポリモーフィズムの2つ目のメリットは,
汎用的な引数のメソッドを実装できることである.

Main.java

```
Maou maou = new Maou("MAOU様",53,53);
Slimi slimi = new Slimi("スライミ", 1, 1);
Wizardn wiz = new Wizardn("魔法マン", 10, 50);

Yousya y = new Yousya("優者", 10);
y.attack(maou);   // maouに攻撃する
y.attack(slimi);  // slimiに攻撃する
y.attack(wiz);    // wizに攻撃する
```

味方プレイヤーのYousyaクラスの攻撃メソッドattack()はYousyaインスタンスに敵インスタンスを引数で渡し，直接ダメージを与えさせる仕様にしたい.

ポリモーフィズム

メリット

Yousya.java (1)

```
// ポリモーフィズムがない場合，各クラス用の処理を実装しなければならない
public void attack(Maou m){          // Maou用
  m.damage(this.ap);                 // 攻撃力ap分のダメージを与える
}
public void attack(Slimi s){         // Slimi用
  s.damage(this.ap);                 // 攻撃力ap分のダメージを与える
}
public void attack(Wizardn w){       // Wizardn用
  w.damage(this.ap);                 // 攻撃力ap分のダメージを与える
}
```

Yousya.java (2)

```
// ポリモーフィズムがある場合，Monster用に一つだけでよい．
// また今後新たに開発されるMonsterに対しても汎用的に動作する．
public void attack(Monster m){       // Monster全般用
  m.damage(this.ap);                 // 攻撃力ap分のダメージを与える
}
```

継承や実装でMonster部分が保証されているなら，何インスタンスでもよい

ポリモーフィズム

練習問題４：SpecialStudentの赤点

第二回の練習問題で開発したStudentとSpecialStudentクラスを，本日のプロジェクトフォルダにコピペせよ.

次に，Main.javaを作成し，mainメソッドにおいて，以下の通りのインスタンスを生成せよ.

インスタンス名	生徒種別	名前	日本語の点数	英語の点数	数学の点数
merio	Student	メリオ	30	60	
luigue	Student	ルイグー	50	60	
peech	SpecialStudent	ペーク	70	90	80
koppa	SpecialStudent	コッパ	30	70	70
kinopuo	SpecialStudent	キノプオ	60	70	60

ポリモーフィズム

練習問題４：SpecialStudentの赤点

赤点（平均点が50点未満）か判定するredScore()
メソッドを開発したい．以下の要件でメソッドを開発せよ．

引数　　：〇〇型：生徒種別を問わず生徒を1名受け取る
戻り値：boolean型：赤点ならばtrueを返す．
他　　：Main.javaの中に定義せよ．
　　main()から呼ぶので，public staticとせよ．

ポリモーフィズム

まとめ

ザックリ捉えると**Monster**だよね，**Slimi**って

```
Monster m = new Slimi();
```

箱の型　　**中身の型**

extendsかimplementsの関係にある場合に限る

箱の型→どのメソッドが使えるのかを決める
中身の型→メソッドが呼ばれた際の動作を決める

キャストで強引に箱の型を変更することができる

ポリモーフィズムができることで，インスタンス配列で
上手く楽をしたり，汎用的な引数のメソッドを実現できる

前回の演習課題２の解説
コールバック

次週予告

※次週以降も計算機室

前半

待ちに待ったGUIプログラミングの第一歩を学ぶ.

後半

講義内容に関するプログラミング演習課題に取り組む.

本日の提出課題

講義パート

課題 1

本日の授業を聞いて，
よくわかったと思う内容を
2点簡潔に述べよ.

課題 2

本日の授業を聞いて，
質問事項または**気になった点**
を1点以上簡潔に述べよ.

課題 3

感想（あれば）

課題 4

なし

演習

- 昼休み，いつものWebページに演習問題をPDFで演習問題をアップロードする．各自実施してプロII同様のWebページから提出すること.
- 質問は３人体制で受け付けるので遠慮なく申し出る．質問の際は，どこまでわかっていて何がわからないのかを申し出ること.
- （ないとは思うが）コピペは発覚次第両成敗する.
  - ✓ {コピペ，カンニング} ∈不正行為
- つまらないミスも今回は問答無用で×とするので，最終チェックを怠らないこと．（昨年は目視で甘めに採点していたが，自動採点を開発している意味がないので．．．）