

継承, カプセル化 1

2017/11/13(月) プログラミングII 第六回
福井大学 工学研究科 情報・メディア工学専攻
長谷川達人

※WindowsでPC起動しておいてください.



よくあった間違い

課題 1 : hp,nameだけでも初期化できる用のコンストラクタも定義する. を読み飛ばしていた人が30名ほど.

attack2()~attack4()を実装する際に旧attack()を消してしまっている人が数名ほど.

本講義の概要

前半：Java 担当：長谷川		後半：Scala 担当：石井先生	
第1回	Java基礎文法/開発環境の使い方	第9回	Scala言語の基本
第2回	Java基礎文法/C言語との差異	第10回	関数型プログラミングの基本
第3回	オブジェクト指向	第11回	関数とクロージャ
第4回	クラス/インスタンス/メソッド1	第12回	関数型プログラミングの簡単な例
第5回	クラス/インスタンス/メソッド2	第13回	静的型付けと動的型付け
第6回	継承/カプセル化1	第14回	ケースクラス/パターンマッチング
第7回	継承/カプセル化2	第15回	多相性やパターンマッチングの例
第8回	中間試験/Java言語のまとめ	第16回	期末試験

※来年以降は全てJavaになる予定

本日の目標

概要

継承を使う場面と使い方を学ぶ.
カプセル化の意図と使い方を学ぶ.

目標

継承とカプセル化を用いたプログラムが書ける.

オブジェクト指向の3大要素

第6回～第7回
継承

第6回～第7回
カプセル化

ポリモρφイズム
プロ4にて

クラスとインスタンス
第3回～第5回

Javaの基礎文法
第1回～第2回

これらは、オブジェクト指向を行う際に役に立つ3つの考え方
みたいなものである。



本日の目次

- 継承
 - 継承とは
 - オーバーライド
 - 親クラスのメソッドへのアクセス
 - コンストラクタ
- カプセル化

継承

Monsterクラスのインスタンスとしてボスモンスターを作ろうと思ったが、ボスはすごい魔法を使ってくるよう**少し改造したい**。

どうすればよいだろうか？

(Monsterインスタンスだとattack()メソッドしかない)

```
public class Monster{  
    int hp = 0;  
    String name = "";  
    int ap = 0;  
    public int attack(){  
        return this.ap;  
    }  
}
```

継承

案1. Monsterクラスにmagicメソッドを実装する.

> ボス以外のMonsterインスタンスもmagic()が使えてしまう.

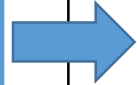
```
public class Monster{
    int hp = 0;
    String name = "";
    int ap = 0;
    public int attack(){
        return this.ap;
    }
    public int magic(){
        System.out.println(this.name+"は魔法を詠唱した. ");
        return this.ap + (int)(Math.random()*10);
    }
}
```


継承

案2. MonsterクラスをコピーしてBossMonsterクラスを作る.

> attack()や共通のフィールドに変更があったとき, 両方直さなければならない.

```
public class Monster{  
    int hp = 0;  
    String name = "";  
    int ap = 0;  
    public int attack(){  
        return this.ap;  
    }  
}
```



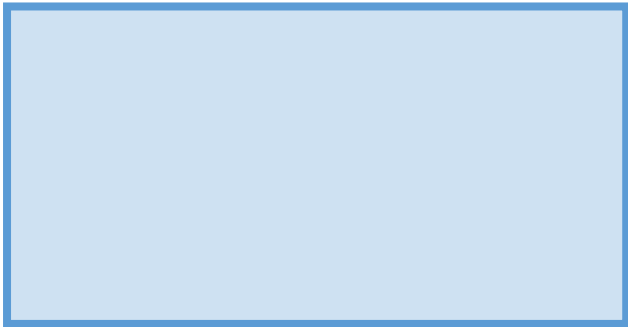
```
public class BossMonster{  
    int hp = 0;  
    String name = "";  
    int ap = 0;  
    public int attack(){  
        return this.ap;  
    }  
    public int magic(){  
        System.out.println(this.name+"は魔法を詠唱した. ");  
        return this.ap + (int)(Math.random()*10);  
    }  
}
```

継承

案3. 継承 (extends) を使ってBossMonsterクラスを作る.

```
public class Monster{
    int hp = 0;
    String name = "";
    int ap = 0;
    public int attack(){
        return this.ap;
    }
}
```



```
public class BossMonster extends Monster{
    
    public int magic(){
        System.out.println(this.name+"は魔法を詠唱した. ");
        return this.ap + (int)(Math.random()*10);
    }
}
```

継承

案3. 継承 (extends) を使ってBossMonsterクラスを作る.

- ＞ 継承を使うと，差分のみを被継承クラスに書くだけで良い.
- ＞＞ 可読性が向上する
- ＞＞ Monsterを修正するとBossMonsterにも反映される.

```
public class BossMonster extends Monster{  
  
    public int magic(){  
        System.out.println(this.name+"は魔法を詠唱した. ");  
        return this.ap + (int)(Math.random()*10);  
    }  
  
}
```

座布団一枚っ



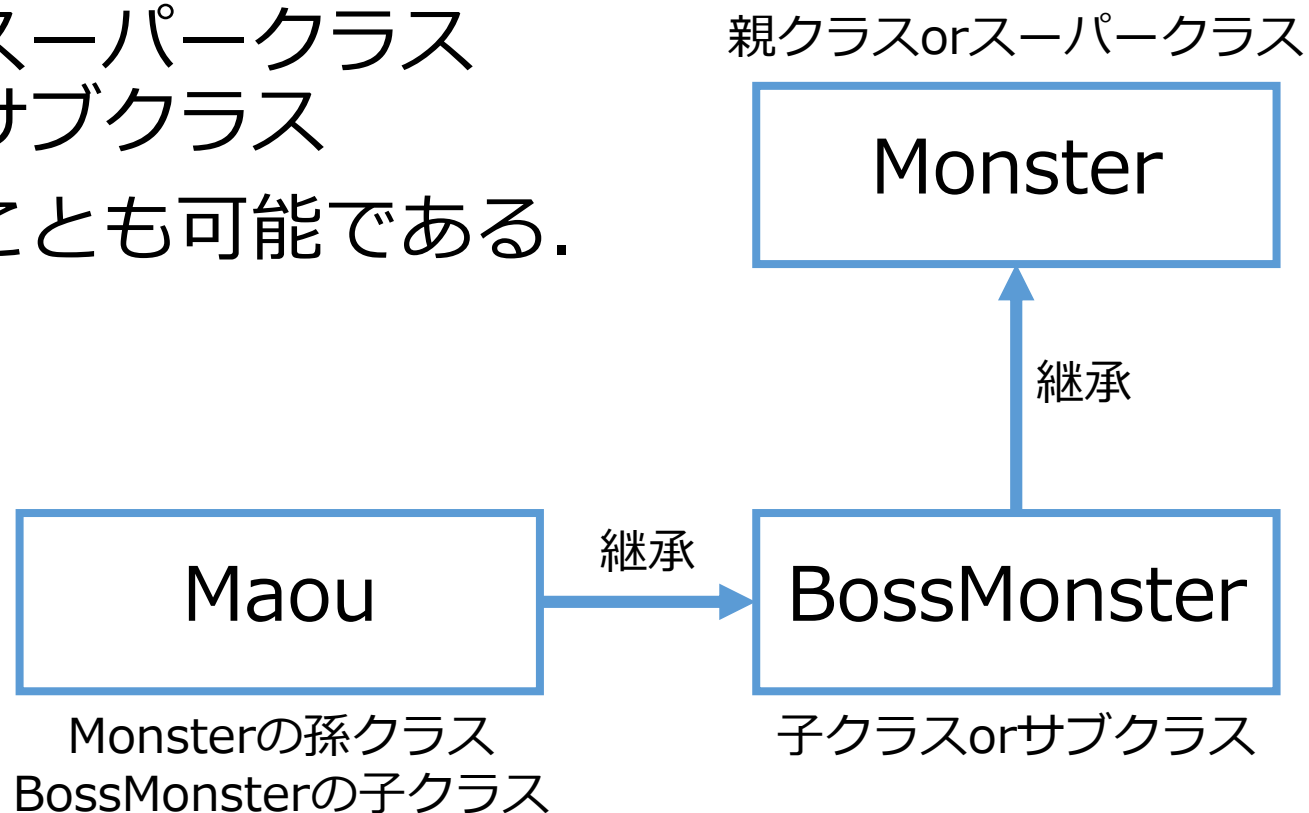
継承

継承関係は右図のように図示する.

継承される側 : **親クラス** or スーパークラス

継承する側 : **子クラス** or サブクラス

子クラスをさらに継承することも可能である.



継承

継承まとめ

似たようなクラスを作るときに，元のクラスを流用し拡張開発を行うことを継承という．新クラスでは元のクラスで記述された内容を再度記述する必要が無くなる．

継承関係では以下のような呼び名を用いる．

継承される側：親クラスorスーパークラス

継承する側：子クラスorサブクラス

継承の使い方は，子クラスを定義する際に以下のように記述する．

```
public class [新クラス名] extends [親クラス名] {}  
example: public class BossMonster extends Monster {}
```

子クラスをさらに継承することも可能である．

継承

練習問題 1 : 継承

事前準備

MonsterクラスとBossMonsterクラスを以下のように定義する。
eclipseを開き、本日のプロジェクトフォルダを作成後、以下のプログラムを記述する。

```
public class Monster{  
    int hp = 0;  
    String name = "";  
    int ap = 0;  
    public int attack(){  
        return this.ap;  
    }  
}
```

```
public class BossMonster extends Monster{  
    public int magic(){  
        System.out.println(this.name+"は魔法を詠唱した. ");  
        return this.ap + (int)(Math.random()*10);  
    }  
}
```

継承

練習問題 1 : 継承

BossMonsterは魔法を使う時に魔法ポイント（mp）を5消費する．この処理を実装せよ．mpはBossMonster独自のフィールドとする．

継承

オーバーライド

BossMonsterクラスを継承により実装したが、ボスは通常攻撃を行うごとに攻撃力（AP）が上昇するよう**少し改造したい**。

どうすればよいだろうか？

(attack()はMonsterクラスにあるが、BossMonsterのみ
attack()を改造したい。)

```
public class Monster{
    int hp = 0;
    String name = "";
    int ap = 0;
    public int attack(){
        return this.ap;
    }
}
```

```
public class BossMonster extends Monster{
    public int magic(){
        System.out.println(this.name+"は魔法を詠唱した。");
        return this.ap + (int)(Math.random()*10);
    }
}
```


継承

オーバーライド

案 1. BossMonsterクラスにattack2()を実装する.

- > 通常Monsterはattack()を呼び出し, BossMonsterはattack2()を呼び出さなければならない.
- >> ヒューマンエラーの原因になるのでは？

```
public class Monster{  
    int hp = 0;  
    String name = "";  
    int ap = 0;  
    public int attack(){  
        return this.ap;  
    }  
}
```

```
public class BossMonster extends Monster{  
    public int magic(){  
        System.out.println(this.name+"は魔法を詠唱した. ");  
        return this.ap + (int)(Math.random()*10);  
    }  
    public int attack2(){  
        this.ap += 5;  
        return this.ap;  
    }  
}
```

こんな感じ

継承

オーバーライド

案2. オーバーライドを使ってattack()メソッドを上書きする.

> 通常MonsterもBossMonsterもattack()で統一できる.

>> ヒューマンエラーのリスクを減らせる.

```
public class Monster{
    int hp = 0;
    String name = "";
    int ap = 0;
    public int attack(){
        return this.ap;
    }
}
```

```
public class BossMonster extends Monster{
    public int magic(){
        System.out.println(this.name+"は魔法を詠唱した. ");
        return this.ap + (int)(Math.random()*10);
    }
    public int attack(){
        this.ap += 5;
        return this.ap;
    }
}
```

継承

オーバーライド

オーバーライドを使う時には**@Override**注釈をつける。

これは今からオーバーライドを書きますよという宣言である。

```
public class Monster{  
    int hp = 0;  
    String name = "";  
    int ap = 0;  
    public int attack(){  
        return this.ap;  
    }  
}
```

```
public class BossMonster extends Monster{  
    @Override  
    public int attack(){  
        this.ap += 5;  
        return this.ap;  
    }  
}
```

ここに書く

```
public class BossMonster extends Monster{  
  
    public int attack(){  
        this.ap += 5;  
        return this.ap;  
    }  
}
```

どちらも正常に動く

継承

オーバーライド

オーバーライドを使う時には**@Override**注釈をつける。

注釈をつけることでヒューマンエラーのリスクを減らせる。

```
public class Monster{  
    int hp = 0;  
    String name = "";  
    int ap = 0;  
    public int attack(){  
        return this.ap;  
    }  
}
```

```
public class BossMonster extends Monster{  
    @Override  
    public int attackk(){  
        this.ap += 5;  
        return this.ap;  
    }  
}
```

たまたま書き
間違えたとき

親クラスに
attackk()が
ないとコンパ
イルエラー

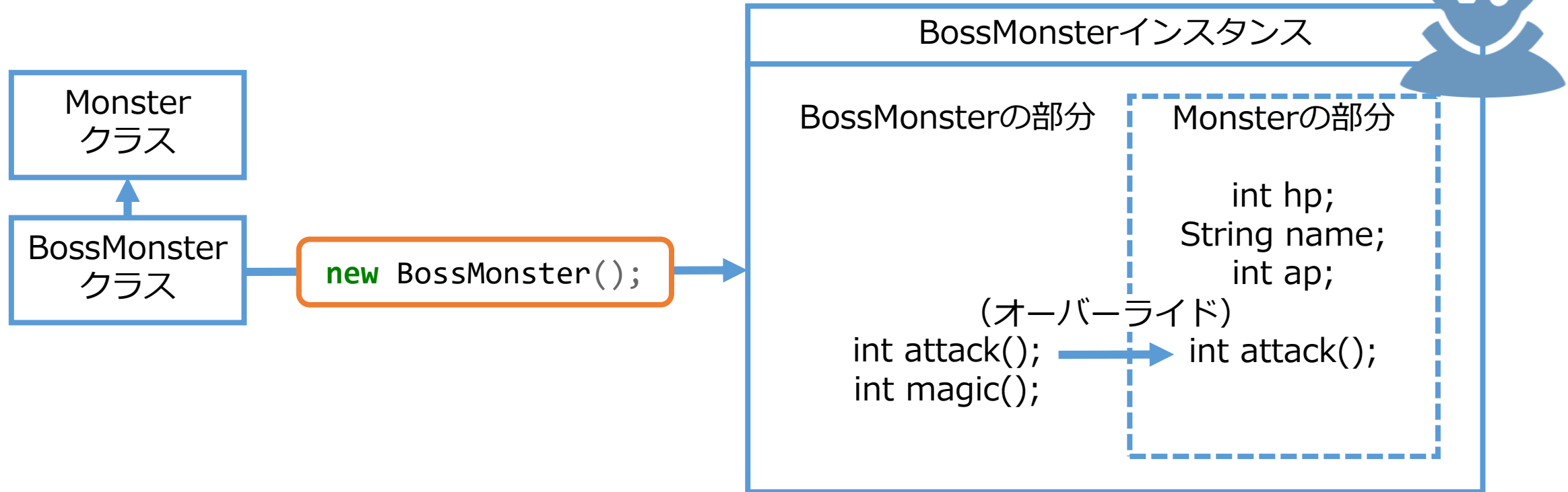
```
public class BossMonster extends Monster{  
  
    public int attackk(){  
        this.ap += 5;  
        return this.ap;  
    }  
}
```

attackk()とい
う新メソッド
と認識される

継承

継承とオーバーライドのイメージ

Monsterクラスを継承したBossMonsterクラスをインスタンス化すると、右側のインスタンスができるイメージとなる。

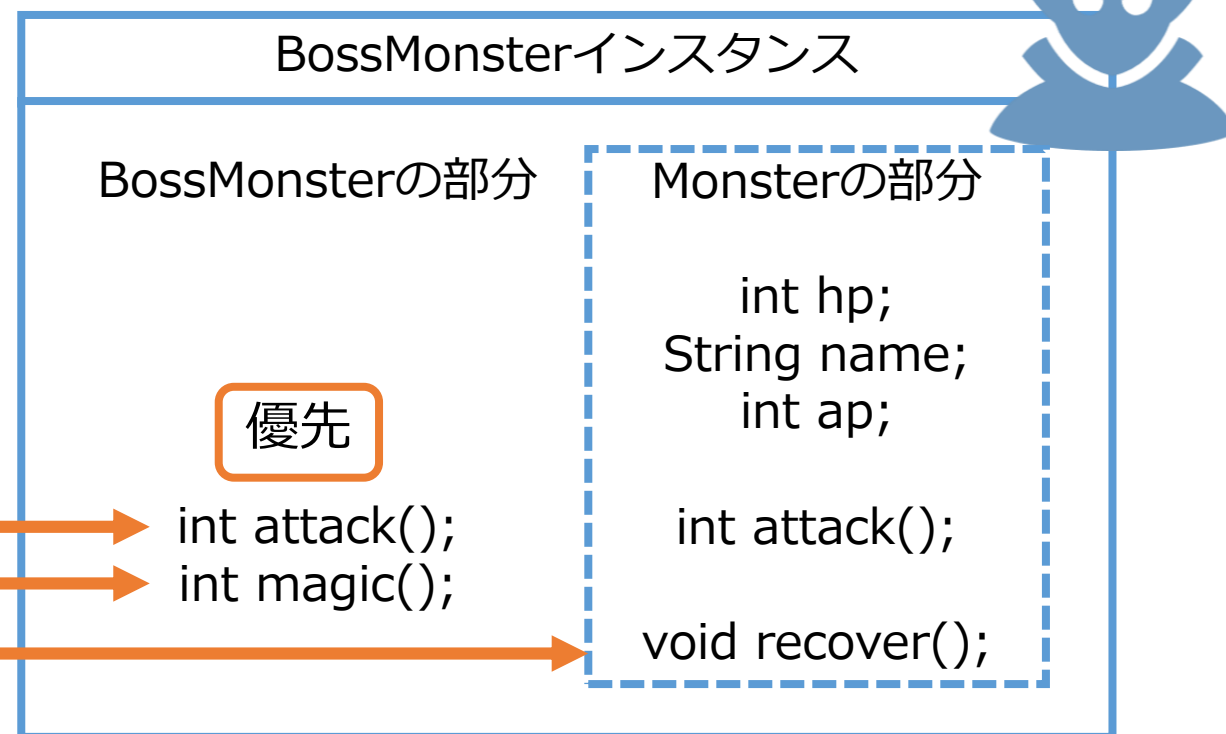


継承

継承とオーバーライドのイメージ

BossMonsterに実装されているメソッドを優先して使用する。
実装されていない場合はMonsterに実装されているメソッドを使用する。

```
public static void main(String[] args){  
    BossMonster boss = new BossMonster();  
    boss.attack();  
    boss.magic();  
    boss.recover();  
}
```

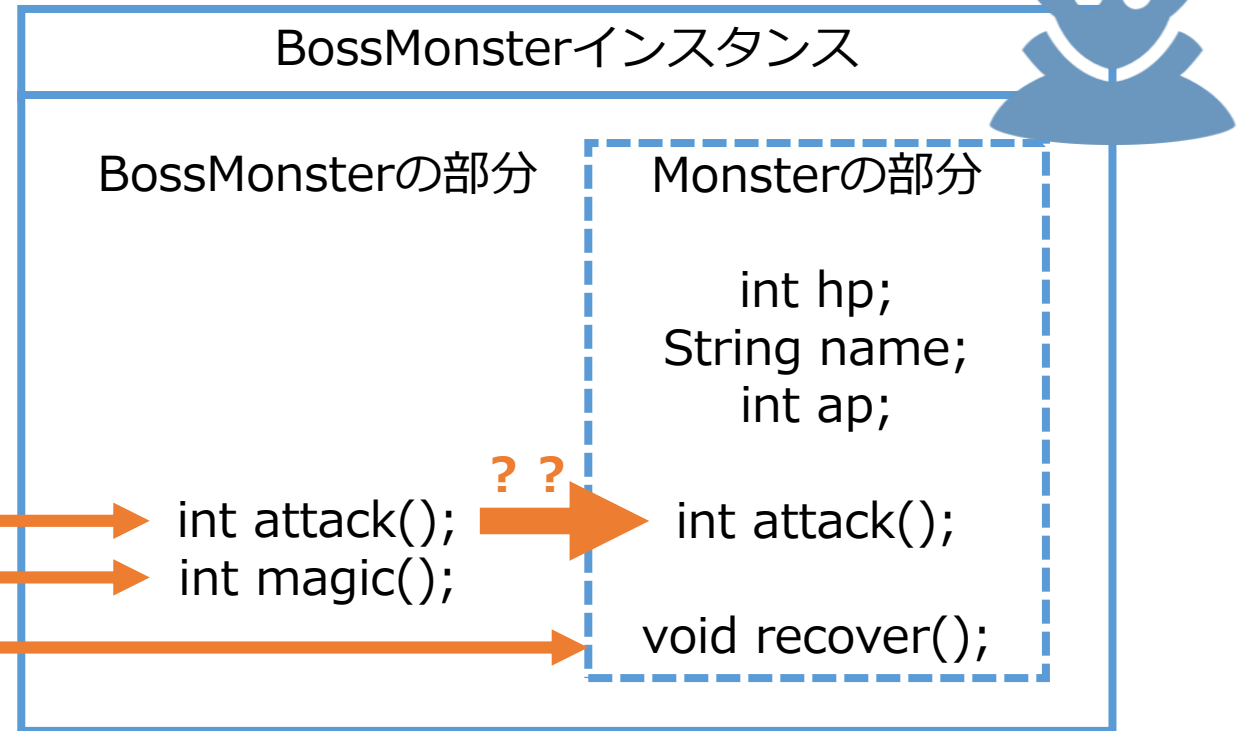


継承

親クラスのメソッドへのアクセス

下記の例の場合, Monster側のattack()にアクセスできないのか?
＞ BossMonster内のメソッドからならアクセスできる.

```
public static void main(String[] args){  
    BossMonster boss = new BossMonster();  
    boss.attack();  
    boss.magic();  
    boss.recover();  
}
```



継承

親クラスのメソッドへのアクセス

例えば、先ほどの例を考える。

もし、Monsterのattack()の仕様が変わった場合どうなる？

＞ BossMonsterのattack()も修正が必要でバグの要因になり得る。

```
public class Monster{  
    int hp = 0;  
    String name = "";  
    int ap = 0;  
  
    public int attack(){  
        return this.ap+5;  
    }  
}
```

```
public class BossMonster extends Monster{  
  
    @Override  
    public int attack(){  
        this.ap += 5;  
        return this.ap+5;  
    }  
}
```

両方修正しなければならない
(将来的に忘れられるかも?)

継承

親クラスのメソッドへのアクセス

super.method()を使うことで、この問題を回避できる.

> Monsterのattack()を修正すれば, BossMonsterから呼び出す **super.method()**も修正される.

```
public class Monster{  
    int hp = 0;  
    String name = "";  
    int ap = 0;  
  
    public int attack(){  
        return this.ap+5;  
    }  
}
```

```
public class BossMonster extends Monster{  
  
    @Override  
    public int attack(){  
        this.ap += 5;  
        return super.attack();  
    }  
}
```

super.method()で
親クラスのメソッドを
呼び出せる

継承

親クラスのメソッドへのアクセス

それに伴い, `this.ap` も, 本来であれば `super.ap` と記述されるべきである.

> `this`. や `super`. を必ず付ける付けないという話は人それぞれ好みによる. しかし, これらの意図を理解しておく意味を込めて, 本講義では必ずつけることにする.

```
public class Monster{
    int hp = 0;
    String name = "";
    int ap = 0;

    public int attack(){
        return this.ap+5;
    }
}
```

```
public class BossMonster extends Monster{
```

```
    @Override
    public int attack(){
        super.ap += 5;
        return super.attack();
    }
}
```

`super.field`で
親クラスのフィールド
にアクセスできる

継承

オーバーライドのまとめ

オーバーライド：親クラスの全く同じメソッドを再定義すること

オーバーロード：同じメソッド名を多重定義すること（復習）

メソッドの直前には「@Override」注釈を記述する。

親クラスのメソッドやフィールドにアクセスする際には**super.**を付ける。

```
public class Monster{
    int hp = 0;
    String name = "";
    int ap = 0;
    public int attack(){
        return this.ap;
    }
}
```

オーバー
ライド

```
public class BossMonster extends Monster{
    @Override
    public int attack(){
        super.ap += 5;
        return super.attack();
    }
}
```

継承

練習問題 2 : オーバーライド

以下の構造のプログラムを考える時, (1)~(4.3)はいずれとなるか.

- ・ (A)を実行する
- ・ (B)を実行する
- ・ エラーになる

Main

```
void main(){
    ClassA a = new ClassA();
    ClassB b = new ClassB();

    a.method(); //(1)
    b.method(); //(2)

    a.testA(); //(3.X)
    b.testB(); //(4.X)
}
```

ClassA

```
void method(){} //(A)

void testA(){
    method(); //(3.1)
    this.method(); //(3.2)
    super.method(); //(3.3)
}
```

ClassB extends ClassA

```
@Override
void method(){} //(B)

void testB(){
    method(); //(4.1)
    this.method(); //(4.2)
    super.method(); //(4.3)
}
```

継承

練習問題 2（発展）：オーバーライド

では, (5.1)~(5.3)はいずれとなるか.

- ・ (A)を実行する
- ・ (B)を実行する
- ・ エラーになる

Main

```
void main(){
  ClassA a = new ClassA();
  ClassB b = new ClassB();

  b.testA(); //(5.X)
}
```

ClassA

```
void method(){} //(A)

void testA(){
  method(); //(5.1)
  this.method(); //(5.2)
  super.method(); //(5.3)
}
```

ClassB extends ClassA

```
@Override
void method(){} //(B)

void testB(){
  method();
  this.method();
  super.method();
}
```

継承

練習問題 3 : オーバーライド (余裕のある人)

先ほど紹介したMonsterクラスのattack()メソッドを実装せよ。
次に, BossMonsterクラスでattack()メソッドをオーバーライドし, 攻撃直前にapを5向上させるように拡張せよ。
プログラムは次の通りである。

```
public class Monster{
    int hp = 0;
    String name = "";
    int ap = 0;

    public int attack(){
        return this.ap;
    }
}
```

```
public class BossMonster extends Monster{

    //この辺にmpやmagic()の記述があるはず

    @Override
    public int attack(){
        super.ap += 5;
        return super.attack();
    }
}
```

継承

コンストラクタ

次のプログラムの実行結果を予測してみよう.

```
public class Main {  
    public static void main(String args[]){  
        ClassB b = new ClassB();  
    }  
}
```

```
public class ClassB extends ClassA{  
    public ClassB(){  
        System.out.println("ClassB");  
    }  
}
```

```
public class ClassA {  
    public ClassA(){  
        System.out.println("ClassA");  
    }  
}
```

【出力】
ClassA
ClassB

ClassAコンストラクタも
実行されるのはなぜ？

継承

コンストラクタ

Javaでは全てのコンストラクタは、その先頭で必ず親クラスのコンストラクタを呼び出さなければならない。

```
public class Main {  
    public static void main(String args[]){  
        ClassB b = new ClassB();  
    }  
}
```

```
public class ClassB extends ClassA{  
    public ClassB(){  
        System.out.println("ClassB");  
    }  
}
```

```
public class ClassA {  
    public ClassA(){  
        System.out.println("ClassA");  
    }  
}
```

ここで暗黙的に親クラスの
引数無しコンストラクタ
が呼び出されている

継承

コンストラクタ

親クラスのコンストラクタを明示的に呼び出すには,
super([引数]);と記述する.

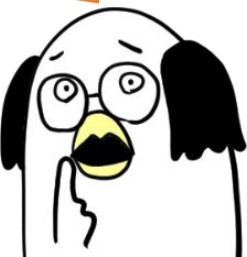
```
public class Main {  
    public static void main(String args[]){  
        ClassB b = new ClassB();  
    }  
}
```

```
public class ClassB extends ClassA{  
    public ClassB(){  
        super();  
        System.out.println("ClassB");  
    }  
}
```

```
public class ClassA {  
    public ClassA(){  
        System.out.println("ClassA");  
    }  
}
```

何も書かなかった場合,
ここに暗黙の**super**();を
自動で挿入していた.

自クラスのコンストラクタ
を呼び出すには**this**();を
使っていましたね (復習) .



継承

コンストラクタ

ただし、暗黙の`super()`は**引数無しコンストラクタしか自動挿入されない**。したがって、以下のケースでは明示する必要がある。

```
public class Main {  
    public static void main(String args[]){  
        ClassB b = new ClassB();  
    }  
}
```

```
public class ClassB extends ClassA{  
    public ClassB(){  
        System.out.println("ClassB");  
    }  
}
```

```
public class ClassA {  
    public ClassA(int a){  
        System.out.println("ClassA");  
    }  
}
```

暗黙の`super()`;を自動挿入しようとしても、親クラスには引数無しコンストラクタがなかった！！

今回の場合、`super(100);`等を手で明記する必要がある。

継承

コンストラクタのまとめ

- コンストラクタ： **new** でインスタンスが生成された時に真っ先に呼ばれる処理（復習）
- 何かを継承したクラスの場合は、親クラスのコンストラクタ → 子クラスのコンストラクタの順に実行される。
- 自クラスのコンストラクタの明示的呼び出しは、 **this**([引数]);
- 親クラスのコンストラクタの明示的呼び出しは、 **super**([引数]);
- 引数無しのコンストラクタ **this()**, **super()** は暗黙的に実装されている。
- 親クラスのコンストラクタに引数が必要な場合、明示的に **super**([引数]) を記述する必要がある。

継承

練習問題 4 : コンストラクタ

現在, BossMonsterクラスにはmpフィールドとmagic()メソッドが実装されていると思う.

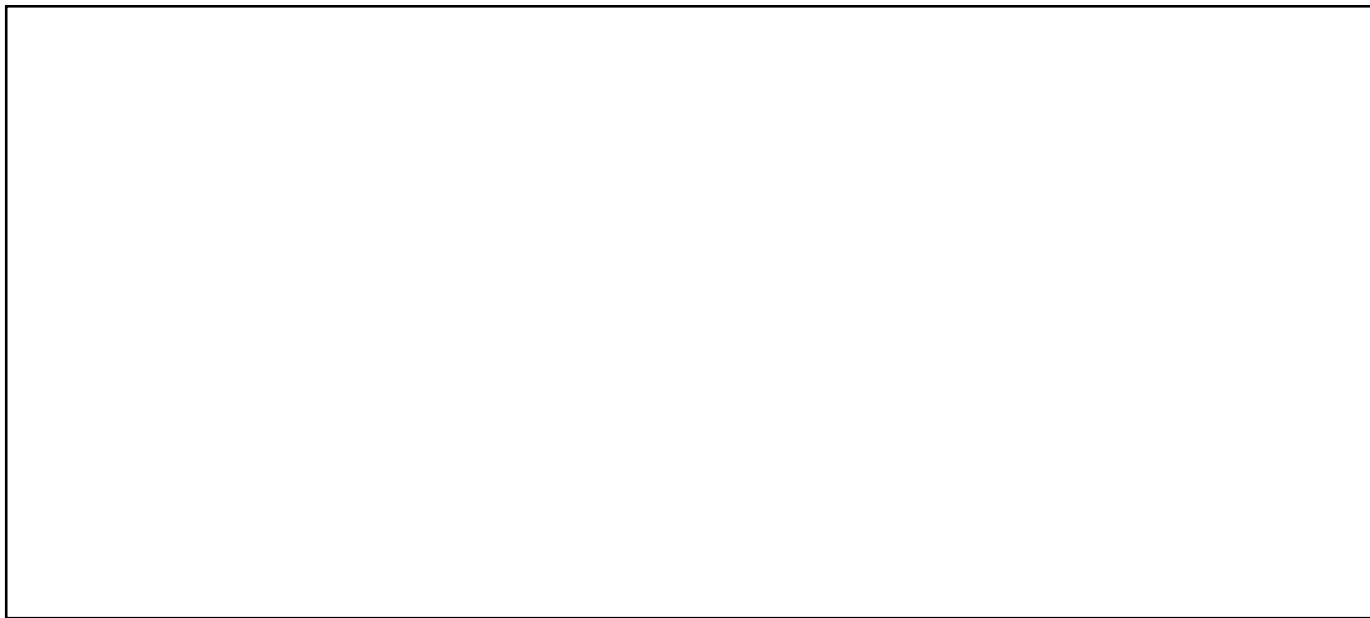
まず, Monsterクラスに対して, hp, name, apを初期化する3引数コンストラクタを実装せよ. (前回課題と同じ)

次に, BossMonsterクラスに対して, hp, name, ap, mpを初期化する4引数コンストラクタを実装せよ.

最後に, BossMonsterクラスとMonsterクラスを引数無しでもインスタンス化できるように, 引数無しコンストラクタを両者に実装せよ.

継承

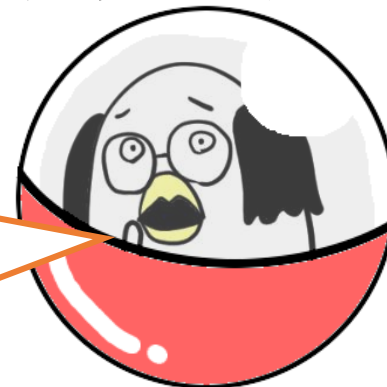
練習問題 4 : コンストラクタ



カプセル化

- **カプセル化** : フィールドやメソッド, クラスに対してアクセス制御を行い, 内部を隠蔽すること.
- アクセス修飾子 : これまでおまじないとしてきたpublic等のこと.
 - **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.

アクセス制御されているので,
誰でも容易に編集させません.



カプセル化

- 次のプログラムを考える。

nameとattack()はpublicなのでMainクラスからアクセス可能

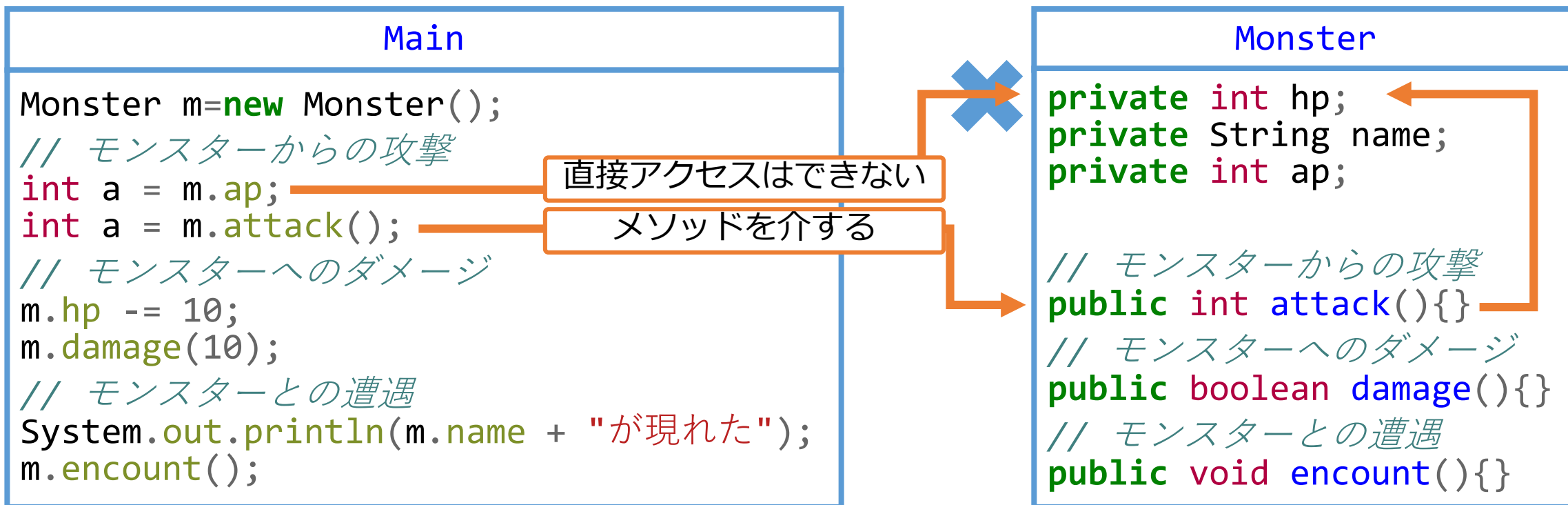
```
public class Main {  
    public static void main(String[] args){  
        BossMonster boss = new BossMonster();  
        System.out.println(boss.name);  
        System.out.println("" + boss.hp);  
        System.out.println("" + boss.attack());  
        System.out.println("" + boss.getRand(10));  
    }  
}
```

hpとgetRand()はprivateなのでMainクラスからのアクセスは不可
(コンパイルエラーになる)

```
public class Monster{  
    private int hp = 0;  
    public String name = "";  
    int ap = 0;  
  
    public int attack(){  
        return this.ap + this.getRand(10);  
    }  
  
    private int getRand(int n){  
        return (int)(Math.random()*n);  
    }  
}
```

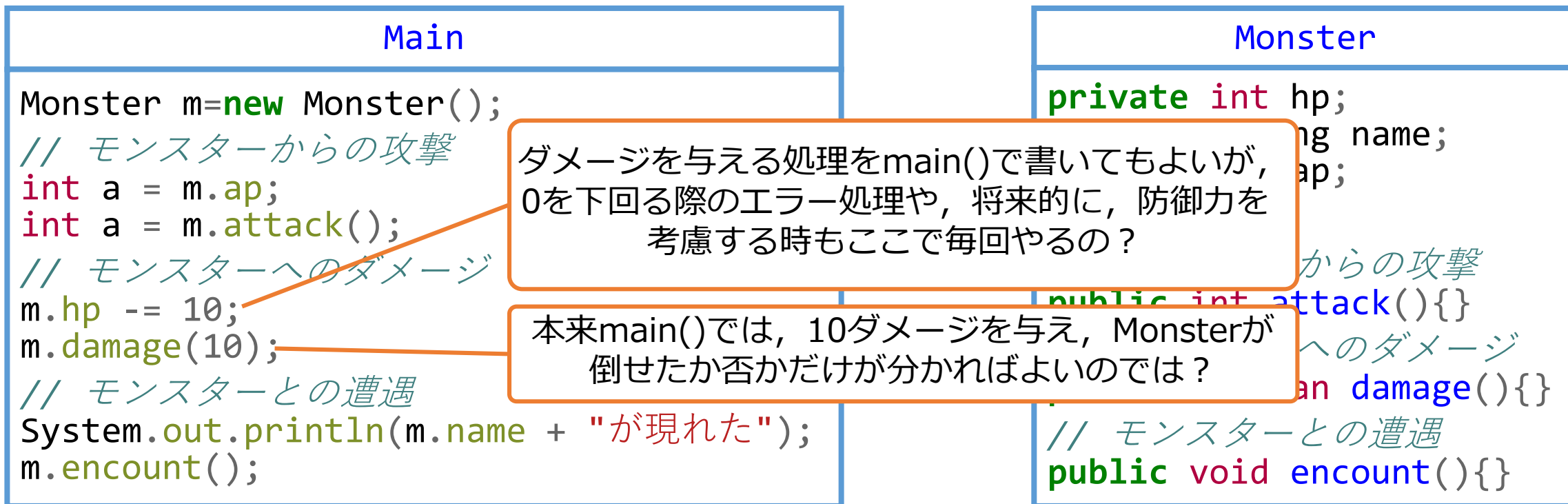
カプセル化

- フィールドは全てprivateにし、メソッドを経由しなければフィールドにアクセスできなくすることが一般的である。



カプセル化

- このように、フィールドを隠蔽し、メソッドを介して必要な情報のみを取り出せるようにすることを、カプセル化という。



カプセル化

カプセル化を意識しない開発を行った場合

Main

```
Monster m=new Monster();  
// モンスターへのダメージ  
m.hp -= 10;  
if(m.hp <= 0){  
    m.hp = 0;  
    System.out.println  
    (m.name + "は敗れた");  
}
```

hpは0以下で負け判定

仕様書は
熟読Aさん

撃退判定表示は「～は敗れた」

Monster

```
public int hp;  
public String name;
```

Poison // 毒ダメージ管理

```
Monster m = [引数]で取得;  
// モンスターへのダメージ  
m.hp -= 5;  
if(m.hp < 0){  
    m.hp = 0;  
    System.out.println  
    (m.name + "は消えた");  
}
```

0未満と勘違い

動けばOK
Bさん

撃退判定表示は「～は消えた」

運用ルールが使用者に委ねられる = ヒューマンエラーのリスク増加

カプセル化

カプセル化を意識しない開発を行った場合

Main

```
Monster m=new Monster();  
// モンスターへのダメージ  
m.damage(10);
```

ダメージを10与えると記述し、あとは任せる。

Monster

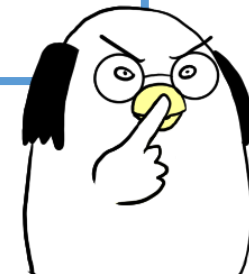
```
private int hp;  
private String name;  
  
void damage(int d){  
    this.hp -= d;  
    if(this.hp<=0){  
        this.hp = 0;  
        print(this.name  
            + "は敗れた");  
    }  
}
```

フィールドは外部から好き勝手させない。

統一した運用ルールを実現

Poison // 毒ダメージ管理

```
Monster m = [引数]で取得;  
// モンスターへのダメージ  
m.damage(5);
```

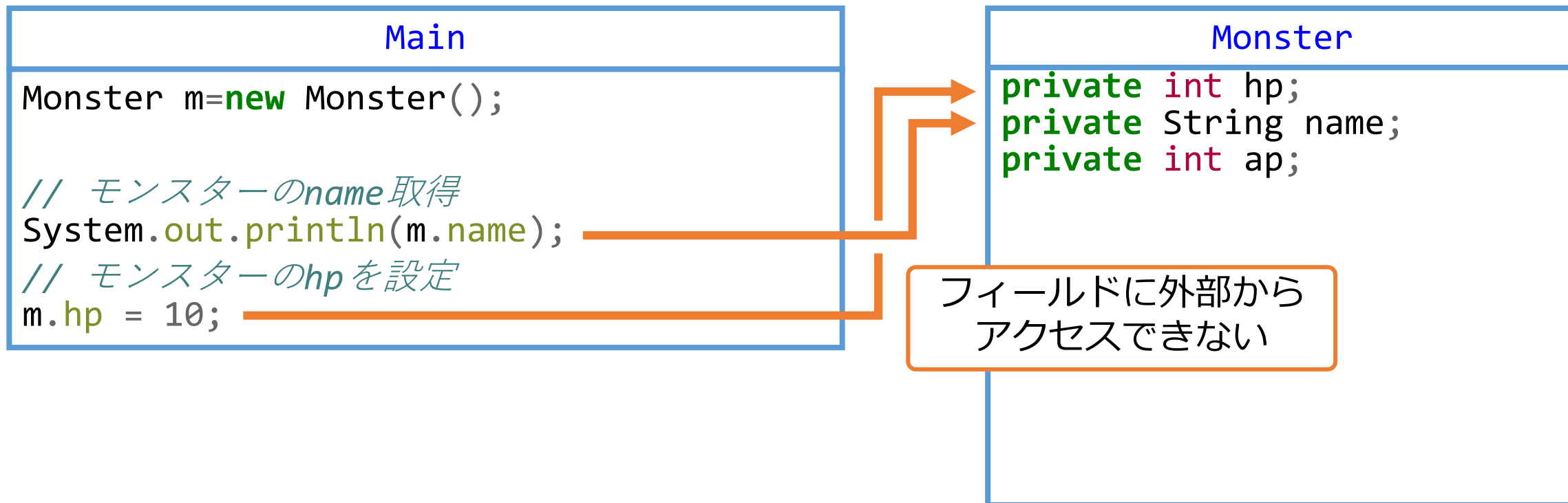


自フィールドへの処理は、自クラス内で実装する（カプセル化）。
＞オブジェクト指向への一歩

カプセル化

getterとsetter

フィールドを隠蔽したら、どうやってhpにアクセスするの？
> getterやsetterを使うという手がある。



カプセル化

getterとsetter

フィールドを単純にget, setするメソッドをgetter, setterと呼ぶ。
main側からアクセスが必要となる際にgetter, setterを実装する。

Main

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

```
// モンスターのname取得
```

```
System.out.println(m.getName());
```

```
// モンスターのhpを設定
```

```
m.setHp(10);
```

メソッドを介してアクセス

ただし, この実装は本来好ましくない。
> setHp()でhpを好き勝手に設定できるため,
実質**public**と同じである。本来は正確性や
エラーの判定処理を明記する必要がある。

Monster

```
private int hp;
```

```
private String name;
```

```
private int ap;
```

```
// nameの取得
```

```
public String getName(){
```

```
    return this.name;
```

```
}
```

```
public void setHp(int hp){
```

```
    this.hp = hp;
```

```
}
```

カプセル化

まとめ

- 今回の例のように、フィールドの値を書き換える際には様々なルールで制限が行われることが少なくない.
- 直接、フィールドを書き換えられるようにしておくと、様々なルールを無視してしまう人が出てくるかもしれない.
 - **ヒューマンエラーのリスク**
- インスタンスを利用する側は、必要な情報のみ**メソッドを介して**やり取りできるよう、アクセス権を最小限にとどめる.
 - 残りはクラス側が様々なルールへの対応を行う方が望ましい.
- これがカプセル化であり、オブジェクト指向では各クラスはほぼ全てカプセル化されている.

継承

練習問題 5 : カプセル化

BossMonsterクラスのmpフィールドをカプセル化しよう.

mpフィールドはmainから操作する必要性はなく, magic()を使用した際に5消費されていればよい.

念のためmagic()使用時に残りmpを表示する一文も追記しよう.

本日のまとめ

- 継承 : 似たクラスを実現する際に元クラスを引きついで開発すること (**extends**)
- オーバーライド : 継承時に親クラスのメソッドを子クラス側で再定義すること
- コンストラクタ : 親クラスには (**super**([引数]))
自クラスには (**this**([引数]))
- カプセル化 : フィールドを隠蔽し必要箇所のみ使えるようアクセス制限を行うこと

本日のまとめ

プログラムの書き方

- 継承

```
public class [新クラス名] extends [親クラス名] {}  
example: public class BossMonster extends Monster {}
```

- オーバーライド

```
public class Monster{  
    int hp = 0;  
    String name = "";  
    int ap = 0;  
    public int attack(){  
        return this.ap;  
    }  
}
```

オーバー
ライド

```
public class BossMonster extends Monster{  
    @Override  
    public int attack(){  
        super.ap += 5;  
        return super.attack();  
    }  
}
```

親クラスにはsuper.でアクセス

次週予告

※次週以降も計算機室

本日の復習と，本日の内容に関する演習問題を実施する．



本日の提出課題

提出はWebClassで

課題 1

本日の授業を聞いて、
初めて知ったと思う内容を2点簡潔に述べよ。
「簡潔に述べよ」≡「1, 2文程度で述べよ」と考えるとよい

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2点簡潔に述べよ。

課題 3, 4

中間テストのサンプル問題2問