第４週　カプセル化・継承・オーバーライド

カプセル化

　オブジェクト指向の三大要素の １ つ目カプセル化について説明していきます。まず、カプセル化をしていないときにどのような問題があるか見てみましょう。

```
//Personクラス
public class Person {
	String name;	//名前
	int hp;		//体力
	//コンストラクタ
	Person(String name, int hp){
		this.name = name;
		this.hp = hp;
	}
	//ステータスを表示する
	void showStatus(){
		System.out.println(name + "'s HP is " + hp);
	}
}
```

```
public class Sample1 {
	public static void main(String[] args) {
		// TODO 自動生成されたメソッド・スタブ
		Person kaimai = new Person("Kaimai", 100);
		kaimai.showStatus();
		//開米のhpを変更
		kaimai.hp = -1;
		kaimai.showStatus();
	}
}
```

このように hp の値を「-1」のようにあり得ない値を設定できてしまいます。そこで、Person クラスに下記のような setHp メソッドを実装することで解決することができます。

```
//Personクラス
public class Person {
	String name;		//名前
	int hp;			//体力
	//コンストラクタ
	Person(String name, int hp){
		this.name = name;
		this.hp = hp;
	}
	//ステータスを表示する
	void showStatus(){
		System.out.println(name + "'s HP is " + hp);
	}
	int getHp(){
		return hp;
	}
	void setHp(int hp){
		if(hp > 0){
			this.hp = hp;
		}
	}
}
```

setHp メソッドを実装することで hp に負の値を設定しようとしてもメッセージを出して hp が負の値になることを防いでくれるようになりました。このように、フィールドの値を設定するメソッドを「setter」、フィールドの値を得るメソッドを「getter」といいます。しかし、まだ問題は残っています。次のように、Person クラスのフィールド hp にアクセスしようとした場合です。

```
kaimai.hp = -1;
```

これは、**アクセス修飾子**を指定することで解決することができます。

アクセス修飾子

private	同じクラスからアクセス可能
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・継承先のクラスからアクセス可能
public	全クラスからアクセス可能

アクセス修飾子はクラス定義、フィールド定義、メソッド定義の直前に付けることで適用する事ができます。継承とパッケージについては今後扱うのでいまは private と public の違いだけを知っていれば大丈夫です。今回の場合は hp の値が勝手に書き換えられてしまう問題は private を付けることで解決できます。

```
private kaimai.hp = -1;
```

これにより、hp に他クラスから直接アクセスするとコンパイルエラーが起こり、想定外の値が代入されることを防ぎます。このように private をつけることで他クラスから隠蔽することを**カプセル化**といいます。他のクラスから hp を編集や参照をしたいときは getHp()メソッド(getter)や setHp()メソッド(setter)を使います。

継承

次に、オブジェクト指向の三大要素の 2 つ目の継承について説明します。まず、以下のソースを見てみましょう。

```
//会員クラス
public class Member {
	private String name;
	private int age;
	private int height;
	private int weight;

	public Member(String name, int age, int height, int weight){
		this.name = name;
		this.age = age;
		this.height = height;
		this.weight = weight;
	}

	public void putStatus(){
		System.out.println(name + " : " + age + "才 - " + height + "cm - " + weight + "kg");
	}
	public String getName(){return this.name;}
	public int getAge(){return this.age;}
	public int getHeight(){return this.height;}
	public void setHeight(int height){if(height > 0){this.height = height;}}
	public int getWeight(){return this.weight;}
	public void setWeight(int weight){if(weight > 0){this.weight = weight;}}
}
```

```
//特別会員クラス
public class SpecialMember {
	private String name;
	private int age;
	private int height;
	private int weight;
	private String privilage;

	public SpecialMember(String name, int age, int height, int weight, String privilage){
		this.name = name;
		this.age = age;
		this.height = height;
		this.weight = weight;
		this.privilage = privilage;
	}

	public void putStatus(){
		System.out.println(name + " : " + age + "才 - " + height + "cm - " + weight + "kg"
+ privilage);
	}
	public String getName(){return this.name;}
	public int getAge(){return this.age;}
	public int getHeight(){return this.height;}
	public void setHeight(int height){if(height > 0){this.height = height;}}
	public int getWeight(){return this.weight;}
	public void setWeight(int weight){if(weight > 0){this.weight = weight;}}

	//↓会員クラスとの違い↓
	public String getPrivilage(){
		return this.privilage;
	}
	public void setPrivilage(String privilage){
		this.privilage = privilage;
	}
}
```

上記のように会員クラスと特別会員クラスがあった場合２つのクラスの違いは特典があるかないかだけです。このようにほとんど同じフィールドとメソッドを持ったクラスを新しく定義することは効率的ではありません。そこで、**継承**を使用します。継承とは既存のクラスからフィールドやメソッドを引き継いで新たにクラスを定義することです。今回のSpecialMember クラスを Member クラスを継承して書いてみましょう。

```
//特別会員クラス
public class SpecialMember extends Member{
	private String privilage;

	public SpecialMember(String name, int age, int height, int weight, String privilage){
		super(name, age, height, weight);
		this.privilage = privilage;
	}

	public void putStatus(){
		System.out.println(name + " : " + age + "才 - " + height + "cm - " + weight + "kg - "
+ privilage);
	}

	public String getPrivilage(){
		return this.privilage;
	}
	public void setPrivilage(String privilage){
		this.privilage = privilage;
	}
}
```

既存のクラスを継承して新しく定義するためにはキーワード「**extends**」をつけます。

```
class SpecialMember  extends  Member{  …  }
```

継承元のクラスをスーパークラス(親クラス)、継承先のクラスをサブクラス(子クラス)と言います。今回の例では Member がスーパークラス、SpecialMember がサブクラスとなります。

継承を使って書き換える前の Member クラスのフィールドのアクセス修飾子は private でしたが書き換えた後では protected に変更されていることに注意してください。private のままだとサブクラスからスーパークラスのフィールドにアクセスする事ができなくなるのでサブクラスで使用したいフィールドは protected にすることを忘れないようにしてください。

継承について注意しなければならない事がもう一つあります。それは多重継承の禁止です。多重継承とは複数のクラスを継承して新しくクラスを定義することです。逆に、一つの親クラスを継承して複数の子クラスを定義することは可能です。

オーバーライド

最後に、オーバーライドについて説明します。オーバーライドとは親クラスで定義されているメソッドを子クラスで上書きして新しく定義することです。先ほど作った Member クラスと SpecialMember クラスで見てみましょう。

```
//会員クラス
public class Member {

                                        ⋮

        public void putStatus(){
                System.out.println(name + " : " + age + "才 - " + height + "cm - " + weight + "kg");
        }

                                        ⋮


```

```
//特別会員クラス
public class SpecialMember {

                                        ⋮

        public void putStatus(){
                System.out.println(name + " : " + age + "才 - " + height + "cm - " + weight + "kg"
+ privilage);
        }

                                        ⋮

}
```

Member クラスは putStatus メソッドを持っていて Member クラスを継承している SpecialMember クラスも putStatus メソッドが定義されています。それぞれの処理を確認すると SpecialMember クラスのメソッドは Member クラスの処理に加えて特典も表示させるようになっています。これが**オーバーライド**です。

オーバーライドは来週説明するポリモーフィズムにおいて重要になります。

演習問題

1. Character クラスをカプセル化しなさい

```
//Characterクラス
public class Character {
	//フィールド
	String name;
	int hp;
	int ap;

	//コンストラクタ
	Character(String name, int hp, int ap){
		this.name = name;
		this.hp = hp;
		this.ap = ap;
	}
	public void attack(){
		System.out.println(name + "が攻撃力" + ap + "で攻撃!!");
	}
	//ゲッタ ・ セッタ
	public String getName(){
		return name;
	}
	public int getHp(){
		return hp;
	}
	public void setHp(int hp){
		if(hp > 0){
			this.hp = hp;
		}
	}
	public int getAp(){
		return ap;
	}
}
```

2. Character クラスを継承して次の機能を満たす Fighter クラスと Wizard クラスを実装し,適当な main 関数を用意して動作することを確認しなさい.

- Fighter クラス

  ✓ atack メソッドを呼ぶと
  ～の会心の一撃(攻撃力××)
  と表示される。(~は name××は ap)

- Wizard クラス

  ✓ フィールドに mp を持つ。
  ✓ atack メソッドを呼ぶと
  ～の魔法攻撃(攻撃力××, 魔力〇〇)
  と表示される。(~は name, ××は ap,〇〇は mp)