Java 講習 第 11 回

1.継承

既存のクラスを流用してさらに機能を追加したいときに継承という仕組みを利用する
継承機能を使うことでコードがすっきりするよ！
継承機能は Java、オブジェクト指向において大事な役割を持つので各自でもよく勉強しよう。

1.1 スーパークラスとサブクラス

継承の元になるクラス　・・・　スーパークラス(親)
継承して作成されるクラス・・・サブクラス(子)
と呼ばれる
　基本的な機能を含むスーパークラスを作っておく。ユーザーはそれを継承し、個別の具体的な機能を追加して活用する
　という具合に利用される

1.1 継承の使い方

継承を行うときには下記のように記述する。

```
class サブクラス名 extends スーパークラス名{
        (追加する機能)
}
```

スーパークラスは既に
作成済みであることに注意

クラス定義の際、extends の後にスーパークラス名を記述する。
継承は java では一度に一つしか出来ません。継承したいクラスが複数あっても一つしか継承出来ないので注意しましょう。
例　class Write extends Board {...}
この記述は　*Board を拡張(extends)して Write というクラスを作る* といった意味です

1.2 継承してみる

継承をすることでスーパークラスのメンバをサブクラス内で使用することができます。
＊メンバ…クラスの中身メソッドとか変数
ただし継承すると、スーパークラス(親)のコンストラクタに引数が必要な場合動作しないので注意

なにはともあれプログラムを書くのが一番なので継承のメカニズムが分かる簡単な例題を出します。

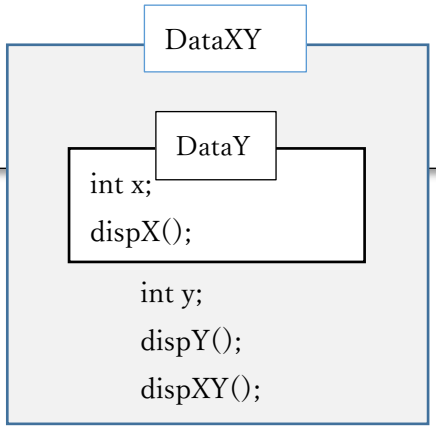
例題:　1. 変数 x の値を管理・x の値を表示する機能のあるクラスの作成（スーパークラス）
　　2. 1 のサブクラスとして変数 y の値を管理・y の値を表示,スーパークラスの x と共に y を表示する機能のあるクラスの作成
　　3. 2 のサブクラスをインスタンス生成し、変数 x,y の値を変更して値を表示する

```
class DataX{                          //スーパークラス（継承元）
        int x;
        void dispX(){                                 //DataX クラスの変数 x を表示するメソッド
                System.out.println("x=" + x);
        }
}

class DataXY extends DataX{           //サブクラス(DataX クラスを継承してる)
        int y;
        void dispY(){                                 //DataXY クラスの変数 y を表示するメソッド
                System.out.println("y=" + y);
        }
        void dispXY(){                                //DataX クラスの変数 x を表示するメソッド
                System.out.println("x=" + y + "    y=" + y);   //スーパークラスの変数を表示
        }
}
//以下のクラスを実行する
public class Rei1{
        public static void main (String[] args){
                DataXY dt = new DataXY();             //作成したサブクラスのインスタンス生成
                dt.x = 100;                           //値の変更
                dt.y = 200;
                dt.dispX();                           //メソッドの呼び出し
                dt.dispY();
                dt.dispXY();
        }
```

実行結果

```
x=100
y=200
x=100 y=200
```



このプログラムからサブクラス DataXY では
DataX の全メンバを利用できているのが分かる

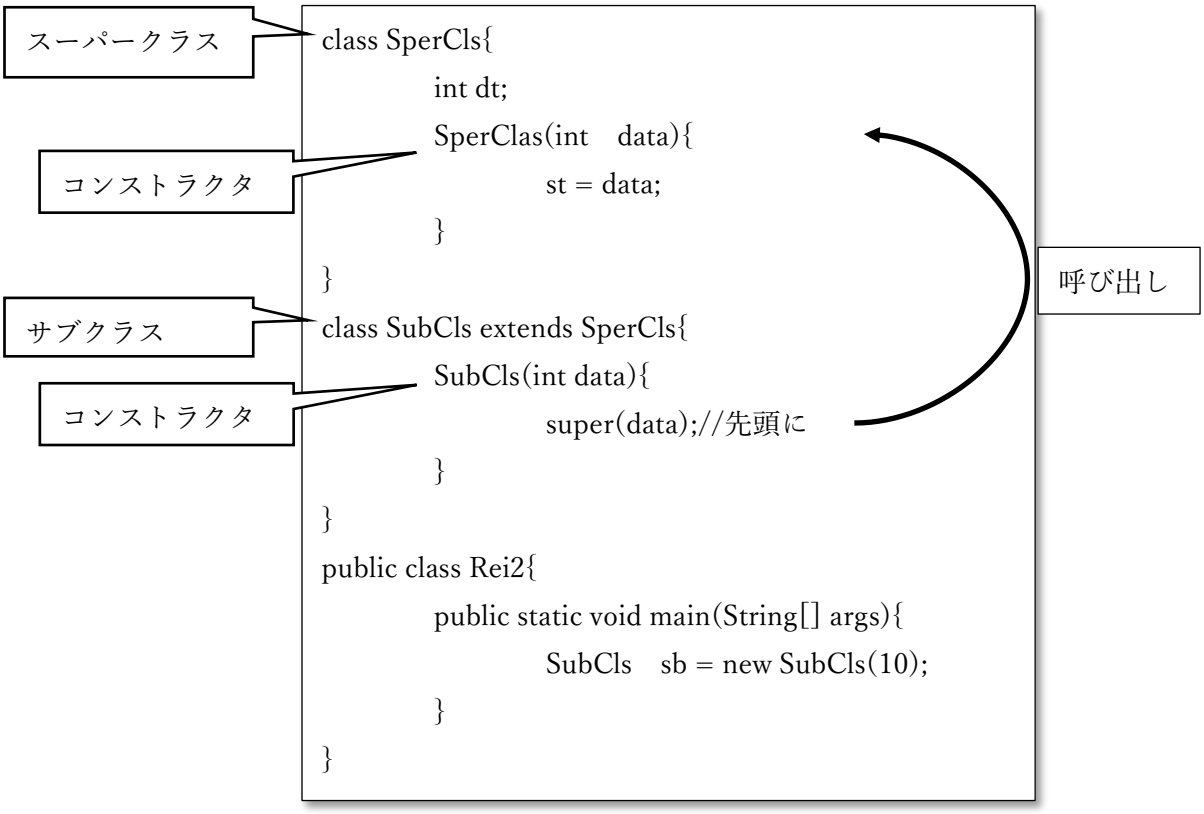
Java 講習 第 11 回

1.3 コンストラクタについて

コンストラクタは継承すると、スーパークラス(親)のコンストラクタに引数が必要な場合動作しません。
しかしながら親のコンストラクタ(引数蟻のもの)を呼びたい場合があります。
その際は以下のように専用のコードを記述する必要があります。

```
super(引数);
```

また、この記述は　**サブクラスのコンストラクタ内の先頭行に記述**　しなければなりません



この例ではインスタンス生成時に引数 10 を渡しているので、
最終的に変数 dt には 10 が入る

Java 講習 第 11 回

1.4 オーバーライドとオーバーロード

メソッドが同じ名前でも引数が違う場合、メソッドを作ることが出来る(多重定義)
このことをオーバーロードという　　また、継承した際も同様にオーバーロードすることが出来る

例:

```
class MyCls{
        int x,y;
        void set(int a){ //1 つめ
                x=a;
        }
        void set(int a,int b){        //2 つめ
                x = a;
                y=b;
        }
        public static void main(String[] args){
                set(1);   //1 つ目が呼ばれる
                set(10,20);//2 つ目が呼ばれる
        }
}
```

```
class MyCls{
        int x,y;
        void set(int a){    //1 つめ
                x=a;
        }
}
class Sub extends MyCls{
        void set(int a,int b){ //2 つめ
                x = a;
                y=b;
        }
}
class Main{
        public static void main(String[] args){
                Sub sub = new Sub();
                sub.set(1);          //1 つめ
                sub.set(10,20); //2 つめ
        }
}
```

また、継承した際にメソッド名が同じで引数も同じ場合は継承元のメソッドが無視される。
これをオーバーライドという

```
class MyCls{
        int x,y;
        void set(int a){
                x=a;
        }
}
class Sub extends MyCls{
        void set(int a){
                y=b;
        }
}
class Main{
        public static void main(String[] args){
                Sub sub = new Sub();
                sub.set(1);
        }
}
```

呼ばれない

Java 講習 第 11 回

2. 修飾子 例 public, private,protected,etc…

クラス、メソッド、コンストラクタ、変数などには修飾子をつけることが出来る。
修飾子をつけることでアクセスの制限をすることなどができる。

＊public,private などの修飾子は特にアクセス修飾子と呼ばれる。
それ以外の修飾子には static,final なんかがある。

2.1 アクセス修飾子

public　…どっからでもアクセスできる。
protected…同じパッケージの中、継承しているサブクラス間限定。
private　…同じクラスの中限定。
何もなし…同じパッケージの中だけ。

パッケージについてはここでは説明しないが、パッケージ≒フォルダとして考えてもらってよい

場所	private	指定なし	protected	public
同じクラス	○	○	○	○
同じパッケージ内のクラス	×	○	○	○
同じパッケージ内のサブクラス	×	×	○	○
他パッケージ内のクラス	×	×	×	○

使用例

```
public class MyCls1{        //とあるクラス
        int a;              //外部からアクセスできる
        public int b;       //外部からアクセスできる(同じパッケージ内のみ)
        private int c;      //外部からアクセスできない
        void disp1(){
        ・・・
        }
}
class MyCls2{               //別のクラス
        void disp2(){
        ・・・
        }
}
```

この例では disp1()メソッドでは変数 a,b,c が利用できる
しかし、別クラスである disp2()メソッドの中からは変数 a,b とメソッド disp1()しか利用できない
このようにアクセス修飾子を使うことでアクセスを制限できる。

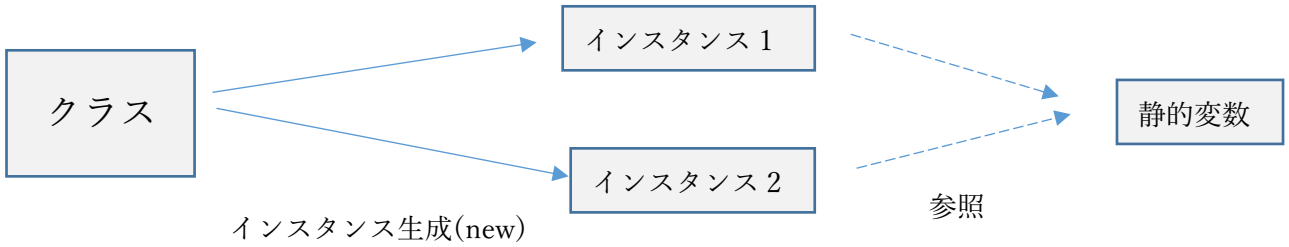
※1 ファイルに public クラスは 1 つしか作れないので注意

Java 講習 第 11 回

2.2 他の修飾子

final…最初に入れた値を変えることが出来ない。固定される。入れたが最後、そいつはもう変わらない。
（変数名はすべて大文字のことが多い）

static…メモリ上に残る頑固なやつ。静的変数/静的メソッド。（クラス自体の修飾子としては使えない）
通常の変数は、インスタンス(new したやつ)ごとに異なる値を保持し、インスタンス変数と呼ばれる。
一方、静的変数はインスタンスによらず共通のメモリ領域を占有する。
つまりインスタンスを作らなくても最初からあるということである
クラスＡをインスタンス化したインスタンス１とインスタンス２があるときに、インスタンス１が静的変数を１０にセットしたら、インスタンス２から参照しても１０になっている。静的変数の利用にはインスタンス化の必要がない。



使い方は

クラス名.静的変数　　クラス名.静的メソッド()

```
class Cls1{          //とあるクラス
        final int A=1000;          //今後は変数 A に代入できない
        static int a;              //インスタンス生成せずに使える変数
        static void disp1(){       //インスタンス生成せずに使えるメソッド
        ・・・
        }
}
public class Main{          //はじめに実行するクラス
        public static void main(String[] args){
                Cls1.a = 10;       //インスタンス生成せずに使える
                Cls1.disp1();      /インスタンス生成せずに使える
                Cls1 cl = new Cls1();
                cl.A = 10;         //final されているのでこれはコンパイルエラー
}
```