第六回　独習Java　ゼミ

第6章　継承

６．１　サブクラス

６．２　継承と変数

６．３　メソッドのオーバーライド

2006/05/24　　神津　健太

６．１　サブクラス

継承　：　オブジェクト指向プログラミング（OOP）の基本の１つ。
あるクラスを特化することによって、ほかのクラスを
定義すること。

クラスYがクラスXを継承する場合、
Yを「Xのサブクラス」、Xを「Yのスーパークラス」と呼ぶ。

「YはXを拡張する」と表現されることもある。

継承を使うと、クラスでそのスーパークラスの機能を再利用できる。

クラスの継承

```
class clsName2 extends clsName1 {
        //クラスの本体
}
```

クラス宣言内のextendsキーワードが2つのクラス間に
継承関係を形成する。
*clsName2*は*clsName1*のサブクラス。
「*clsName1*は*clsName2*のスーパークラス」とも言える。
extends *clsName1* をはずすとObjectクラスがスーパークラスで
あると解釈する。(Objectクラス　→　6.10)

クラスが継承できるのは1つのスーパークラスだけだが、サブクラスを
複数持つことはできる。サブクラスにさらに複数のサブクラスがあっても
かまわない。

オブジェクトを参照する変数の宣言

```
clsName   varName;
```

*varName*は変数名、*clsName*はそのクラス名である。
*VarName*は*clsName*クラスの任意のオブジェクトを参照することが可能になる。また、*clsName*のサブクラスの任意のオブジェクトを参照することもできる。

スーパークラスは、そのクラスのオブジェクトまたはそれから派生した任意のオブジェクトを参照することができる。

サブクラスの例

```
class X {
}

class X1 extends X {
}

class X11 extends X1 {
}

class InheritanceHierarchy{
   public satatic void main(String args[]){
      X x;
      System.out.println("Instantiating X");
      x = new X();
      System.out.println("Instantiating X1");
      x = new X1();
      System.out.println("Instantiating X11");
      x = new X11();
   }
}
```

クラスXは暗黙的にObjectクラスを拡張する。

クラスX1はクラスXを、
クラスX11はクラスX1を拡張する。

main()メソッドでクラスXの変数xを宣言し、

その後で各クラスのオブジェクトをインスタンス化し、

xにオブジェクト参照を代入している。

６．２　継承と変数

サブクラス内の変数が、スーパークラスの変数と同じ名前の場合

→ スーパークラスの変数が隠される。
名前が同じでもデータ型が同じである必要はない。

隠されたスーパークラスの変数の参照

super.varName　　varName:スーパークラスの変数名

superキーワードの利点
他人が作成したクラスを拡張して、新しいクラスを作成するとき、
サブクラスで使用する変数名と同じ変数名がスーパークラスで
使われていても、どちらの変数も参照することができる。

継承と変数の例（その１：クラス継承階層の具体例）

```
class X{
   float f;
}

class Y extends X{
   String s;
}

class Xy{
   public satatic void main(String args[]){
      Y y = new Y();
      y.f = 4.567f;
      y.s = "Teach Yourself Java";
      System.out.println("y.f = " + y.f);
      System.out.println("y.s = " + y.s);
   }
}
```

XクラスはObjectクラスを拡張し、
float型インスタンス変数 f を持つ。

YクラスはXクラスを拡張し、
Sring型インスタンス変数 s を持つ。

Yクラスのオブジェクトには、
スーパークラス（Xクラス）で定義された
インスタンス変数 f がある。

main()では、new演算子を使って
Yクラスをインスタンス化し、
インスタンス変数を初期化して、
その値を表示する

継承と変数の例（その２：変数が隠される具体例）

```
class E{
   int x;
}

class F extends E{
   String x;
}

class Ef{
   public satatic void main(String args[]){
      F f = new F();
      f.x = "This is a String";
      System.out.println("f.x = " + f.x);
      E e = new E();
      e.x = 45;
      System.out.println("e.x = " + e.x);
   }
}
```

EクラスはObjectクラスを拡張し、
int型インスタンス変数 x を持つ。

FクラスはEクラスを拡張し、
Sring型インスタンス変数 x を持つ。

main()では、
まずFクラスのオブジェクトを作成し、
このオブジェクトの参照を
F型のローカル変数 f に代入。
よって、f.x はString型になる。

次にEクラスのオブジェクトを作成、
E型のローカル変数 e に代入。
よって、e.x はint 型になる。

（例ではインスタンス変数だが、
静的変数を隠すこともできる。）

継承と変数の例（その３：superキーワードの具体例）

```
class M100{
   int i = 100;
}

class M200 extends M100{
   int i = 200;
   void display(){
      System.out.println("i = " + i);
      System.out.println("super.i = " + super.i);
   }
}

class SuperKeyword{
   public satatic void main(String args[]){
      M200 m200 = new M200;
      m200.display
   }
}
```

M100クラスはint型の変数 i を
宣言して初期化。

M200クラスはM100クラスを拡張し、
int型の変数 i を宣言して初期化。

display()メソッドでは、まずM200で
宣言された変数 i を表示する。
次にsuperキーワードを使って、
M100で宣言された変数 i を
表示する。

注）super.super.varName といった
表現を使っても、継承階層の
2レベル上の隠された変数には
アクセスできない

６．３　メソッドのオーバーライド

クラスで宣言したメソッドと同じシグネチャを持つメソッドが、
そのスーパークラスで宣言されている場合には、
サブクラスで再定義される。

→　メソッドのオーバーライド

サブクラスのメソッドがスーパークラスのメソッドを
オーバーライドすると、スーパークラスのメソッドは、
サブクラスオブジェクトから隠される。

スーパークラスの参照を使って、サブクラスのオブジェクトを
参照することもできる。

Javaでは、実行中のオブジェクトのクラスに基づいて
適切なオーバーライドメソッドを選択する。
そのため、どのオーバーライドメソッドが使用されるかは、
コンパイル時ではなく実行時に決まる。

オーバーライドの例（その1）

```
class A1{
   void hello(){
      System.out.println("Hello from A1");
}

class B1{
   void hello(){
      System.out.println("Hello from B1");
}

class C1{
   void hello(){
      System.out.println("Hello from C1");
}

class MethodOverriding1{
   public satatic void main(String args[]){
      C1 obj = new C1();
      obj.hello();
   }
}
```

継承階層を形成する3つのクラスに同じシグネチャのhello()メソッドを宣言。
サブクラスのhello()メソッドはスーパークラスのhello()メソッドをオーバーライドする。

表示結果は

Hello from C1

となる。

これは、変数 obj から参照されるオブジェクトがC1クラスのオブジェクトだから。

オーバーライドの例（その2）

```
class A2{
   void hello(){
      System.out.println("Hello from A2");
}

class B2{
   void hello(){
      System.out.println("Hello from B2");
}

class C2{
   void hello(){
      System.out.println("Hello from C2");
}

class MethodOverriding1{
   public satatic void main(String args[]){
      A2 obj = new C2();
      obj.hello();
   }
}
```

大きな変更点はmain()メソッドの
最初の行である。
変数 obj をA2型にしている。
ただし、オブジェクトそのものはC2型。

表示結果は

Hello from C2

となる。

どのメソッドを実行するかの決定は、
オブジェクトそれ自体の型であり、
変数 obj の型ではない。

クラスA2の変数を使って、C2のオブ
ジェクトを参照できる。

練習問題(1)

以下のプログラムはコンパイルエラーになる。
どの部分がエラーとなるのか、またその理由も答えなさい。

```
class Parent{
}

class Child extends Parent{
}

class Test{
	public static void main(String args[]){
		Parent p;
		p = new Parent();
		p = new Child();
		Child c;
		c = new Child();
		c = new Parent();
	}
}
```

練習問題（2）

2つの整数値に対して、指定した四則演算を行うプログラムを作成せよ。
ただし、以下の手順をふまえること。

A，B，C，Dの4つのクラスを作る。
BはAを拡張し、CはBを拡張、DはCを拡張するようにする。

それぞれのクラスに同じ名前のメソッドを1つずつ作る。
メソッド内容は別々のものにする。
内容については、それぞれ四則演算をするようにする。
例） A：加算　B：減算　C：乗算　D：除算

コマンドライン引数を3つとるようにする。
1番目の引数はクラス名（計算方法）、
2番目と3番目は計算する2つの数値とする。

実行例（Aを加算、Bを減算とした場合）

```
>java Shisoku A  2  3
2 + 3 = 5
>java Shisoku B  2  3
2 - 3 = -1
```