プログラミングIII

黒瀬　浩

kurose@neptune.kanazawa-it.ac.jp

居室　67・121

学習方法について

オブジェクト指向の考え方を学ぶ科目なので
　　ネットで探して写す
　　答えがどこにあるがまず確認する
やりかたでは対応できません

各用語がどのようなものは自分の言葉で説明出来るようになること
各用語はJavaではどのように記述するかすぐ答えられること
Pythonに比べ余計な記述をしなければならない理由を説明できること

ソースコードを理解するのにノートやマーカーを使わずにやれる人はほとんどいません（ながめているだけの人は無理です）

以下を復習すること
　　クラス，メソッド，フィールド，インスタンス，コンストラクタ，
　　インスタンス生成方法，自分自身のインスタンスを示す，オーバーロード
　　クラスメソッド，インスタンスメソッド
　　クラス変数，インスタンス変数
以降，上記を使ったことを学習します。

Javaプログラム作成の注意　別クラスファイルの影響

Eclipseで警告やプログラムが正しく動かない原因

Javaは実行時にそのディレクトリ（正確にはclasspahtという環境変数)のファイルをチェックする

6章の例では、
List6-1StudentCard.java　　クラス定義のみ
List6-2ConstructorExample.javaで、
StudentCardクラスを定義しているが
同じディレクトリにStudentCard.javaがあるので多重定義
(正確にはコンパイルしてStudentCard.classができている場合)

クラス名を変えれば重複しない（またはファイル名変更）

注： 明示的に外部のクラスを取り込むには import 文を書く

レポート2 配点4点　期限：　6回開始時

タイトル　プログラミングIIIレポート2
クラス名列、氏名、提出日明記
A4で1枚(2ページの場合は両面印刷)
紙は縦長でも横長でも見やすい大きさの字になれば良い
プログラムソース、実行結果の画像を貼り付けること

コンストラクタのオーバーロードを使い引数の数、型により動作の異なるメソッドを作れ。クラス変数で呼ばれた回数のカウンタを作ること(初期値は３)
整数値と文字列を保持するフィールド変数を用意すること。コンストラクタでフィールドに値をセットする。値を取得するgetInt(), getStr()メソッドも実装しインスタンス生成後、値を取得して表示せよ
。
以下はインスタンス生成の例。

a = new OlEx1("abc", 3)	第1引数の文字を第2引数回連結する
b = new OlEx1(4, 5)	引数2つが整数どうしなら加算する
c = new OlEx1(5)	カウンタ値と第1引数を加算する

この後、値を a.getStr(), b.getInt(), c.getInt() などで値を取得し 表示する
引数組み合わせの異なったインスタンスを最低3つ生成すること。

継承(inheritance) 上位クラスを引き継ぐ

```
class A {
     int counter;
     method1(){    }
     method2(){    }
}
```

クラスAに似たクラスBを作る

```
class B {
     int counter;
     int counter2;
     method1(){    }
     method2(){    }
     method3(){    }
}
```

クラスAのmethod1を修正すると
クラスB側も修正しなければならない
矛盾が起きやすい
管理が面倒
定義は1箇所で行いたい

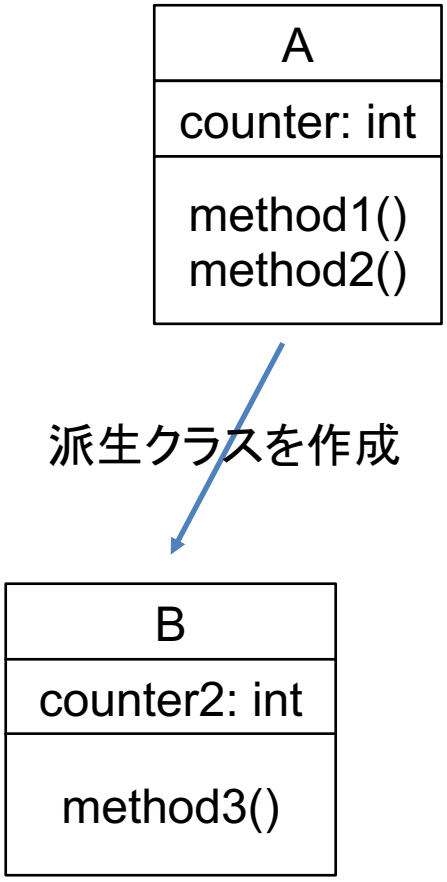
```
class A {
     int counter;
     method1(){    }
     method2(){    }
}
```

```
class B extends A {
    int counter2;
    method3(){    }
}
```

extends Aはクラスを拡張する(継承)
クラスAで定義したものはクラスBで使える

定義は必要なところで行う
継承により拡張する

クラスの関係



クラスAを拡張してクラスBを作る

上位（元、親）クラス： スーパー(super)クラス
下位（派生、子）クラス： サブ(sub)クラス

スーパー、サブは相対的な呼び方なので
CのスーパークラスであるbはAのサブクラス
Aのサブクラスであるbはcのサブクラス
ということも有り得る

最上位のクラス　　基底（base)クラス
Java言語の最上位は　Objectクラス
クラス定義で　Objectを継承する場合は
extendsの記述を省略できる
Class A extends Object { }

あるクラスから見て親は1つ、子は0個以上

多重継承：　親クラスが複数でも良い。Javaは多重継承不可

継承 フィールド、メソッドは継承される

```
class A {
     int counter;
     method1(){    }
     method2(){    }
}
class B extends A {
    int counter2;
    method3(){    }
}

public static void Ex {
    public static void main(){
         B b = new B();              // クラスBのインスタンス生成
         b.counter = 1;              // クラスAで定義したフィールド
         b.counter2 = 2;             // クラスBで定義したフィールド
         b.method1();                // クラスAで定義したメソッド
         b.method3();                // クラスBで定義したメソッド
    }
}
```

クラスBの中にクラスAを全て
取り込んだように使える

オーバーライド(override) オーバーロードでは無い

```
class A {
     int counter;
     method1(){   }
     method2(){   }
}
class B extends A {
    int counter;
    method1(){   }
}
```

メソッド
クラスBで継承元(親クラス）と同じメソッドを作成すると
下位クラスのメソッドが動作する（上位クラスのメソッドは隠蔽される）

すなわち、同じ名前のメソッドは下位クラスが上位クラスを上書きする

フィールド
簡単ではない。
親のメソッドでは親のフィールドを見てしまう。
そのフィールドを参照しているのは、どのクラスのメソッドか意識が必要

親クラスのメソッドを呼び出す

```
class A {
    int counter;
    void method1(){ }
}

class B extends A {
    void method1(){
        super.method1()
        // クラスBでの追加を書く
    }
}


//  mainクラス定義省略
B b = new B();      // インスタンンス生成
b.method1();        //  クラスBのmethod1()を呼ぶ
```

クラスBではクラスAと同じ名前のメソッドmethod1()を定義

クラスBのmethod1()はクラスAのmethod1()と同じ処理をさせたい場合も多い

その場合は、親クラスのメソッドを呼べば良い

オーバーライドは上書きだが親のメソッドがなくなってしまうわけではない

コンストラクタの継承

```
class A {
    A()      { System.out.println("A arg none"); }
    A(int a){ System.out.println("A arg "+a);   }
}
class B extends A {
    B()      { System.out.println("B arg none"); }
    B(int a){ System.out.println("B arg "+a);   }
}
public class Inh {
    public static void main(String[] args){
        A a1 = new A();
        A a2 = new A(5);
        B b1 = new B();
        B b2 = new B(5);
   }
}
```

結果

```
A arg none
A arg 5
A arg none
B arg none
A arg none
B arg 5
```

クラスBのインスタンスを作ると継承元のコンストラクタも実行されるが
b2生成時クラスAのコンストラクタは引数なしの方が実行されている

デフォルトコンストラクタ

```
class A {
    A()      { System.out.println("A arg none"); }
    A(int a){ System.out.println("A arg "+a);   }
}
class B extends A {
    super();        // デフォルトコンストラクタ呼び出し
    B()      { System.out.println("B arg none"); }
    B(int a){ System.out.println("B arg "+a);   }
}
```

継承のあるクラスからインスタンスを生成すると
実際は、上位クラスの引数なしコンストラクタを呼び出してしまう

引数あり上位コンストラクタの呼び出し

```
class A {
    A()     { System.out.println("A arg none"); }
    A(int a){ System.out.println("A arg "+a);   }
}
class B extends A {
    B(int a){
        super(a);
        System.out.println("B arg "+a);
    }
}
public class Inh1 {
    public static void main(String[] args){
        A a1 = new A();
        A a2 = new A(5);
      // B b1 = new b();
        B b2 = new B(5);
    }
}
```

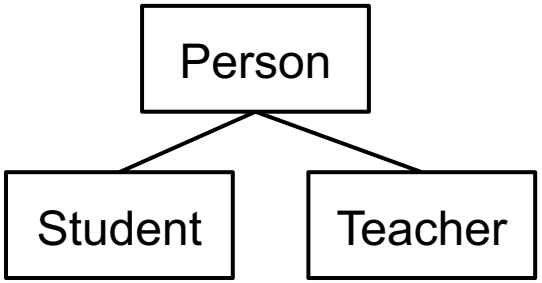
引数なしコンストラクタを作らない
引数あり上位コンストラクタを意識して呼ぶ

実行結果

```
A arg none
A arg 5
A arg 5
B arg 5
```

クラスの継承

クラスPersonからStudent, Teacherを派生




```
class Person { }
class Student extends Person { }
class Teacher extends Person { }

// インスタンス生成
Person  p = new Person();
Student s = new Student();
Teacher t = new Teacher();

// 以下のようにも書ける
Person s = new Student();
Person t = new Teacher();
```

インスタンス変数 s はStudentクラスの生成結果、
インスタンス変数 t はTeacherクラスの生成結果
だが、s, tの型は、Person

上位クラスの型で指定する場合の注意

```
class Person { int id =0; }
class Student extends Person { int id =1; }
class Teacher extends Person { int id =2; }

public class Poly1 {
    public static void main(String[] args){
        Person  p = new Person();
        Student s = new Student();
        Teacher t = new Teacher();
      // p.id -> 0, s.id ->1, t.id -> 2
    }
}
```

main()の処理を以下にすると

```
Person s = new Student();
Person t = new Teacher();
// s.id -> 0, t.id -> 0
```

クラスの確認　instanceof

```
Person[] persons = new Person[3];
person[0] = new Person();
person[1] = new Student();
person[2] = new Teacher();

for(int i =0; i<persons.length; i++){
    if( persons[i] instanceof Person ){
        // personクラスの処理
    } else if( persons[i] instanceof Student ){
        // Studentクラスの処理
    } else if( persons[i] instanceof Tehacher ){
        // Teacherクラスの処理
    }
```

ポリモーフィズム（多態性、多様性）

```
class Person {
    void work(){ System.out.println("P"); }
}
class Student extends Person {
    void work(){ System.out.println("S"); }
}
class Teacher extends Person {
    void work(){ System.out.println("T"); }
}

public class Poly2 {
    public static void main(String[] args){
        Person s = new Student(); s.work();  // Sを出力
        Person t = new Teacher(); t.work();  // Tを出力
    }
}
// オーバーライドによりメソッドが上書きされた（フィールドはだめ）
```

ポリモーフィズム（polymorphism, 多態性,多様性）

同じ機能でもクラスにより動作を変えられる

上位クラスは汎用的、下位クラスは個別
特化：汎用的から個別
汎化：個別から汎用的（一般化）

動物クラスの下位に鳥クラス、哺乳類クラスを作る

動物クラスは　移動する　という機能を持つ（メソッド）
　鳥クラスで　移動する　は飛ぶ動作を行う
　哺乳類クラスで　移動する　は歩く動作を行う

＋演算子　は　数値どうしなら加算を、文字列どうしなら連結を行う
　オーバーロード機能でも実装できたが
　ポリモーフィズムで実装することもできる（intクラス、Stringクラス）

ポリモーフィスムの活用例

メソッドに渡す複数の型を共通化する

先の例では、　Person, Student, Teacherの3クラスがある

それぞれの型に対応する場合

1) オーバーロードを使う

```
work(Person p) {    }
work(Student s){    }
work(Teacher t){    }
```

2) ポリモーフィズムを使う

```
work(Person p){    }
```

上位クラスでインスタンスを生成すれば
メソッドを集約でき
動作は下位クラスの動作となる