ソフトウェア第三
— オブジェクト指向プログラミング, Java(2) —

機械情報工学科　水内郁夫
http://www.jsk.t.u-tokyo.ac.jp/~ikuo/lec/soft3/
(ID:soft3, pass:MechanoI)

2008 年 12 月 8 日 (月)

1　オブジェクト指向プログラミング (10 月 2 日資料その 2)

1.1　抑えるべきキーワード

クラス，メソッド（メンバ関数），スロット変数（メンバ変数），インスタンス，コンストラクタ，デストラクタ，継承，スーパークラス，サブクラス，ポリモーフィズム，オーバーライド，オーバーロード，仮想関数，フレンドクラス，スコープ

1.2　オブジェクト指向プログラミングの背景

オブジェクト指向プログラミングという考え方が生まれた背景にはこの数十年で計算機の性能が爆発的に向上したことにより，従来より大規模なソフトウェアが書かれるようになってきた．大規模なソフトウェア開発が乱発，複雑化したために，ソフトウェア開発コストが上昇し，1960 年代にはソフトウェア危機という言葉も登場するようになった．そこでソフトウェアの再利用，部品化といったような概念をベースにしたソフトウェア工学が誕生することとなった．

プログラムを構成するコードとデータのうち，コードについては手続き，関数といった構成によって全体をブラックボックスとすることで再利用性の向上が提唱された．このようなスタイルは構造化プログラミングと呼ばれ 1967 年にエドガー・ダイクストラらによってまとめられた．

このように，オブジェクト指向プログラミングは，大規模なソフトウェア開発には欠かすことのできない概念であり，バグの対策や，複数人でのプロジェクト管理にも役立つ概念である．そのベースとなっている概念は，カプセル化と抽象化による，モジュールのブラックボックス化と，機能の明瞭化といえる．

1.3　クラスとインスタンス

オブジェクト指向プログラミングでは，従来までに学んだ基本的な言語（例えば C 言語や BASIC）などでは，関数を定義し，その関数を次々と呼び出すようなスタイルのプログラミングであった．しかしながら，大規模なソフトウェアを構築する場合や，複数人のチームでソフトウェアを構築する場合，必ずしもこのようなスタンダードなスタイルが有効であるとは限らない．他人の書いた

ソフトウェアを理解し，自分の書くソフトウェアをそれと同調させることは一般的に難しい．オブジェクト指向プログラミングとは，機能，内部状態，が定義されたオブジェクトと呼ばれる単位のモジュールを使って，ソフトウェアを構築しやすくする技法の一つである．

オブジェクトの型に相当するものがクラスであり，内部状態に相当するものがスロット変数（メンバ変数）機能に相当するものがメソッド（メンバ関数）となる．オブジェクト同士の間で,「機能」に対するメッセージ通信を行うことでのみ内部状態を読み込んだり，変更したりすることができる．逆に言えば，許可なく相手のオブジェクトの内部状態を変更することはできない．このこのように必要最小限の情報だけを外部に見せ，複雑で外部に必要ではない部分を隠蔽するスタイルをカプセル化と呼ぶ．これはオブジェクト指向プログラミングの特徴の一つとなっている．

1.4 インスタンス

オブジェクトは実態のない宣言のようなもので，C 言語で言えば構造体の定義分に相当する．具体的なオブジェクトの実態はインスタンスと呼ばれ，コンストラクタと呼ばれるメソッドで構築される．逆にインスタンスはデストラクタによって削除される．例えて言うと，クラスは概念を表す一般名詞（例えばイヌ）で，インスタンスは文脈の中で現れた固有名詞（例えばポチ）と考えると分かりやすい．当然のことながらインスタンスは適宜複数個作られ，必要に応じて削除されて行く．

1.5 継承，スーパークラス，サブクラス

いくつかのクラスを使ってプログラミングを進めて行くうちに，かなり似た「機能」や「内部状態」を持つケースが出てくることがある．そのような「似たような部分が沢山存在する」場合，従来の関数定義型のプログラミング言語では，サブルーチンやライブラリを構築することで対応してきた．しかしながら，オブジェクト指向プログラミングの場合はクラス自体に重なりがあるため，サブルーチンの概念を適用することはできない．そこで，オブジェクト指向プログラミングの特徴の一つでもある，継承を用いることとなる．

継承とは，あるクラスのメソッドやスロット変数をそのまま「継承」し，新たにメソッドやスロット変数を追加することで新しいクラスを簡単に定義する概念である．例えば，イヌというクラスを継承して盲導犬というクラスを作ったり，学生というクラスを継承して高校生や大学生というクラスを作ったりすることに相当する．この場合，継承元になるクラス（学生）をスーパークラス（あるいは親クラス），継承したクラス（大学生）をサブクラス（あるいは小クラス）と呼ぶ．

1.6 ポリモーフィズム

ポリモーフィズム (polymorphism) とは，日本語で「多様性」のことを意味する．オブジェクトの機能を使用する立場（ユーザの立場）で見ると，異なるクラスに存在するそれぞれの異なる機能（メソッド）にある共通性がある場合がある．ユーザからすればなるべくシンプルな形で機能を使用したいのだが，クラスごとに異なるメソッドが用意されていると，面倒なことが起こる場合がある．この場合にポリモーフィズムの概念を用いたクラスとメソッドの定義を行う場合がある．

例えば，コンビニエンスストアで，荷物を送る場合を考える．ユーザの立場ではダンボール箱を持ってコンビニに行き,「宅配をお願いします」と店員に声をかければよい．しかしながら，実際には,「セブンイレブンでは宅急便のみを受け付ける」「ampm ではペリカン便のみ受け付ける」「ファ

ミリーマートではゆうパックのみ受け付ける」というような制限がある.（もちろん例えばの話）しかしながらユーザは「宅配を依頼する」というインタフェースで目的を達成することが日常生活では当たり前になっている．これがポリモーフィズムの概念であり，異なるクラス（コンビニ）に共通するインタフェース（宅配）を親クラスのレベル（コンビニ）で定義し，各サブクラス（セブンイレブン，ampm）では，そのインタフェースの実際の挙動を決めるメソッド（宅急便，ペリカン便）を各自定義しなおすことを行う．この定義しなおすことを，オーバーライドと呼び，親クラスに共通のレベルで定義するメソッドを仮想関数と呼ぶ．先週の課題の `Benchmark.java` の構造を思い出そう．

似た用語として，オーバーロードがある．これは同じ関数名にもかかわらず，引数の型の違いによって異なる挙動をさせたい場合に，複数の同じ名前の関数の定義をすることである．感覚的には上記のポリモーフィズムと同じような概念のように思えるが，このオーバーロードは同じクラス内で複数の同じ名前の関数を定義することになるので，全く違う概念である．

1.7 フレンドクラス，多重継承

C++ や Java などでは，public や private などの修飾子を用いてカプセルの内部なのか外部なのかを区別する．private で定義されたメソッドやスロット変数は外部から直接アクセスすることができなくなる．public で定義された場合はいつでも誰からでもアクセス可能である．フレンド関数は，基本的には外部からアクセスできないメソッドを半ば強引にアクセスさせることができる．フレンドクラスはクラスの全てのメソッドに対してフレンド関数を適用することと同等の概念である．

このほかにも，複数のクラスのメソッドを同時に使いたいという場合には，多重継承と呼ばれる手法も存在する．これは二つのクラスを同時に継承することに相当する．ただし，C++では可能であるが，Java では禁止されている継承のスタイルなので，注意が必要である．

このように異なる複数のクラスを考慮しながらプログラミングを進めていくと，同じスロット変数やメソッドが重複して登場することがある．基本的にはある名前のメソッドを呼ぶ場合には自分のクラスのメソッドが呼ばれるが，親クラスやフレンドクラスのメソッドを故意に呼びたい場合にはスコープ解決を行う必要がある．C++の場合には，[クラス名]::[メソッド名] という具合に，::でスコープを指定することができる．

1.8 オーバーライドとオーバーロード (先週の資料から)

響きが似ている言葉で紛らわしいこの二つの用語を，確認しておこう．

オーバーライド: 親クラスで定義されているメソッドを子クラスで再定義 (上書き定義) すること．コンストラクタ以外の通常のメソッドでは，親クラスのメソッドは呼ばれず子クラスでオーバーライドしたメソッドが実行される．

オーバーロード: 引数の型や個数が異なる同じ名前のメソッドを複数定義すること．メソッドを呼び出す際に与えられた引数の型・個数に応じて実行されるメソッドが選択される．

2 JNI

JNI とは，Java Native Interface の略で，本来仮想マシン (JavaVM) の命令セット (Java バイトコード) しか実行できない Java 言語のプログラムから，物理 CPU (native CPU) の上で実行されるプログラムを呼び出すための仕組みである．ロボットを動かしたり，携帯電話の機種による違いを吸収するために同じ名前のクラス・メソッドで機種によって異なる実行をしたりということを可能にする．

物理的なロボットと，コンピュータ上のシミュレータの中で動くロボットを，共通のプログラムから動かすというようなこともできる．たとえば，ロボットを動かすためのメソッドを集めた抽象クラスを用意し，そのクラスを継承した物理ロボット操作クラスと，同じクラスを継承した仮想ロボット操作クラスを作成し，物理ロボット操作クラスは JNI により物理ロボットを操作するネイティブコードを実行し，仮想ロボットは Java3D で描かれたグラフィックスを動かす Java コードを実行するというようなことが可能である．

`http://www.jsk.t.u-tokyo.ac.jp/~ikuo/lec/soft3/docs/20081208-sample.zip` は，Cygwin 又は Linux で JNI をテストするためのサンプルプログラムである．

```
% cd 20081208-sample
% make
% java JNITest
```

3 パッケージ・クラスパス

パッケージは，Java の関連するクラスとインタフェースをまとめたクラスライブラリである．

パッケージに含まれるクラスのクラスファイルは，パッケージ名と同じ名前のディレクトリに格納する．パッケージ名は，ディレクトリの階層と対応し，ディレクトリの階層をドット (.) でつないだ名前となる．パッケージ内のクラスは，パッケージ名．クラス名で参照することができる．

パッケージの定義には **package** 宣言を行い，パッケージ参照には，**import** 宣言を行う．同じディレクトリ内に別々のファイルにパッケージ宣言のないクラスを定義しておいた場合には，それらのクラスはそのディレクトリの同じパッケージにあるとみなされる．

パッケージの基点となるディレクトリを**クラスパス**という．javac によるコンパイル時に，そのオプション-classpath によってそのコンパイル時のクラスパスを与えることが可能となっている．**環境変数**の **CLASSPATH** にパッケージの基点をコロン (:) で仕切って複数指定しておくことによって複数のパッケージ参照が可能となる．

3.1 パッケージとアクセス制御

あるパッケージの中で，public のクラスの中に，public のついていないメソッドを定義した場合にはそのメソッドへのアクセスはできない．

```
1  package test;
2
3  public class Point {
4    int x,y;
5    public Point() {
6     System.out.println("Point constructor");
7    }
8    void pos() {
```

```
 9       System.out.println("( "
10             + x + " , " + y + " )" );
11     }
12   }
```

```
 1   import test.*;
 2
 3   public class Line {
 4     public Line() {
 5      System.out.println("Line Constructor");
 6     }
 7     public static void main(String[] args) {
 8       Point p = new Point();
 9       // p.pos(); // アクセス不可
10     }
11   }
```

3.2 JDK でのパッケージ

JDK のシステムには，jdk1.6.0_03/jre/lib/rt.jar に含まれる基本クラスの定義されたパッケージ群をはじめとして，各種システムクラスライブラリのパッケージが用意されている．

3.3 パッケージの例

パッケージ内のライブラリを使うには **import 文**を使う．自分でパッケージを作る場合は package 文を使う．パッケージ名は，そのクラスファイルが置かれているディレクトリ名と同じ名前とする．

```
 1   /*
 2    * ap/application/Main.java
 3    * 下村：Java によるインターネットプログラミングより.
 4    *    近代科学社.
 5    */
 6   package application;
 7   import resource.*;
 8   /**
 9    * ガソリンスタンドで給油を行います.
10    */
11   public class Main {
12     public static void main (String[] args) {
13       GasStation gs = new GasStation(1000);
14       Car car1 = new Car(100);
15       Car car2 = new Car(200);
16       Tank tank1 = new Tank(400, 350);
17       gs.fillUp(car1);
18       gs.fillUp(car2);
19       gs.fillUp(tank1);
20       gs.fillUp(gs);
21     }
22   }
```

```
 1   /*
 2    * ap/application/GasStation.java
 3    */
 4   package application;
 5   import resource.*;
 6   /**
 7    * ガソリンスタンドの役割を果します.
 8    */
 9   public class GasStation extends Tank {
10     /**
11      * ガソリンスタンドを生成します.
12      * @param capacity タンクの容量と残油量
13      */
14     public GasStation(int capacity) {
```

```
15      super(capacity, capacity);
16    }
17
18    /**
19     * タンクを満タンにします.
20     * @param tank タンク
21     */
22    public void fillUp(Tank tank) {
23      int amount = tank.fillUp();
24      int remain = getFuel() - amount;
25      setFuel(remain);
26      System.out.println(tank + ": 給油量 = " + amount);
27      System.out.println(this + ": 残油量 = " + remain);
28    }
29  }
```

```
1   /*
2    * ap/application/Car.java
3    */
4
5   package application;
6   import resource.*;
7
8   /**
9    * 自動車の役割を果します.
10   */
11  public class Car extends Tank {
12    /**
13     * 指定された容量のタンクをもつ, 自動車を生成します.
14     * @param capacity タンクの容量
15     */
16    public Car(int capacity) {
17      super(capacity, 0);
18    }
19  }
```

```
1   /*
2    * ap/ressource/Tank.java
3    */
4   package resource;
5
6   /**
7    * タンクの役割を果します.
8    */
9   public class Tank {
10    private int capacity = 0;
11    private int fuel = 0;
12
13    /**
14     * タンクを生成します.
15     * @param capacity タンクの容量
16     * @param fuel 残油量
17     */
18    public Tank(int capacity, int fuel) {
19      this.capacity = capacity;
20      this.fuel = fuel;
21    }
22
23    /**
24     * 残油量を取得します.
25     * @return 残油量
26     */
27    public int getFuel() {
28      return fuel;
29    }
30
31    /**
32     * 残油量を設定します.
33     * @param amount 残油量
34     */
35    public void setFuel(int amount) {
```

```
36        fuel = amount;
37      }
38
39      /**
40       * タンクを満タンにします.
41       * @return 給油量
42       */
43      public int fillUp() {
44        int amount = capacity - fuel;
45        fuel = capacity;
46        return amount;
47      }
48    }
```

```
% javac application/Main.java
% java application.Main
application.Car@7b7072: 給油量 = 100
application.GasStation@136228: 残油量 = 900
application.Car@113750: 給油量 = 200
application.GasStation@136228: 残油量 = 700
resource.Tank@4672d0: 給油量 = 50
application.GasStation@136228: 残油量 = 650
application.GasStation@136228: 給油量 = 350
application.GasStation@136228: 残油量 = 650
```

3.4 java.util パッケージの利用

java.util.* のように*を使うと，java.util パッケージにおけるすべてのクラスを import する．

```
1   // Property.java
2   import java.util.*;
3
4   public class Property {
5     public static void main(String[] args) {
6       System.out.println(new Date());
7       Properties p = System.getProperties();
8       p.list(System.out);
9       System.out.println("--- Memory Usage:");
10      Runtime rt = Runtime.getRuntime();
11      System.out.println("Total Memory = "
12                  + rt.totalMemory()
13                  + " Free Memory = "
14                  + rt.freeMemory());
15      }
16  }
```

```
% java Property
Mon Dec 08 08:11:07 JST 2008
-- listing properties --
java.runtime.name=Java(TM) 2 Runtime Environment, Stand...
sun.boot.library.path=C:\cygwin\usr\local\Java\j2sdk1.4.2_1...
java.vm.version=1.4.2_16-b05
java.vm.vendor=Sun Microsystems Inc.
java.vendor.url=http://java.sun.com/
path.separator=;
```

```
java.vm.name=Java HotSpot(TM) Client VM
file.encoding.pkg=sun.io
user.country=JP
sun.os.patch.level=Service Pack 3
java.vm.specification.name=Java Virtual Machine Specification
user.dir=C:\cygwin\home\ikuo\doc\kyomu\soft3\2...
java.runtime.version=1.4.2_16-b05
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
java.endorsed.dirs=C:\cygwin\usr\local\Java\j2sdk1.4.2_1...
......(中略)
java.vendor.url.bug=http://java.sun.com/cgi-bin/bugreport...
sun.cpu.endian=little
sun.io.unicode.encoding=UnicodeLittle
sun.cpu.isalist=pentium i486 i386
--- Memory Usage:
Total Memory = 2031616 Free Memory = 1699816
```

というような表示がなされる．

3.5 クラスパス

クラスパスは，Java のクラス定義のファイルを探す場所をリストアップしたものである．環境変数 CLASSPATH や，java 起動時に-classpath オプションを付けて，クラスパスを指定することができる．クラスパスが設定されていない場合，カレントディレクトリの*.class ファイルとデフォルトのシステムライブラリをクラスパスとしてクラス定義を探索する．クラスパスが設定されている場合，CLASSPATH にカレントディレクトリが含まれなければ，カレントディレクトリの *.class を探索しない．

カレントディレクトリのファイルは常に見て欲しい場合は，~/.bashrc に，

```
if [ "$CLASSPATH" != "" ]; then
export CLASSPATH=".;${CLASSPATH}"
fi
```

を追加する．CLASSPATH がすでに設定されている場合，カレントディレクトリを追加する．

Windows の java は，Cygwin の世界ではなく，Windows の世界で動くので，CLASSPATH のディレクトリは Windows の名前で表記し，複数のディレクトリを並べる場合は ; で区切る．たとえば，"C:\Windows;C:\cygwin\home\mechuser\javalib"のようにする．

環境変数 CLASSPATH の代わりに，

```
java -classpath ".;C:\cygwin\home\mechuser\javalib" HelloWorld
```

のようにすることもできる．

3.6 Java アーカイブ

Java 環境には，jar ファイル形式を扱うユーティリティがある．jar コマンドの使い方は次のようになっており，options は，tar コマンドと同様の，c(生成), t(内容一覧), x(抽出), f (ファイル名

指定), v(詳細表示) がある.

```
jar [options] destination [manifest] inputfiles(s)
```

jar ファイル形式は，CLASSPATH 等で指定すると jar ファイルの中の*.class ファイルを参照することができる.

例 1：`CLASSPATH="lib1.jar;lib2.jar;C:\cygwin\home\root\javalib"`
例 2：`java -classpath "../TestLib.jar;." libtest`

ちなみに，jar ファイルの実態は zip 形式である.

4 Java プログラミング

4.1 配列

```
1  // ArrayTest1.java
2
3  public class ArrayTest1 {
4    public static void main(String argv[]){
5      int a1[]={1, 3, 5, 4};
6      int a2[]={2, 4, -1};
7      int a3[];
8      int[] a4;
9      int a5[]= new int[5];
10     int a6[]= new int[5];
11     Integer a7[]= {new Integer(7), new Integer(8)};
12     Integer a8[]= new Integer[2];
13
14     a3=a1;
15     a6=a2;
16     a4=a3;
17     a2=a1;
18     a8=a7;
19 //    a8=a2;  -> error
20
21     for (int i=0; i<a1.length; i++)
22       System.out.print("a1["+i+"]= "+a1[i]+" ");
23     System.out.println();
24     for (int i=0; i<a2.length; i++)
25       System.out.print("a2["+i+"]= "+a2[i]+" ");
26     System.out.println();
27     for (int i=0; i<a3.length; i++)
28       System.out.print("a3["+i+"]= "+a3[i]+" ");
29     System.out.println();
30     for (int i=0; i<a4.length; i++)
31       System.out.print("a4["+i+"]= "+a4[i]+" ");
32     System.out.println();
33     for (int i=0; i<a5.length; i++)
34       System.out.print("a5["+i+"]= "+a5[i]+" ");
35     System.out.println();
36     for (int i=0; i<a6.length; i++)
37       System.out.print("a6["+i+"]= "+a6[i]+" ");
38     System.out.println();
39     for (int i=0; i<a7.length; i++)
40       System.out.print("a7["+i+"]= "+a7[i]+" ");
41     System.out.println();
42     for (int i=0; i<a8.length; i++)
43       System.out.print("a8["+i+"]= "+a8[i]+" ");
44     System.out.println();
45   }
46 }
```

```
% java ArrayTest1
a1[0]= 1 a1[1]= 3 a1[2]= 5 a1[3]= 4
a2[0]= 1 a2[1]= 3 a2[2]= 5 a2[3]= 4
a3[0]= 1 a3[1]= 3 a3[2]= 5 a3[3]= 4
a4[0]= 1 a4[1]= 3 a4[2]= 5 a4[3]= 4
a5[0]= 0 a5[1]= 0 a5[2]= 0 a5[3]= 0 a5[4]= 0
a6[0]= 2 a6[1]= 4 a6[2]= -1
a7[0]= 7 a7[1]= 8
a8[0]= 7 a8[1]= 8
```

4.2 制御文

制御の流れを変えるための制御文としては，(1)if-else 文，(2) swtich 文，(3)while 文，(4) do-while 文，(5)for 文，(6)break 文，(7)return 文，(8)continue 文, (9)try-catch-finally 文などがあり，メソッド呼び出し文や例外をスローする throw 文なども制御の流れを変える．文には，名前で参照できるように，文の前にラベルを付けることができる．break 文や continue 文では，そのラベルを指定することが可能であり，goto 文は Java では用意されていない．

4.3 break 文

label 無しの break 文は，もっとも内側の switch, for, while, do 文を終了させ，それらの文の中だけに記述できる．label 有りの break 文は，その label が付いた文を終了させる．

```
1   public boolean workOnFlag(flaot flag) {
2     int y, x;
3     search: {
4       for (y = 0; y < matrix.length; y++) {
5         for (x = 0; x < matrix[y].length; x++) {
6           if (matrix[y][x] == flag)
7             break search;
8         }
9       }
10      // 発見されなかった時にはここに来る.
11      return false;
12    }
13    // 発見された時の処理をここで行う.
14    return true;
15  }
```

4.4 continue 文

label 無しの continue 文は，ループ (for, while, do) の中だけで使用でき，ループの本体の終わりへ制御を移し，次にループ式が評価される．for ループでは，ループ式の前に，increment 式が評価される．

label 有り continue 文は，その label を持つループの終わりに制御を移す．2 重のループがある場合にそのループの外側のループの終わりへ制御を移したい場合などに利用する．

```
1   static void doubleUp(int[][] matrix) {
2     int order = matirx.length;
3     column:
4       for (int i = 0; i < order; i++) {
5         for (int j = 0; j < order; j++) {
6           matrix[i][j] = matrix[j][i] = matirx[i][j]*2;
```

```
7           if (i == j) continue column;
8         }
9       }
10    }
```

行列の対角の要素に達するごとに，列に対して繰り返している外側のループに戻って継続することで行の残りの処理がスキップされている．

4.5 例外処理

プログラムの中ではいろいろなエラーが発生する場合がある．それは，すべてのエラーが起こりうる可能性をあらかじめ調べつくしておくということが困難な場合や，事前に調べるようなコードを入れておくとプログラムが煩雑になったりするために，エラーが起こってから処置をするという具合にせざるを得ない形になる．

そのため，Java では，そういったエラーをきれいに扱うための例外 (exception) をクラスとして定義している．Java では，異常なエラー状態を検出したメソッドは，例外を**スロー (throw)** する．例外は，呼び出しスタック上のコードによって**キャッチ**する．メソッドを呼び出しているコードは，そのスローされた例外を処理して実行を続けることができる．例外がキャッチされないと実行中のスレッドは終了する．例外は，オブジェクトで，型，メソッド，データを伴うことができ，かつ，**スロー**されるオブジェクトとなるようにクラス Throwable やそのサブクラスの Exception クラスを拡張した形で定義される形となっている．

例外型の作成は，以下のように行う．

```
1   public class NoSuchAttributeException
2               extends Exception {
3     public String attrName;
4     public NoSuchAttributeException(String name) {
5       super("No attribugte named \"" +
6                       name + "\" found ");
7       attrName = name;
8     }
9   }
```

4.6 throws 節

例外をスローする throw 文は，

```
throw expression;
```

という具合になる．

例外をスローするメソッドには，throws 節を宣言する必要がある．メソッドは，メソッド名と両カッコの間の引数のパラメタを**シグニチャ(signature)** というが，例外をスローする場合には，シグニチャの後に **throws 節**を付ける．複数の例外をスローする可能性がある場合には，それらを列挙する．

```
1   public void replaceValue(String anme,
2               Object newValue)
3               throws NoSuchAttributeException {
4     Attr attr = find(name); // attr を探す.
5     if (attr == null) // 見つからない場合
6        throw new NoSuchAttributeException(name);
7     attr.setValue(newValue);
8   }
```

4.7 try-catch-finally 文

try-catch-finally 文は try の後のブロックを実行して catch 節の中の例外が発生すればそれを catch 節のブロックを実行し，finally 節は try の中でエラーが起きても起きなくても必ず実行される．

下のような構文で，catch 節は複数記述することができ，finally 節を省略した，try-catch 文や，catch 節を省略した try-finally 文なども利用可能である．

```
try {
        try-statements
} catch (exception_type1 identifier1) {
        catch1-statements
} catch (exception_type2 identifier2) {
        catch2-statements
...
} finally {
        finally-statements
}
```

複数の catch 節がある場合には，最初から順に exception-type を調べてゆくため，最初に上位の exception-type があるとエラーとなる．下位の Exception 型からチェックしてゆくような形にする必要がある．

catch 節と finally 節の中でエラーが起きたり，例外をスローしたりした場合には，この try-catch-finally 文の外の try-catch-finally 文で catch されることになる．

```
1   //TryTest.java
2   class TryTest{
3     public static void main (String[] args) {
4       if (args.length == 0) {
5         System.out.println("suuply one arg");
6         return;
7       }
8       try {  //(1)try
9         int x = Integer.parseInt(args[0]);
10        System.out.print  ("100 / " + x + " is ");
11        System.out.println(100/x);
12      } catch(ArithmeticException e) {  //(2)catch
13        System.err.println("Error" + e.getMessage());
14      } finally {  //(3)finally
15        System.out.println("This program was finished.");
16      }
17    }
18  }
```

4.8 オーバロード

あるクラスに，引数の数や型は異なるが同じ名前のメソッドを複数定義しておき，引数の違いによって自動的にメソッドが選ばれて実行される機構のことをオーバロードという．

```
1   // ArrayTest2.java
2   public class ArrayTest2 {
3     static void printArray(String name, int a[]) {
4       for (int i=0; i<a.length; i++)
5         System.out.print(name + "["+i+"]= "+a[i]+" ");
6       System.out.println();
7     }
8     static void printArray(String name, Object a[]) {
9       for (int i=0; i<a.length; i++)
10        System.out.print(name + "["+i+"]= "+a[i]+" ");
11      System.out.println();
12    }
```

```
13      public static void main(String argv[]){
14        int a1[]={1, 3, 5, 4};
15        int a2[]={2, 4, -1};
16        int a3[];
17        int[] a4;
18        int a5[]= new int[5];
19        int a6[]= new int[5];
20        Integer a7[]= {new Integer(7), new Integer(8)};
21        Integer a8[]= new Integer[2];
22
23        a3=a1;
24        a6=a2;
25        a4=a3;
26        a2=a1;
27      //    a8=a2;  -> error
28
29        printArray("a1",a1);
30        printArray("a2",a2);
31        printArray("a3",a3);
32        printArray("a4",a4);
33        printArray("a5",a5);
34        printArray("a6",a6);
35        printArray("a7",a7);
36        printArray("a8",a8);
37        printArray("a9",new Object[] {"a", "b", "c"});
38        printArray("a10",new Object[]
39                   {new Float(3.14),
40                      new Integer(4),
41                      new Double(2.3)});
42      }
43    }
```

```
% java ArrayTest2
a1[0]= 1 a1[1]= 3 a1[2]= 5 a1[3]= 4
a2[0]= 1 a2[1]= 3 a2[2]= 5 a2[3]= 4
a3[0]= 1 a3[1]= 3 a3[2]= 5 a3[3]= 4
a4[0]= 1 a4[1]= 3 a4[2]= 5 a4[3]= 4
a5[0]= 0 a5[1]= 0 a5[2]= 0 a5[3]= 0 a5[4]= 0
a6[0]= 2 a6[1]= 4 a6[2]= -1
a7[0]= 7 a7[1]= 8
a8[0]= null a8[1]= null
a9[0]= a a9[1]= b a9[2]= c
a10[0]= 3.14 a10[1]= 4 a10[2]= 2.3
```

4.9 オーバーライド

親クラスと子クラスで同じメソッド名のものがあった場合に，子クラスは親クラスのメソッドをオーバライドしているという．子クラスのインスタンスにあるメソッドを作用させた場合に，子クラスに定義されていなければ親クラスのメソッドが実行されるが，子クラスに専用のメソッドを作用させたい場合に子クラスの方にそのメソッドを定義しておく．子クラスのメソッドの中で，親クラスのメソッドも実行したければ明示的に super. メソッド名で呼び出す必要がある．

宇宙空間の浮遊する物体クラス Particle とエンジンをもったロケットクラス Rocket の例を，以

下に示す.

```
// Universe.java

import java.util.*;

public class Universe {
  static int xg = 0, yg = 4;
  static int height = 25, width = 25;

  static class Particle {
    String name;
    int pos[] = {0, 0};
    int dot[] = {0, 0};
    int acc[] = {0, 0};

    Particle(String n, int px, int py,
             int vx, int vy) {
      name = n; pos[0] = px; pos[1] = py;
      dot[0] = vx; dot[1] =vy;
    }
    Particle(String n, int px, int py) {
      this(n,px,py,0,0);
    }
    int[] pos() { return pos; }
    int[] pos(int x, int y) {
      pos[0] = x; pos[1] = y;
      return pos;
    }
    int[] vel() { return dot; }
    public int[] vel(int x, int y) {
      dot[0] = x; dot[1] = y;
      return dot;
    }
    int[] acc() { return acc; }
    int[] acc(int x, int y) {
      acc[0] = x; acc[1] = y;
      return acc;
    }
    void step() {
      pos[0] += dot[0]; pos[1] += dot[1];
      dot[0] += acc[0]; dot[1] += acc[1];
    }
    void status1() {
      System.out.print(
        name + "'s P:(" +
        pos[0] + ", " + pos[1]
        + ")  V:(" +
        dot[0] + ", " + dot[1]
        + ")  A:(" +
        acc[0] + ", " + acc[1] + ") ");
      if (iscrashed()) {
        System.out.println(";");
        System.out.print(
          "        " + name + " crashed ");
      }
    }
    void status() {
      status1();
      System.out.println();
    }
    boolean iscrashed() {
      return( (pos[0] < width) &&
              (pos[1] > height));
    }
  }

  static class Rocket extends Particle {
    int motor[] ={0, 0};

    public Rocket(String n, int x0, int y0,
                  int vx, int vy, int mx, int my) {
      super(n, x0, y0, vx, vy);
      motor(mx, my);
    }
    int[] motor() { return motor; }
    int[] motor(int x, int y) {
      motor[0] = x; motor[1] = y;
      acc[0] = xg + x; acc[1] = yg + y;
      return motor;
    }
    void status() {
      super.status1();
      System.out.println(" M:(" +
                         motor[0] + ", " +
                         motor[1] + ") ");
    }
  }

  public static void main(String argv[]) {
    Particle[] ps = new Particle[6];

    ps[0] = new Particle(new String("p1"),4,3,2,5);
    ps[1] = new Particle("p2",0,0,0,1);
    ps[2] = new Particle("p3",1,1,9,4);
    ps[3] = new Rocket("r1",2,3,4,3,2,1);
    ps[4] = new Rocket("r2",7,5,2,1,7,6);
    ps[5] = new Rocket("r3",2,4,4,2,2,3);

    for (int t=0; t<5; t++) {
      for(int i=0; i<6; i++) {
        ps[i].status();
        ps[i].step();
      }
    }
  }
}
```

実行例は，以下のようになる．

```
% java Universe
p1's P:(4, 3)  V:(2, 5)  A:(0, 0)
p2's P:(0, 0)  V:(0, 1)  A:(0, 0)
p3's P:(1, 1)  V:(9, 4)  A:(0, 0)
r1's P:(2, 3)  V:(4, 3)  A:(2, 5)  M:(2, 1)
r2's P:(7, 5)  V:(2, 1)  A:(7, 10)  M:(7, 6)
r3's P:(2, 4)  V:(4, 2)  A:(2, 7)  M:(2, 3)
p1's P:(6, 8)  V:(2, 5)  A:(0, 0)
p2's P:(0, 1)  V:(0, 1)  A:(0, 0)
p3's P:(10, 5)  V:(9, 4)  A:(0, 0)
r1's P:(6, 6)  V:(6, 8)  A:(2, 5)  M:(2, 1)
r2's P:(9, 6)  V:(9, 11)  A:(7, 10)  M:(7, 6)
r3's P:(6, 6)  V:(6, 9)  A:(2, 7)  M:(2, 3)
p1's P:(8, 13)  V:(2, 5)  A:(0, 0)
p2's P:(0, 2)  V:(0, 1)  A:(0, 0)
p3's P:(19, 9)  V:(9, 4)  A:(0, 0)
r1's P:(12, 14)  V:(8, 13)  A:(2, 5)  M:(2, 1)
r2's P:(18, 17)  V:(16, 21)  A:(7, 10)  M:(7, 6)
r3's P:(12, 15)  V:(8, 16)  A:(2, 7)  M:(2, 3)
p1's P:(10, 18)  V:(2, 5)  A:(0, 0)
p2's P:(0, 3)  V:(0, 1)  A:(0, 0)
p3's P:(28, 13)  V:(9, 4)  A:(0, 0)
r1's P:(20, 27)  V:(10, 18)  A:(2, 5) ;
        r1 crashed  M:(2, 1)
r2's P:(34, 38)  V:(23, 31)  A:(7, 10)  M:(7, 6)
r3's P:(20, 31)  V:(10, 23)  A:(2, 7) ;
        r3 crashed  M:(2, 3)
p1's P:(12, 23)  V:(2, 5)  A:(0, 0)
p2's P:(0, 4)  V:(0, 1)  A:(0, 0)
p3's P:(37, 17)  V:(9, 4)  A:(0, 0)
r1's P:(30, 45)  V:(12, 23)  A:(2, 5)  M:(2, 1)
r2's P:(57, 69)  V:(30, 41)  A:(7, 10)  M:(7, 6)
r3's P:(30, 54)  V:(12, 30)  A:(2, 7)  M:(2, 3)
```

4.10 ファイル

File クラスは，ファイルやディレクトリを扱うことができ，`File(String path)`，`File(String dirName, String name)`，`File(File fileDir, String name)` などでファイルを作り，`length()`，`lastModified()`，`reanmeTo(File newName)`，`mkdir()`，`delete()`，などのメソッドがあり，`list` メソッドでディレクトリ内のファイルを列挙することができる．

```
1   import java.io.*;
2
3   public class FileList {
4       public static void main(String[] args) {
5         try {
6             File path = new File(".");
7             String[] list = path.list();
8             for(int i = 0; i < list.length; i++)
9                 System.out.println(list[i]);
10        } catch(Exception e) {
11            e.printStackTrace();
12        }
13    }
14  }
```

以下は，ファイルから 1 文字ずつ読み込む例である．DataInputStream の readByte() メソッドを用いている．

```
1  // TestEOF.java
2  // Copyright (c) Bruce Eckel, 1998
3  // Testing for the end of file while reading
4  // a byte at a time.
5  import java.io.*;
6
7  public class TestEOF {
8    public static void main(String[] args) {
9      try {
10       DataInputStream in =
11         new DataInputStream(
12          new BufferedInputStream(
13           new FileInputStream("TestEof.java")));
14
15       while(in.available() != 0)
16         System.out.print((char)in.readByte());
17     } catch (IOException e) {
18       System.err.println("IOException");
19     }
20   }
21 }
```

次の例は，ファイルをコピーするための例である．ファイルが無い場合や，ディレクトリを指定した場合にはエラー表示がなされるように Exception をキャッチして，err 出力へ表示を行っている．そのために，main メソッドには，throws 節が宣言されている．read(buf) は，buf.length バイトのデータを読み出す.

```
1  import java.io.*;
2
3  public class FileInputOutput {
4    public static void main (String[] args)
5              throws Exception {
6      if (args.length < 2) {
7        System.err.println(
8           "使用方法: java FileInputOutput " +
9              "コピー元 コピー先");
10       return;
11     }
12
13     final int BUF_SIZE = 1024;
14     byte buf[] = new byte[BUF_SIZE];
15     int numBytes;
16
17     try {
18       FileInputStream fis =
19           new FileInputStream(args[0]);
20       FileOutputStream fos =
21           new FileOutputStream(args[1]);
22
23       while ((numBytes = fis.read(buf))
24                                != -1) {
25         fos.write(buf, 0, numBytes);
26       }
27       fis.close();
28       fos.close();
29     } catch (Exception e) {
30       System.err.println(e);
31     }
32   }
33 }
```

5 Java アプレット

アプレットは，HTML ビューワから実行される Java オブジェクト (Applet クラスを継承) で，main メソッドから実行されるのではなく，init(ロード時に実行される処理)，start(開始時に実行され

る処理)，stop(そのページを離れるときに実行する処理)，paint(描画を行う処理)，destroy(HTML ビューワを終了する時に実行する処理) 等のメソッドを定義する．

アプレットの実行は，HTML に次のように記述しブラウザから見ることができる．

```
1  <APPLET code="AppletTest.class" width=220 height=120>
2  </APPLET>
```

手軽にアプレットを表示するには，appletviewer [1]を用いるのが便利である．

```
1  // AppletCar.java
2  //<APPLET code="AppletCar.class" width=220 height=120>
3  // </APPLET>
4  import java.awt.*;
5  import java.applet.*;
6  public class AppletCar extends Applet
7  {
8    public void paint(Graphics g)
9      {
10       g.clearRect(0,20,400,100);
11       g.drawRect(50,60,150,30);
12       g.drawOval(80,90,20,20);
13       g.drawOval(150,90,20,20);
14       g.drawLine(80,60,120,30);
15       g.drawLine(120,30,170,30);
16       g.drawLine(170,30,190,60);
17       g.drawString("Hello",120,80);
18     }
19 }
```

上のプログラムは Java awt (abstract window tool) パッケージを用いて 2 次元のグラフィックスを表示するサンプルである．これを，<APPLET タグを記述した HTML から開くには，Cygwin では次のようにすることができる．

```
% javac AppletCar.java
% appletviewer AppletCar.html
```

AppletCar.java のコメントに HTML ファイルに記述するタグと同じものを入れておけば，

```
% appletviewer AppletCar.java
```

とすることも可能である (図 1).

先週の資料に JDK に付属の demo/applets/を見てみようというのがあった．

```
% ls /java/jdk1.6.0_03/demo/applets
Animator/  CardTest/    Fractal/       JumpingBox/      SortDemo/
ArcTest/   Clock/       GraphLayout/   MoleculeViewer/  SpreadSheet/
BarChart/  DitherTest/  GraphicsTest/  NervousText/     TicTacToe/
Blink/     DrawTest/    ImageMap/      SimpleGraph/     WireFrame/
```

appletviewer で html ファイルを開いてみると，手軽にサンプルを実行してみることができるだろう．

[1]先週の JDK インストールを行っていれば，/Java/jdk1.6.0_03/bin/appletviewer.exe にある．

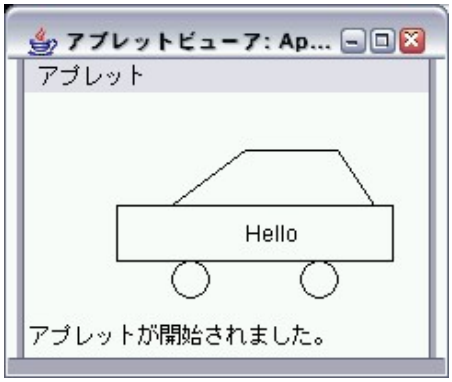

図 1: Javacar

6 ロボットの行動プログラム

人工知能の行動システムのアーキテクチャは，感覚情報から問題空間を記述し，行動計画をたてて，動作実行するというように，感覚から動作への流れを推論システムを通してシーケンシャルに進めてゆくという考え方が普通に考えられている．しかし，高度な知能はないけれども実世界で環境に適応しながら生き続けている昆虫や動物はそのような形ではなく，感覚から動作系への並列機構を持ち，下位の反射行動が上位の行動から管理されているような構造があるのではないかということで, 新しい行動システムの構成法としてサブサンプションアーキテクチャが MIT の人工知能研究所の Rodney Brooks 教授により 1986 年に提唱された．感覚から動作へという処理の流れのなかで，すべて問題空間の記述と行動計画をたててから実行というのではなく，反射行動のようにあらかじめ決められた動作を即座に行動する流れもなければ知能をもつように見える生物の仕組みを理解することは困難であるということを移動ロボットを作りながらの提唱がなされた．サブサンプションアーキテクチャは，さらに反射層の流れが上位の流れから制御されるように構成し，上位層が追加されてゆく際に下位層の作り直しが起こらないように下位層が整ってゆく構造を目指したものである．

6.1 subsumption1 サンプル

LEGO MINDSTORMS RCX 及び LEGO MINDSTORMS NXT は，LEGO 社が MIT(マサチューセッツ工科大学) と共同開発した，LEGO をベースとしたブロックにモータやセンサのブロックを追加し，それらを制御するマイクロプロセッサモジュールを用意し，自由にプログラミングができるという製品である．1998 年に登場した初代の LEGO MINDSTORMS の後継として 2006 年に登場した LEGO MINDSTORMS NXT のプロセッサブロックである LEGO NXT は，ARM 系の CPU が搭載されているが，この上で走る Java をベースとしたロボットプログラミング環境として，leJOS と呼ばれる環境が世界中の有志によって開発されている．

`http://lejos.sourceforge.net/`

Lejos の example プログラムには，サブサンプションアーキテクチャを Java の Thread を用いた形のプログラムとして実装したプログラムがいくつか提供されている．Subsumption1 はその一つの例である．

`http://lejos.sourceforge.net/p_technologies/nxt/nxj/api/` には，MINDSTORMS NXT 用の leJOS のクラス一覧がある．

`http://lejos.sourceforge.net/` の左上の download をクリックして，NXT 向け leJOS をダウンロードして，`samples/Subsumption1/Main.java` を見てみよう．

プログラムの流れを読むために，次のことを調べてみるとよい．

1. Subsumption1 の Action インタフェースの役割は何か．
2. Subsumption1 の Task クラスの fsm の役割は何か．
3. Subsumption1 の LeftBumber, RightBumber, Wander の働きは何か．
4. Subsumption1 の synchronized メソッドの働きは何か．
5. Subsumption1 によりロボットの行動がどのようなことになるか．