ソフトウェア第三 — Java に触れる —

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 1 日 (月)

1　Java 言語のプログラミング環境

Java 言語は，“Write Once, Run Anywhere” を哲学とし，どの計算機でも一度書いたプログラムが走るように設計されたプログラミング言語である．計算機ハードウェア，計算機 OS に依存しないようにするために，仮想的な計算機 (Java virtual machine) を考え，その計算機でプログラムが走り，その仮想計算機がそれぞれの計算機の上で走るという構造を想定している．言語自体も，大きなプログラムを作る際には不可欠となっているオブジェクト指向言語となっており，さまざまなクラスライブラリを利用して高度な応用プログラムも容易に作れるようになっている．Sun Microsystems という会社が開発し，Java 言語のコンパイラ，仮想計算機が公開されており，

- Java ホームページ http://java.sun.com/
- チュートリアル http://java.sun.com/docs/books/tutorial/index.html

からダウンロードすることができる．PC でよく使うものは Java Standard Edition(Java SE 又は J2SE) で，実行環境は JRE(J2SE Runtime Environment) と呼ばれ，開発キット (コンパイラ等) は JDK(J2SE Development Kit) と呼ばれる．最新のバージョンは Java SE 6 となっている．

1.1　参考書

1. K.Arnold, J.Gosling, D. Holmes,The Java Programming Language, Third Edition, Pearson Education Japan, 2000.(柴田訳, プログラミング言語 Java 第 3 版，ピアソンエデュケーションジャパン，2001.)
2. Bruce Eckel（安藤慶一訳），Thinking in Java(Bruce Eckel のプログラミングマスターコース, 上下）, ピアソン・エデュケーション，1999
3. Joseph O’neil, Teach Yourself Java, 1999.(ジョゼフ・オニール著，トップスタジオ訳，武藤健志 監修，独習 Java ，SHOEISHA，2000.
4. Steven Holzner 著，武藤健志 監修，トップスタジオ訳, Java Black Book, インプレス, 2000
5. Michael Morrison , Jerry Ablan(福井真吾，久野禎子，久野靖), Teach Yourself More Java in 21 days(続・Java 言語入門 - 新しいフレームワークとＡＰＩ), ピアソン・エデュケーション,1998
6. 下村隆夫,Java によるインターネットプログラミング，近代科学社，2002
7. Patrick Henry Winston, Sundar Narasimhan(鬼頭繁治，滝沢　徹，牧野祐子訳），On to Java(ウィンストンの Java), アジソンウェスレイ，1997.
   オンラインバージョン：http://www.ai.mit.edu/people/phw/OnToJava/
8. Laura Lemay, Charles L. Perkins(武舎広幸，久野禎子，久野靖訳), Teach Yourself Java in 21 days (Java 言語入門 - アプレット，ＡＷＴ，先進的機能), ピアソン・エデュケーション, 1996

1.2 Java 言語開発環境

Java のプログラミング環境である JDK(Java Development Kit) は `http://java.sun.com` からダウンロードすることができる．Windows 上で，走らせるパッケージ JDK(Java development Kit) は，`jdk-6u3-windows-i586-p-iftw.exe*` というような実行形式となっていて，これを実行するとコンパイラやクラスライブラリが展開される．

展開される場所を cygwin の環境ように `c:/cygwin/java/jdk1.6.0_03` を指定すると，Cygwin の bash の中では，`/java/jdk1.6.0_03` と参照される場所に入ることになる．

上記の場合，実行プログラム探索用パスを指定する環境変数 PATH に，`/java/jdk1.6.0_03/bin` を加える．version 1.6 の JDK 環境は以下のようになる．

```
/Java/jdk1.6.0_03> ls
COPYRIGHT    README_ja.html    db/       lib/     LICENSE      README_zh_CN.html  demo/     sample/
LICENSE.rtf  THIRDPARTYLICENSEREADME.txt  include/  src.zip  README.html  bin/             jre/

/Java/jdk1.6.0_03/bin> ls
HtmlConverter.exe*  java-rmi.exe*  jconsole.exe*    jstack.exe*   native2ascii.exe*  schemagen.exe*
appletviewer.exe*   java.exe*      jdb.exe*         jstat.exe*    orbd.exe*          serialver.exe*
apt.exe*            javac.exe*     jhat.exe*        jstatd.exe*   pack200.exe*       servertool.exe*
beanreg.dll*        javadoc.exe*   jinfo.exe*       keytool.exe*  packager.exe*      tnameserv.exe*
extcheck.exe*       javah.exe*     jli.dll*         kinit.exe*    policytool.exe*    unpack200.exe*
idlj.exe*           javap.exe*     jmap.exe*        klist.exe*    rmic.exe*          wsgen.exe*
jar.exe*            javaw.exe*     jps.exe*         ktab.exe*     rmid.exe*          wsimport.exe*
jarsigner.exe*      javaws.exe*    jrunscript.exe*  msvcr71.dll*  rmiregistry.exe*   xjc.exe*

/Java/jdk1.6.0_03/demo> ls

applets/  applets.html  jfc/  jpda/  jvmti/  management/  nbproject/  plugin/  scripting/

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

という具合に demo プログラムがたくさん用意されている．

1.3 コンパイラ javac

コンパイラは `javac` という名前である．`javac` は，Java のソースプログラム (通常は拡張子として java をつけたファイル名とする．) を Java 仮想計算機 (JavaVM) の機械語コードへ変換する．ソースプログラムは，通常はクラス名をつけておく．

コンパイルによって，プログラム内部のクラス名に class という拡張子のついたオブジェクトファイルができあがる．

このプログラムを実行するには，オブジェクトコードファイルを順に実行するプログラムが必要で java という名前のプログラムがそれである．

```
public class HelloWorld {
/**
ここから

ここまでがコメントとなる．
javadoc HelloWorld.java とすると
ドキュメントの html が作られる．
**/
  public static void main(String argv[]){
    System.out.println("Hello World!");
  }
}

コンパイル
% javac HelloWorld.java

確認
% ls *.class
HelloWorld.class

実行
% java HelloWorld
Hello World!
```

この例で，main は HelloWorld のメソッドだが，public がついているのでこのファイルの外からも呼び出すことができるメソッドになっている．また，static がついているので，HelloWorld クラスのクラスメソッドという意味になる．クラスメソッドというのは，クラスのインスタンスが無くても呼び出すことができるメソッドのことである．main メソッドは，クラスメソッドになる．

System.out.println は，System クラスのクラス変数 out に対してメソッド println を送っているものである．

クラス変数 out の型は，PrintStream クラスのインスタンスになっていて，PrintStream のクラスには，checkError, close, flush, print, println, write などのメソッドがある．メソッド println の引数は，引数無，boolean, char, char[], double, flot, int long, Object, String が可能となっている．

Java では，static がついていないメソッドは，インスタンスメソッドまたはただメソッドと呼びる．これは，インスタンスが作られたあと，そのインスタンスに対して実行するメソッドという意味になる．

また，クラス内の変数に対しても，static がついているものはクラス変数，ついていないものはインスタンス変数といいる．クラス変数は，そのクラスのインスタンスすべてが共有する変数となる．

static 宣言が変数についた場合には，その変数はクラス変数になり，つかないものはインスタンス変数となる．

クラスのインスタンスを作った場合に，そのインスタンスからクラス変数もそれ以外の変数 (インスタンス変数) も両方がそのインスタンス内部にあるかのように扱うことができるが，クラス変数はそのクラスから作られたすべてのインスタンス間で共通のものとなる．すなわちあるインスタンスでそのクラス変数の値を変更した場合には，他のインスタンスから見るそのクラス変数の値がその変更された値になって見える．クラス変数は，クラスにひとつ定義され，インスタンス変数は，インスタンスごとに定義されるということになる．

1.4 ドキュメント生成 javadoc

Java 言語環境には，Java のソースプログラムからプログラム内部から，クラスの構造，コメントなどを抜き出して，html 形式のドキュメントファイルを出力するツール javadoc が用意されている．

```
% mkdir HelloWorldDoc
% cd HelloWorldDoc
ドキュメントの生成
% javadoc ../HelloWorld.java
ドキュメントの閲覧 (cygwin の場合)
% cygstart index.html
```

これにより，図 1 のようなドキュメントが作られる．

public 宣言のないクラスについては，`javadoc -private file.java` のように，`-private` を指定すればドキュメントが作られる．

1.5 逆アセンブラ javap

`javap` は，逆アセンブラで，`.class` ファイルを読み込む．

```
% javap HelloWorld
Compiled from HelloWorld.java
public class HelloWorld extends java.lang.Object {
    public HelloWorld();
    public static void main(java.lang.String[]);
}
```

2 オブジェクト指向

オブジェクト指向プログラミングで用いられる概念定義を復習しておこう．

クラス (class): 同じふるまいをするオブジェクトのグループ．

図 1: HelloWorldDoc

インスタンス (instance): あるクラスの性質をもつひとつのオブジェクト．具体例．プログラムの実行は，クラスのインスタンスの集合が互いにメッセージをやりとりすることでなされる．

メッセージ (message): オブジェクトからオブジェクトへ伝えられる内容．

メソッド (method): クラスのメッセージを扱うための手段．

クラス変数 (class variable): クラス内で宣言され，そのクラスに属するすべてのインスタンスにより共有される変数．

インスタンス変数 (instance variable): クラス内で宣言され，そのクラスのインスタンスを作ると，そのインスタンスごとに記憶領域が用意される変数．

クラスメソッド (class method): インスタンスではなく，クラスに対して適用できるメソッド．

継承 (inheritance): すでに存在しているクラスに基づいて新しいクラスを定義するための手段．継承する親のクラスをスーパクラスという．ひとつのスーパクラスのみ継承できるシステムは単一継承といい，複数の親を継承できるシステムは多重継承という．

委託 (delegation): あるオブジェクトに対してメッセージが送られてきた場合に，そのオブジェクトの構成要素のひとつに対してメッセージを送ること．

カプセル化 (encapsulation): オブジェクトの構成要素を外からダイレクトに見えないようにすること．

抽象クラス (abstract class): メソッド名などを統一的に表現しておき，サブクラスで実際にそのメソッドの中身を定義してゆくための抽象化されたクラス．

コンストラクタ (constructor): インスタンス生成メソッド．Java では，クラス名 obj = new クラス名 (); として生成される．

ガベージコレクタ (GC: garbage collector): 参照されなくなったオブジェクト (インスタンス) を破棄する．Java では一般にインスタンスを破棄するデストラクタは無く，ごみ集め (GC(garbage collection)) により不要なオブジェクトが破棄される．

3 クラス宣言

3.1 Java プログラムの構成要素

Java プログラムは，パッケージ，クラス，インタフェース，メンバー，メソッドという 5 つの要素から構成される．最上位はパッケージであり，下位要素を上位要素のメンバーであると言う．階層構造は，次のようになっている．

```
パッケージ
  |
  ├パッケージ
  |
  └クラス／インタフェース
      |
      ├クラス／インタフェース
      |
      ├フィールド (メンバ変数)
      |
      └メソッド
```

クラスの内部の構成要素 (メンバー) には以下の 3 つの種類がある．

フィールド (field) クラスやそのインスタンスで保持するためのデータを定義する．

メソッド (method) 手続きを定義する．

ネストしたクラスとネストしたインタフェース そのクラスの中でフィールドやメソッドを共有できるクラスやインタフェースをさす．

3.2 修飾子

メンバーには修飾子を指定することができる．修飾子にはアクセス修飾子とそれ以外の修飾子がある．アクセス修飾子はアクセスに関する制約を規定するもので，private，protected，public，何も付けない，の四種類がある．

3.3 アクセス修飾子 (access modifier)

各種メンバーに対してアクセス制御を行うためのもので，以下のものがある．これによりカプセル化 (encapsulation) やデータ隠蔽 (data-hiding) を実現する．

private そのクラス自身からのみアクセス可能

何もない そのクラス自身と同じパッケージ内の他のクラスからもアクセス可能

protected そのクラスのサブクラス，同じパッケージのクラスおよびそのクラス自身からアクセス可能

public ぞのクラスにアクセス可能なクラスからアクセス可能．

3.4 クラス修飾子

クラス宣言の前に付く修飾子には，以下のものがある．

public 全ての場所からアクセス可能．

abstract サブクラスで実装しなければならない abstract メソッドを持つことを示す．インスタンスを作成することができない．

final サブクラスを作ることができないクラスを示す．abstract と同時には指定できない．

strictfp そのクラス内で定義されたすべての不動小数点計算が，CPU の処理系に依存せずに厳密に同じ演算結果となる．

3.5 フィールド修飾子

フィールド要素につく修飾子としては以下のものがある.

アクセス修飾子 **private, protected, public** のキーワードをつけたもの，何もつけないものがある.

static そのクラスでひとつの実体だけ存在するクラス変数.

final 初期化された後にその値を変更できない変数を示す.

transient オブジェクトのシリアライズすべきではないことを示す.

volatile その変数の最新の値を必ず用いるようにすることを示す.

3.6 メソッド修飾子

アクセス修飾子 **private, protected, public** のキーワードをつけたもの，何もつけないものがある.

abstract そのクラスの中で本体が定義されていないメソッド.

static クラスに対して呼び出されるメソッドを示す．クラスメソッドと呼ばれる．クラスメソッドでは this 参照がなく，そのクラスの static フィールドと他の static メソッドにしかアクセスできない．static メソッドは abstract にすることはできない.

final サブクラスでオーバライドできないメソッド.

synchronized 同じクラスの同一のインスタンスに作用させる synchronized メソッドと同期が取られることを示す.

native ハードウェア操作など Java 言語以外の言語で記述されたルーチンを呼び出すことを示す.

strictfp そのクラス内で定義されたすべての不動小数点計算で厳密な評価が行われる.

4 クラスの継承とメソッド

4.1 オーバーライドとオーバーロード

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.

4.2 コンストラクタ

200811-sample.zip のサンプルの中の，`SuperConstruct.java`，`SuperConstruct2.java`，`Construct.java` 等について，確認しておこう.

コンストラクタは，オブジェクトを生成するための特殊なメソッドのようなものだが，継承時の振る舞いは通常のメソッドとは異なる．メソッドのオーバーライドの考え方からすると違和感があるので気をつける必要がある.

- クラスのインスタンス生成時には，継承階層の最上位から順にコンストラクタが呼ばれる.
- スーパークラスのコンストラクタを明示的に (`super()` 等によって) 呼び出す場合以外は，自動的に引数の無いコンストラクタ (スーパークラスの) が呼び出される.

- 明示的にコンストラクタが定義されていない場合，引数の無いコンストラクタが自動的に定義される．
- 引数の有無にかかわらずコンストラクタを一種類でも定義すると，コンストラクタの自動定義は行われない．Construct2.java がエラーになるのは，class A に引数付きコンストラクタを定義したため，引数無しのコンストラクタが自動生成されず，子クラスのインスタンス生成の際に引数無しコンストラクタを呼ぼうとするとエラーになる．

5 抽象クラスと抽象メソッド

抽象メソッド (abstract method) をもつクラスを**抽象クラス (abstract class)** という．抽象クラスはインスタンスを生成できない不完全なクラスのようなもので，いくつかのサブクラスに共通の性質を定義するために使う．抽象メソッドは，手続きの本体がないメソッドで，サブクラスのメソッドにおいて，その手続き本体を定義する．

```
1
2  abstract class Benchmark {
3      abstract void benchmark();
4      public final long repeat(int count) {
5          long start = System.currentTimeMillis();
6          for (int i=0; i < count; i++)
7              benchmark();
8          return(System.currentTimeMillis() - start);
9      }
10 }
11
12 class LoopBenchmark extends Benchmark {
13     long count=0;
14     LoopBenchmark(int c) { count = c; }
15     void benchmark() {
16         for (int i=0; i < count; i++);
17     }
18 }
19
20 class FactBenchmark extends Benchmark {
21     int count=0;
22     FactBenchmark(int c) { count = c; }
23     int Fact(int n) {
24         return((n <= 1)? 1 : Fact(n-1)*n);
25     }
26     void benchmark() {
27         int f =        Fact(count);
28     }
29 }
30
31 class MethodBenchmark extends Benchmark {
32     void benchmark() {
33     }
34     public static void main(String[] args) {
35         int count = Integer.parseInt(args[0]);
36         long time = new MethodBenchmark().repeat(count);
37         System.out.println(count +
38                            " methods in " +
39                            time +
40                            " milliseconds");
41         time = new LoopBenchmark(1000000).repeat(count);
42         System.out.println("Loop(1000000) " +
43                            count +
44                            " methods in " +
45                            time +
46                            " milliseconds");
47         time = new FactBenchmark(10000).repeat(count);
48         System.out.println("Fact(10000) " +
49                            count +
50                            " methods in " +
51                            time + " milliseconds");
52     }
53 }
```

実行は，

```
% java MethodBenchmark 1000
```

のようにする．(main メソッドがあるクラスはどれか？引数は？)

6 Object クラス

Java のクラスシステムののルートクラスが Object クラスである．Object クラスがもつメソッドには以下のようなものがある．

public boolean equals(Object obj) レシーバオブジェクトと obj で参照されているオブジェクトの値が等しいどうかの同値性 (equivalence) を調べる．デフォルトでは，== を用いる同一性 (identity) を調べるメソッドになっている．

public int hashCode() 同一のオブジェクトは同一の整数値を返す．同一 (identity) であり同値 (equivalence) ではない．

protected Object clone() 複製を作る．複製を作るためのインタフェースとして Cloneable がある．複製は同値であるが同一ではない．

public final Class getClass() このオブジェクトのクラスをあらわす Class 型のオブジェクトを返す．すべての型に対して Class オブジェクトがある．

protected void finalize() throws Throwable ガーベッジコレクション時にオブジェクトに終了処理をさせる．オブジェクトの領域が回収される前に実行される手続きをクラスは実装することができる．

public String toString() オブジェクトの文字列表現を返す．

6.1 オブジェクトの同等性 equals

```
1  public class Test {
2      static int x,y;
3      static Integer w,z;
4      public static void main(String args[]) {
5          x = 10;
6  //        y = new Integer(10);  エラー
7  //        w = 20;   // エラーになる.
8          z = new Integer(20);
9          y = x + z.intValue();
10         System.out.println("( " + x + " , " + y + " )" );
11     }
12 }
```

```
1
2  public class Equivalence {
3    public static void main(String[] args) {
4      Integer n1 = new Integer(100);
5      Integer n2 = new Integer(100);
6      System.out.println(n1 == n2);
7      System.out.println(n1 != n2);
8    }
9  }
```

```
1  public class EqualsMethod {
2    public static void main(String[] args) {
3      Integer n1 = new Integer(100);
4      Integer n2 = new Integer(100);
5      System.out.println(n1.equals(n2));
6    }
7  }
```

```
1  class Value {
2    int i;
3  }
4
5  public class EqualsMethod2 {
6    public static void main(String[] args) {
7      Value v1 = new Value();
8      Value v2 = new Value();
9      v1.i = v2.i = 100;
10     System.out.println(v1.equals(v2));
11   }
12 }
```

これらのテストプログラムは，サンプルとしてダウンロードできるので，実行してみること．

6.2 Class クラス

オブジェクトに `getClass()` メソッドを送ると `Class` クラスのインスタンスを得ることができる．

```
// ClassDemo.java
class ClassDemo {
  public static void main(String args[]) {

    Integer obj = new Integer(8);
    Class cls = obj.getClass();
    if (obj instanceof Object)
      System.out.println("obj is instance of Object");
    if (obj instanceof Integer)
      System.out.println("obj is instance of Integer");
    if (cls instanceof Class)
      System.out.println("cls is instance of Class");
  }
}
```

実行すると次のようになる.

```
% java ClassDemo
class java.lang.Integer
obj is instance of Object
obj is instance of Integer
cls is instance of Class
```

7 親クラスの参照

7.1 super 変数

```
class That {
    protected String nm() {
        return "That";
    }
}

class More extends That {
    protected String nm() {
        return "More";
    }
    protected void printNM() {
        That sref = (That) this;
        System.out.println("this.nm() = " + this.nm());
        System.out.println("sref.nm() = " + sref.nm());
        System.out.println("super.nm() = " + super.nm());
    }
    public static void main(String argv[]) {
        (new More()).printNM();
    }
}
```

という具合に More.java を定義し，実行すると，次のようになる.

```
% java More
this.nm() = More
sref.nm() = More
super.nm() = That
```

以下は，何が表示されるか.

```
class That {
    protected String nm() {
        return "That";
    }
}

class More extends That {
    protected String nm() {
        return "More";
    }
    protected void printNM() {
      That sref = (That) this;

      System.out.println("this.nm() = " + this.nm());
      System.out.println("this.nm() = " + sref.nm());
      System.out.println("this.nm() = " + super.nm());
    }
}

class Super {
    static void main(String str[]) {
```

```
        More m = new More();
        m.printNM();
    }
}
```

7.2 super コンストラクタ

```
// SuperConstruct.java
class That {
    That() {
      System.out.println("That constructed");
    }
    That(String str){
      System.out.println("That constructed " + str);
    }
}
class More extends That {
    More() {
        super("abc");
        System.out.println("More constructed");
    }
}

public class SuperConstruct {
    public static void main(String str[]) {
        More m = new More();
    }
}
```

```
// SuperConstruct2.java
class This {
    This() {
        System.out.println("This constructed");
    }
}
class That extends This {
    That() {
        System.out.println("That constructed");
    }
    That(String str){
        System.out.println("That constructed " + str);
    }
}
class More extends That {
    More() {
        super("abc");
        System.out.println("More constructed");
    }
}
public class SuperConstruct2 {
    public static void main(String str[]) {
        More m = new More();
    }
}
```

7.3 コンストラクタの呼び出し

```
//Construct.java
class A {
    A() {System.out.println("A");}
}
class B extends A {
    B() {System.out.println("B");}
}
class C extends B {
    C() {System.out.println("C");}
}
public class Construct {
    public static void main(String str[]) {
        C m = new C();
    }
}
```

次はエラーがでる.

```
//Construct2.java
class A {
    A(String str) {System.out.println(str);}
```

```
}
class B extends A {
    B() {
        System.out.println("B");
    }
}
class C extends B {
    C() {System.out.println("C");}
}
public class Construct2 {
    public static void main(String str[]) {
        C m = new C();
    }
}
```

8 インタフェース

Java のクラスは，処理系の効率を優先して**単一継承 (single inheritance)** であるが，**多重継承 (multiple inheritance)** のように異なるクラスの特長を継承するような機能を実現するために，インスタンスを作れないクラスのようなものを宣言するための**インタフェース**宣言が可能である．

インタフェース宣言では，変数はもたず，定数の定義やメソッドの宣言を行う．クラスがそのインタフェース機能を継承する際には，スーパクラス宣言を行う **extends** の代わりに，**implements** を用いる．extends はひとつのクラスしか示せない (単一継承) が，implements には複数のインタフェースをカンマで区切って宣言できる．インタフェースを継承することを**実装 (implement)** するという．

インタフェースの中のメソッド宣言は，宣言だけで，メソッドの実体の定義はそれを実装しているクラスで行う．そのメソッド定義には **public** をつける．

8.1 インタフェース修飾子

インタフェース宣言の前に付く修飾子には，以下のものがある．

public これがなければそのパッケージの中からのみアクセス可能.

abstract すべてのインタフェースは暗黙に abstract．慣習として abstract 修飾子は常に省略される．

strictfp そのインタフェース内で定義されたすべての不動小数点計算で厳密な評価が行われる．

8.2 インタフェースのメンバー

インタフェースの内部の構成要素には以下の 3 つの種類がある．インタフェースのメンバは基本的に public なのでアクセス修飾子はとくにつけず，public も省略する．

定数 名前付き定数を宣言できる．暗黙として，public, static, final がついたデータとみなされる．必ず初期化子を持たなければならない．初期化の内ブランク final(blank final) は許されない.

メソッド (method) 実装を与えることができないメソッドで，abstract が暗黙として仮定され，abstract 修飾子は省略される．

ネストしたクラスとインタフェース

8.3 インタフェースの階層

インタフェースもクラスと同様階層性をもたせて拡張できる．extends を用い，複数のインタフェースを extends できる．

```
public interface SerializableRunnable
    extends java.io.Serializable, runnable {
}
```

8.4 システムのインタフェース

システムにあるインタフェースの例としては，次のようなものがある．

Cloneable この型のオブジェクトは複製が可能であるということを示すインタフェース．

Comparable この型のオブジェクトは比較のための順序を持っている．

Runnable この型のオブジェクトは独立した制御スレッドで実行できるメソッドをもっている．

Serializable この型のオブジェクトは別の仮想マシンへ転送するためや，永久に保存するためにオブジェクトバイトストリームへ書き出すことができる．

8.5 インタフェースの例

```
1  // Person.java
2  public class Person {
3    public String name;
4    protected int age;
5    private int phone;
6    Person() {
7    }
8    Person(String name, int age) {
9      this.name = name;
10     this.age = age;
11   }
12   public int getAge() {
13     return age;
14   }
15   public static void main (String[] args) {
16     Person person = new Person();
17     System.out.println("名前 = " + person.name);
18     System.out.println("年齢 = " + person.age);
19     System.out.println("年齢 = " + person.getAge());
20   }
21 }
```

インタフェースとして以下のようなものを定義したとする．これは，display() メソッドをもつインタフェース型を定義している．

```
1  // MyInterface.java
2  interface MyInterface {
3    void display();
4  }
```

```
1  // Parent.java
2  public class Parent extends Person {
3    Parent() {
4      super("Parent",30);
5    }
6    void display() {
7      System.out.println("Parent: " + name + " " + age);
8    }
9    void performClass(Parent p) {
10     p.display();
11   }
12   void performInterface(MyInterface m) {
13     m.display();
14   }
15
16   public static void main (String args[]) {
17     Person a = new Person();
18     Parent p = new Parent();
19     Child c = new Child();
20     Other o = new Other();
21     // p.performClass(a);   compile error
22     p.performClass(p);
23     p.performClass(c);
24     // p.performClass(o);   compile error
25     // p.performInterface(p); compile error
```

```
26      p.performInterface(c);
27      p.performInterface(o);
28    }
29  }
```

performClass は，Parent とそのサブクラスの Child のインスタンスを受け付ける．performInterface は，MyInterface をもつ Child と Other のインスタンスを受け付ける．

以下の 2 つの Child と Other を MyInterface インタフェースを実装したクラスとして定義する．それぞれに，display() メソッドの本体を定義する．

```
1  // Child.java
2  public class Child
3    extends Parent
4    implements MyInterface {
5    public void display() {
6      System.out.println("Child: " + age);
7    }
8  }
```

```
1  // Other.java
2  public class Other
3    implements MyInterface {
4    public void display() {
5      System.out.println("Other");
6    }
7  }
```

main メソッドをもったクラスを呼び出すことができる．

```
1  % java Person
2  名前 = null
3  年齢 = 0
4  年齢 = 0
5  count = 0, total = 48
```

```
1  % java Parent
2  Parent: Parent 30
3  Child: 30
4  Child: 30
5  Other
```

9 マルチスレッド

Java の特徴のひとつに，スレッドを利用できる点にある．

Java では，スレッドもオブジェクトの一つで，new オペレータで生成される．生成されただけでは実行がはじまらず，生成されたスレッドに start メソッドを適用するとスレッドは Java システムに内蔵されている実行スケジューラに渡される．実行スケジューラはスレッド実行の待ち行列にこのスレッドを置いて，待ち行列先頭のスレッドを順に実行する．

9.1 スレッドクラスを継承したクラスとして

Java ではスレッドは 2 つの方法で定義できる．

```
1  // ThreadTest.java
2
3  class EchoTread extends Thread {
4    int i = 0;
5    public void sleep(int a) {
6    }
7    public void run() {
8      while (i++ < 3) {
9        System.out.println("hello " + i + " in " + getName());
10       try {
11         sleep(1000,300);
12       } catch (InterruptedException e) {
13       }
14     }
```

```
  }
}

public class ThreadTest {
  public static void main(String argv[]){
    EchoTread th1 = new EchoTread();
    EchoTread th2 = new EchoTread();
    System.out.println("Thread test");
    th1.start();
    th2.start();
    System.out.println("end Start");
  }
}
```

Thread クラスでは，start メソッドが呼ばれると，実行キューに実行すべきスレッドが登録される．登録されたスレッドが実行状態になるとその Thread の run メソッドが呼ばれるという具合に処理が進む．

実行は以下のようになる．

```
% javac ThreadTest.java
5 java ThreadTest
Thread test
end Start
hello 1 in Thread-1
hello 1 in Thread-2
hello 2 in Thread-1
hello 2 in Thread-2
hello 3 in Thread-1
hello 3 in Thread-2
```

start メソッドが送られてから次のように，システムの gc(garbage collection) を呼び出すことによって，時間がかかる処理を挿入すると，実行はその下のようになる．

```
// ThreadTest2.java
import java.io.*;

class EchoTread extends Thread {
  int i = 0;
  public void sleep(int a) {
  }
  public void run() {
    while (i++ < 3) {
      System.out.println("hello " + i + " in "
                    + getName());
      try {
        sleep(1000,300);
      } catch (InterruptedException e) {
      }
    }
  }
}

public class ThreadTest2 {
    public static Runtime run = Runtime.getRuntime();
    public static void main(String argv[]){
        int i=0;
        EchoTread th1 = new EchoTread();
        EchoTread th2 = new EchoTread();
        System.out.println("Thread test2");
        th1.start();
        while (i++ < 100) {
            run.gc();
        }
        System.out.println("end Start1");
        th2.start();
        while (i++ < 100) {
            run.gc();
        }
        System.out.println("end Start2");
    }
}
```

```
% javac ThreadTest2.java
% java ThreadTest2
Thread test2
hello 1 in Thread-1
hello 2 in Thread-1
end Start1
end Start2
hello 1 in Thread-2
hello 3 in Thread-1
hello 2 in Thread-2
hello 3 in Thread-2
```

ごみ集め

いらなくなったデータを回収して再利用するための処理のことをごみ集め (GC: garbage collection) と呼ぶ．Java では自動ガーベッジコレクタが用意され，ヒープ領域に獲得したデータをユーザ自身が明示的に解放処理を実行する必要はない．複数の Thread が走っている場合には，それぞれの処理の最中にごみが発生するが，それらを自動的に回収するということが自動的に行われる．

9.2 Runnable インタフェースを実現したクラスとして

Thread クラスを継承するのではなく，Thread の実行に必要なメソッドを宣言している **Runnable インタフェース** が，java.lang パッケージの中に用意されており，これを実装することによってスレッド実行がなされるクラスを定義することができる．Runnable は，run() メソッドを宣言したインタフェースである．(インタフェースについては，**8** 節参照)

Runnable を実装するクラスのインスタンスを Thread コンストラクタに渡して，スレッドクラスのインスタンスを作る形となる．

Thread クラス自体も Runnable インタフェースを実装している．

```
// RunnableTest.java
class Echo implements Runnable {
  int i = 0;
  public void run() {
    while (i++ < 3) {
        System.out.println("hello " + i);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
        }
    }
  }
}

public class RunnableTest {
  public static void main(String argv[]){
      Echo e1 = new Echo();
      Thread th1 = new Thread(e1);
      Echo e2 = new Echo();
      Thread th2 = new Thread(e2);
      System.out.println("Echo test");
      th1.start();
      th2.start();
      System.out.println("end start");
  }
}
```

```
% javac RunnableTest.java
% java RunnableTest
Echo test
hello 1
end start
hello 1
hello 2
hello 2
hello 3
hello 3
```

9.3 スレッドの名前

スレッドには名前がついている．getName メソッドによってその名前を取り出すことができる．

```
// mainthread.java
class mainthread
{
    public static void main(String args[])
    {
        Thread thread =
            Thread.currentThread();

        System.out.println(
            "Main thread is named " +
            thread.getName());
    }
}
```

実行すると，次のように main スレッドは main という名前を持っていることがわかる．

```
% javac mainthread.java
% java mainthread
Main thread is named main
```

スレッドは生成した後から名前を付けることができる．

```
// setname.java
class setname
{
    public static void main(String args[])
    {
        Thread thread =
            Thread.currentThread();

        System.out.println(
            "Main thread's original name is " +
            thread.getName());

        thread.setName("The Main Thread");

        System.out.println(
            "Main thread's name is now " +
            thread.getName());
    }
}
```

実行すると以下のようになる．

```
% javac setname.java
% java setname
Main thread's original name is main
Main thread's name is now The Main Thread
```

10 課題

課題は，ikuo-soft3-08@jsk.t.u-tokyo.ac.jp 宛てに提出してください．メールのサブジェクトは，「soft3-kadai:20081201:xxxxx」(xxxxx は学生証番号) とする．締切は来週 12 月 8 日 (月) の講義時間前を目標としてください．その他，質問・要望・意見などがあれば，遠慮なく ikuo@jsk.t.u-tokyo.ac.jp にメールをください．

10.1 課題 1

JDK のデモプログラムを実行してみよ．コマンドラインから java コマンドで起動する方法と，demo/applets.html を開いてブラウザから見る方法がある．興味を持ったプログラムのソースファイルも見てみよう．いくつかのプログラムを実行し，内容などを調査し，わかったことやプログラムの概要等をまとめ，メールで提出すること．

10.2 アドバンスト課題 1

JNI の使い方を調べ，テストプログラムを書いて実行してみよ．必要なソースファイルと Makefile を作成し，make とすると必要な*.dll(Linux なら*.so) や*.class 等がコンパイルされるようにして，それらを tar.gz 等で束ねてメールで提出すること．