第7章　パッケージとライブラリ

他人の作成したクラス（やインタフェース）を利用したり、自分の作成したクラスを利用してもらうときには、クラス名やインタフェース名がぶつからないように気を付ける必要がある。

Java ではクラス名の衝突を避けるために＿＿＿＿＿＿（空欄 7.0.1）(package) という仕組みを用いる。また、パッケージの名前の付け方にも一定の慣習がある。

この章では、パッケージを JAR (Java archive) と呼ばれる Java のライブラリを配布するためのファイル形式と併せて紹介する。

7.1　パッケージ

すでに、Java のクラスは、正式にはパッケージという階層的な名前空間に属している（例えば、標準ライブラリの `Graphics` クラスの正式名は `java.awt.Graphics`、`PrintWriter` クラスの正式名は `java.io.PrintWriter` である）こと、パッケージに属するクラスを利用するときは、通常 import 文を使って短い名前を使うことなど、を学習してきた。

ここでは、パッケージ内にクラスを作成する方法を紹介する。

7.1.1　package 宣言

クラスが属するパッケージを宣言するには、ソースファイルの先頭に、`package` というキーワード、続けてパッケージ名とセミコロン（`;`）を書く。例えば、`foo.bar` というパッケージに `Baz` というクラスを定義するときは

```
package foo.bar;

public class Baz {
   ...
}
```

のようにする。これで、正式な名前（完全修飾名）が `foo.bar.Baz` のクラスが定義される。

package 宣言のないクラスは、＿＿＿＿＿＿＿＿＿＿（空欄 7.1.1）という特別なパッケージに属することになる。アプレットやサーブレットのように他のクラスから利用することがないクラスは、無名パッケージでも特段不都合はない。しかし、他のクラスから利用するクラスは、名前の衝突を避けるために無名でないパッケージに入れることが望ましい。

7.1.2　パッケージとディレクトリ階層の関係

クラス (.class) ファイルは、そのパッケージに対応するディレクトリに配置する。例えば、foo.bar.Baz クラスなら無名パッケージのクラスを置くディレクトリ（java コマンドで実行する場合はカレントディレクトリ、アプレットの場合は HTML ファイルが置かれているディレクトリ）の foo というディレクトリの bar というサブディレクトリに、 Baz.class というファイルを置く。（ただし、後述するようにクラスファイルを JAR ファイルというアーカイブファイルにまとめてしまう方法もある。）

main メソッドを持つクラス Main が、例えば、abc.xyz というパッケージにあるとき、これを実行するには　abc というディレクトリの xyz というサブディレクトリに、 Main.class を置き、abc ディレクトリの親ディレクトリ（Main.class のあるディレクトリからは 2 つ上のディレクトリ）で、

```
> java abc.xyz.Main
```

というコマンドを実行する必要がある。

javac コマンドも、java コマンドと同じ場所で、与えられた java のソースファイルをコンパイルするために必要なクラスファイルを探索する。例えば、XXX.java というソースファイルの中で、aaa.bbb.CCC というクラスが使われている場合、aaa というディレクトリの bbb というサブディレクトリの CCC.class というファイルを探す。クラスファイルが見付からなかった場合は、同じ場所で .java という拡張子を持つソースファイル（CCC.java）を見つけ、再帰的にコンパイルする。

つまり、foo.bar.Baz という完全修飾名を持つクラスをコンパイルするためには、Baz.java というソースファイルを、foo というディレクトリの中の bar というディレクトリに置き、

```
> javac foo\bar\Baz.java
```

というコマンドでコンパイルする。（UNIX 環境では \ の代わりに / になる。）

問 7.1.1 第 5 章で定義した Point クラスを point というパッケージに、ColorPoint クラスを point.ext というパッケージに 入れ、PointTest は無名パッケージのままにして、コンパイルし、実行せよ。

7.1.3　CLASSPATH 環境変数

java や javac コマンドがユーザー定義のクラスファイルを探す場所は、通常カレントディレクトリの下だが、＿＿＿＿＿＿（空欄 7.1.2）という環境変数で、クラスを探すディレクトリを指定することができる。探すディレクトリを複数指定する場合は、Windows の場合、セミコロン（;）で区切る。また、java コマンドに -classpath というオプションを与えても、コマンドごとにクラスを探すディレクトリを指定することができる。例えば、次のようにする。

```
> java -classpath .;C:\foo\bar;C:\aaa\bbb\ccc Baz
```

詳しくは java コマンドのマニュアルを参照すること。

7.1.4 パッケージ名の付け方の慣習

パッケージの名前を適当に付けたのでは、やはり衝突の可能性が出てくる。そこで Java では次のような約束ごとで、ドメイン名に基づいてパッケージ名をつけることにしている。

例えば example.com というドメイン名を持つ組織では＿＿＿＿＿＿＿＿（空欄 7.1.3）という接頭辞を持つパッケージ名を使用する。つまり、ドメイン名の単語順を逆にした接頭辞を用いる。それ以降のパッケージの階層名の規約は組織ごとに定めれば良い。ドメイン名にパッケージ名として使えない文字（例えばハイフン「-」）が入っている場合は、適宜アンダースコア「_」などに置き換える。

7.1.5 パッケージとアクセス指定

パッケージは情報隠蔽の単位としても使用される。

public や private というアクセス指定は、パッケージと関連はないが、アクセス指定には他に、protected と無指定（つまり、public, protected, private のいずれの指定もない場合）がある。

アクセス指定がないクラス、メソッド、フィールドは、＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿（空欄 7.1.4）からのみアクセス可能という意味になる。また、protected と指定されたメソッドやフィールドは、同一パッケージのコードと（他のパッケージに属するかもしれない）＿＿＿＿＿＿（空欄 7.1.5）のコードからのみアクセス可能という意味になる。

Q 7.1.2 foo.Bar クラス（foo パッケージの Bar クラス）, foo.BarTest クラス（foo パッケージの BarTest クラス）, BarTest クラス（無名パッケージの BarTest クラス）をそれぞれ次のように定義する

パス: foo/Bar.java

```
package foo;

public class Bar {
  public int x;
  private int y;
  int z;

  public Bar(int a, int b, int c) {
    x = a; y = b; z = c;
  }
}
```

パス: foo/BarTest.java

```
package foo;

public class BarTest {
  public static void main(String[] args) {
    Bar bar = new Bar(1, 2, 3);
    System.out.println(bar.x);   // い __
```

```
    System.out.println(bar.y);   // ろ ___
    System.out.println(bar.z);   // は ___
  }
}
```

パス: BarTest.java

```
import foo.Bar;

public class BarTest {
  public static void main(String[] args) {
    Bar bar = new Bar(1, 2, 3);
    System.out.println(bar.x);   // に ___
    System.out.println(bar.y);   // ほ ___
    System.out.println(bar.z);   // へ ___
  }
}
```

い〜へ、の行でコンパイル時にエラーにならない行には○を、エラーになる行には×をつけよ。

問 7.1.3 point.Point クラスの x や y などのフィールドを protected と指定し（あるいは無指定にし）、さらに、point.ext.ColorPoint クラスや PointTest クラスのメソッドからこれらのフィールドに直接アクセスを試みて、protected と無アクセス指定の効果を確認せよ。

7.2　JAR ファイルの作成

複数個の Java のクラスファイル（と、場合によってはそこから使用する画像や音声ファイルなど）を、一つにまとめて扱うことができる。Java は JAR 形式というファイル形式を用いる。JAR 形式の拡張子は.jar である。

JAR ファイルの作成には、jar というコマンドを用いる。例えばカレントディレクトリ以下のすべてのファイルを親ディレクトリの nantoka.jar という JAR ファイルにまとめるには、以下のようにする。

```
> jar -cvf ..\nantoka.jar .
```

カレントディレクトリ（.）以下を JAR にまとめるときに、生成される jar ファイル自身をカレントディレクトリ以下に置いてしまうと、うまく生成できないので注意が必要である。-C というオプションで、jar コマンドの作業ディレクトリを変更することもできる。例えば、aaa というディレクトリ以下のすべてのファイルを kantoka.jar という JAR ファイルにまとめるには、以下のようにすることもできる。

```
> jar -cvf kantoka.jar -C aaa .
```

これは、aaa に移動してから、

```
> jar -cvf ..\kantoka.jar .
```

を実行するのと同等である。

JAR 形式は実は ZIP 形式そのものなので、拡張子を .zip に書き換えると、標準的な解凍ツールで中身を見ることができる。ただし、META-INF/というディレクトリ内には、jar コマンドが作成するメタ情報が格納されている。

JAR ファイルは CLASSPATH 環境変数や java コマンドや javac コマンドの -classpath オプションの中にディレクトリと同じように指定することができる。例えば、

```
> java -classpath .;nantoka.jar XXX
```

のようにする。

アプレットの場合は、applet タグの archive という属性で、使用する JAR ファイルを指定することができる。複数の JAR ファイルを指定するときは、コンマ (,) で区切る。例えば、

```
<applet code="XXX.class" width="150" height="50"
        archive="nantoka.jar,kantoka.jar">
</applet>
```

のように書くことができる。

問 7.2.1 point.Point クラスと point.ext.ColorPoint クラスを JAR ファイルにアーカイブし、java コマンドの-classpath オプションでこの JAR ファイルを指定して PointTest クラスを実行せよ。

7.3 署名とマニフェスト

2014 年 1 月の Java 7 update 51 から、新しいセキュリティ要件が導入され、その結果、Java アプレットに信頼できる認証局のコード署名が必要となった。

ただし、Java 7 update 51 以降でもブラウザーの例外サイト・リストに URL を追加することで、特定の URL に置かれたアプレットを従来のセキュリティ要件で実行することができる [1]。その場合でも、アプレットからローカルファイルやネットワークにアクセスするためにはアプレットにコード署名が必要である。（この場合、信頼できる認証局の証明でなく、自己証明—いわゆるオレオレ証明—でも可能である。）

この節では、JAR ファイルに（オレオレ証明による）署名を追加する方法を説明する。また、ついでに、Java 7 update 51 以降に対応したマニフェストを追加する方法も説明する。

[1] http://www.java.com/ja/download/faq/exception_sitelist.xml 参照

7.3.1　署名

JAR ファイルに署名をするためには、まず、証明書を作成する必要がある。証明書の作成は、keytool コマンドを用いる。

```
> keytool -genkey -alias mykey
```

いくつかの質問が表示されるので、これに答える必要がある。以下に香川大学工学部に所属する、“さぬきたろう” さんの応答例を挙げる。(↙ は改行を表す。)

```
Enter keystore password: abcdefgh↙
Re-enter new password: abcdefgh↙
What is your first and last name?
  [Unknown]:  Taro Sanuki↙
Taro Sanuki
What is the name of your organizational unit?
  [Unknown]:  Faculty of Engineering↙
Facalty of Engineering
What is the name of your organization?
  [Unknown]:  Kagawa University↙
Kagawa University
What is the name of your City or Locality?
  [Unknown]:  Takamatsu↙
Takamatsu
What is the name of your State or Province?
  [Unknown]:  Kagawa↙
Kagawa
What is the two-letter country code for this unit?
  [Unknown]:  JP↙
JP
Is CN=Taro Sanuki, OU=Faculty of Engineering, O=Kagawa University, L=Takama
  [no]:  yes↙
yes

Enter key password for <mykey>
 (RETURN if same as keystore password):  ↙
```

これで、*mykey* という名前の証明書が、キーストア（デフォルトでは、ホームディレクトリの.keystore という名前のファイルだが、-keystore オプションで変更可能）に作成される。一つの証明書で複数の JAR ファイルに署名できるので、keytool コマンドは、一度だけ実行しておけば良い。

信頼できる認証局にこの証明書に署名してもらうには、さらに作業が必要になるが、ここではその説明を割愛する。

作成した証明書を使って、JAR ファイルに署名をするには jarsigner コマンドを用いる。

```
> jarsigner nantoka.jar mykey
```

mykey は、keytool コマンドの -alias オプションで指定した証明書の名前である。パスワードの入力を促されるので、keytool コマンドで指定したパスワード（上の応答例では *abcdefgh*）を入力する。

7.3.2 （参考）マニフェスト

自己証明書ではなく、信頼できる認証局から署名してもらった証明書を用いると、例外サイト・リストにサイトを追加しなくてもアプレットが実行できるようになる。その場合、JAR ファイルに Permissions, Codebase というマニフェスト属性を記述する必要がある。この節ではマニフェスト属性の追加方法を説明する。

次のような追加するマニフェストの内容を持つファイルを作成する。

ファイル: MANIFEST.MF

```
Permissions: sandbox
Codebase:    www.example.com
```

この例は、sandbox で実行される（つまりファイルやネットワークにアクセスしない）アプレットの場合である。もし、ローカルファイルやネットワークにアクセスするアプレットなら、sandbox の部分を all-permissions に変更する。*www.example.com* の部分は、アプレットを配備するサーバーのホスト名に変更する。

マニフェストを追加するには jar コマンドを実行するときに、-m オプションで上記のファイルを指定する。

```
> jar -cvmf MANIFEST.MF kantoka.jar -C aaa .
```

あるいは、

```
> jar -cvmf ..\MANIFEST.MF ..\kantoka.jar .
```

すでに作成済みの JAR ファイルに後からマニフェストを追加するには、-c ではなく、-u オプションを使用する。

```
> jar -uvmf MANIFEST.MF kantoka.jar
```

これで JAR ファイルにマニフェスト属性が追加される。

キーワード　パッケージ、package 宣言、無名パッケージ、CLASSPATH 環境変数、-classpath オプション、protected, JAR 形式、jar コマンド、archive 属性、署名、jarsigner コマンド、