独習Java第3版

7.5　パッケージ
7.6　クラスパス
7.7　importステートメント
7.8　アクセス制御とパッケージ

パッケージとpackageステートメント

- パッケージとは
  - クラスやインターフェイスの集まりのこと
- 次のpackageステートメントを指定すれば
  1つのソースファイルに含まれるクラスとインターフェイスを特定のパッケージに割り当てられる

  package *packageName* ;
  //*packageName*はパッケージ名

packageステートメント

- packageステートメントを指定しないと
  ソースファイルのクラスとインターフェイスは
  規定のパッケージに割り当てられる
- 1つのファイル内で指定できる
  packageステートメントは1つ
- パッケージ名はJava命名規則に従った識別名

packageステートメントの使用例

- packageステートメントの例

  package enfineering ;

- 複数のパッケージを階層構造にできる
  - 複数のパッケージ名をピリオド(.)で区切って指定する

  package engineering.electrical.signals ;

packageステートメント使用時の注意

- パッケージの階層に対応した
  ディレクトリ階層を構成しなくてはいけない
  - 先ほどの例の場合、パッケージのソースファイルは
    engineering¥electrical¥signals
    というディレクトリの下になければならない

Javaのパッケージ

- Javaのクラスライブラリはパッケージで体系化
  - パッケージの一例

パッケージ	説明
java.applet	アプレットを作成するためのクラス群
java.awt	グラフィカルユーザーインターフェイスを作成する AWT（Abstract Window Toolkit）のクラス群
java.awt.event	AWTコンポーネントからのイベントを処理するクラス群
java.io	入出力動作を扱うクラス群
java.lang	Javaの中核的な機能を提供するクラス群
java.net	ネットワーク機能を提供するクラス群
java.util	ユーティリティ機能を提供するクラス群

使用例

PackageDemo.java, A.java, B.java
という3つのソースファイルがあるとする

```
//p¥PackageDemo.javaのソースコード
package p ;
class PackageDemo {
  public static void main( String args[] ) {
    A a = new A() ;
    a.a1() ;
    B b = new B() ;
    b.b1() ;
  }
}
```

実行結果
a1
b1

```
//p¥A.javaのソースコード
package p ;
class A {
  void a1() {
    System.out.println( “a1” ) ;
  }
}
```

```
//p¥B.javaのソースコード
package p ;
class B {
  void b1() {
    System.out.println( “b1” ) ;
  }
}
```

パッケージ構築と実行

- 先ほどの例のパッケージを構築するには
  pの親ディレクトリで次のコマンドを実行
  - javac p¥*.java
- 実行するにはpの親ディレクトリで
  次のコマンドを実行
  - java p.PackageDemo
- 実行時にはパッケージ名とクラス名を併せた
  完全修飾クラス名で指定する必要がある

クラスパス

- クラスパス（CLASSPATH）とは
  - JDK/JREツールに対して.classファイルの検索先を指定するための環境変数
  - ディレクトリや.jarファイル、.zipファイルが順番に格納されている（拡張子が.jarや.zipのファイルは複数のファイルを格納するためのアーカイブ（保存用ファイル））
- パッケージを利用する場合はCLASSPATH環境変数を正しく設定することが重要

環境変数の設定方法

- Windowsプラットフォームでの
  環境変数の設定方法
  - 以下のコマンド（例）を実行
    set CLASSPATH=c:¥project1;c:¥project2;c:¥project3

- クラスパスの確認方法
  - 以下のコマンドを実行
    echo %CLASSPATH%
    （これで環境変数の現在の設定が表示される）

クラスの捜索

- 先ほどの例のクラスパスを設定したときに
  abc.Testというクラスを見つける場合、
  JDK/JREツールは次のファイルを順番に探す

  1．カレントディレクトリ下のabc¥Test.classファイル
  2．c:¥project1¥abc¥Test.classファイル
  3．c:¥project2¥abc¥Test.classファイル
  4．c:¥project3¥abc¥Test.classファイル
  5．Javaクラスライブラリ

- 最初に見つかったファイルを使用し、
  最後まで見つからなかったらエラーとなる

importステートメント

- importステートメントとは
  - 別パッケージのクラスやインターフェイスに
    完全修飾名を指定しなくても
    アクセスできるようにするもの

importステートメントの構文

- 構文は次の2種類
  - import *fullyQualifiedTypeName ;*
  - import *packageName.* ;*
- 構文の特徴
  - 1つ目はfullyQualifiedTypeNameに
    指定したクラス・インターフェイスに
    短い名前で直接アクセスできる
  - 2つ目はpackageNameに指定したパッケージ内の
    型を短い名前で直接参照できる

使用例

- 次のように指定すると
  - import java.awt.event.MouseEvent ;

    →完全修飾名のjava.awt.event.MouseEventを
    MouseEventだけで代用できる

  - import java.awt.event.*;

    →java.awt.event内の全ての型を
    短い名前で参照できる

サブパッケージについて

- java.langパッケージはコンパイラによって
  自動で全てのソースファイルに取り込まれる
  - java.langパッケージに含まれる型はいつでも
    短い名前で参照できる
- 1つのパッケージを取り込んでも
  そのサブパッケージは取り込まれない

  import java.awt.*; → java.awt.event内の
  パブリックな型は取り込まれない

  ↓サブパッケージを取り込むために指定する

  import java.awt.event.*;

使用例

- Demo.java, R.java, S.java　という3つのファイルがあるとする

```
//q¥Demo.java
package q ;
import r.* ;
import s.* ;
class Demo {
  public static void main( String args[] ) {
    R r = new R() ;     r.r1() ;
    S s = new S() ;     s.s1() ;
  }
}
```

```
//r¥R.java
package r ;
public class R {
  public void r1() {
    System.out.println( “r1” ) ;
  }
}
```

```
//s¥S.java
package r ;
public class S {
  public void s1() {
    System.out.println( “s1” ) ;
  }
}
```

構築と実行

- 先ほどの3つのパッケージを構築方法するには次の3つのコマンドを実行する
  - javac q¥*.java
  - javac r¥*.java
  - javac s¥*.java

  (javac q¥*.java javac r¥*.java javac s¥*.java
  と1つまとめることもできる)
- 実行には次のコマンドを実行
  - java q.Demo

実行結果:
r1
s1

アクセス制限とパッケージ

- public, protected, privateの意味

キーワード	説明
public	全てのクラスにアクセスを許可する
protected	同じパッケージ内のコード、または別の パッケージ内のサブクラスにのみアクセスを許可
private	同じクラス内のコードにのみアクセスを許可

- これらのキーワードは同時に複数指定できない
- １つも指定しなければ既定で同じパッケージ内のコードにしかアクセスが許可されない

例(1/3)

- e¥E.java, f¥F.java, g¥G.javaの3つのソースコードについて

```
//e¥E.java
package e;
public class E {
  public int e1 = 11;
  protected int e2 = 22;
  private int e3 = 33;
}
```

```
//f¥F.java
package f;
import e.*;
public class F extends E {
  public void display() {
  System.out.println(e1);
  System.out.println(e2);
  // System.out.println(e3);
  }
}
```

Eのインスタンスを作成

サブクラスなので
protectedメンバにアクセスできる

privateメンバなので
アクセスできない

publicメンバなので
アクセスできる

例(2/3)

```
//g¥G.java
package g;
import e.*;
public class G {
  public void display() {
    E e = new E();
    System.out.println(e.e1);
    // System.out.println(e.e2);
    // System.out.println(e.e3);
  }
}
```

Eのインスタンスを作成

publicメンバなので
アクセスできる

サブクラスではないので
protectedメンバにアクセスできない

privateメンバなので
アクセスできない

例(3/3)

- 次に示すコードはh¥ProtectedDemo.javaのソースコードで、f.Fクラスとg.Gクラスを使用

```
//h¥ProtectedDemo.java
package h;
import f.F;
import g.G;
class ProtectedDemo {
  public static void main(String args[]) {
    F f = new F();
    f.display();
    G g = new G();
    g.display();
  }
}
```

別パッケージ内のサブクラスをテストする

別パッケージ内の非サブクラスをテストする

構築と実行

- パッケージの構築には次のコマンドを実行

```
javac e¥*.java
javac f¥*.java
javac g¥*.java
javac h¥*.java
```

or

```
javac  e¥*.java  f¥*.java  g¥*.java  h¥*.java
```

- 実行するには親ディレクトリで次のコマンドを実行
  - java h.ProtectedDemo

```
実行結果:
11
22
11
```

課題

- コマンドラインから受け取った引数（整数）が
  3の倍数であるかを調べるクラス、
  5の倍数であるかを調べるクラスを作成せよ
  ただし、この2つのクラスは
  2つソースファイルに別々に定義し、
  baisuという1つのパッケージにまとめよ