2011 年度 メディアプロジェクト演習 1「Java プログラミング」

担当: 前田亮，森勢将雅，健山智子，木村朝子

1. 演習の目的

本演習では，プログラミング言語 Java を用いて，オブジェクト指向プログラミングを修得する．また，Web アプリケーションサーバで使用するサーブレットを実装する技術を身につける．演習を行うにあたり必要なクラスライブラリの情報は，以下のドキュメントあるいは参考文献を参考にすること．

Java™ Platform, Standard Edition 6 API 仕様:
`http://java.sun.com/javase/ja/6/docs/ja/api/index.html`

本演習の進捗の目安は以下の通りである．(2.9 節～2.13 節，および 3.8 節以降はオプションである)．

第一週:2.1～2.6 第二週:2.7～2.8 第三週:3.1～3.5 第四週:3.6～3.7 第五週:3.7

オプションの節の見出しには右上に「†（ダガー）」が付けられている．これらは発展的な内容である．

2. オブジェクト指向と Java 言語

オブジェクト指向(OO: Object Oriented)とは，実世界の「もの」を抽象化した「オブジェクト」を構成単位として，プログラムを記述していく枠組みである．本演習で取り扱う Java 言語はオブジェクト指向言語（OOPL: Object-Oriented Programming Language)の一つである．Java の基本的な式，制御構造，演算子などの構文は C 言語に類似しているが，オブジェクト指向に基づいている点が大きく異なる．

オブジェクト指向における主要な概念として，**「クラスとインスタンス」「フィールドとメソッド」「継承」**が挙げられる．

クラスとは構造体のようなものであり，ユーザが独自にプログラム中で定義することができる型である．**インスタンス**とはクラスを型として実行時に生成されるデータであり，変数によって参照される．実世界になぞらえるなら，クラスが鋳型，インスタンスが製品といった関係にある．

クラスが構造体と異なる点は，そのクラスのインスタンスからのみ実行できる固有の関数を定義できることである．これを**メソッド**と呼ぶ．メソッドは C の関数のようにどこからでも呼ぶことができるわけではなく，常にインスタンスを表す変数を通して呼ぶことしかできない．(ただしクラスを通して呼ぶメソッドも存在する)．一方，クラスにおいて定義された変数を**フィールド**と呼ぶ．これは構造体のメンバに類似する．
実世界に存在する「もの」は通常，属性と機能を持っている．たとえば自動車であれば「車体の色」「排気量」といった属性に加えて，「加速する」「減速する」という機能がある．プログラミングにこの考え方を持ち込み，属性(フィールド)や機能(メソッド) を持つクラスを多数定義し，それらを連携させる形でプログラムを書いていくのがオブジェクト指向言語の特徴である．

継承とはひとつのクラスのフィールドとメソッドを継承させつつ，新たなフィールドとメソッドを付け加えた新しいクラスを作成する技法である．すなわち最初に一般的なクラス(スーパークラス)を作っておき，それに様々な機能を付け加えることでより個別的なクラス(サブクラス)を作り出す． スーパークラスとサブクラスの関係はいわば「自動車」と「トラック」の関係に類似している．「自動車」に具体的な属性と機能を付け加えて得られるのが「トラック」であるが，トラックは自動車の持つ属性と機能をすべて継承している．

これらの概念によってプログラムの書きやすさや読みやすさ，再利用性が高まっているのがオブジェクト指向言語の利点である．本演習では Java プログラミングを通してオブジェクト指向の考え方に習熟することも目的としている．

2.1 ソースファイルとクラスファイル

まず，ターミナル（端末）を開き，以下のスクリプトを実行して初期設定を行う．

```
% /kyozai/amaeda/2011mp1/scripts/javaSetup.sh
```

このスクリプトは Java を使用するために必要な設定を各自のホームディレクトリにある~/.cshrc に書き込んでいる．これによってログイン時に Java のコンパイルやサーブレットの実行に必要な環境変数が自動的に設定されるようになる．なお，javaSetup.sh の実行後，新しくターミナルを開いた場合は.cshrc の設定が自動で読み込まれるが，**はじめて変更した直後には読み込まれないので，以下のように入力することで現在のターミナルに読み込ませる．**

```
% source ~/.cshrc
```

設定が正しくできているかを確認するには以下を実行する．もし「servlet-api.jar」を含む文字列が表示されれば正しく設定できている．

```
% echo $CLASSPATH
```

> 設定内容を見るには cat ~/.cshrc を実行すればよい．環境変数 PATH を設定するのは，最新の Java 開発環境を使用するためである．また，servlet-api.jar はサーブレットが使用するクラスファイルをまとめたアーカイブファイルである．拡張子".jar"は Java Archive を表す．

Java プログラムは以下に示す手順で実行する．

1. ソースファイルを Emacs 等のエディタで作成する．
2. ソースファイルを javac コマンドでコンパイルする．
3. java コマンドを使い，Java 実行環境で実行する．

ソースファイルとは，Java ソースプログラムを指し，拡張子".java"を持つテキストファイルである．C 言語のプログラム作成と同様に，Emacs や Windows の「メモ帳」などのテキストエディタを用いて作成すればよい．
ソースファイルを Java コンパイラ（javac）でコンパイルすると，クラスやインタフェースごとにクラスファイルと呼ばれるバイナリファイルが作成される．クラスファイルは，拡張子".class"を持つ．クラスファイルはバイトコード（bytecode）とも呼ばれ，プラットフォームに依存しないという特徴を持つ．ソースファイル Sample.java のコンパイルは，以下のコマンドにより行う．（なお，以下では Linux 上での Java の実行環境を前提としている．）

```
% javac Sample.java
```

バイトコードは，Java 仮想機械（JVM: Java Virtual Machine）と呼ばれるインタプリタ上で解釈実行する．インタプリタは，バイトコードを各計算機に向けの機械語に逐次変換して，その処理を実行する．
クラスファイル Sample.class の実行は，以下のコマンドにより行う．（javac ではないので注意）

```
% java Sample
```

なお，本章におけるサンプルプログラムのソース（.java ファイル）は以下のディレクトリに置かれている．

```
/kyozai/amaeda/2010mp1/java
```

mkdir コマンドで本演習用のディレクトリ（たとえば mp1）を作り，cp コマンドを使ってコピーするとよい．

```
% mkdir ~/mp1
% cp /kyozai/amaeda/2011mp1/java/*.java ~/mp1
```

2.2 Java プログラム

簡単な Java プログラム Sample1.java を示す.

```
/* Sample1 */
public class Sample1 {  // クラスの宣言
    public static void main(String[] args) {
        System.out.println("Enjoy Java programming.");  // 文字列を画面に出力
    }
}
```

予約語 class によりクラス Sample1 を定義する. public は, このクラスを他のパッケージに対して公開することを表す. main は Java プログラムにおいて特殊なメソッド(手続き)であり, java コマンドの引数として指定したクラスの実行開始点である. java Sample1 と実行した場合は, クラス Sample1 の main から実行が始まる. System.out.println()の System.out は標準出力を意味し, println()は文字列を画面に表示するメソッドである. また, //からその行の終りまでの文字, /*と*/にはさまれる文字はコメントとなる. Sample1 のプログラムをコンパイルし, 実行すると, 以下のようになる.

```
% javac Sample1.java
% java Sample1
Enjoy Java programming.
```

Java ソースファイルを作成する際の注意として, ファイル内の公開(public)クラスの名前(Sample1)とファイル名(Sample1.java)は一致させる必要がある.

2.3 クラスとインスタンス

クラス(class)とはデータ(属性)とそれに対する処理(操作)をひとまとめにして定義した抽象データ型である. Java プログラムは 1 つ以上のクラスで構成され, 属性をフィールド(field), 操作をメソッド(method)と呼ぶ. 実行時には, クラスからオブジェクトを生成し, 生成したオブジェクトが動作することで処理を行う. 生成したオブジェクトのことを, クラスに対してインスタンス(instance)と呼ぶ. つまり, クラスとはインスタンスを生成する雛形(鋳型)のようなものである. インスタンスを生成し, 処理を実行する Java プログラムを Sample2 に示す.

```
/* Sample2 */
public class Sample2 {
    public static void main(String[] args) {

        /* Book クラスからインスタンスを生成 */
        Book myBook = new Book("The Java Tutorial", 55);
        System.out.println("My book title = " + myBook.getTitle());
        System.out.println("My book title = " + myBook.title);
        System.out.println("My book price = " + myBook.getPrice());

        /* Book クラスからインスタンスを生成 */
        Book yourBook = new Book("The Java Programming Language", 40);
        System.out.println("Your book title = " + yourBook.getTitle());
        System.out.println("Your book price = " + yourBook.getPrice());
    }
}

class Book {  // クラス Book の宣言
    String title;       // 題名
    private int price;  // 価格(ドル)
```

```
    Book(String t, int p) {  // コンストラクタ
        title = t;  // 題名の設定
        price = p;  // 価格の設定
    }

    public String getTitle() {  // 題名の取得
        return title;
    }

    public int getPrice() {  // 価格の取得
        return price;
    }
}
```

Book は本を表すクラスである. 属性として本の題名と価格を持ち, それぞれ title という文字列型(String)および price という整数型 (int)のインスタンス変数で参照する. さらに, Book は本の題名を取得するメソッド getTitle()と本の価格を取得するメソッド getPrice()を持つ. クラス名と同じメソッドは, コンストラクタ(constructor)である. コンストラクタはインスタンス生成時に自動的に実行されるため, インスタンスの初期化に用いる.

クラス Sample2 では, myBook と yourBook という Book 型のインスタンス変数を宣言し, `new` 演算子によりクラス Book からインスタンスを 2 つ生成する. 生成したインスタンスは, それぞれ変数 myBook と yourBook を用いて扱う. たとえば, `myBook.getTitle()` により myBook のメソッド getTitle()を呼び出したり, `myBook.title` により myBook のインスタンス変数 title の値を参照したりすることができる. Sample2 のプログラムを実行すると, 以下のようになる.

```
% java Sample2
My book title = The Java Tutorial
My book title = The Java Tutorial
My book price = 55
Your book title = The Java Programming Language
Your book price = 40
```

Java では, インスタンスではなくクラスに属するフィールドやメソッドを定義することができる. これらは, 宣言や定義において修飾子 `static` を付け, それぞれクラス変数(class variable)およびクラスメソッド(class method)と呼ぶ. たとえば, `System.out` の out はクラス変数, `Math.sqrt()` の sqrt()はクラスメソッドである. クラス変数やクラスメソッドは, インスタンスを生成しなくともクラス名を指定するだけで参照や呼び出しが可能である. 特に, クラス変数は同一のクラスから生成したインスタンス群で値を共有したい場合や定数を保持するために使う.

【基本課題 1】
`price` フィールドの値はドル単位である. 価格を円($1 = ￥92.70)で返すメソッド `double getYenPrice()` を追加し, `main` メソッドの中でドルでの価格, 円での価格の両方を出力するように変更せよ.

2.4 変数と型

Java において変数はすべて型を持つ. Java のデータ型には, 次に示す基本型 (primitive type)と参照型(reference type)がある. 基本型には, 論理型の `boolean`(`true` あるいは `false` をとる), 整数型の `byte`, `short`, `int`, `long`, 浮動小数点型の `float`, `double`, Unicode 文字型の `char` がある. 配列, クラス, インタフェースは参照型であり, その変数の値はすべてポインタとなる.

さらに, Java の変数はスコープ(scope)を持つ. スコープとは, その変数を単純名(名前のみ)で参照できる範囲である. インスタンス変数のスコープは宣言されたクラスであり, メソッド引数のスコープは宣言されたメソッドである. ローカル変数のスコープは, それが宣言された位置から始まり, それが宣言されたブロックの終りまでとなる.

【基本課題 2】

Book クラスに出版年(西暦)を保持するフィールド(int 型)を追加せよ. さらに, そのフィールドにアクセスするためのメソッド void setYear(int year) と int getYear()を作成し, main メソッドの中でインスタンス myBook に対して出版年の設定と出力を行うように変更せよ(本の出版年は任意に設定してよい).

【基本課題 3】

題名, 価格, 出版年を一度に設定してインスタンスを生成するためのコンストラクタを追加せよ.

2.5 演算子と文

Java の主な演算子には, 算術演算子(+, -, *, /, %, ++, --, ?:), 代入演算子(=, +=, -=, *=, /=, %=), 比較演算子(<, <=, >, >=, ==, !=), 論理演算子(&&, ||, !)がある. また, Java は, 条件文(if), 繰り返し文(while, for), break 文, continue 文, return 文を備える. これらの記法と意味は, C 言語と同じである.

これら以外の重要な演算子として, インスタンスを生成する new, 型を変換するキャスト(型を括弧で括る), インスタンスの型を判定する instanceof, 配列の生成やアクセスに用いる[], 限定名を形成するドット(.)がある.

【基本課題 4】

価格が x より低いかどうかを判定するメソッド boolean isLess(int x)を追加せよ. これを用いて, main メソッドの中で 50 ドルより安い場合のみ, 本の情報を出力させるようにせよ.

2.6 文字列

配列を用いて文字列を実現する C 言語と異なり, Java では文字列を扱うクラス String と StringBuffer が用意されている. String から生成したインスタンスでは, 格納されている文字列を変更することはできない. これに対して, StringBuffer から生成したインスタンスでは, 文字の追加や挿入ができるようになっている. ここで, String のインスタンスは頻繁に利用されるため, new 演算子を使わなくとも, 次のようにインスタンスの生成が可能である.

```
String str = "Java";  // String str = new String("Java");と同じ
```

String から生成したインスタンスは参照オブジェクトであるため, 文字列の比較には==演算子ではなく, クラス Object のメソッド equals()を用いる. たとえば String 型の a と b を比較する if 文は以下のようになる.

```
if(a.equals(b)){ }
```

また, 文字列の先頭が一致するかどうかを確かめる startsWith(), 末尾が一致するかを確かめる endsWith()などのメソッドもある. たとえば a の先頭が"The"であるかどうかを確かめることは以下の if 文によって可能になる.

```
if(a.startsWith("the")){ }
```

2.7 配列

Java では, 複数のオブジェクトをまとめて扱うために配列とコレクションクラスが用意されている.

配列(array)とは, 同じ型の複数の変数(要素)を並べたものである. 配列は []を用いて宣言し, i 番目の要素にアクセスする場合は, 添字 i を[]に入れて指定する. Java では, 配列もクラス (オブジェクト)であるため, new 演算子を用いて, 要素数に応じた領域を確保しなければならない. 配列の利用例を示す.

```
int numbers[];           // int[] numbers でもよい
numbers = new int[10];   // 10 個分の領域を確保(添字 0～9)
```

```
numbers[5] = 1;                                          // 5 番目の要素に 1 を代入
System.out.println("5th = " + numbers[5]);               // 5 番目の要素を表示
System.out.println("length = " + numbers.length);       // 配列の長さを表示

int max = 10;
String strings[] = new String[max];  // max 個分の領域を確保(添字 0～max-1)
```

配列の長さを取得したい場合は，配列インスタンスのフィールド length を用いる．また，配列の要素数は生成時に変数により指定可能である．

2.8 継承と合成

オブジェクト指向で構築されたクラスやインスタンスを拡張する方法には，継承と合成がある．

継承（inheritance）とは，既存クラスのインスタンス変数やメソッドを引き継いで新しいクラスを定義することである．継承を用いると，すでに存在するクラスに機能を追加したり，一部の機能を書き換えるだけで新しい機能を持つクラスを作成することができる．継承を用いてクラスを拡張するプログラムを Sample3 に示す．

```
/* Sample3 */
public class Sample3 {
    public static void main(String[] args) {
        Book book = new Book("The Java Tutorial", 55);
        System.out.println("Title = " + book.getTitle());
        System.out.println("Price = " + book.getPrice());

        OnlineBook obook = new OnlineBook(
        "The Java Virtual Machine Specification", 0,
        "http://java.sun.com/docs/books/vmspec/index.html");
        System.out.println("Title = " + obook.getTitle());
        System.out.println("Price = " + obook.getPrice());
        System.out.println("Website = " + obook.getWebsite());
    }
}

class Book {
    ... // Sample2 と同じ
}

class OnlineBook extends Book {  // クラス Book を継承
    public String website;  // URL

    OnlineBook(String t, int p, String website) {
        super(t, p);              // スーパークラスのコンストラクタの呼び出し
        this.website = website;  // URL の設定
    }

    public String getWebsite() {  // URL の取得
        return website;
    }

    public String getTitle() {
        return "Online: " + title;
    }
}
```

予約語 extends は継承の宣言を表す. つまり, クラス OnlineBook はクラス Book を拡張している. Book を親クラスあるいはスーパークラス（super-class）, OnlineBook を子クラスあるいはサブクラス（sub-class）と呼ぶ. Java では, すべてのクラスは Object（java.lang.Object）のサブクラスであり, extends を省略した場合は, Object の直接の子クラスになる.

Sample3 において, OnlineBook から生成したインスタンスは, OnlineBook のインスタンス変数 website, メソッド getWebsite()に加えて, Book の title, price, getTitle(), getPrice()が利用できる. ただし, コンストラクタは継承されない. 継承により引き継がれる機能は, 直接の親クラスだけでなく, 間接的に継承している親クラスの祖先からも引き継がれる（よって, OnlineBook は Object の機能も引き継ぐ）. さらに, OnlineBook ではメソッド getTitle()を再定義することで, Book の機能の一部を修正している. このように, 親クラスに存在するメソッドと同じシグニチャを持つメソッドを子クラスで再定義することをメソッドのオーバーライド（override）という. シグニチャとは, メソッドの名前, 引数の型と並びを指す.

ここで, super は親クラス, this は自分自身（のインスタンス）を指す. また, super() および this() はコンストラクタ内部でのみ利用可能でそれぞれ親クラスおよび自分自身の別のコンストラクタを呼び出す際に用いる.

Sample3 のプログラムを実行すると, 次のようになる.

```
% java Sample3
Title = The Java Tutorial
Price = 55
Title = Online: The Java Virtual Machine Specification
Price = 0
Website = http://java.sun.com/docs/books/vmspec/index.html
```

合成（composition）とは, 別のオブジェクトを参照あるいは包含することで機能を拡張することである. 新規クラスへのメソッド呼び出しを既存クラスのメソッドに委譲（delegetion）あるいは転送（forwarding）することで, 既存クラスの機能の一部を再利用し, さらに独自の機能を追加することができる. 合成により機能を拡張するプログラムを Sample4 に示す.

```
/* Sample4 */
public class Sample4 {
    public static void main(String[] args) {
        BookInLibrary book = new BookInLibrary("The Java Tutorial", 45);
        book.setBorrower("Taro Ritsumei");
        System.out.println("Title = " + book.getTitle());
        System.out.println("Borrower = " + book.getBorrower());
    }
}

class Book {
    ... // Sample2 と同じ
}

class BookInLibrary {
    private Book book;         // Book クラスのインスタンスへの参照
    private String borrower;   // 借用者

    BookInLibrary(String t, int p) {
        book = new Book(t, p);  // Book クラスのインスタンスの生成
        borrower = null;
    }

    public void setBorrower(String name) {  // 借用者の設定
        borrower = name;
    }
```

```
    public String getBorrower() {   // 借用者の取得
        return borrower;
    }

    public String getTitle() {    // 題名の取得
        return book.getTitle();   // インスタンス book へ転送
    }
}
```

クラス BookInLibrary のコンストラクタは，クラス Book のインスタンスを生成し，その参照先をインスタンス変数 book に格納する．BookInLibrary は，独自にインスタンス変数 borrower およびそれにアクセスするメソッド setBorrower()と getBorrower()を持つ．また，getTitle()への呼び出しをインスタンス book のメソッド getTitle()に転送することで，その機能を再利用している．Sample4 のプログラムを実行すると，次のようになる．

```
% java Sample4
Title = The Java Tutorial
Borrower = Taro Ritsumei
```

【基本課題 5】

OnlineBook クラスを継承した OnlineMagazine クラスを作成し，出版月と出版日を保持するフィールド（int 型）を追加せよ．これらに登録を行うメソッド void setMonth(int month)，void setDate(int date)，および出版年／出版月／出版日をまとめて出力するメソッド String getPublicationDate()を作成せよ．出力のフォーマットはどのようなものでもよい．

2.9 抽象クラスと多態性†

継承を用いると，類似した機能を持つクラスの共通部分をまとめて親クラスで定義しておき，それぞれ異なる機能だけをその子クラスに定義することができる．その際，親クラスではメソッドの実装を定義せずに，そのシグニチャだけを規定しておきたいことがある．このような要求に対して，Java では抽象クラス（abstract class）が用意されている．実装が定義されていないシグニチャだけのメソッドを抽象メソッド（abstract method）という．抽象クラスは実装を持たないメソッドを含むため，直接インスタンスを生成することはできない．抽象クラスの子クラスからインスタンスを生成するためには，その子クラスで抽象メソッドをすべて実装する必要がある．抽象クラスを含むプログラムを Sample5 に示す．

```
/* Sample5 */
public class Sample5 {
    public static void main(String[] args) {
        Publication pub;

        pub = new Article("Java Compiler", "IEEE Software");
        pub.showInfo();

        pub = new TechReport("Java Programming Explained",
                             "Ritsumeikan Univ.");
        pub.showInfo();
    }
}

abstract class Publication {  // 抽象クラスの宣言
    protected String title;  // 題名

    Publication(String t) {
        title = t;
```

```
    }

    abstract public void showInfo();  // 抽象メソッドの宣言
}

class Article extends Publication {  // クラス Publication を継承
    private String journal;  // 雑誌名

    Article(String t, String j) {
        super(t);
        journal = j;
    }

    public void showInfo() {  // showInfo の()実装
        System.out.println(title + " in " + journal);
    }
}

class TechReport extends Publication {  // クラス Publication を継承
    private String institution;  // 発行機関

    TechReport(String t, String i) {
        super(t);
        institution = i;
    }

    public void showInfo() {  // showInfo の()実装
        System.out.println(title + " by " + institution);
    }
}
```

修飾子 abstract は抽象クラスおよび抽象メソッドに付加する. Publication は抽象クラス, showInfo()は抽象メソッドである. Publication を継承しているクラス Article および TechReport では, メソッド showInfo()を実装しているため, インスタンスが生成可能である. また, クラス Sample5 では, クラス Article および TehcReport から生成したインスタンスを Publication 型のインスタンス変数 pub で保持している. Java では, このように子クラスのインスタンスを親クラスのインスタンスとして保持することが可能である.

Sample5 のプログラムを実行すると, 次のようになる.

```
% java Sample5
Java Compiler in IEEE Software
Java Programming Explained by Ritsumeikan Univ.
```

この実行例をみると, メソッド pub.showInfo()の呼び出しに対して, Article か TechReport のインスタンスのどちらかが選択され, そのインスタンスのメソッド showInfo()が実行されていることがわかる. このように, インスタンス変数 pub に格納されている値の型に応じて実行時に呼び出しメソッドが決定されることを動的束縛(dynamic binding)と呼ぶ. また, showInfo()のように 1 つのメソッド呼び出しで, 異なる振る舞いを実現することを多態性(polymorphism, 多相性)と呼ぶ.

ここで, メソッドのオーバーロード(overload)も多態性の一種である. オーバーロードとは, シグニチャの異なるメソッドを定義することを指す. Java では, シグニチャが異なれば, 同一クラス内でも同じ名前を持つメソッドを多重に定義することができる. たとえば, 次の 2 つのメソッドは, オーバーロードの関係にあり, 別々に扱われる.

```
public int getPrice() { ... }
public int getPrice(int rate) { ... }
```

Java では, 抽象メソッド(と定数)だけで構成されているクラスをインタフェースと呼ぶ. インタフェースの利用例を以下に示す.

```
interface Printable {
    public void print();
}
class Article extends Publication implements Printable {
    public void print() { ... }
}
```

インタフェースは抽象メソッドしか持たないため，修飾子 abstract は不要である．インタフェースを実装（継承）する際には，extends ではなく implements を用いる．インタフェースはメソッドの実装を持たずにそのシグニチャを規定でき，さらにインタフェースを継承するクラスにメソッドの実装を強要するため，フレームワークの構築によく使われる．

2.10 パッケージとアクセス制御†

大規模なプログラムを開発する際には，複数の人間で効率的にクラスを作成したい．また，Java ではクラスごとにクラスファイル（拡張子が .class のファイル）を作成するため，複数のクラスを同じファイルに記述するよりもクラスごとに Java ソースファイルを作成する方が，不必要なコンパイルを削減できる．たとえば，Sample2 において，クラス Book をクラス Sample2 以外のクラスから利用する場合は，Book のコードをソースファイル Book.java に分けて記述するとよい．これにより，Sample3.java では Book のコードをあらためて記述する必要はなく，Sample2 と Book, Sample3 と Book は独立に開発することができる．

また，大規模なプログラムを作成する場合は，ほかの人が作成したクラス（あるいはインタフェース）を再利用する場合が多い．さまざまな人が作成したクラスを混在させて利用する場合に名前の衝突を避けるため，Java にはパッケージという仕組みが組み込まれている．ソースファイルの先頭に package 文が存在すると，そのファイルで定義したクラスやインタフェースはそのパッケージ（名前空間， name-space）に属することになる．たとえば，次のように記述すると，

```
package ritsumei;
class File {
    ...
}
```

クラス File はパッケージ ritsumei に属し，完全限定名（fully-qualified name）は， ritsumei.File となる．よって，J2SE のクラスライブラリに含まれる java.io.File（java.io パッケージ）と区別される．

ソースファイルに package 文が存在しない場合は，そのファイルで定義されているクラスは無名パッケージに属することになる．同じパッケージに属するクラスどうしは，特にパッケージ名を指定しなくともお互いに利用可能である．異なるパッケージに属するクラスを利用する場合は，パッケージ名を含む完全限定名で指定する．たとえば，パッケージ ritsumei に含まれるクラス File を指定する場合は，ritsumei.File とすればよい．また，import 文を用いると，完全限定名を指定しなくともクラス名だけでクラスを指定可能となる．よって，次のコードにおいて，File は ritsumei.File と同じものを指す．

```
import ritsumei;
class Directory {
    ritsumei.File file1;   // 完全限定名で指定
    java.io.File file2;    // ritsumei.File とは異なるクラス
    File file3;            // ritsumei.File と同じクラス
}
```

特定のパッケージの直下に属するクラス群をまとめて指定する場合は，次のように"*"を用いて import 文を記述することができる．

```
import java.io.*;
class Directory {
    ...
}
```

このように記述することで，パッケージ java.io に属するクラスをクラス名だけで指定できる．ここで，java.lang パッケージについては，特に import 文を記述しなくとも自動的に取り込まれている．

パッケージに関連して，Java では情報隠蔽（information hiding）を実現するために，アクセスを制御する（access control）修飾子がいくつか用意されている．修飾子とアクセス制御の関係を以下にまとめる．

`public`: 異なるパッケージのクラスから，そのクラス，変数，メソッドにアクセス可能
`protected`: 同じパッケージとサブクラスから，その変数，メソッドにアクセス可能
なし: 同じパッケージから，そのクラス，変数，メソッドにアクセス可能
`private`: クラス内からのみ，その変数，メソッドにアクセス可能

【発展課題 1】

現在までに作成した各種クラスをひとつのパッケージ `bookinfo` にまとめよ．

2.11 例外処理†

C 言語によるプログラミングでは，コンパイル時には判定できないエラーを起こす可能性がある文の実行に対して，その直後に条件文を用いてエラー検査を行う．この方法では，エラー検査のコードがプログラムに散らばり再利用を妨げる．これに対して，Java は正常な処理とエラー処理を分離する例外（exception）という仕組みを持つ．Java の例外処理は，例外を送出する部分と送出された例外を捕捉して決められた処理を実行する部分からなる．例外の送出と捕捉を含むプログラムを Sample6 に示す．

```
/* Sample6 */
public class Sample6 {
    public static void main(String[] args) {
        int numbers[] = new int[10];    // 配列の宣言と確保

        try {                                                  // ここから
            numbers[20] = 0;  //  エラー発生
        } catch (ArrayIndexOutOfBoundsException e) {  // ここまでの例外を捕捉
            System.out.println("Exception occured: " + e);  // 例外処理
        }

        System.out.println("Finish!");  // 終了メッセージの表示
    }
}
```

ArrayIndexOutOfBoundsException は，配列の添字が範囲を越えた際に送出される例外である．Java では，エラーが発生する可能性のあるコードを try 文と catch 文にはさむことで例外を捕捉し，catch 文のブロックに例外処理を記述する．捕捉された例外は，catch ブロックの実行後に消滅する．Sample6 のプログラムを実行すると，次のようになる．

```
% java Sample6
Exception occured: java.lang.ArrayIndexOutOfBoundsException
Finish!
```

エラーが発生しているにもかかわらず，例外処理が実行され，プログラムは正常に終了している．

さらに，Java では，独自の例外を作成することや throw 文を用いて明示的に例外を送出することができる．Sample7 は，例外 NoTitleException を独自に定義するプログラムである（クラス Exception のサブクラスとして

NoTitleException を定義している). クラス Manual のメソッド setTitle()の処理中に例外 NoTitleException が送出される可能性があるため, throws 文を用いてその例外を送出する可能性があることを宣言する.

```
/* Sample7 */
public class Sample7 {
    public static void main(String[] args) {
        try {
            Manual manual = new Manual();
            manual.setTitle("");  // 文字数 0 の題名を設定

        } catch (NoTitleException e) {  // 例外の捕捉
            System.out.println("No title");
        }
    }
}

class Manual {
    String title;

    public void setTitle(String t) throws NoTitleException {  // 例外の送出
        if (t.length() == 0) {
            throw new NoTitleException();  // 例外の送出
        }
        title = t;
    }
}

class NoTitleException extends Exception {  //  例外の定義
}
```

2.12 入出力処理†

Java ではデバイスへの入出力をストリーム(stream)で扱う. これらは, java.io パッケージに含まれている. ストリームは 4 つのクラス InputStream, OutputStream, Reader, Writer を基礎とする. InputStream と OutputStream はバイト指向ストリーム(バイトデータの入出力に用いる)であり, Reader と Writer は文字指向ストリーム(テキストデータの入出力に用いる)である.

ファイルから文字列を入力する場合は, ファイル名を指定して FileReader オブジェクトを生成する. ファイルに文字列を出力する場合は, FileWriter オブジェクトを生成する. ファイル入出力のプログラムを Sample8 に示す.

```
/* Sample8 */
import java.io.*;

public class Sample8 {
    public static void main(String[] args) {
        String filename = "test8.txt";

        try {
            PrintWriter pw = new PrintWriter(
              new BufferedWriter(  // バッファを挿入
              new FileWriter(filename)));  // 出力ストリームの作成
            pw.println("Java");
            pw.println("Programming");  // 文字列を書き出す
            pw.println("Language");
```

```
            pw.close();  // 出力ストリームのクローズ

        } catch (IOException e) {
            System.out.println("Cannot write: " + filename);
            System.exit(1);  // プログラムの終了
        }

        try {
            BufferedReader br = new BufferedReader(  // バッファを挿入
              new FileReader(filename));  // 入力ストリームの作成

            String line;
            while ((line = br.readLine()) != null) {  // 文字列を読み込む
                System.out.println(line);
            }
            br.close();  // 入力ストリームのクローズ

        } catch (FileNotFoundException e) {
            System.out.println("File Not Found: " + filename);

        } catch (IOException e) {
            System.out.println("Cannot read: " + filename);
            System.exit(1);  // プログラムの終了
        }
    }
}
```

この Sample では，ファイルの入出力を効率的に行うためにクラス BufferedReader によりバッファリングを行っている．Sample8 のプログラムを実行すると，次のようになる．

```
% java Sample8
Java
Programming
Language
% cat test8.txt
Java
Programming
Language
```

2.13 コレクションクラス†

コレクションクラス（collection class）とは，任意のオブジェクトをまとめて格納するためのクラスである．Vector, ArrayList, HashSet, TreeSet などのクラスと，List や Set などのインタフェースが，java.util パッケージに含まれている．これらを利用するにあたってはプログラムの冒頭に import java.util.*; と記載すること．配列を用いた場合，要素数を一度指定すると，その大きさを変えることはできない．これに対して，コレクションクラスを用いた場合は，その大きさを必要に応じて変更することができる．コレクションクラスの利用例（Vector）を示す．

```
import java.util.*;
 (略)
Vector<String> v = new Vector<String>();     // Vector のインスタンスの生成
v.add(new String("Java"));                   // 0 番目の要素を追加
v.add(new String("Programming"));            // 1 番目の要素を追加
v.add(new String("Langugae"));               // 2 番目の要素を追加
```

```
System.out.println(" size = " + v.size());  // 要素数の表示
String first = v.get(1);              // 1 番目の要素を取得(0 から始まる)
v.remove(1);                                  // 1 番目の要素を削除
```

Vector<String>と書くのは「ジェネリックス」と呼ばれ，Vector クラスにどのようなクラスのインスタンスを入れるかをあらかじめ指定するためのものである．また，Java にはコレクションクラス内部の要素に順番にアクセスするために反復子（インタフェース Iterator)が用意されている．Iterator を用いた各要素への反復アクセスの例を以下に示す．

```
Vector<String> v = new Vector<String>();    // Vector のインスタンスの生成
v.add(new String("Java"));
v.add(new String("Programming"));
v.add(new String("Langugae"));

Iterator it = v.iterator();           // v に対する反復子の取得
while (it.hasNext()) {                // 要素が存在する場合はループ内部へ
  String str = (String)it.next();     // 要素を取り出す
  System.out.println(str);
}
```

反復子を用いることで，コレクションクラスの型にとらわれずに，各要素にアクセス可能となる．たとえば，クラス Vector をクラス TreeSet（要素の格納に木構造を用いる）に変更した場合でも，インスタンスの生成以外はコードを変更する必要はない．

【発展課題 2】

複数の著者の名前を保持するフィールド authors（String 型の配列で，要素数は最大 10 とする）を追加せよ．さらに，複数の著者の名前を一度に設定するメソッド void setAuthors(Vector names)を追加せよ．引数 names（java.util.Vector クラス）の各要素は著者名を表す文字列とする．このメソッドが正常に動作していることを main メソッドで確認せよ．

【発展課題 3】

著者の数を取得するメソッド int getAuthorsNum()と，著者らの名前を取得するメソッド String getAuthors()を追加せよ．getAuthors()では，著者が 2 人以上の場合は著者名の間をスラッシュ(/)で区切ること．このメソッドが正常に動作していることを main メソッドで確認せよ．

【発展課題 4】

最大 10 冊の書籍を登録できるクラス BookSeries（シリーズ本）を作成せよ．ただし以下の条件を満たすこと．

- 各書籍の情報は Book クラスのインスタンスの配列として保持すること．
- Book クラスのインスタンスの登録を一冊ずつ行うためのメソッドを実装すること．
- シリーズに含まれる書籍タイトルの一覧を表示するメソッドを実装すること．
- 配列の上限を超えて書籍の登録が行われた場合，ArrayIndexOutOfBoundsException を catch し，エラーメッセージを表示させよ．

【発展課題 5】

上記の BookSeries クラスを配列ではなくコレクションを使った形で実装せよ．なお，ArrayIndexOutOfBoundsException に対する対応は行わなくてよい．

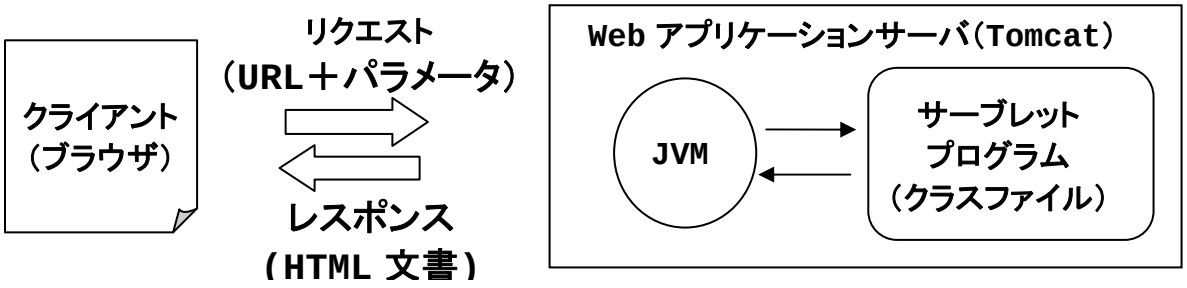
3. サーブレット

サーブレットとは Web アプリケーションサーバ上で Java プログラムを実行し，Web ブラウザ（クライアント）との情報のやりとりを行わせる仕組みである．

あらかじめ完全な形で記述された HTML 文書を静的なページと呼ぶのに対し，アクセス時にプログラムによって生成される HTML 文書を動的なページと呼ぶ．サーブレットは Java によって HTML 文書を生成する仕組みである．

どのような HTML を作成するかをプログラムによって決められるため，静的な Web ページと比べて格段にインタラクティブな Web サイトを作成することができる．また，JPEG 等の画像ファイルを生成させることも可能である．

なお，Web アプリケーションサーバとは Web に対して公開され，プログラムの実行によって高度な機能を提供するサーバの総称である．サーブレットは Web アプリケーションのひとつの形態と位置づけられる．



本演習では代表的な Web アプリケーションサーバである Tomcat を利用する．

また，Web アプリケーションサーバで Java を利用する仕組みにはもうひとつ代表的なものとして JSP (JavaServer Pages) がある．こちらは HTML ファイルの中に Java のソースを埋め込んで記述し，自動的なコンパイルによってサーブレットの機能を提供するものであるが，本演習では扱わない．

3.1 Tomcat の初期設定

Tomcat は Apache Software Foundation によって開発されている Web アプリケーションサーバであり，サーブレットを提供する手段として代表的なものである．

Tomcat のディレクトリ内にサーブレットのクラスファイルを作成しておくことで，クライアント（Web ブラウザ）からリクエスト（アクセス）があった際，JVM によって HTML 文書が生成され，レスポンスとして返される．

Tomcat の利用にあたっては，Tomcat 本体を動かす実行ファイルと自作の Web アプリケーション（サーブレット）を分けて設置することができる．その場合，環境変数 `CATALINA_HOME` によって実行ファイルやライブラリを置いたディレクトリを指定し，環境変数 `CATALINA_BASE` によって Web アプリケーションを置いたディレクトリを指定する．これによって個人のホームディレクトリの下に Web アプリケーションを作成し，Tomcat の実行ファイルは共通のものを利用することができる．（`CATALINA_HOME` と `CATALINA_BASE` を同一のディレクトリに設定することももちろん可能である）．

以下のスクリプトを実行して Tomcat の初期設定を行う．

```
% /kyozai/amaeda/2011mp1/scripts/tomcatSetup.sh
```

ホームディレクトリ下に作成される`~/mytomcat` というディレクトリがサーブレットの開発に使用するディレクトリである．実際のサーブレットのクラスファイルは `~/mytomcat/webapps/mysite/WEB-INF/classes` に設置する．本演習ではサーブレット用の Java プログラムの作成とコンパイルはこの `classes` ディレクトリにて行うこと．

また，新しいサーブレットを作成した際には`~/mytomcat/webapps/mysite/WEB-INF/web.xml` に `servlet` 要素を追加する必要があるので注意すること．

「mysite」は今回作成するサーブレットの集まりにつけた名称であり，実際は自由につけてよい．ただしその場合はプログラム中で mysite となっている箇所をすべてそのディレクトリ名に合わせること．「mytomcat」も同様に自由に付けて良い名前であるが，webapps，WEB-INF，classes 等は変えることができない．

以上によって初期設定は完了である．スクリプト tomcatSetup.sh によって実行される内容を見るには cat コマンドや less コマンドを使えば見ることができる．

```
% cat /kyozai/amaeda/2011mp1/scripts/tomcatSetup.sh
```

3.2 Tomcat の起動

Tomcat は以下のように起動する．

```
% $CATALINA_HOME/bin/startup.sh
```

なお，Tomcat のディレクトリ内で startup.sh を実行した場合，Tomcat のトップページは見えるがサーブレットが起動しないという不具合が生じるので，以下のようにホームディレクトリに移動してから起動するのがよい．すなわち以下のようになる．

```
% cd
% $CATALINA_HOME/bin/startup.sh
```

二つのターミナルを開いておき，javac でのコンパイル用と shutdown.sh / startup.sh での Tomcat の再起動用に使い分けると便利である．

3.3 Web アプリケーションへのブラウザからのアクセス

起動後，ブラウザに以下の URL を入力することで Tomcat のトップページにアクセスできる．猫のイラストの描かれたページが表示されれば正常に起動している．

```
http://localhost:8080
```

コロンの後の 8080 はポート番号を表す．Web サーバは通常，80 番をポートとして使用しており，一般的な Web ページの URL では省略されているが，80 番以外のポートを使用するサーバにアクセスする場合はコロンの後に番号を記述することになる．（ブラウザのアドレスバーに http://www.google.co.jp:80 と入力してアクセスしてみよ．http://www.google.co.jp:81 ならどうか．）．Tomcat はデフォルトで 8080 番ポートを使用するが，設定ファイルによって変更することも可能である．

また，World Wide Web に対して Web アプリケーションサーバを公開した場合，たとえばドメインが www.mydomain.net であれば，以下のようにアクセスできる．

```
http://www.mydomain.net:8080
```

本演習では実験室外に向けての Web アプリケーションサーバの公開は行わないが，実験室内では localhost の代わりにホスト名を入れることで他の計算機上の Tomcat にアクセスできる．

情報処理演習室（RAINBOW 環境）ではたとえば以下のようにアクセスできる．

```
http://pri1s007:8080
```

学科実験室では以下のようになる．ホスト名は計算機本体にシールで貼られている．

```
http://ccjpt052:8080
```

ホスト名の代わりに IP アドレスで指定しても良い．

```
http://172.24.14.29:8080
```

3.4 Tomcat の停止

Tomcat を停止する場合は以下のように行う. **サーブレットのプログラムを書き換えてコンパイルを行った後，結果を反映させるためには再起動が必要になる.** これは停止したのちに起動することで行う.

```
% $CATALINA_HOME/bin/shutdown.sh
```

上記のコマンドで止められない場合(エラーメッセージが表示される場合),

```
% ps –aux | grep java
```

を用いて java で動作しているプロセスを見つけ, その中のひとつのプロセス番号を使い,

```
% kill プロセス番号
```

という形で終了させる.

自作のサーブレットが正常に動作しない場合, ~/mytomcat/logs/ディレクトリの中に出力されるログファイルをエラーの原因を確かめるのに利用できる.

なお, 自分のホームディレクトリの残り容量に余裕がある場合, 以下のようにサーブレットのサンプル集をコピーしてきてもよい. (残り容量は du コマンドで調べることができる. 単位は MB. 最大が 50MB.)

```
% cp -r $CATALINA_HOME/webapps/examples/ ~/mytomcat/webapps/
```

これらのサンプルは Tomcat のトップページ(http://localhost:8080 で表示されるページ) の左下にある「Servlet Examples」というリンクを辿ることで見ることができる.

3.5 サーブレットの作成

サーブレットは javax.servlet パッケージで定義された HttpServlet クラスを継承することで作成する. ここでは Myserv というクラス名で自作のサーブレットを作成する.
サーブレットは~/mytomcat/webapps/mysite/WEB-INF/classes の下に設置する.
以下のファイル Myserv.java を classes ディレクトリの下に作成し, javac でコンパイルする. **本章においては SampleX. java といったファイル名で作成しないので注意すること.**
本章におけるサンプルプログラムのソースは以下のディレクトリに置かれている.

```
/kyozai/amaeda/2011mp1/mytomcat/webapps/mysite/WEB-INF/classes/
```

しかし初期設定時にスクリプト tomcatSetup.sh を実行したことによって, 各自のホームディレクトリ下にある ~/mytomcat/webapps/mysite/WEB-INF/classes/ にもコピーされている.

```
/* Sample9 Myserv.java */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Myserv extends HttpServlet {
```

```
    public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("<html><body><h1>Hello World!</h1></body></html>");
    }
}
```

今までに書いてきた Java プログラム同様，ファイル名と同じ名前のクラスがプログラム中で定義されていなくてはならない．この場合，public class Myserv という風に Myserv クラスが定義されているため，ファイル名は Myserv.java でなくてはならない．HttpServlet はスーパークラスであり，これを継承する形で Myserv クラスが作られている．

クライアント(ブラウザ)からサーバにリクエスト(アクセス)があった時，Myserv クラスの doGet メソッドあるいは doPost メソッドが実行される．そのため doGet メソッドあるいは doPost メソッドを定義しておくことで，アクセス時の挙動を指定できる．

> *GET と POST は HTTP による通信で使用される代表的なメソッドである．(なお，ここで言う「メソッド」は Java とは関係なく，HTTP におけるリクエストの形式を指す)．たとえばブラウザに URL を打ち込んでアクセスする時のように，何のパラメータも付いていない要求はブラウザからサーバに GET メソッドとして送られる．一方，フォームを使い，パラメータを付けて情報要求を行う場合，GET と POST のいずれかを使用できる．GET の場合は URL の中に `http://www.site.com/page?param=value` のようにパラメータの名称と値が「?」に続いて明記される．POST の場合はフォームで送信した内容は URL 内に現れない．*

クライアントへのレスポンスは response メソッドを通して行われる．MyServ.java では doGet メソッドの中で response.getWriter() を用いてレスポンスに書き込むためのストリーム out を作成し，その println メソッドを実行することでクライアントに送られるテキストを作成している．

なお，新しくサーブレットを作成した場合，~/mytomcat/webapps/mysite/WEB-INF/web.xml を編集し，以下のように servlet 要素と servlet-mapping 要素を追加する必要がある．追加した部分を太字で示した．これによってクラスファイル名とサーブレット名（クライアントからアクセスする時のパス）の対応付けを行う．（/kyozai/amaeda/2011mp1/web.xml からコピーしてきたものにはすでに登録されている）．

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
<servlet>
      <servlet-name>Myserv</servlet-name>
      <servlet-class>Myserv</servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>Myserv</servlet-name>
      <url-pattern>/Myserv</url-pattern>
</servlet-mapping>
</web-app>
```

url-pattern 要素はスラッシュ(/)から始める点を注意すること．このように設定を行った後，Tomcat を再起動する．javac によってクラスファイルを新たに作成した場合，あるいは修正を行った場合，Tomcat の再起動が必要である．

ブラウザからは以下の URL でアクセスできる.

```
http://localhost:8080/mysite/Myserv
```

3.6 HTML のフォームからサーブレットを呼び出す

サーブレットからのレスポンスはブラウザに URL を直接入力するだけでなく, HTML のフォームを利用して取得することもできる.

クライアントから送られた要求(リクエスト)は doGet メソッドあるいは doPost メソッドに第一引数(HttpServletResponse クラス)として渡される. リクエストに含まれるパラメータの値は第一引数(以下のプログラムでは request)の getParameter メソッドの戻り値として読み出すことができる.

ReadForm.java はフォームからの入力を読み込み, 出力に利用するプログラムである.

```
/* Sample10 ReadForm.java */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ReadForm extends HttpServlet {

    //----------------------
    /* HTTP の GET メソッドで呼び出される関数 */
    public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

      response.setContentType("text/html;charset=utf-8");
      PrintWriter out = response.getWriter();

      out.println("<html><head><meta http-equiv=\"Content-Type\"
content=\"text/html; charset=utf-8\"></head><body>");
      out.println("<form action=\"ReadForm\" method=\"post\">");
      out.println("名前を入力してください:");
      out.println("<input type=\"text\" name=\"username\">");
      out.println("<input type=\"submit\" value=\"送信\">");
      out.println("</form></body></html>");
    }

    //----------------------
    /* HTTP の POST メソッドで呼び出される関数 */
    public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

      request.setCharacterEncoding("UTF-8");
      String username = request.getParameter("username");

      response.setContentType("text/html;charset=utf-8");
      PrintWriter out = response.getWriter();

      out.println("<html><head><meta http-equiv=\"Content-Type\"
content=\"text/html; charset=utf-8\"></head><body><h1>こんにちは, " + username + "
さん</h1></body></html>");
    }
}
```

なお，下から 5 行目～6 行目の `http-equiv=\"Content-Type\"` と `content=\"text/html; charset=utf-8\">` の間には改行を入れず，スペースで区切るようにすること．

`request.setCharacterEncoding("UTF-8");` はパラメータの文字コードを UTF-8 であるとみなすための命令である．また，`response.setContentType("text/html;charset=utf-8");`は出力の文字コードを UTF-8 に設定するための命令である．

ブラウザから以下の URL にアクセスし，フォームに入力して「送信」ボタンを押すと，送られたパラメータの内容を踏まえた形でサーブレットが実行される．

`http://localhost:8080/mysite/ReadForm`

ブラウザに上記の URL を入力してこのページにアクセスした場合，何のパラメータも与えられていないため，ブラウザからサーバに GET メソッドのリクエストが送られる．そのためサーバ ReadForm クラスの doGet メソッドを実行する．その結果としてブラウザに送られるのは以下の HTML ファイルである．

```
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"></head>
<body>
<form action=”ReadForm” method=”post”>
<input type=”text” name=”username”>
<input type=”submit” value=”送信”>
</form>
</body></html>
```

この HTML ファイルに含まれるフォームはテキストボックスと submit ボタンから構成される．

1．`<input type=”text” name=”username”>` という HTML 要素によって username という名前を持つパラメータを入力させるためのテキストボックスが表示される．

2．`<input type=”submit” value=”送信”>` という HTML 要素によって「送信」と書かれた submit ボタンが表示される．

3．`<form action=”ReadForm” method=”post”>` によって submit ボタンが押された時の動作を指定している．すなわち POST メソッドを ReadForm サーブレットに送れという要求がサーバに送られることになる．

POST メソッドを受け取ったサーバは ReadForm クラスの doPost メソッドを実行する．この結果，「こんにちは，～さん」という挨拶がブラウザにレスポンスとして送られることになる．ただし「～」の中にはリクエストの一部として送られてきた username パラメータの値が入る．

【基本課題 6】

ReadForm.java を書き換え，フォームで送られた人名によって，一部の人間には挨拶し，一部の人間には挨拶しないプログラムを作れ．文字列の比較には String クラスが持つ equals メソッドが使える．

【発展課題 6】

ReadForm.java を書き換え，名前とパスワードを入力させ，正しい場合に「ログインに成功しました」と表示されるプログラムを作れ．なお，`<input type=”password” name=”pass1”>` という input 要素を使うと，入力時に文字が*で隠されるテキストボックスを作ることができる．（name 属性の値，すなわち「pass1」の部分には任意の名前を付けることができる）．

3.7 クイズゲーム

Quiz.java はフォームを利用し，クイズを提示するプログラムである．

```
 /* Sample11 Quiz.java */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Quiz extends HttpServlet {

    int qNum = 0;
    int qMax = 100;
    String[] problems = new String[qMax];
    String[] answers = new String[qMax];
    String[] descriptions = new String[qMax];

    //-------------------
    /* HTML のヘッダを表示 */
    public void printHeader(HttpServletResponse response) throws
ServletException, IOException {

       //-------------------
       /* 出力の文字コードを utf-8 に設定 */
       response.setContentType("text/html;charset=utf-8");

       //-------------------
       /* HTML のヘッダの出力 */
       PrintWriter out = response.getWriter();
       out.println("<html><head><meta http-equiv=\"Content-Type\"
content=\"text/html; charset=utf-8\">");
       out.println("<title>MyQuiz</title>");
       out.println("<link rel=\"stylesheet\" type=\"text/css\"
href=\"quiz.css\">");
       out.println("</head><body>");
       out.println("<div class='contents'>");
    }

    //-------------------
    /* 問題ファイルの読み込み */
    public void readFile(HttpServletResponse response) throws
ServletException, IOException {

       PrintWriter out = response.getWriter();
       String filename = System.getenv("CATALINA_BASE") +
"/webapps/mysite/WEB-INF/classes/problems.txt";

        try {
            BufferedReader br = new BufferedReader(new
FileReader(filename));  // 入力ストリームの作成

            String line;
           int lineCnt = 0;
            while ((line = br.readLine()) != null) {  // 文字列を読み込む
             String[] elems = line.split(",");
             if(elems.length>1){
                 for(int i = 0; i < elems.length; i++){
```

```
                    elems[i] = elems[i].trim();
                }
                problems[lineCnt] = elems[0];
                answers[lineCnt] = elems[1];
                if(elems.length > 2){
                    descriptions[lineCnt] = elems[2];
                }
                lineCnt++;
             }
            }
           qNum = lineCnt;
            br.close();  // 入力ストリームのクローズ

        } catch (FileNotFoundException e) {
            out.println("<div id='error'>ファイル " + filename + " が見つかりません.</div>");

        } catch (IOException e) {
            out.println("<div id='error'>ファイル " + filename + " を読み込みません.</div>");
            System.exit(1);  // プログラムの終了
        }
    }

    //---------------------
    /* 問題を表示 */
    public void printProblem(HttpServletResponse response, int problemID)
throws ServletException, IOException {

       PrintWriter out = response.getWriter();

       if(problemID < 0){
           problemID = (int)(Math.random() * qNum);
       }
       out.println("<div id='problem'>" + problems[problemID] + "</div>");

       out.println("<div id='answerform'>");
       out.println("<form action='/mysite/Quiz' method='post'>");
       out.println("<input type='hidden' name='problemID' value='" +
problemID+ "'>");
       out.println("<input type='text' name='answer' width=60>");
       out.println("<input type='submit'>");
       out.println("</form>");
       out.println("</div>");

    }

    //---------------------
    /* HTMLを閉じる */
    public void printFooter(HttpServletResponse response) throws
ServletException, IOException {

       PrintWriter out = response.getWriter();
       out.println("</div>");
       out.println("</body></html>");
    }

    //---------------------
```

```
    /* HTTP の GET メソッドで呼び出される関数 */
    public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
       printHeader(response);
       readFile(response);
       printProblem(response,-1);
       printFooter(response);
    }

    //---------------------
    /* HTTP の POST メソッドで呼び出される関数 */
    public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
       printHeader(response);
       readFile(response);
       PrintWriter out = response.getWriter();

       //--------------------
       /* POST のパラメータの読み込み */

       request.setCharacterEncoding("UTF-8");

       String submittedAnswer = request.getParameter("answer");
       int problemID = -1;
       String problemIDStr = request.getParameter("problemID");
       if(problemIDStr!=null){
           try{
              problemID = Integer.parseInt(problemIDStr);
           }catch (NumberFormatException e){
              out.println("NumberFormatException: " + problemIDStr);
           }
       }

       if(submittedAnswer!=null && problemID >= 0){
           submittedAnswer = submittedAnswer.trim();
           out.println("<div id='useranswer'>「" + submittedAnswer + "」
</div>");

           //---------------------
           /* 正解か不正解かを表示 */
           if(submittedAnswer.equals(answers[problemID])){
              out.println("<div id='correct'>正解です！</div>");
              out.println("<div id='description'>" + descriptions[problemID]
+ "</div>");

              int oldProblemID = problemID;
              problemID = (int)(Math.random() * qNum);
              if(qNum>2){
                  while(problemID==oldProblemID){
                     problemID = (int)(Math.random() * qNum);
                  }
              }

           }else{
              out.println("<div id='incorrect'>残念！</div>");
              out.println("<div id='anotherproblem'><a href='/mysite/Quiz'>別
の問題へ</a></div>");
```

```
            }
        }
        printProblem(response, problemID);
        printFooter(response);

    }
}
```

実行するにあたっては，mysite の下にスタイルシート quiz.css を置き，mysite/WEB-INF/classes の下に問題を記述した csv 形式のファイル problems.txt を置いておく必要がある．

quiz.css はクイズのレイアウトを決めるためのスタイルシートである．Quiz.java の出力は問題／ユーザが入力した回答／解説など，様々な div 要素から構成されている．これらのそれぞれについて文字の色やサイズ，背景色や行間の大きさを指定している．problems.txt は各行ごとに問題，回答，解説の順でコンマで区切って並べたテキストファイルである．

Quiz.java の書き換えや Tomcat の再起動を行わなくても quiz.css を変えるだけでレイアウトは変更でき，problems.txt を変えるだけで問題を変更できる．

Quiz.java は ReadForm.java と同様，GET リクエストに対応する doGet メソッドと POST リクエストに対応する doPost メソッドから構成されている．また，HTML の冒頭部分を出力する printHeader メソッド，末尾部分を出力する printFooter メソッドがある．readFile メソッドは問題のファイル problems.txt を読み込む．printProblem メソッドは問題を出力する．

HTML のフォームでは<input type="hidden" name="（パラメータ名）" value="（パラメータの値）">という要素を入れると，データを送信するたびに「パラメータ名」と「パラメータの値」のペアが送信される．これを利用して現在対象となっている問題の番号（problemID）が送られるようにし，正解／不正解の判定に利用している．

問題の番号を表す problemID は int 型でなくてはならないが，request.getParameter の戻り値は文字列となるため，String クラスのインスタンスから int 型の値を読み取る parseInt メソッドを使用している．これはクラスメソッドであるので，Integer.の後に直接メソッドを書き込み，引数として problemIDStr を与えることになる．

【発展課題 7】

problems.txt を書き換え，自分独自の問題からなるクイズを作成せよ．問題，回答，解説の順でコンマで区切り，1 行にひとつ組み合わせとなるように注意すること．

【発展課題 8】

Quiz.java を変更し，現在までの正解数と不正解数が表示されるようにせよ．たとえば correctnum（正解数）が 3 であることを送りたいのであれば，以下のように input 要素で type='hidden'にしたものを form 要素の中に入れておけばよい．

```
<input type='hidden' name='correctnum' value='3'>
```

【発展課題 9】

フォームから送られてきたすべての回答がそれに対応する問題文と共にサーバ上のテキストファイルに書き込まれていくようにせよ．ファイルへの書き込みに関しては 2.12 節「入出力処理」に記載がある．

2.12 節の形ではファイルに上書きして書き込みされるが，追加書き込みを行いたい場合は以下のように FileWriter の第二引数として true を付ければよい．

```
PrintWriter pw = new PrintWriter(new BufferedWriter(
                        new FileWriter(filename,true)));
```

3.7 エラーへの対応

実習時によく起きるエラーと対策の一覧を以下に挙げる.

■ Tomcat を起動してもトップページ http://localhost:8080/ が表示されない.

Tomcat を停止後, 3.1 節に記載した以下のコマンドを実行すると設定が初期化され, 動く可能性がある.
/kyozai/amaeda/2011mp1/scripts/tomcatSetup.sh

これでも動かなかった場合は、いったん ~/mytomcat/webapps/mysite を ~/Desktop 等にコピーしておき、~/mytomcat 全体を消す。そして tomcatSetup.sh を実行後、~/Desktop/mysite の中身を ~/mytomcat/webapps/mysite に戻す。コマンドとしては以下のようになる。

```
cp -rf ~/mytomcat/webapps/mysite ~/Desktop
rm -rf ~/mytomcat
/kyozai/amaeda/2011mp1/scripts/tomcatSetup.sh
cp -rf ~/Desktop/mysite/* ~/mytomcat/webapps/mysite/
```

■ サンプルのサーブレット(ReadForm や Quiz)は動くが, 新しく作ったサーブレットが動かない.

もしそのサーブレットが正常にコンパイルされているのであれば, web.xml にそのサーブレット名が追加されていない, あるいは webl.xml でタイプミスやタグの閉じ忘れの可能性がある.
web.xml への追加に関しては 3.5 節に説明がある.
あるいは逆に存在しないサーブレットが web.xml に登録されている可能性もある.
その場合は以下の場所にある web.xml だけ再びオリジナルのものをコピーしてくるという方法がある.
/kyozai/amaeda/2011mp1/mytomcat/webapps/mysite/WEB-INF/web.xml

■ Tomcat のトップページは表示されるが, サーブレットがひとつも動かない.

$CATALINA_HOME/bin/startup.sh を ~/mytomcat の下位ディレクトリで実行している可能性がある.
3.2 節の太字部分に説明がある.

■ 一部のサーブレットは動くが, 一部は動かず, ClassNotFound 等のエラーが表示される.
たとえば Quiz は動くが ReadForm は動かない, など.
サンプルのソースを再度コピーしてきてコンパイルし, Tomcat を再起動しても変わらない.

何らかのキャッシュが原因だと思われるが, たとえば動かないサンプルソースを別の名前に変えてコンパイルし, web.xml に追加した上で起動すると, そのサーブレットは動くことがある.
(ReadForm.java を CheckForm.java に変更し, クラス名も変えてコンパイル)
web.xml への追加に関しては 3.5 節に説明があります.

■ サーブレットで生じた例外をブラウザ上で見たい.

PrintWriter out = response.getWriter(); というように out が定義されている場合, 以下のように out を printStackTrace メソッドに与えることで, キャッチされた例外の内容がブラウザ上に表示される.

```
try {
(中略)
} catch (Exception e) {
   e.printStackTrace(out);
}
```

■ emacs で文書を入力する際，かな漢字変換がフリーズする．

~/.anthy/ を削除すると解決することがある．削除ではなく mv ~/.anthy/ ~/.anthy.old という形で名称変更するのでもよい．

■ Firefox を起動しようとすると，「別のプロセスで起動しているので終了してください」と表示され起動しない．

~/.mozilla/ を削除し，Firefox を再起動すると解決することがある．ただしブックマーク等は消えてしまう．

■ 原因不明の不具合

ホームディレクトリの使用量が 50MB を越えている場合はいろいろな不具合が生じるので，du コマンドでサイズの大きいファイル／ディレクトリを見つけ，消去することで動くことがある．たとえばブラウザのキャッシュ（~/.mozilla の下位にある Cache ディレクトリの中身）などは消しても良い．

参考文献

[1] Java 言語仕様 第 3 版， Gosling, J.他著，村上雅章訳，ピアソンエデュケーション，2006.
[2] プログラミング言語 Java 第 4 版， Arnold, K.他著，柴田芳樹訳，ピアソンエデュケーション，2007.
[3]* 独習 Java， O’Neil, J.著，トップスタジオ他訳，翔泳社，2002.
[4]* 10 日でおぼえる Java 入門教室 第 2 版， 丸の内とら著，翔泳社，2005.
[5] Tomcat ハンドブック 第 2 版， Brittain, J.他著，村上雅章訳，オライリージャパン，2008 .
[6]* 10 日でおぼえる JSP/サーブレット入門教室， 山田祥寛著，翔泳社，2002.

*がついているものが初級者向けの入門書であり，学習に適している．それ以外はリファレンス的な書籍である．