目 次

1　はじめに

- オブジェクト指向言語である Java 言語によるプログラミングを体験する
- コンピュータ言語設計の基礎でもある構文解析について学ぶ

2　第 1 週目

2.1　課題

1. 数値を 2 つ受け取って，その合計値を表示する Java GUI プログラムを作成せよ．
2. 四則演算を可能とするプログラムに書き換えよ．

2.2　概要

作成するプログラムの実行画面と使用するコンポーネント（部品）の名称などを図 1 に示す．テキストボックスを 3 つ配置し，第 1, 第 2 のボックスに数値を入力した後，ボタンをクリックすると入力した 2 つの数値の和を 3 番目のテキスト・ボックスに表示するプログラムを作成する．

- jTextField1 と jTextField2 に数値を入力する
- 「加算」ボタンをクリックすると入力した値の和が jTextField3 に表示される

図 1: プログラムの実行画面

グラフィックスには Swing と呼ばれる GUI ツールキットを利用する．

2.3　原理と利用するライブラリ

- GUI
- イベント処理
- 例外処理

2.4　設計

2.5　加算ボタンクリック時のイベント処理

ボタンがクリックされたとき，次の処理を行う．

1. テキスト・フィールド jTextField1 に入力されている値を変数 v1 に格納する

2. テキスト・フィールド jTextField2 に入力されている値を変数 v2 に格納する

3. y ← v1+v2 とする

4. y の値をテキスト・フィールド jTextField3 に書き出す

テキスト・フィールドからの数値の取得，あるいは，テキスト・フィールドへの数値の表示を行う場合に，例外 (NumberFormatException) が発生する可能性がある．そこで，この例外が発生した場合の処理を try 〜 catch 文で記述する．

2.5.1 MyTextField クラス

テキスト・フィールド (JTextField クラス) を利用して文字列の表示，および，書き込まれた文字列の取得を行うことができるが，数値を表示する際には，一旦，数値を文字列に変換した後に表示する必要がある．数値の取得についても同様である．そこで，JTextField を拡張して，直接に数値を入出力することができる MyTextField クラスを作成する．

追加する機能など

- MyTextField クラス
  - JTextField クラスを継承
- `public void setText(double)`
  - setText 関数をオーバーロード
  - 引数の値を表示する
- `public double getValue()`
  - 入力されている数値を返す

2.6 プログラムの作成

2.6.1 NetBeansIDE(統合開発環境) によるプログラム作成

NetBeansIDE は Java プログラムの開発を目的として作成された統合開発環境であり，現在では，C, Javascript, HTML など様々なプログラムの開発に利用することができる．

プログラム作成の手順:

1. NetBeansIDE の起動
2. プロジェクトを作成する
3. プロジェクト内にファイルを作成する
4. ファイル内にプログラムを作成する
   (a) ボタンなどの GUI コンポーネントを配置する
   (b) プログラムコードを記述する
5. プログラムを実行/デバッグする

作成するプログラム

図 1 に示すように，テキストボックスを 3 つ配置し，第 1, 第 2 のボックスに数値を入力した後，ボタンをクリックすると入力した 2 つの数値の和を 3 番目のテキストボックスに表示するプログラムを作成する．

プログラムの作成:

1. NetBeansIDE の起動
   - アイコンをクリックして NetBeansIDE を起動する
2. 新規プロジェクトの作成
   (a) Java プログラムを新たに作成する場合は，先ず，プロジェクトを作成する
   (b) メニューを「ファイル新規プロジェクト」と進み
   (c) カテゴリに「Java」，プロジェクトに「Java アプロケーション」を選択して,「次 >」をクリックする
   (d) プロジェクト名に「CalcProgram」と入力して，「メイン・クラスの作成」のチェックを外す．その後、「終了 (F)」をクリックする
3. 新規ファイルの作成
   (a) メニューを「ファイル (F)」「新規ファイル (N)」と進む．
   (b) 「新規ファイル」ダイアログ・ボックスで，カテゴリに「Swing GUI フォーム」，ファイル・タイプに「JFrame フォーム」を選択して,「次」をクリックする
   (c) クラス名として「Calc」を入力する
   (d) 「終了」をクリックする

   ファイル Calc.java 内に Calc クラスの基本的な部分が生成され，デザイン領域に GUI コンポーネントのグラフィカル・ビューが表示される．

   「ソース」ボタンを選ぶとソース・プログラムが表示され,「デザイン」バタンでグラフィカル・ビューが表示される．
4. プログラムの作成 1 (ボタンなどの GUI コンポーネントを配置する)

   「デザイン」ボタンを選択し，GUI コンポーネントのグラフィカル・ビューを表示した状態する．最初，ビューにはフレーム (JFrame) が配置されている．この状態で，パレットからコンポーネントを選択してビュー上に配置する．

   (a) 「パレット」の「Swing コントロール」から「ラベル」を選択して，図 2 のようにフレーム (JFrame) 上に配置する．
   (b) 同様に「テキスト・フィールド」,「ボタン」配置する

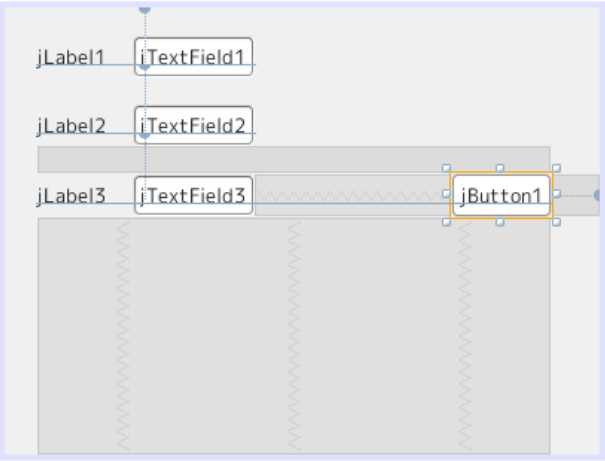


図 2: コンポーネントの配置

(c) 図 1 のように，各コンポーネントの表示やサイズを変更する．
コンポーネントを右クリックして表示されるメニューから「テキストを変数」を選び，キーボードから表示する文字列を入力する．コンポーネントの位置やサイズは、マウス・クリックとドラッグにより変更することができる

配置した各部品 (コンポーネント) には jTextField1, jTextField2 などと自動的に変数名が付けられる．

5. プログラムの作成 2 (イベント処理を記述する)

(a) イベントの発生元となるボタンを右クリックし，表示されるメニューより,「イベント」「Action」「actionPerformed」と進んでマウスを左クリックする．
表示が「デザイン」から「ソース」に切り替えられ，ソース・プログラムが表示される

(b) 表示されるプログラム中の関数 jButton1ActionPerformed

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

を書き換えて，このイベントに対する処理内容を記述する．

イベント処理関数　=ボタンのクリックに対して以下の処理を行うことにする．

表 1: イベント処理

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    double v1, v2, y;
     i. jTextField1 に入力されている数値を読み取り，変数 v1
        に格納する

     ii. jTextField2 の数値を変数 v2 に格納する

    iii. 加算: y ←v1 + v2

    iv. y に格納されている値を jTextField2 に表示する
}
```

記述の具体化　表 1 に示す処理 i.〜 iv. を具体に検討する．

i. jTextField1 への入力値を v1 に格納する
JTextField への入力されてものを直接に数値として読み取ることができない．そこで，JTextField クラスのメンバ関数 getText を使用して入力値を文字列として受け取り，その後，数値に変換して代入する．処理は次のようのプログラムコードになる．

```
String st;                    // 文字変数 (String 型) を宣言
st = jTextField1.getText(); // 入力文字列を取り出し
v1 = Double.valueOf(st);     // 数値に変換して格納
```

ii. jTextField2 への入力値を v2 に格納する
v1 の場合と同様にして代入する

```
st = jTextField2.getText();
v2 = Double.valueOf(st);
```

iii. 加算: y = v1 + v2;

iv. y を jTextField2 に表示する．
数値を文字列に変換して表示する

```
            st = Double.toString(y);  // 文字列に変換
            jTextField2.setText(st);
```

6. プログラムを実行/デバッグする

   (a) 「実行 (R)」「プロジェクト (CalcProgram) を実行」を選択して，プログラムを実行する

   (b) 「プロジェクト CalcProgram にはメイン・クラスが設定されていません。」と表示された場合には，メイン・クラスの選択リストから Calc を選び,「OK」をクリックすると，プログラムがコンパイルされ，続いて，エラーがなければ，実行が開始される

2.6.2 ソースプログラムを概観する

「ソース」タブを選択して作成したソースプログラムを確認する．

全体の構成

- Calc クラスの定義内に変数 (メンバ変数，プロパティ) および関数 (メンバ関数, メソッド) の定義を記述する
- `jButton1` などこのクラスに属する変数は，クラス内部で有効であり，このクラスのプロパティとも呼ぶ
- `initComponents` などの関数をメソッドとも呼ぶ
- クラス名と同じ名前を持つメソッド `Calc()` は，クラス・オブジェクトが生成されるときに 1 度だけ呼び出されてクラス・プロパティ（変数）の初期化のために使用される．このことから コンストラクタ と呼ばれている．
- public や private はアクセス修飾子と呼ばれ，プロパティ (変数) やメソッド (関数) がクラス外部からアクセス可能 (public) であるか，クラス内部のみアクセス可能 (private) であるかを制限する．
- この Calc クラスは main メソッドを含むので，このプログラムはシステムから直接実行可能であり，システムから呼び出されると Calc クラスが生成され，コンストラクタが呼び出された後に，main メソッドから処理が開始される

```
public class Calc extends javax.wing.JFrame {
    public Calc() {
        initComponents();
    }

    private void initComponents() {
        ...
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        ...
    }

    public static void main(String args[]) {
        ...
    }
    // 変数の宣言
    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel1;
    ...
}
```

initComponents

```
private void initComponents() {
    jLabel1 = new javax.swing.JLabel();  // オブジェクトの生成
    jLabel2 = new javax.swing.JLabel();
    ...
    jLabel1.setText("値１");
    ...
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
        }
    });   //  ボタンをクリックした時の処理をイベントハンドラ (ActionListener) に登録
    ...
    // コンポーネントの配置
    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    ...
}
```

イベント処理関数

```
private void jButton1ActionPerformed(java.awt.event.Action evt) {
    double v1, v2, y;
    String st;
    st = jTextField1.getText();
    v1 = Double.valueOf(st);
    ...
    st = Double.toString(y);
    jTextField2.setText(st);
}
```

- 変数の有効範囲

  変数はそれが宣言されたブロック内 ('{' と'}' で囲まれた部分) でのみ有効である.

  - イベント処理関数内で宣言された v1, v2, y, st はこの関数内でのみ有効であり，関数の外では参照や代入はできない.
  - jButton1，jLabel1 は Calc クラス内の変数として宣言されていて，このクラスのメンバ関数 Calc()，initComponents() などで参照や代入が可能である．このようなクラス内部全体で有効な変数をメンバ変数と呼ぶ.

- new 演算子

  ```
  jLabel1 = new javax.swing.JLabel();
  ```

  JLabel クラスのオブジェクトを生成して，そのアドレスを変数 jLabel1 に格納する．この動作をインスタンス化と呼ぶ．(図 3)

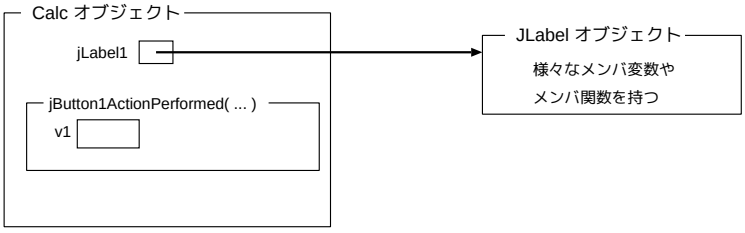


図 3: 変数とオブジェクトの関係

- 関数の呼び出し

  ```
  jTextField2.setText(st);
  ```

  st をパラメータとして jTextField3 のメンバ関数 setText を呼び出す．(図 4)

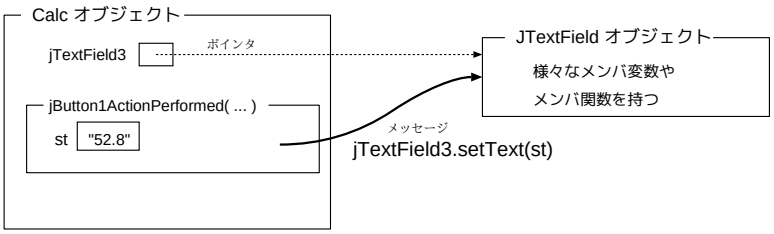


図 4: メンバ関数の呼び出し

2.6.3 例外処理

テキスト・フィールドに数値以外のものを入力した状態で,「加算」ボタンをクリックするとエラーが発生する (図 5). このエラーメッセージは，テキスト・フィールドから読み取った文字列が数値に変換できないために例外 (Exception) が発生したことを述べている (NumberFormatException).

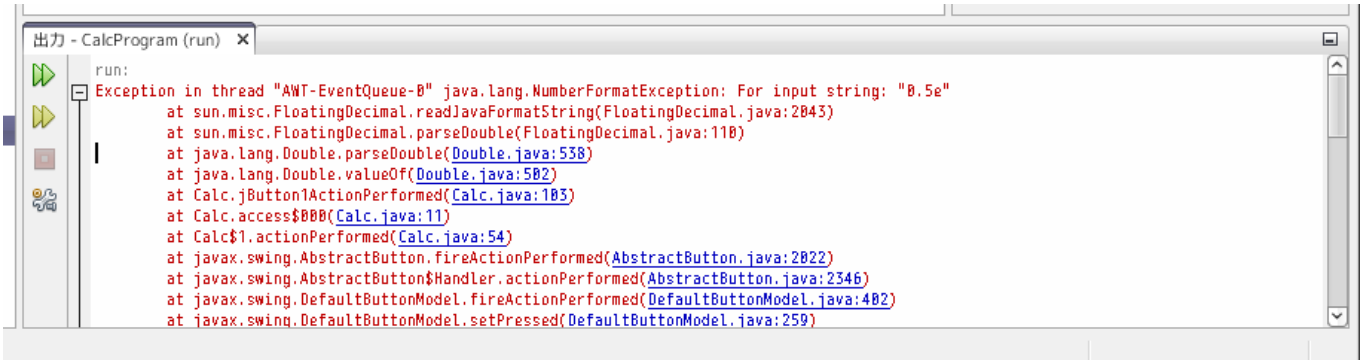


図 5: 数値以外を入力した場合に表示されるエラーメッセージ

このエラーはテキストフィールドに入力した文字列を数値に変換できない場合に発生する．図 5 中に表示されている NumberFormatException がそれを意味する．このような実行中のエラーの発生を java では例外と呼び，次のような try〜catch 文を用いて例外に対応することができる．

```
try {
    try ブロック
}
catch(ExceptionType param) {
    例外処理ブロック
}
```

例外が発生する可能性があるプログラム部分を try ブロック内に配置し，*ExceptionType* の例外が発生した場合の処理を例外処理ブロック内に記述する．

NumberFormatException への対応

図 5 に示す例外の発生に対して，メッセージダイアログと呼ばれるウィンドウを表示するように jButton1ActionPerformed メソッドを表 2 のように書き換える．

ただし，上のように jButtonActionPerformed メソッドを書き換えると，showMessageDialog メソッドの部分でエラーが表示される．ここで，図 6 に示すように，行番号下のエラー・アイコンを左クリックして表示されるヒントに対して

javax.swing.JOptionPane.showMessage... をインポートに追加

を選択して，エラーを解決する．(2 行目に import 文が追加される)

java ではクラス・ライブラリはパッケージとしてまとめて管理することができ，import 文はパッケージを読み込むことを指示する．import 文は C 言語での include 命令と類似した機能を持つ．

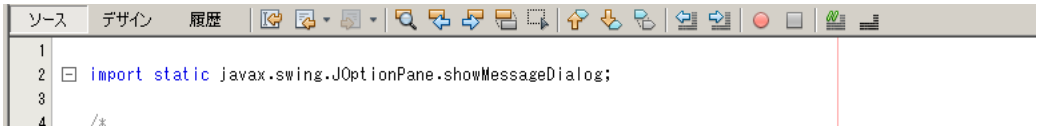
表 2: try〜catch による例外処理

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        double v1, v2, y;
        String st;                    // 文字列を格納する変数 st を宣言

        ...  (変更なし)

        jTextField3.setText(st);      // jTextField3 に合計値を表示する
    }
    catch (NumberFormatException e) {
        showMessageDialog(null, "数値が正しく入力されていません。");
    }
}
```

エラーメッセージ



生成された import 文

図 6: showMessageDialog に対するエラー表示と import 文

2.6.4 MyTextField クラスの作成

GUI ツールキット Swing に用意されている JTextField はテキストフィールド・コンポーネントを提供する．表 2 で使用したように，このコンポーネントはキーボートからの入力や表示を文字列として扱う必要がある．このため，前章で作成したプログラムでは，数値と文字列の間での変換が必要であった．

JTextField を拡張して数値を扱えるようにする．作成するクラスを MyTextField クラスと名付ける．

MyTextField クラス

- jTextField の拡張クラス (jTextField クラスを継承する)
- 以下のメソッドを追加する
  - `setText(double)`
  - `double getValue()`

1. 新規にファイル MyTextField を作成する
   (a) 「ファイル (F)」「新規ファイル」と進み
   (b) カテゴリとして「Java」，ファイル・タイプとして「Java クラス」「次 >」
   (c) クラス名「MyTextField」「終了 (F)」

表 3: MyTextField クラス

```
public class MyTextField extends javax.swing.JTextField {
    public void setText(double v) {
        setText(Double.toString(v));
    }

    public double getValue() {
        String st = getText();
        return Double.valueOf(st);
    }
}
```

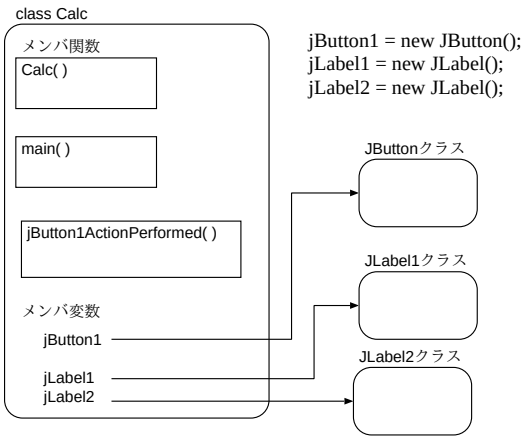
2. MyTextField.java に表 3 に示すプログラムを入力する
3. 「ファイル」⇒「保存」あるいは「すべてを保存」を選択して，作成した MyTextField.java を保存する
4. Calc.java タブをクリックした後,「デザイン」を選択してコンポーネントのグラフィカル・ビューを表示する
5. テキスト・フィールド jTextField1〜jTextField3 を削除する
6. 画面左側のプロジェクト・ウィンドウから「MyTextField.java」をドラッグし，中央のデザイン・ビューの適切な位置にドロップして JFrame 内に MyTextField コンポーネントを上から順に 3 つ配置する．各コンポーネントの変数名は，順に，myTextField1, myTextField2, myTextField3 となる．
7. 「ソース」タブを選択して，表示をソースプログラムに変更する．ボタンのクリック処理を定義する jButton1ActionPerformed メソッドを表 4 に変更する

表 4: 変更した jButton1ActionPerformed メソッド

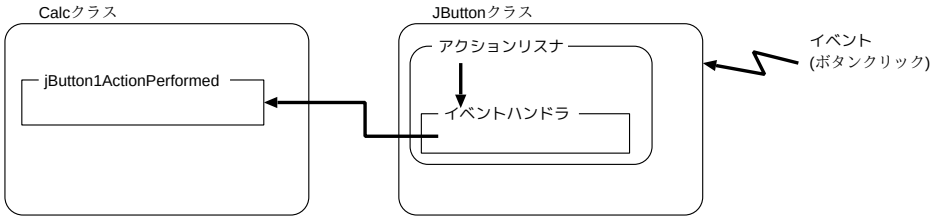
```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    double v1, v2;
    try {
        v1 = myTextField1.getValue(); // myTextField1 に入力された値を v1 に格納
        v2 = myTextField2.getValue(); // myTextField2 に入力された値を v2 に格納
        myTextField3.setText(v1+v2);  // それらの和を myTextField3 に表示
    }
    catch(NumberFormatException e) {
        showMessageDialog(null, "数値が正しく入力されていません。");
    }
}
```

2.7 まとめ

- プロジェクト
- クラス (クラス名とファイル名)



- イベント処理



- 例外処理

  `try〜catch`

- オブジェクト指向プログラミング
  - カプセル化

    private, public
  - ポリモーフィズム (多態性)

    `setText(`*String*`)`

    `setText(`*double*`)`
  - インヘリタンス (継承)

    `extends`

2.7.1 GUI プログラムの作成

NetBeansIDE での GUI プログラム作成

1. プロジェクトの作成
   - 「メイン・クラスの作成」のチェックを外す
2. ファイルの作成 (クラス)
   - 「Swing GUI フォーム」→「JFrame フォーム」
3. コンポーネント (部品) の配置
4. イベント処理プログラムのコーディング
5. 実行/デバッグ

3　演習問題２ (第２週目)

3.1　課題

括弧を含む計算式を入力すると，式を評価してその値を表示する Java GUI プログラムを作成せよ．

4　原理

「12+4.5*(10-6)」のような計算式を入力とする電卓プログラムを作成する．このためには，

- 入力された文字列を 12, +, 4.5 といった字句に区切る字句解析機能
- 計算のルールに従って，字句の並びを解析する構文解析機能
- 構文を理解して、計算を実行する機能

が必要になる．

ここでは，数値に対する四則演算に加えて，括弧'(', ')' を含む計算式を処理するプログラムを作成する．

4.1　字句解析

図 7 に示すように，与えられた文字列を字句 (トークン, token) に分割する．計算機言語を例に取ると，C 言語では，キーワード，変数，定数，演算子，括弧などが字句である．通常，改行コード，空白などの区切り記号は，字句解析の段階で読み捨てられる．

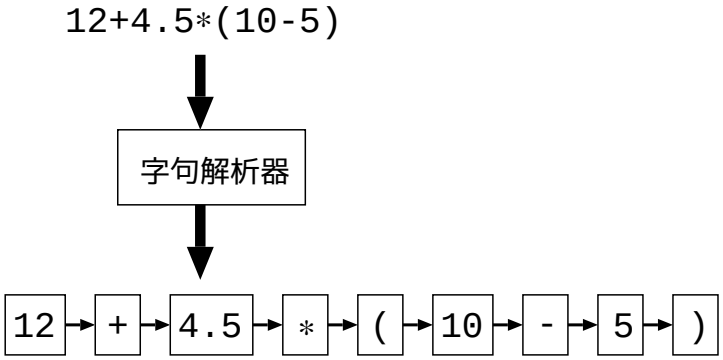


図 7: 字句解析

4.2　計算式の構文解析

文法に従って字句の間の関係を解析し，構文木などのデータ構造で表現する．

$$12 \ + \ 2 \ * \ 3 \ - \ (10 \ + \ 5)/3$$

を計算するとき，演算には優先順位があるので次のように解釈し，処理する．

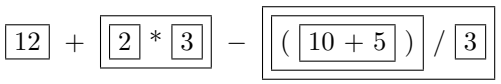
- この式は 3 つの項 (term) の加減算により構成されている

$$\boxed{12} \ + \ \boxed{2 * 3} \ - \ \boxed{(10 \ + \ 5)/3}$$

- さらに，各項は因子 (factor) の乗除算で構成されている

$$\boxed{12} \ + \ \boxed{\boxed{2} * \boxed{3}} \ - \ \boxed{\boxed{(10 \ + \ 5)} \ / \ \boxed{3}}$$

- 各因子は数値，あるいは，括弧で括られた式 (expr) で構成されている．



これは次のように，バッカス・ナウア記法 (BNF, Backus-Naur form) を用いて表現することができる．

〈式〉 ::= 〈項〉 | 〈式〉 '±' 〈項〉
〈項〉 ::= 〈因子〉 | 〈項〉 '*' 〈因子〉 | 〈項〉 '/' 〈因子〉
〈因子〉 ::= 〈数値〉 | '(' 〈式〉 ')'

上式において,「::=」は左辺が右辺で定義されることを表し，縦棒「 | 」は「あるいは (or)」を意味する．
第 1 行目は,

〈式〉とは,「〈項〉」，あるいは、「〈式〉に続いて，'+' あるいは '−' 記号と〈項〉」である

と読むことができる．
上で示した式などの定義は正規表現を用いて次のように再定義することができる．

〈式〉 ::= 〈項〉 ('±' 〈項〉) *
〈項〉 ::= 〈因子〉 { ('*' 〈因子〉) | ('/' 〈因子〉) } *
〈因子〉 ::= 〈数値〉 | '(' 〈式〉 ')'

上式において,「*」はメタ文字 (メタ記号) と呼ばれ，0 回を含めた繰り返しを意味する．したがって，〈項〉の定義

〈項〉 ::= 〈因子〉 { ('*' 〈因子〉) | ('/' 〈因子〉) } *

の右辺は，次のように読む：

〈因子〉に続いて,
('*' に続く〈因子〉)，あるいは，(/ に続く〈因子〉)
の繰り返し (0 回も含む) である．

5 設計

5.1 MyTokenizer

java.io.StreamTokenizer のサブクラスとして字句解析器を作成する．ここで作成する字句解析器は，与えられた文字列（計算式）を先頭から順にスキャン（検索）し，字句に分割して、その種類 (コード) を返す (図 7)．この字句解析器のクラス名を MyTokenizer と呼ぶことにする．

5.1.1 MyTokenizer クラス

- 機能

  java.io.StreamTokenizer クラスの属性を継承し，文字列を字句に分解する．認識する字句は以下の通りである．

  1. 符号なしの数値
  2. 上記以外の文字 (1 文字)

  ただし，空白は区切り記号として読み飛ばす．

- コンストラクタ

  StreeamTokenizer はキーボードなどの標準入力やファイルなどのストリームからケータを読み取るが，ここでは，文字列 (テキストボックスに入力されたもの) からデータを読み取るように変更する．

コンストラクタ	機能
MyTokenizer(String)	引数として与えられた文字列をバッファに格納し，字句解析器を初期化する．その際に，区切り記号などの字句解析ルールを決定する．

- 主なメンバ変数

変数	機能
double nval	現在のトークンが数値の場合，その値を記憶する．
int ttype	現在のトークンの種類を表すコードを保持する．
static int TT_EOF	バッファからの読み込みが終了したことを示すコード．バッファを最後まで取り出したあとに nextToken が呼ばれるとこのコードを返す．
static int TT_NUMBER	字句が数値であることを示すコード．

- 主なメンバ関数

メソッド	機能
int nextToken()	バッファの先頭から順に空白を読み飛ばした後，字句を 1 つ取り出し，その種類を表すコード (整数値) を返す．ただし，取り出した字句はバッファから取り除く．また，文字列が数値の場合はその値をメンバ変数 nval に格納する．
void pushBack()	現在のトークンをバッファに戻す．nextToken が呼び出されると，この戻された字句が再び取り出される．

  - `int nextToken()` が戻す値

    戻り値　取り出した字句 (現在のトークン) の種類をコード (整数値で返す)

値	説明
`TT_NUMBER`	符号なしの数値
`TT_WORD`	単語
`TT_EOF`	最後まで読み終えて，現在のトークンがない状態
読み取った文字のアスキーコード	数値以外の字句 (1 文字) を読み取った場合，そのアスキーコードを戻り値とする

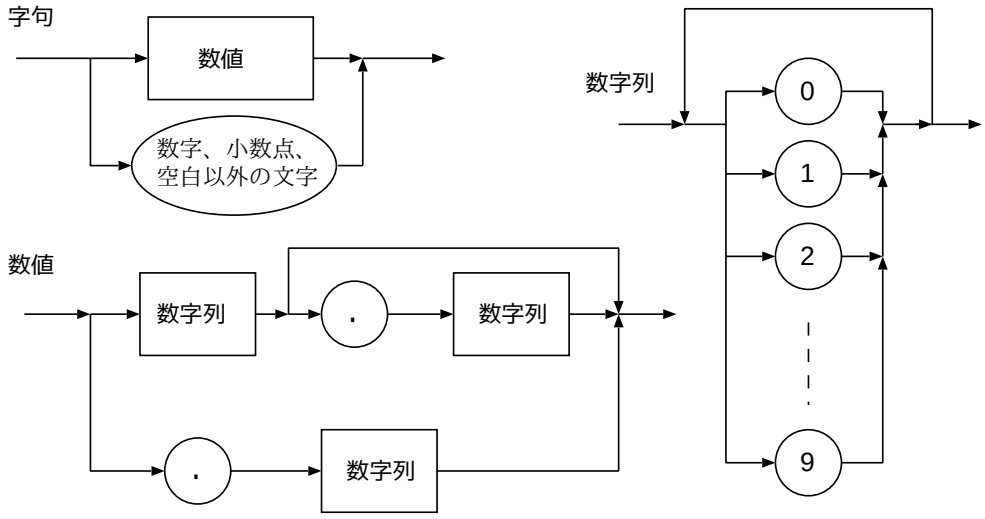
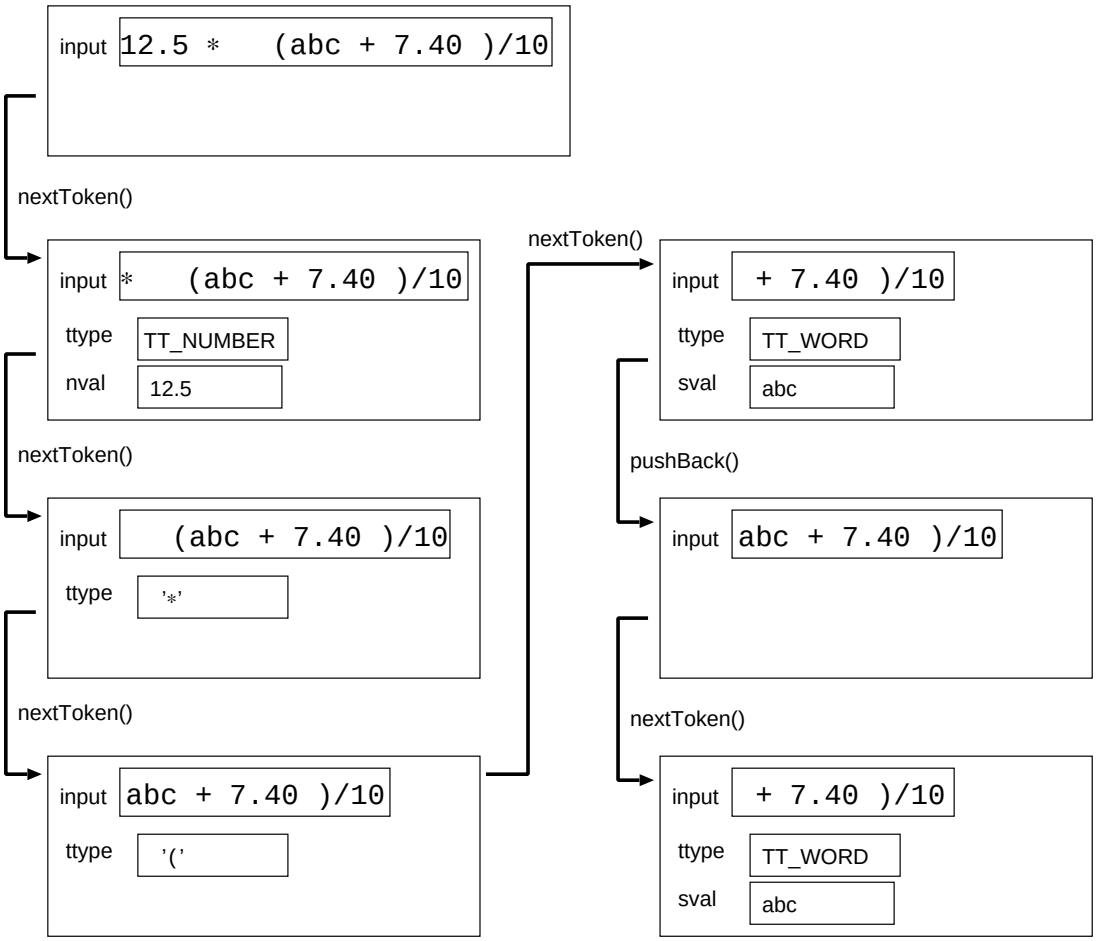


図 8: MyTokenizer が識別する字句のルール



[図] MyTokenizer オブジェクトの変化 (字句解析の実行例)

5.2 構文解析と計算処理

5.2.1 Calc クラス

計算式に対する記述の規則を図 9 のように定める．この図は，式は単独の項，あるいは，項に続く ± 演算記号

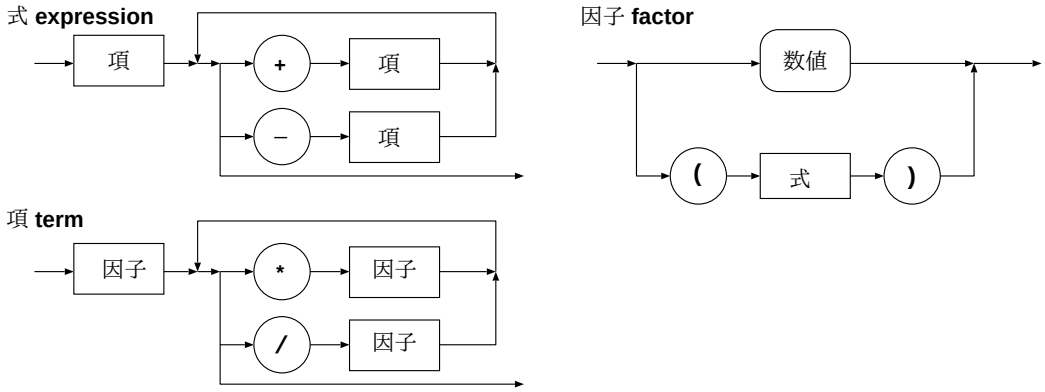


図 9: 状態遷移図

(加減算) と項の繰り返しであること定義している．さらに，項は因子あるいは，因子に続く乗除算記号と因子の繰り返しであり、因子は、数値，あるいは，式を括弧で挟んだものと定義している．

式「0.4 - 1.5*(8-3)」,「5+(10-(2+4)/2)/2」などはこの構文にマッチする．

5.2.2 プログラム

- CalcProgram2 という名称で新規にプロジェクトを作成する
- 字句解析器には，既に作成済の MyTokenizer クラスを使用する．(付録参照: MyTokenizer.java)
- プロジェクト内に新規ファイル Clac.java を作成し，計算式を実行するプログラムを作成する

以下，説明の都合で，計算式を入力するテキスト・ボックスを jTextField1，計算結果 (数値) を表示するテキスト・ボックスを jTextField2，計算開始を指示するボタンを jButton1 と表現する．

- 主なメンバ変数 (プロパティ)

変数	機能
MyTokenizer token	字句解析器

- 主なメンバ関数 (メソッド)

関数	機能
ボタン・アクション	jButton1 がクリックされたときの処理
double expression()	「式」を処理し，その結果得られた値を返す
double term()	「項」を処理し，その結果得られた値を返す
double factor()	「因子」を処理し，その結果得られた値を返す

1. token 変数

   メンバ変数として token を宣言する：

   ```
   MyTokenizer token;
   ```

2. jButton1 がクリックされたときの処理

   前章と同様に，デザイン画面上でアクション・リスナへの自動登録を行い，以下のようなイベントハンドラを記述する．

   処理手順

   (a) 入力された計算式 (文字列) を初期値として，字句解析器 token を初期化する

   ```
   String str = jTextField1.getText();
   token = new MyTokenizer(str);
   ```

   (b) 式を評価する (expression)

   ```
   double y = expression();
   ```

   (c) 評価した結果 (数値) を jTextField2 に表示する

   ```
   jTextField2.setText(Double.toString(y));
   ```

   (d) 後処理: (この部分は省略可)
   字句 (トークン) を 1 つ読み取り，
   - それが TT_EOF(終了コード) なら正常に終了
   - 数値などの場合は処理を終えていない文字列が残っていることを意味するので，showMessageDialog 関数[1] を呼び，メッセージ・ダイアログにエラーメッセージを表示する

[1] ファイルの先頭 (クラス宣言の前) に import 文が必要
```
import static javax.swing.JOptionPane.*;
```

```
            switch(token.nextToken()) {
                case MyTokenizer.TT_EOF:
                    break;
                case MyTokenizer.TT_NUMBER:
                    showMessageDialog(null,"error! [" + token.nval + "]");
                    break;
                default:
                    showMessageDialog(null,"error! ["+(char)token.ttype+"]");
            }
```

プログラム

```
    public void jButton1ActionPerformed(java.awt. ...) {
        (a) token の初期化
        try {
            (b) 式の評価
            (c) 結果の表示
            [(d) 後処理]   // ここは省略可
        }
        catch(IOException e) {
            showMessageDialog(null, "式に誤りあり");
        }
    }
```

3. expression() 関数

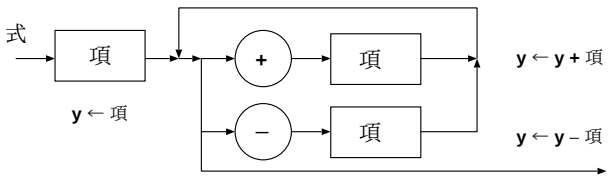


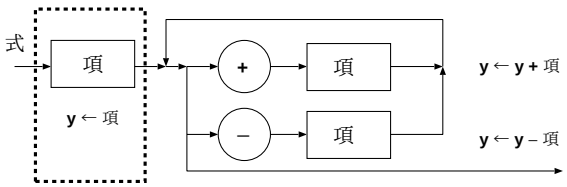
図 10: 「式」の状態遷移図

式を評価してその結果 (値) を返す．この関数は状態遷移図 10 にしたがって処理を実行する．その手順は次のようになる．

処理手順

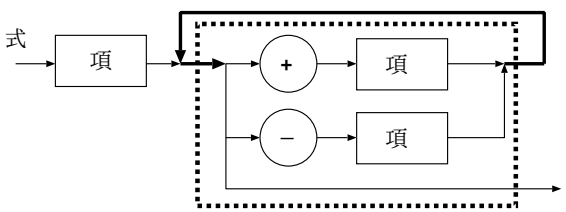
(a) 項を処理し，その値を変数 y に格納する

```
y = term();
```



(b) ループ内で次の処理を繰り返す

```
while(true) {
   ...
}
```

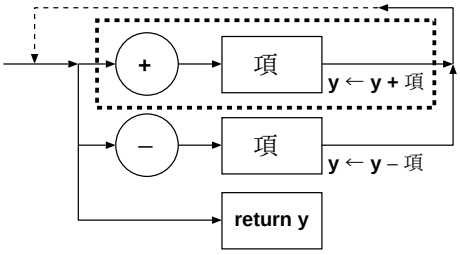


i. 字句 (トークン) を 1 つ読み取り，字句の種類に応じて処理を選ぶ

```
token.nextToken();
switch(ttype) {
    case ...;
    case ...;
}
```

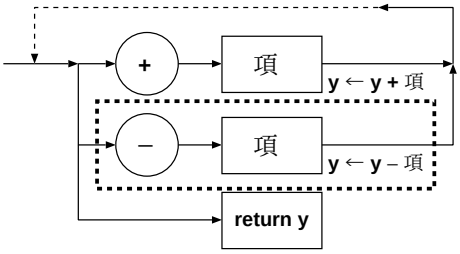
ii. その字句が'+' の場合は，項を読み取り，その値を y に加える

```
case '+':
    y += term();
    break;
```



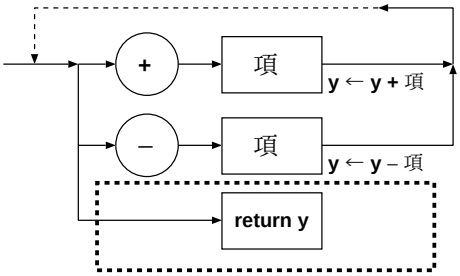
iii. 字句が '– の場合は，項を読み取り，y からその値を差し引く

```
case '+':
    y -= term();
    break;
```



iv. 上記以外の場合:
'+', '–' 以外の場合は，次の処理に備えて現在のトークンを構文解析器に戻し (pushBack())，y を戻り値として，この関数を呼び出したところへ戻る

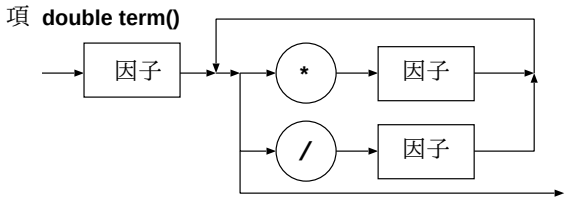
```
default:
    token.pushBack();
    return y;
```



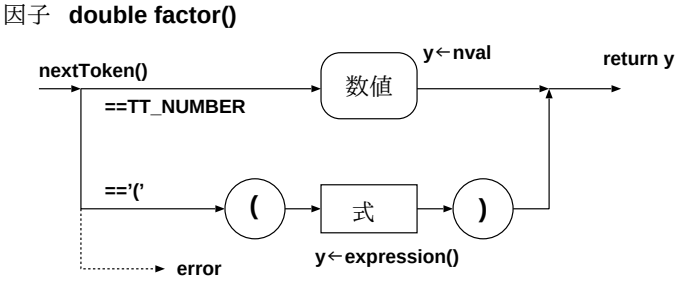
プログラム

```
private double expression() throws IOException {
    double y;
    (a) 項を評価し，値を y に代入する
    while(true) {
        switch(token.nextToken()) {
            ii. '+' の場合は次の項を評価した値を y に加える
           iii. '-' の場合は次の項を評価した値を y から引く
            iv. 上記以外の場合
        }
    }
}
```

4. term() 関数 – 項を処理して，その値を返す.



5. factor() 関数 – 因子を処理して，その値を返す.



(a) token から字句を 1 つ読む

- それが数値なら
  i. 読み込んだ数値を返す
- 字句が開き括弧なら
  i. expression を呼んで，括弧内の式を処理し，その値を受け取る
  ii. 式の後に閉じ括弧があるか確認する
  – 閉じ括弧がなければエラー
  iii. 式の値を返す

5.3 演習問題

1. メンバ関数 term および factor を作成し，四則演算と括弧が使用できる計算プログラムを完成せよ
2. 時間に余裕があれば，以下の内容にチャレンジせよ
   - 「+10+2」や「-(2+3)」のように符号で始まる式も処理できるようにプログラムを拡張する
   - その他の (思いつく) 機能を付けたす

A　Java 言語とオブジェクト指向プログラミング

オブジェクト指向プログラミング言語であり，クラス (class) と呼ばれるテンプレート (ひな形) を定義することでプログラムを記述する．詳しくは,「JAVA プログラミング」(3 年次後期) で.

Java では 1 つのファイルに 1 つのクラスを記述する．クラス名とファイル名は一致している必要があり，ファイル名にはクラス名に続いて拡張子「.java」を付ける

- オブジェクト指向言語では，関連するデータと関数をひとかたまりにして取り扱う．その集まりをオブジェクトと呼ぶ
- オブジェクト指向プログラミングでは，オブジェクトに対してメッセージを送ると考えでプログラムを構築する
- コンピュータ・メモリ上のデータと関数の集まりをあたかも、そこにある物 (object) のようにとらえることから，オブジェクトという名称が用いられている

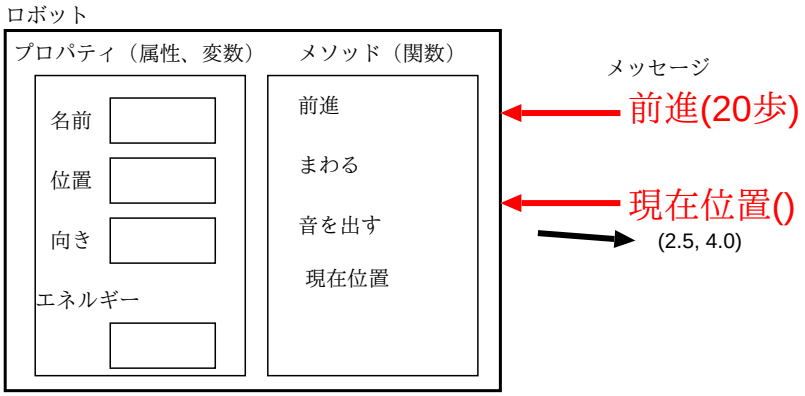


図 11: オブジェクト指向プログラミングの例

変数に文字型，整数型などがあるように，様々な種類のオブジェクトが存在し得る．オブジェクトの種別をクラスと呼ぶ.

- クラスはオブジェクトのひな形（テンプレート）である
- クラスからメモリ上にオブジェクトを生成する過程をインスタンス化という
- オブジェクトをインスタンスとも呼ぶ

関数の使用例　図 12 の例において，クラス内の関数「給油する」を呼び出すコードを以下に示す.

```
b.給油する(100); //　b に対してメッセージを送る
a.給油する(500); //　a に対してメッセージを送る
```

このように，オブジェクトを示す変数名に続いて，コンマ (.) で区切って関数と引数を書く.

クラスの構造

```
class クラス名 {　定義部　}
```

定義部には，変数宣言と関数定義をセミコロン (;) で区切って置く．変数宣言，関数定義の出現する順番と数は任意.

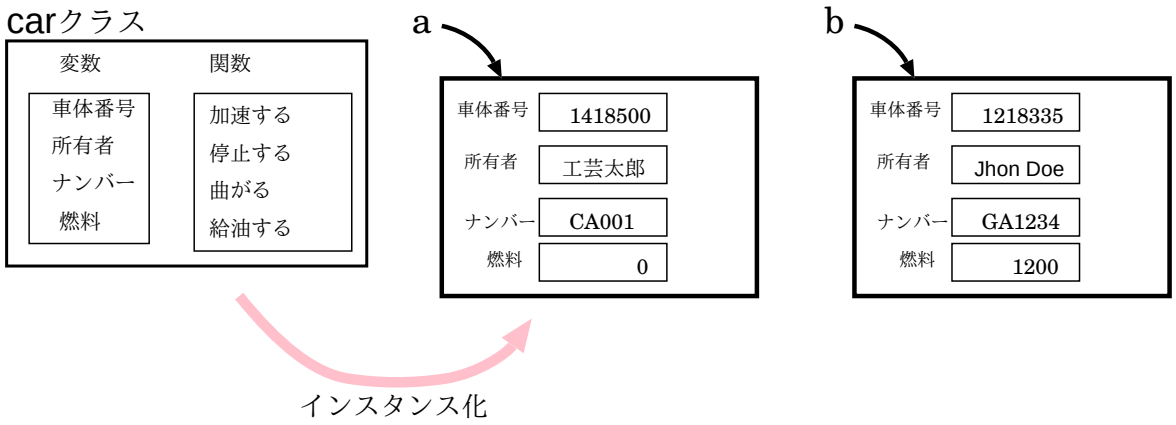


図 12: クラスとオブジェクトの例

クラスは，変数宣言と関数定義で構成され．関数定義の内側に関数を定義することはできない．関数は，しばしば，メソッドとも呼ばれる．C 言語と同様に，関数内で変数を宣言することもできる．関数内で宣言した変数はその関数内でのみ有効である．

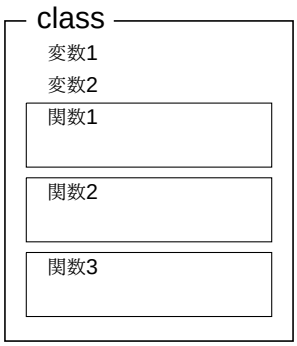


図 13: クラスの構成

関数 (メソッド) の構造

戻り値の型名 関数名（引数リスト）{　定義部　}

A.1　簡単な java プログラムの例

example.java ファイル

[プログラムリスト] プログラムの例

```
public class example {
    String name;
    public static void main(String args[]) {
        String localStr;
        name = "工芸";
        localStr = "太郎";
        System.out.println("私の名前は、"+name+localStr+"です。");
        set("山田");
        System.out.println("あなたは"+name+localStr+"さんですか。");
    }

    void set(String st) {
        name = st;
    }
}
```

C 言語と同様に，main 関数から処理が開始される．このプログラムを実行すると，3 行目の main 関数から実行が開始されて，コンソール画面に「私の名前は、工芸太郎です。」と表示した後，次の行に「あたなたは山田太郎さんですか。」と表示される．

この例では，プログラム全体は example クラスの定義であり，このクラスの内部で main 関数および set 関数が定義されている．

プログラム 2 行目中は String クラスのオブジェクト (を指し示す変数) として name を宣言している．String は文字列を扱うクラスであり，文字列 (String) に対する演算子「+」は連結演算子と呼ばれ，文字列の連結を行う．

A.2 Java 言語の特徴

1. カプセル化

   private, public などのアクセス修飾子を用いて，クラスの外部からのアクセスを制御する．

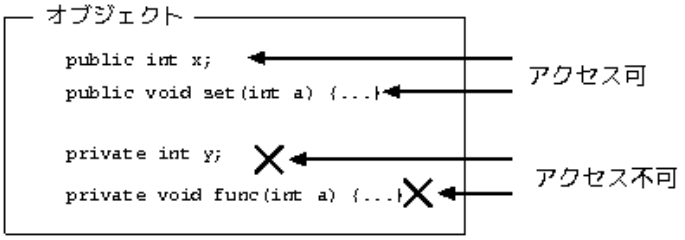


図 14: カプセル化

2. ポリモーフィズム (多態性)

   様々な異なるオブジェクトに対して，共通したメソッドの呼び出し方を提供することができる．

3. インヘリタンス (継承)

   あるクラスを拡張して新たに別のクラスを作成することができる．新しいクラスは元のクラスのプロパティやメソッドを継承する．もとのクラスをスーパークラス，新たなクラスをサブクラスと呼ぶ．

B GUI

C MyTokenizer.java

C.1 プログラム・リスト

```
import java.io.*;

public class MyTokenizer extends StreamTokenizer {
    MyTokenizer(String st) {
        super(
                new BufferedReader(
                        new InputStreamReader(
                                new ByteArrayInputStream(st.getBytes()))));
        resetSyntax();
        parseNumbers();
        whitespaceChars(0, 32);
        ordinaryChar('-');
        wordChars('A', 'Z');
        wordChars('a', 'z');
    }
}
```

D 演習に当たって

- プログラミングは難しくない．コンピュータは所詮，機械であるからルール通りに記述すれば，ルール通りの動作をする．
  - ルールがわからないときは調べる
  - どこがわからないかを明確にする
- 処理内容を理解しないまま真似をする手抜きの習慣を付けたまま，複雑な問題に取り組むとプログラミングに自身を失い，嫌いになる．(自分の可能性を失う)
- 好奇心を持ち，自分なりに工夫する．
- 人間ならば間違い，思い違いはあって当然
  - 文法的なミス: エラーメッセージを読む
  - 実行結果がおかしい: デバッガを使って原因を調べ，修正する

E ワークルームでの PC の利用

ユーザ名　wr
パスワード　cs
[注意]

- Windows をシャットダウンするとデータは消滅する
- 作成したプログラムの保存
  - USB メモリなど外部記憶装置に保存
  - ファイルをセンター・システムに転送

E.1 FFFTP によるファイルの転送

F レポート

1. 学籍番号，氏名
2. 日付と課題名
3. 自分で定義した変数の役割
4. データ構造などで工夫したこと (もしあれば)
5. 自分で記述した関数の説明 (IDE が自動作成した関数については説明不要)
    - 引数，戻り値，処理内容
6. 実行結果
7. 考察
8. プログラム・リスト

F.1 考察の内容

- オリジナルの考察を書く．他人と同じものは低く評価される (努力をして，考える力やプログラミング力など，実力を付けることが目的)
- 問題解決のためどのような工夫をしたか
- 未解決の問題点，実行してわかったこと．期待した通りの動作をしたか？予想に反した結果，意外な結果が得れれた場合，その理由は何か．