Ｊａｖａ入門

(№.５)

科　名		氏　名	
Tutorial group			

2005.1

1 Abstract Window Toolkit(AWT)

Abstract Window Toolkit(AWT)は、Java の GUI(Graphical User Interface)構築の最も基本となる**ライブラリパッケージ**である。

Abstract Window Toolkit(AWT)の特徴は、次のとおりである。

(1) AWT では、**アプレット**(applet)と**アプリケーション**の GUI を作成することができる。

(2) 一般的なプログラムに必要な最小限の GUI 部品であるため、基本的な GUI 部品は一通り揃っているが、高度な GUI 部品は実装されていない。

(3) 直接 OS の API を利用してコンポーネントの表示を行うため、look&feel は OS に依存する。
⇒ "pure java"ではない。

(4) OS 固有のネイティブコードを利用するため、システムリソースを大量に使用する。このため、"heavy weight component"と呼ばれる。

(5) Swing と比較して、OS の API を利用するため動作速度が速い。

また、Java2 からは AWT とは別に Swing が標準で搭載されている。

Swing は、OS 固有のネイティブコードを全く利用しないため、完全な Java の look&feel を実現できる。しかし、その反面 AWT よりも動作速度がかなり遅い。⇒ **"pure java"**である。

【例題 29】

[ファイル]

P287applet.java

[プログラムの説明]

テキスト・P287～288 applet クラスのプログラムを作成しなさい。ただし、クラス名"applet"はクラス名"P287applet"とすること。

【解説】

アプレット(applet)とは、Web ブラウザに GUI を提供する目的で作成されるクラスファイルである。通常、**アプレット**は HTML の<APPLET>タグによって Web ページに埋め込まれる。

アプレットを作成する場合は、"java.applet.Applet"クラスを継承して新たにクラス定義する。また、"java.applet.Applet"クラスは、"java.awt.Panel"を継承して作成されており、コンテナとしての機能を備えている。

したがって、**アプレット**内に **AWT コンポーネント**を組み込むことが可能である。

Java の**ライブラリパッケージ**の構成は、パスが"java"、"javax"及び"org.omg"で始まるものの三つに分類される。

"java"で始まるものは「基本 API パッケージ」で、プログラミングする上で最も基本的な機能がまとめられている。この分野の**ライブラリパッケージ**には次のようなものがある。

(1) java.applet　… **アプレット**の機能を提供する。

(2) java.awt　　… Abstract Window Toolkit の機能を提供する。
(3) java.lang　　… Java 言語の最も基本的な機能を提供する。

"javax"で始まるものは、Swing 関係の機能がまとめられている。この分野の**ライブラリパッケージ**には次のようなものがある。

(1) javax.accessibility … Swing コンポーネントへのアクセス手段を提供する。
(2) javax.rmi　　… CORBA 準拠のオブジェクト間通信プロトコルを提供する。
(3) javax.swing　　… Swing のためのパッケージを提供する。

"org.omg"で始まるものは、CORBA のためのパッケージである。

これらの**ライブラリパッケージ**は、import 文によって指定する。

【構文】 － import

impoet　*package-name*.*class-name*;

package-name　… *class-name* クラスの検索先を指定する。
class-name　… インポートするクラス名を指定する。特定のパッケージの複数のクラスを指定する場合は、"*"を指定する。

"java.applet.Applet"クラスには、**アプレット**の実行と終了に関する次のメソッドがあらかじめ実装されている。

(1) void　init()
アプレットがロードされた時に、一度だけ呼び出される。**アプレット**の初期化のために処理を用意する場合に用いられる。**アプリケーション**の main()メソッドに相当し、**コンストラクタ**の役割もする。**アプレット**では、**コンストラクタ**の実装は可能だが一般的には実装しない。
(2) void　start()
init()メソッドの実行後に呼び出される。**アプレット**の開始のための処理に用いられる。
(3) void　stop()
アプレット終了時に呼び出される。
(4) void　destroy()
void　stop()メソッドの後の**アプレット**破棄の際に呼び出される。なお、**アプリケーション**における JVM の終了処理(System.exit())は不要である。

また、"java.applet.Applet"クラスは"java.awt.Panel"クラスを継承しているので、そのスーパークラスである"java.awt.Container"クラスや"java.awt.Component"クラスに実装されているメソッドなども使用することができる(テキスト・P281)。

【構文】 － APPLET タグ

<APPLET　CODE=*class-file*　WIDTH=*width*　HEIGHT=*height*　ALIGN=*align*>
</APPLET>

class-file … ブラウザから呼び出すクラスファイルを指定する。
width　　… **アプレット**の横幅を指定する。
height　　… **アプレット**の縦幅を指定する。
align　　… **アプレット**の整列位置(RIGHT、LEFT、CENTER)を指定する。

解説：ブラウザから**アプレット**を呼び出す。

【手順】

① コードウィンドウ[新規 1]に、テキスト・P287～288 <APPLET>タグと applet クラスのプログラムをコーディングする。

② メニューバーから、[ファイル(F)] ⇒ [名前を付けて保存(A)...]と操作し、[ファイル名を付けて保存]ダイアログの[ファイル名(N):]欄に"P287applet.java"と入力し、保存(S)ボタンを click する。

③ Ctrl+F8 を押下して、コンパイルのみを行なう。

⇒ コンパイルエラーが無く、"P287applet.class"が作成されることを確認する。

④ [コマンド(C)]バーから、"appletviewer　P287applet.java"と入力し、実行を click する。

⑤ [Command Prompt]ウィンドウが起動し、**アプレット**が表示される。

⑥ 動作を確認後、[Command Prompt]ウィンドウを閉じる。

【考察】

【例題 30】

[ファイル]

P289applet.java

[プログラムの説明]

テキスト・P289 applet クラスのプログラムを作成しなさい。ただし、クラス名"applet"はクラス名"P289applet"とすること。

また、次のコーディングを変更しなさい。

(1) setBackground(Color.white); → setBackground(Color.yellow);

(2) setForeground(Color.blue); → start()メソッドに追加

【解説】

AWT コンポーネントは、フォアグラウンドとバックグラウンドと言う二つの色を組み合わせて表示される。それぞれ、setForeground()メソッドと setBackground()メソッドによって、表示色の値を設定することで変更できる。

この時、表示色の値として"java.awt.Color"クラスのインスタンスを使用することができる。あらかじめ用意されているインスタンスのフィールドには、次のようなものがある。

black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow

【考察】

【例題 31】－ getParameter()メソッド

［ファイル］

P293applet.java

［プログラムの説明］

テキスト・P293 applet クラスのプログラムを作成しなさい。ただし、クラス名"applet"はクラス名"P293applet"とすること。

getParameter()メソッドについては、テキスト・P282 を参考にすること。

【考察】

【例題 32】－ Java console

［ファイル］

P294applet.java

［プログラムの説明］

テキスト・P294 applet クラスのプログラムを作成しなさい。ただし、クラス名"applet"はクラス名"P294applet"とすること。

【考察】

【例題 33】－ TextField

［ファイル］

P296applet.java

［プログラム］

```
//  Text フィールドを扱うプログラム(P296)
import  java.lang.*;
import  java.applet.Applet;
import  java.awt.*;

/*<APPLET
```

```
    CODE = P296applet.class
    WIDTH = 200
    HEIGHT = 200>
</APPLET>*/

public  class   P296applet  extends Applet  {
    public  TextField   text1;              //  データメンバ

    public  void    init(){
        text1 = new TextField(20);          //  オブジェクトの生成
        text1.setText("Hello from Java!");  //  TextFieldに文字列を格納
        add(text1);                         //  コンテナへコントロールの追加

        setBackground(Color.yellow);        //  追加コーディング
    }
}
```

【考察】

【例題 34】－ Button

［ファイル］

P297applet.java, P297applet1.java

［プログラム１］

"P296applet.java"からコピーして良い。

```
//  Buttonコントロールを扱うプログラム(P297)
import  java.lang.*;
import  java.applet.Applet;
import  java.awt.*;

/*<APPLET
    CODE = P297applet.class
    WIDTH = 200
    HEIGHT = 200>
</APPLET>*/

public  class   P297applet  extends Applet  {
    public  TextField   text1;              //  データメンバ
    Button  button1;

    public  void    init(){
        text1 = new TextField(20);          //  オブジェクトの生成
        text1.setText("Hello from Java!");  //  TextFieldに文字列を格納
        add(text1);                         //  コンテナへコントロールの追加
```

```
        button1 = new Button("Click Here!");//  オブジェクトの生成
        add(button1);                       //  コンテナへコントロールの追加

        setBackground(Color.cyan);          //  追加コーディング
    }
}
```

【考察】

・ Button コントロールを Click して、動作を確認しなさい。

[プログラム２]

"P297applet.java"からコピーして良い。

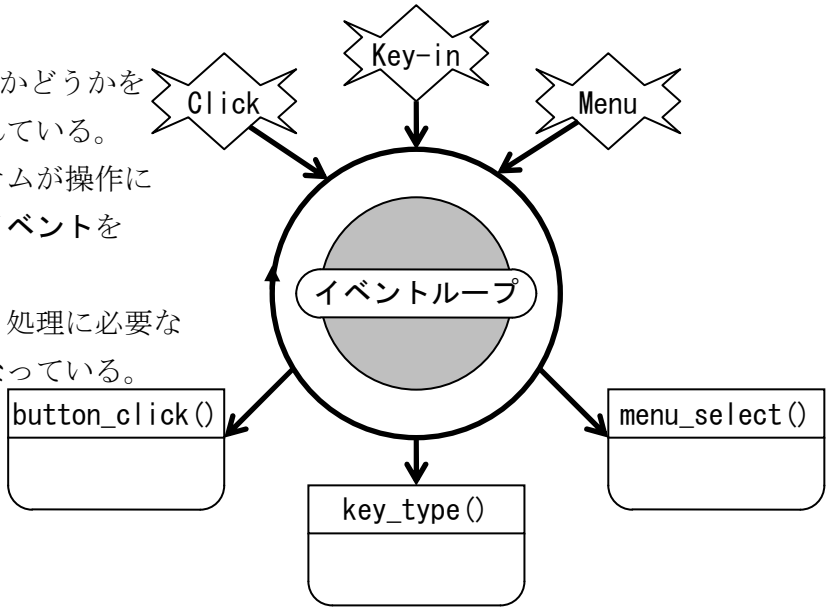
```
//  Button コントロールを扱うプログラム(P297)
import  java.lang.*;
import  java.applet.Applet;
import  java.awt.*;

/*<APPLET
    CODE = P297applet1.class
    WIDTH = 200
    HEIGHT = 200>
</APPLET>*/

public  class   P297applet1 extends Applet  implements  ActionListener  {
    public  TextField   text1;              //  データメンバ
    Button  button1;

    public  void    init(){
        text1 = new TextField(20);          //  オブジェクトの生成
        text1.setText("Hello from Java!");  //  TextFieldに文字列を格納
        add(text1);                         //  コンテナへコントロールの追加

        button1 = new Button("Click Here!");//  オブジェクトの生成
        button1.addActionListener(this);
        add(button1);                       //  コンテナへコントロールの追加

        setBackground(Color.cyan);          //  追加コーディング
    }
    //  追加処理コーディング
    public  void    actionPerformed(ActionEvent e){
        setBackground(Color.green);
    }
}
```

【考察】

イベント処理(event processing)

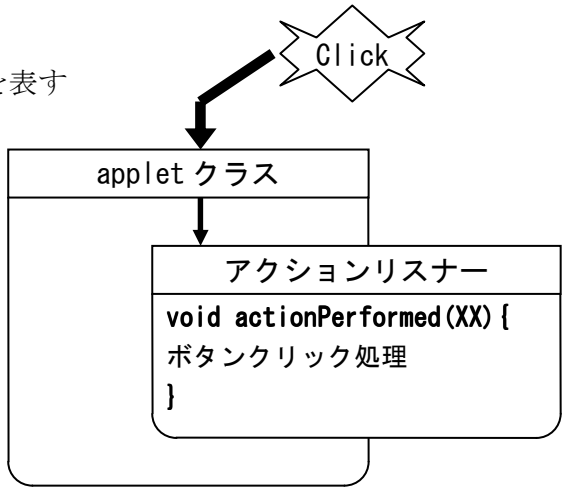
イベントとは、GUI における各種の操作や処理の要求であり、**メッセージ**である。
イベント処理とは、この**イベント**が発生した場合、あらかじめ組み込まれた**イベントハンドラ**を呼び出してその**イベント**に対処する処理を行ない、その後実行中のプログラムに制御を返すことである。

GUI では、常に**イベント**が発生したかどうかを確認する**イベントループ**が実行されている。
ユーザが GUI を操作すると、システムが操作に応じた**イベント**を発生させ、この**イベント**を**イベントループ**内で検出させる。
次に、発生した**イベント**に応じて、処理に必要なサブルーチンを呼び出す仕組みになっている。
これらのプログラムによる一連の操作を**イベントシステム**と言う。



Java では、**代理イベントモデル(delegate event model)**と呼ばれる**イベントシステム**を実装する(Java 1.1 以降)。
代理イベントモデルでは、コンポーネント本体と**イベント処理**を分離し、コンポーネント本体に**イベント処理**の機能を実装しない。すなわち、**イベント**は**イベント処理**専用の**イベントリスナー(event listener)**と言うクラスによって処理される。また、**イベントリスナー**は**イベント**の種類ごとに多数用意されており、各**イベントリスナー**内にはその**イベント**で使用する**イベントハンドラ**のメソッドが用意されている。
Java における**イベント**は、**ソース**における状態変更を表すオブジェクトであり、**ソース**により生成される。
したがって**ソース**には次のような機能が必要となる。

(1) **イベント**の生成。
(2) **リスナー**が特定のタイプの**イベント**に関する通知を登録あるいは登録解除するメソッドの提供。
(3) 登録されているすべての**リスナー**への**イベントメッセージ**の送信。



【構文】－ イベントリスナー登録メソッド(ソース:java.awt.Button)

void addActionListener(ActionListener *al*)

al … ActionListener と呼ばれるイベントリスナーオブジェクトを指定する。

解説：Button クラスに ActionListener を追加する。これにより、Button クラスは Button が Click された時にイベントメッセージを受け取ることができる。

【構文】－ イベントリスナー登録解除メソッド(ソース:java.awt.Button)

void removeActionListener(ActionListener *al*)

al … ActionListener と呼ばれるイベントリスナーオブジェクトを指定する。

解説：Button クラスから ActionListener を削除し、Button クラスからイベントメッセージを受け取らないようにする。

次に、リスナーに必要となる機能を示す。

(1) 特定のイベントメッセージを受け取ることを登録。

(2) 前述(1)で登録したイベントメッセージを受け取るインタフェースの実装。

(3) 不要となった前述(1)の登録解除。

【構文】－ インタフェース実装メソッド(ActionListener インタフェース)

void actionPerformed(ActionEvent *ae*);

ae … アクションイベントオブジェクトを指定する。

解説：イベントメッセージを受け取るアクションリスナーを実装する。

なお、ActionListener インタフェースには、このメソッドだけが定義されている。

表 イベントクラスの種類とソース

ActionEvent	Menu、List、Button 操作
AdjustmentEvent	Scroll 操作
ComponentEvent	Component の状態変化
ContainerEvent	Component の追加、削除
FocusEvent	Component Focus の変化
ItemEvent	CheckBox、List 選択の変化
KeyEvent	Key 操作
MouseEvent	Mouse 操作
TextEvent	TextArea、TextField の変化
WindowEvent	Window 状態の変化

【例題 35】

[ファイル名]

P301applet.java

[プログラムの説明]

テキスト・P301 applet クラスのプログラムを作成しなさい。ただし、クラス名"applet"はクラス名"P301applet"とすること。

なお、実習については次のとおりとする。

(1) P301 のプログラムを入力後、Ctrl+F8 を押下して、コンパイルのみを行なう。

(2) [CPad: **コンパイル終了**]を確認後、actionPerformed()メソッドをコメントアウトする。

(3) Ctrl+F8 を押下して、コンパイルのみを行い、メッセージを確認する。

(4) 前述(2)でコメントアウトした actionPerformed()メソッドを元に戻し実行しなさい。

【構文】－ Button クラスコンストラクタ

Button()　　… ラベル無しの Button オブジェクトを作成する。

Button(String *label*)　… 指定のラベルを持つ Button オブジェクトを作成する。

label　　… Button オブジェクトに表示するラベル(文字列)を指定する。

解説：指定のラベルを持つ Button オブジェクトを作成した場合、Button オブジェクトの Caption にその文字列が格納され、表示テキストとなる。また、デフォルトでは、この Caption 値が**アクションイベント**に関連付けられた**コマンド文字列**となる。

【構文】－ getSource()メソッド

Object　getSource()

解説：**イベント**を生成した Object クラスのオブジェクトを返す。

【考察】

・ 前述(3)の意味を考察しなさい。

【例題 36】

［ファイル名］

P303applet.java

［プログラムの説明］

テキスト・P302～P303 applet クラスと a クラスのプログラムを作成しなさい。ただし、クラス名"applet"はクラス名"P303applet"と、クラス名"a"はクラス名"P303a"とすること。

【考察】

【例題 37】

［ファイル名］

P304applet.java

［プログラムの説明］

テキスト・P304 applet クラスのプログラムを作成しなさい。ただし、クラス名"applet"はクラス名"P304applet"とすること。

【構文】－ getActionCommand()メソッド

String getActionCommand()

解説：ActionListener が、Button オブジェクトで生成された**アクションイベント**に関連付けられた**コマンド文字列**を返す。

【考察】

【実験 1】

［ファイル名］

P301applet1.java

［プログラムの説明］

"P301applet.java"のプログラムを、次のように変更して実行しなさい。ただし、クラス名は"P301applet1"に変更すること。

button1 = new Button("Click Here!"); ⇨ button1 = new Button();

【考察】

- Button オブジェクトを、**デフォルトコンストラクタ**で初期化した場合の動作を考察しなさい。

【実験 2】

［ファイル名］

P304applet1.java

［プログラムの説明］

"P304applet.java"のプログラムを、次のように変更して実行しなさい。ただし、クラス名は"P304applet1"に変更すること。

button1 = new Button("Click Here!"); ⇨ button1 = new Button();

【考察】

- Button オブジェクトを、**デフォルトコンストラクタ**で初期化した場合の動作を考察しなさい。

【例題 38】

［ファイル名］

P305applet.java

［プログラムの説明］

テキスト・P305 First クラスのプログラムを作成しなさい。ただし、クラス名"First"はクラス名"P305First"とすること。

【考察】

【例題 39】

［ファイル名］

P307applet.java

［プログラムの説明］

テキスト・P306～P307 newButton クラスと applet クラスのプログラムを作成しなさい。ただし、クラス名"newButton"はクラス名"P306newButton"と、クラス名"applet"はクラス名"P307applet"とすること。

【解説】

AWTEvent 抽象クラスは、EventObject クラスを拡張したもので、"java.awt"パッケージに含まれている。したがって、"AWTEvent.ACTION_EVENT_MASK"は AWTEvent 抽象クラス内に定義されている定数である。

【考察】

アダプタクラス(adapter class)

Java における**インタフェース**とは、**定数**と**抽象メソッド**のみを宣言したクラスであるため、implements キーワードによって**インタフェース**を実装したクラスでは、その**インタフェース**中に定義されているすべてのメソッド(**イベントハンドラ**)の定義をしなければならない。

しかし、**イベント処理**では**インタフェース**中のすべてのメソッド(**イベントハンドラ**)が必要であるとは限らないが、その**インタフェース**で定義されているすべてのメソッド(**イベントハンドラ**)の定義を求められ、コーディングが煩雑になる。

このため、Java ではあらかじめ**リスナークラス**を implements キーワードによってクラスに実装し、その**リスナークラス**中に定義されているすべてのメソッド(**イベントハンドラ**)を定義したクラスを用意している。このクラスを**アダプタクラス**と言う。

このため、**アダプタクラス**は通常のクラスであることから、**extends** キーワードによって継承するため、多重継承は許可されない。

表 **アダプタクラス**と**リスナークラス**

ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

アダプタクラスか**リスナークラス**か？

アダプタクラスを使用すると**リスナークラス**に比べてコーディングは低減されるが、常に別のクラスとして用意しなければならない。

したがって、いつも**アダプタクラス**を使うことが正しい選択とは言えない。

【例題 40】

［ファイル名］

P308applet.java

［プログラムの説明］

テキスト・P307～P308 applet クラスと ma クラスのプログラムを作成しなさい。ただし、クラス名"applet"はクラス名"P308applet"と、クラス名"ma"はクラス名"P308ma"とすること。

【考察】

【実験 3】

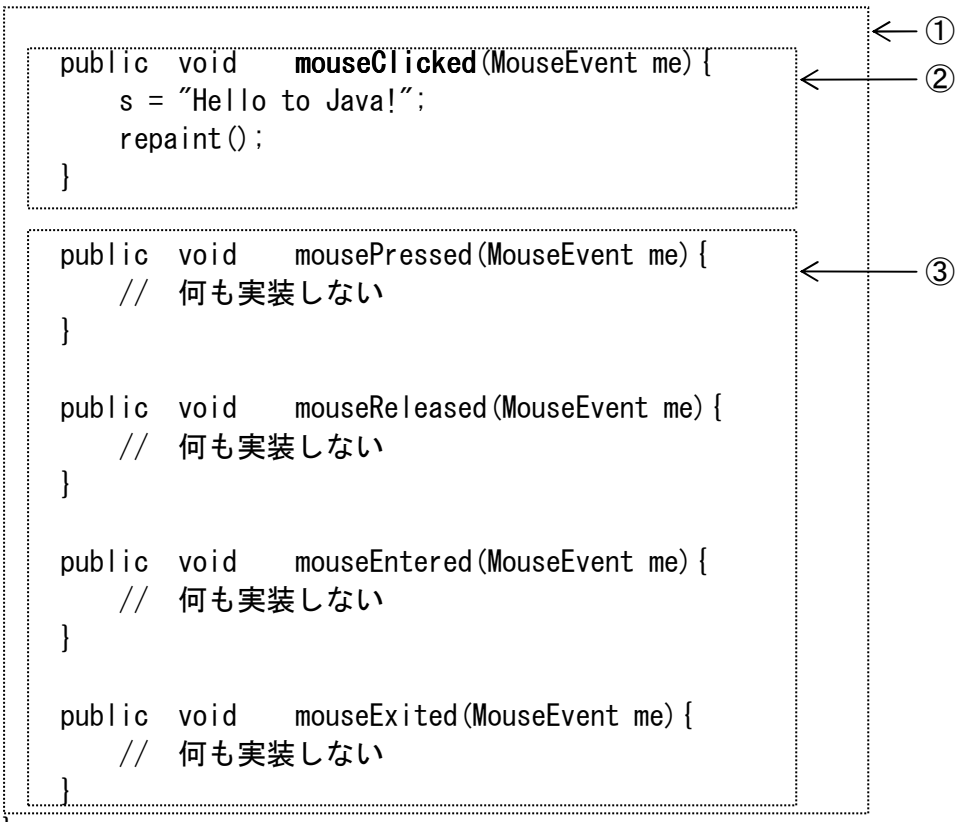
［ファイル名］

P308applet1.java

［プログラム］

```
import  java.lang.*;
import  java.applet.Applet;
import  java.awt.*;
import  java.awt.event.*;

/*<APPLET
    CODE = P308applet1.class
    WIDTH = 200
    HEIGHT = 200>
</APPLET>*/

public  class   P308applet1 extends Applet  implements  MouseListener   {
    public  String  s = "Hello from Java!";

    public  void    init(){
        addMouseListener(this);             //  イベントリスナーの登録
```

```
    }

    public  void    paint(Graphics g){
        g.drawString(s, 60, 100);
    }
```



```
    public  void    mouseClicked(MouseEvent me){
        s = "Hello to Java!";
        repaint();
    }

    public  void    mousePressed(MouseEvent me){
        //  何も実装しない
    }

    public  void    mouseReleased(MouseEvent me){
        //  何も実装しない
    }

    public  void    mouseEntered(MouseEvent me){
        //  何も実装しない
    }

    public  void    mouseExited(MouseEvent me){
        //  何も実装しない
    }
}
```


【解説】

- MouseListener **インタフェース**中に宣言されている**抽象メソッド**　… ①
- 今回、**アプレット**の動作の上で必要となるメソッドの実装　　… ②
- 今回、**アプレット**の動作の上で不要なメソッドの定義　　… ③
- ③ブロック全体をコメントアウトして、コンパイルしなさい。また、コンパイル結果について考察しなさい。

【考察】

- **【例題 40】**のコーディングと比較し、考察しなさい。

【例題 41】

［ファイル名］

P310applet.java

［プログラムの説明］

テキスト・P309～P310 applet クラスのプログラムを作成しなさい。ただし、クラス名"applet"

はクラス名"P310applet"とすること。

【解説】

匿名内部アダプタクラスを使用すると、**アダプタクラス**のサブクラスとして別に作成する必要が無くなる。

【考察】

.....

.....

.....

.....

ウィンドウ型アプリケーション(Frame を使ったアプリケーションクラス)

Java AWT を利用するアプリケーションは、Frame クラスを継承して作成する。

【例題 42】

［ファイル名］

P313app.java

［プログラムの説明］

テキスト・P312～P313 app クラスと AppFrame クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P313app"と、クラス名"AppFrame"はクラス名"P313AppFrame"とすること。

［プログラム］

```
import  java.awt.*;
//  Frame クラスのサブクラスを定義
class   P313AppFrame    extends Frame   {
    public  void    paint(Graphics g){      //  paint()メソッドのオーバーライド
        g.drawString("Hello from Java!", 60, 100);
    }
}

public  class   P313app {
    public  static  void    main(String args[]){
        P313AppFrame    f = new P313AppFrame();
        //  setSize()メソッドによるコンポーネントサイズの変更
        f.setSize(200, 200);
        //  show()メソッドによるコンポーネントの表示
        f.show();
    }
}
```

【解説】

AWT の Frame クラスアプリケーションは、GUI の表示や操作に OS の API を使用しているため、表示する Look&Feel は OS に依存する。

このため、ウィンドウの最大化や最小化の機能は、アプリケーションで実装しなくても機能する。しかし、アプリケーションの終了は、アプリケーションに実装しなければならない。

【考察】

【例題 43】

［ファイル名］

P316app.java

［プログラムの説明］

テキスト・P315～P316 app クラスと AppFrame クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P316app"と、クラス名"AppFrame"はクラス名"P313AppFrame"とすること。

【考察】

【例題 44】

［ファイル名］

Samp44.java, Samp44a.java

［プログラム１］

```
import  java.lang.*;
import  java.awt.*;
import  java.awt.event.*;

public  class   Samp44  {
    public  static  void    main(String args[]){
        //  直接Frameクラスのオブジェクトを作成する
        Frame   mf = new Frame();
        mf.addWindowListener(new WindowAdapter(){
            public  void    windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        mf.setSize(300,200);
        mf.show();
    }
}
```

［プログラム２］

```
import  java.lang.*;
import  java.awt.*;
```

```
import  java.awt.event.*;

public  class   Samp44a {
    public  static  void    main(String args[]){
        //  直接Frameクラスのオブジェクトを作成する
        Frame       mf = new Frame();
        TextField   tf = new TextField("Hello from Java!");  }追加
        Button      bt = new Button("Click Here!");          }
        mf.addWindowListener(new WindowAdapter(){
            public  void    windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        mf.add(bt);  }追加
        mf.add(tf);  }
        mf.setSize(300,200);
        mf.show();
    }
}
```

※［プログラム１］に"追加"部分を、追加コーディングしたものである。

【考察】

・ **【例題 43】**のプログラムと比較し、なぜFrameクラスを**継承**しなければならないのかを考察しなさい。

【例題 45】

［ファイル名］

P317applet.java

［プログラムの説明］

テキスト・P316～P317 appletクラスとAppFrameクラスのプログラムを作成しなさい。ただし、クラス名"applet"はクラス名"P317applet"と、クラス名"AppFrame"はクラス名"P318AppFrame"とすること。

【考察】