ウィンドウ操作を見直す

1. クラス間で変数の共有

　staticをつけることで、クラス内のどのメソッドからもその変数を利用することができるようになる。変数を定義したクラス内では、その変数名で使うことができる。

　もし、他のクラスからその変数を使いたいときはどのクラスの変数なのかを示す必要があるので、クラス名.変数名のように書けば利用できる。

　下のプログラムでは、他のクラスで変数に代入してから元のクラスでちゃんと代入ができているかを確かめている。

|  |
| --- |
| public class NetworkAnalyzer {  static String device;  public static void main(String[] args){  device = "any";  Window w = new Window();  System.out.println(device);  }  }  class Window{  /\*  \* コンストラクタ  \*/  Window(){  NetworkAnalyzer.device = "enp2s0";  }//コンストラクタおわり    }//Windowクラスおわり |

2. インターフェースの選択を組み合わせる

　ネットワークアナライザを起動したら最初に表示するインターフェースの選択操作を実装する。以下のプログラムでは、次のような手順で動作する。

(1) デバイスの一覧を取得

(2) ウィンドウを生成し、ラジオボタンでデバイス名を選択可能にする

(3) キャプチャスタートボタンが押されたら、選択されたデバイス名を格納して終了。✕ボタンが押されたら、何　もせずに終了。

　これからプログラムを作っていく上で大事なのは、変数のスコープ。青地で示した部分は、わざわざコンストラクタの外で一度宣言だけを行っている。こうすることで、ActionListenerやWindowListenerに関係するメソッドの中でも同じ変数を扱えるようになる。

　緑字の部分はActionListenerに、橙字の部分はWindowListenerに関係する箇所である。ActionListenerとWindowListenerはどちらもインターフェースのため、使う使わないにかかわらずメソッドだけは宣言しておく必要がある。そのため、特にWindowListenerは使っていないけど宣言だけしているメソッドがいくつもある。

|  |
| --- |
| public class NetworkAnalyzer{  static String device;  /\* main関数 すべてを統括する \*/  public static void main(String[] args){  device = "any";  Window frame = new Window();  }  }  class Window extends JFrame implements ActionListener,WindowListener{  static {System.loadLibrary("finddevice");}  native static void finddevice(String[] device\_list);  /\*いろんなメソッドから使えるようにフィールドで宣言\*/  static JRadioButton[] radio;  static JButton start\_button;  static JFrame SelectDevice;  /\*  \* コンストラクタ  \* デバイス一覧の取得→ウィンドウの生成まで  \*/  Window(){  String[] device\_list = new String[10]; //デバイスリスト格納用  /\*デバイスリスト格納\*/  finddevice(device\_list);  SelectDevice = new JFrame("インターフェースの選択");  SelectDevice.setSize(500,100);  SelectDevice.setLocationRelativeTo(null);  SelectDevice.setVisible(true);  SelectDevice.addWindowListener(this); //WindowListenerに登録    /\*ラジオボタンにデバイス名をセット\*/  int count=0;  /\*nullじゃないデバイス名の個数だけラジオボタンを作成\*/  while(device\_list[count] != null){  count++;  }  radio = new JRadioButton[count];  for(int i=0;i<radio.length;i++){  radio[i] = new JRadioButton(device\_list[i]);  }    /\*初期状態ではradio[0]を選択した状態\*/  radio[0].setSelected(true);    /\*1つしか選択できないようにグループ化する\*/  ButtonGroup group = new ButtonGroup();  for(int i=0;i<radio.length;i++){  group.add(radio[i]);  }  /\*デバイスの確定ボタン（キャプチャスタート）\*/  start\_button = new JButton("キャプチャスタート");  start\_button.addActionListener(this);  　　　　　　//thisは自分のクラス内のactionPerfomedのみ有効    /\*パネルにラジオボタンとボタンをセット\*/  JPanel p = new Jpanel();  　　　　　　p.setLayout(new GridLayout(radio.length+1,1));  for(int i=0;i<radio.length;i++){  p.add(radio[i]);  }  p.add(start\_button);  /\*コンテナにパネルをセット\*/  Container cPane = SelectDevice.getContentPane();  cPane.add(p);  }//コンストラクタおわり    /\*  \* Windowクラスでのみ有効なイベントクラス  \*/  public void actionPerformed(ActionEvent e) {  Object object = e.getSource();    /\*  \* 最初のインターフェースの選択  \* ラジオボタンで指定されているデバイス名を取得  \*/  if(object == start\_button){  for(int i=0;i<radio.length;i++){  if(radio[i].isSelected()){  NetworkAnalyzer.device = radio[i].getText();  }  }  System.out.println("Windowで表示："+ NetworkAnalyzer.device);  SelectDevice.setVisible(false);  System.exit(0);  }    }//antionPerformedおわり  /\*✕ボタンが押されたとき\*/  public void windowClosing(WindowEvent e){  System.exit(0); //プログラムを終了  }  /\*dispose()が使われたとき（今んとこ使ってない）\*/  public void windowClosed(WindowEvent e){  System.out.println(NetworkAnalyzer.device);  System.exit(0);  }  public void windowOpened(WindowEvent e){    }  public void windowIconified(WindowEvent e){    }  public void windowDeiconified(WindowEvent e){    }  public void windowActivated(WindowEvent e){    }  public void windowDeactivated(WindowEvent e){    }  }//Windowクラスおわり |

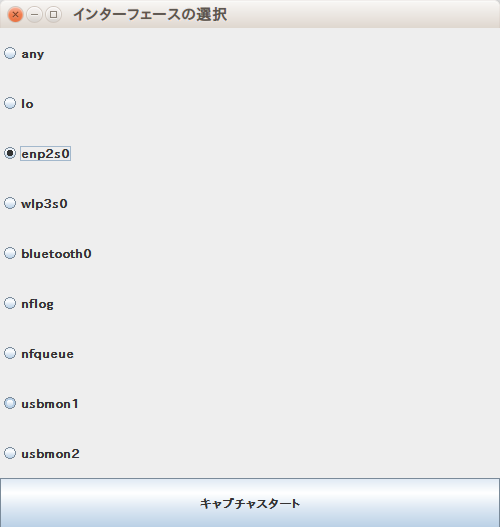
　コンパイル。青字の部分に注意。最初のJavaファイルのコンパイルのときはJavaファイル名を使うけど、ヘッダファイルを作るときには、Çプログラムを呼び出す役割がある**Javaクラス名**を指定するので気をつける。

|  |
| --- |
| notti@notti:~/static$ javac NetworkAnalyzer.java  notti@notti:~/static$ javah -cp ~/static Window  notti@notti:~/static$ gcc -fPIC -shared finddevice.c -lpcap -I /usr/lib/jvm/java-8-oracle-64/include -I /usr/lib/jvm/java-8-oracle-64/include/linux/ -o libfinddevice.so |

　実行結果。ここでも注意が必要で実行するときに指定するのはmainメソッドがあるJavaクラス名である。緑字は、soファイルの場所を教えるために書く。

　この実行結果は、キャプチャスタートボタンを押したときの実行結果である。図のようにenp2s0が選択されていた状態でキャプチャスタートボタンを押したので、enp2s0が表示されている。

|  |
| --- |
| notti@notti:~/static$ java -cp ~/static -Djava.library.path=. NetworkAnalyzer  Windowで表示：enp2s0 |



3. メインウィンドウをつける

　メインウィンドウを表示するmainWindowメソッドをWindowクラス内に作成する。

　まず、メインウィンドウで使用するコンポーネントと、変数をWindowクラスのフィールドに宣言する。

|  |
| --- |
| /\*メインウィンドウで使うコンポーネント\*/  　　　static JFrame MainWindow;  static JMenuBar JBar = new JMenuBar(); //メニューバー  static JMenu[] cJMenu = new JMenu[3]; //メニューバーに並べる項目  static JMenuItem[] menuitem = new JMenuItem[10]; //各項目の中身  static JButton[] btn = new JButton[5]; //アイコン  static JTextField filter = new JTextField(""); //表示フィルタ  static JFileChooser filechooser = new JFileChooser(); //ファイルを開くとき  static JTextArea Packet = new JTextArea(); //パケット一覧]  static JTextArea Packet\_details = new JTextArea(); //パケット詳細  static JTextArea Packet\_bytes = new JTextArea(); //パケットデータ  static JLabel footer=new JLabel(); //フッタ  static String[][] packet ={{"","","","","","",""}}; //表の中身を格納  static String[] column = {"No","Time","Source","Distination","Protocol","Length","Information"};//カラム名  static DefaultTableModel tablemodel = new DefaultTableModel(column, 0 ) ;//テーブルモデルを作成  static JTable Packet\_list = new JTable(tablemodel); //テーブルを作成  static int pcount=4869; //パケット数  static int pdisp =4869; //内表示数  static double percent=100.0; //表示割合  static String profile ="Default"; //プロファイル |

　mainWindowメソッドの中身は長いので省略。

　次に、mainWindowメソッドの呼び出しタイミングを考える。このメソッドはデバイスの選択を行ったあとに表示されればいいので、キャプチャスタートボタンのActionEventでデバイスを取得したあとにmainWindowメソッドの呼び出しを追加する。

|  |
| --- |
| else if(object==Window.start\_button){  while(true){  System.out.printf("%d",NetworkAnalyzer.c);  if(NetworkAnalyzer.device != "0") break;  //c++;  }  mainWindow();  }  } |

4. Functionクラス

　このクラスは、メインウィンドウ上のボタンが押されたときに、Window.actionListenerメソッドを通じて呼び出される機能を持つクラスである。

5. JNIクラス

　このクラスは、ネイティブライブラリを呼び出すクラスである。複数クラスでネイティブライブラリのロードを行うと、ヘッダファイルが増えるしよくわかんなくなるから1つにまとめるために作成した。

　以下のプログラムはまだ作成途中だけど、こんな感じで一括してライブラリのロードをしている。。だから、ネイティブライブラリを使いたいと思ったら、このクラスを必ず通じて使うことになる。

|  |
| --- |
| public class JNI extends JFrame implements ActionListener{  /\*ライブラリのロードとネイティブメソッドの宣言\*/  static {System.loadLibrary("finddevice");}  static {System.loadLibrary("packet");}  //static {System.loadLibrary("capture");}  native static void finddevice(String[] device\_list);  native static float packet(int i\_packet[],String[] s\_packet);  //native static float capture(String str,int i\_packet[],String[] s\_packet);  static String[] device\_list = new String[10]; //デバイスリスト格納用  /\*  \* デバイスの一覧を取得して、配列device\_listに格納  \*/  public static void GetDeviceList(){  finddevice(device\_list);  }  /\*  \* パケットをキャプチャして配列に格納  \* ある程度たまったらデータベースに格納する予定  \*/  public static void PacketCapture(){    }  } |

6. Databaseクラス

　データベースに接続してパケット情報を挿入したり、検索したりするクラス。パケットの情報を挿入するinsertDB()とログの閲覧を行うopenDB()がある。

6.1 実行までの流れ

　データベースを操作するためには、JDBCドライバが必要となる。これはjarファイルとして用意されておりコンパイル時と実行時にjarファイルを指定してやらないといけない。以下にコンパイルと実行が成功するまで入力したコマンドを示す（青字は成功したコマンド）。

　まず-”メイン・クラスNetworkAnalyzerが見つからなかったかロードできませんでした”というエラーが出力される原因を説明する。-cpをなにも設定しない場合、このエラーは出力されない。なぜならクラスパスはデフォルトではカレントディレクトリが設定されているからである。しかし、-cpでjarファイルなどのクラスパスを設定すると、元のカレントディレクトリの設定は上書きされなくなってしまう。そのため、カレントディレクトリにあるファイルでさえわからなくなってしまう。だからカレントディレクトリの設定も同時に行う必要がある。-cpで2つ以上のクラスパスを同時に使用する場合、コロン：で区切る。最初の方のコマンドではカレントディレクトリをhomeからのパスで表している。しかし、実行できていない、以下のリストでは、他にもパスの設定がうまくできていないと考えてsudoをなくしたり、-Eをつけたりしている。結局、カレントディレクトリを表す部分をピリオドに変えるとうまく動作した。何が違うんだろう？ homeからじゃいけないのかな？

|  |
| --- |
| notti@notti:~/static$ javac -cp ~/db\_jni/mysql-connector-java-5.1.40-bin.jar NetworkAnalyzer.java Function.java JNI.java Database.java  notti@notti:~/static$ sudo java -cp ~/static/mysql-connector-java-5.1.40-bin.jar:~/static NetworkAnalyzer  エラー: メイン・クラスNetworkAnalyzerが見つからなかったかロードできませんでした  notti@notti:~/static$ sudo -E java -cp ~/static/mysql-connector-java-5.1.40-bin.jar:~/static -Djava.library.path=. NetworkAnalyzer  エラー: メイン・クラスNetworkAnalyzerが見つからなかったかロードできませんでした  notti@notti:~/static$ java -cp ~/static/mysql-connector-java-5.1.40-bin.jar:~/static -Djava.library.path=. NetworkAnalyzer  エラー: メイン・クラスNetworkAnalyzerが見つからなかったかロードできませんでした  notti@notti:~/static$ sudo java -cp ~/static/mysql-connector-java-5.1.40-bin.jar:. -Djava.library.path=. NetworkAnalyzer  any  Fri Jan 06 10:02:00 JST 2017 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCertificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.  MySQLに接続  成功 |