第七週 Java 講座

今日は、難しい話はなしで、プログラミングがしたくなるようなことをまとめた。

目次

API（Application Program Interface）とは

ソフトウェア同士をつなぐインターフェース。プログラミングをする際に、API を使用することで誰かが作ったデータ構造や特殊な操作を自分のプログラム上で呼び出せる。
例えば、グラフィック処理を行う OpenglAPI を使うことで手軽に CG を描画させることができる（Minecraft はこの Opengl を使って作られている）。

Swing

Java で GUI アプリケーションを作る際に使われる API。フレーム（ウィンドウ）の中にコンポーネント（ボタンやラベル）を追加していき、アプリケーションを作成する。

Swing では、それぞれのコンポーネントにイベントを設定することでアプリを作成していく。

簡単に Swing アプリケーションの作成方法をまとめると以下のようになる。

1. フレームを作成する。
2. フレームにコンポーネントを追加する。
3. コンポーネントにイベント追加する。

実際に、Swing アプリケーションを作っていく。上記の流れに沿って作成していく。

Swing 1.フレームを作成する

```
package week7;

import javax.swing.JFrame;

public class Week7{

	public static void main(String[] args){
		//フレームを作成する
		JFrame jframe = new JFrame("Swing App");
		//表示サイズを決定する
		jframe.setSize(300, 200);
		//フレームを表示する
		jframe.setVisible(true);
	}
}
```

フレームを作成する過程では、JFrame クラスをインスタンス化させる。コンストラクタに文字列を渡すと、ウィンドウのタイトルを決めることができる。サイズの指定もしておくと実行した時に見やすくなる。

JFrame は生成しただけでは、ウィンドウとして現れないので、setVisible メソッドを使い表示させる必要がある。

Swing 2.フレームにコンポーネントを追加する

```
package week7;

import java.awt.BorderLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class Week7{

	public static void main(String[] args){
		JFrame jframe = new JFrame("Swing App");
		jframe.setSize(300, 200);
		//コンポーネントを生成
		JLabel label = new JLabel("Hello Swing");
		JButton btn = new JButton("Button");
		JTextField field = new JTextField("field");
		//フレームにコンポーネントを追加する
		jframe.add(label,BorderLayout.NORTH);
		jframe.add(btn,BorderLayout.CENTER);
		jframe.add(field, BorderLayout.SOUTH);
		jframe.setVisible(true);
	}
}
```

今回使用したコンポーネントは JLabel,JButton,JTextField である。JLabel は、文字を表示するコンポーネントでキーボードを使って書き込むことができない。JButton はマウスでクリックできるボタンを提供する。JTextField はキーボードで書き込むことのできる領域を作成する。
フレームにコンポーネントを追加する際には、配置を指定することで好きな場所にコンポーネントを表示させられる。現在は、BorderLayout を使って相対的にコンポーネントを配置しているが、座標で指定することも可能である。
次に、イベントを追加する。

ここで、一度ソースをリファクタリングする。

```
public class Week7{
	JLabel label;
	JButton btn;
	JTextField field;
	JFrame jframe;

	public static void main(String[] args){
		new Week7();
	}

	public Week7(){
		jframe = new JFrame("Swing App");
		jframe.setSize(300, 200);

		label = new JLabel("Hello Swing");
		btn = new JButton("Button");
		field = new JTextField("field");

		jframe.add(label,BorderLayout.NORTH);
		jframe.add(btn,BorderLayout.CENTER);
		jframe.add(field, BorderLayout.SOUTH);
		jframe.setVisible(true);
	}
}
```

Swing 3.コンポーネントにイベントを追加する

今回は、JTextField に書き込まれた内容を JButton の押下に対応して、JLabel に書き込むイベントを追加する。

```
package week7;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class Week7{
	JLabel label;
	JButton btn;
	JTextField field;
	JFrame jframe;

	public static void main(String[] args){
		new Week7();
	}

	public Week7(){
		jframe = new JFrame("Swing App");
		jframe.setSize(300, 200);

		label = new JLabel("Hello Swing");
		btn = new JButton("Button");
		field = new JTextField("field");
		//JButtonがクリックされた時のイベントを追加する
		btn.addActionListener(new ButtonClickListener());
	}
```

次のページに続く

```
                jframe.add(label,BorderLayout.NORTH);
                jframe.add(btn,BorderLayout.CENTER);
                jframe.add(field, BorderLayout.SOUTH);
                jframe.setVisible(true);
        }
        //イベントを追加するには、ActionListenerを継承して
        //押下時にする処理を書きたい場合は、
        //actionPerformedメソッドをオーバーライドする
    public class ButtonClickListener implements ActionListener{

                public void actionPerformed(ActionEvent event){

                        String tmp = field.getText();
                        label.setText(tmp);
                }
        }
}
```

コンポーネント内の addActionListener メソッドにイベント処理を記述したクラスのコンストラクタを渡すことでコンポーネントにイベントを追加することができる。
イベントを作る際には、ActionListener クラスを継承して、actionPerformed メソッドをオーバライドして、その中にコンポーネントをクリックした時に行って欲しい処理を記述する。
コンポーネントには、setText(),getText()メソッドがあり、コンポーネントに書かれた文字列を変更したり、文字列を受け取ったりできる。

リスナ(Listener)とは、指定されたアクション（ActionListener の場合はコンポーネントのクリック）が起きた時に予め決められた処理を実行するオブジェクト。
ウィンドウに対して作用する WindowListener、マウスに対して作用する MouseListener などが存在している。

JavaFx

Java7 までは、Swing が java 標準の GUI として使われていた。しかし、最新バージョン Java8 では代わりに JavaFx が標準 GUI に選ばれた。C 棟演習室の PC には Java8 が導入されていないので、スクリーンで実演だけを行う。
先ほどの Swing で作成したアプリと同じ操作をするコードを書いてみる。

```
package fxapp;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class Fxapp extends Application{

        Label label;
        TextField field;
        Button button;

    public static void main(String[] args) {
            launch(args);
    }
```

次のページに続く。

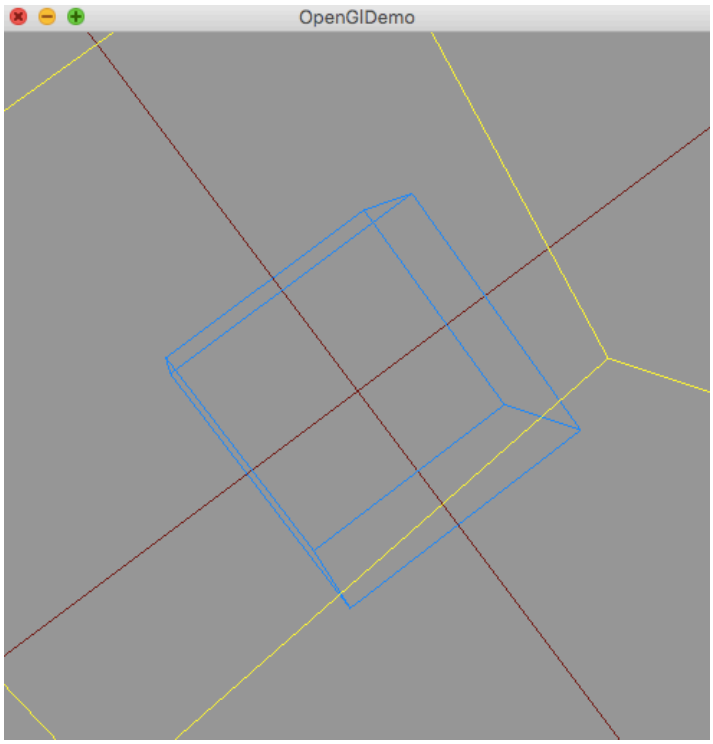
```
	public void start(Stage stage) throws Exception {
		label = new Label("This is JavaFX!");
		field = new TextField();
		button = new Button("Click");
		BorderPane pane = new BorderPane();
		pane.setTop(label);
		pane.setCenter(field);
		pane.setBottom(button);

		button.setOnAction((event)->{label.setText(field.getText());});

		Scene scene = new Scene(pane, 320, 120);
		stage.setScene(scene);
		stage.show();
	}
}
```

JavaFx では、start メソッドに処理をオーバライドする。メソッド名やクラス名が違うが、基本的には Swing と大差ないソースの書き方をしている。ただ、イベント処理部分では、クラスを作成せずに、イベント発生時に行って欲しい処理をラムダ式で記述し、setOnAction メソッドに渡す。ラムダ式は、Java8 で新しく追加された。

JavaFx では、xml で画面の配置を指定することもできる。

OpenGl

OpenGl を使用すると、3DCG を描画させることができる。二年後期の PBL で使用するので、やっておくといいカモ。
プログラムソースは、ほぼトレスなので、割愛。



swing のリスナと合わせて、キーボード処理やマウス処理を行うことができる。

AndroidAPI

AndroidOS で動くアプリを作る際に使用する API。OS のバージョン毎に API が用意されている。基本的には、IDE と平行して利用する。主な IDE としては、AndroidStudio,Eclipse などがある。AndroidStudio の方が特化しているので、オススメ。
Android のアプリを作る際には、Java と簡単な xml を理解する必要がある。
基本的な作成方法は、xml でウィジェット（ボタン、テキストフィールド等）を配置して画面を構成して、Java でイベントを追加する。基本的には、swing や javafx と変わらない。
Swing で作ったものと同じアプリを作ってみた。

MainActivity.Java

```
package com.example.kaimaitakumi.testapp;


import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
   //ウィジェット
    Button button;
    TextView textview;
    EditText edittext;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //xml で定義した画面と連結させる
        button = (Button)findViewById(R.id.button);
        textview = (TextView)findViewById(R.id.textView);
        edittext = (EditText)findViewById(R.id.editText);
        //リスナを設定
        button.setOnClickListener(new ButtonClickListener());
}
}
```

次のページに続く

```
//リスナ
 class ButtonClickListener implements View.OnClickListener {
       public void onClick(View V){
           textview.setText(edittext.getText());
       }
    }



}
```

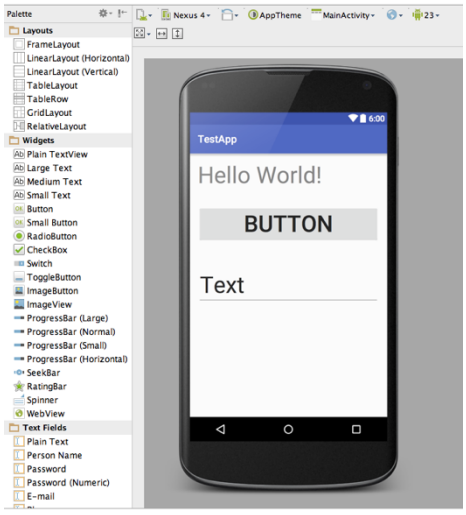
Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.kaimaitakumi.testapp.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:id="@+id/textView"
        android:textSize="20pt"
        android:layout_alignParentEnd="true"
        android:layout_alignParentStart="true" />
```

次のページに続く

```
<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        android:id="@+id/button"
        android:layout_below="@+id/textView"
        android:layout_alignParentStart="true"
        android:layout_marginTop="32dp"
        android:textSize="20pt"
        android:layout_alignEnd="@+id/textView" />


    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:text="Text"
        android:layout_below="@+id/button"
        android:layout_alignParentStart="true"
        android:layout_marginTop="46dp"
        android:textSize="20pt"
        android:layout_alignEnd="@+id/button" />



</RelativeLayout>
```

Androidstudio では、xml をグラフィカルに配置することもできる。



その他 API

TwitterApi

ツイッターには公式以外にも Twitter クライアントと呼ばれる外部の開発者が作成したアプリがある。そのアプリは、ツイッター機能や、フォロー機能などを簡単に実装できる TwitterApi で作られているのがほとんどだ。自分だけの Twitter クライアントを作るのもいいかもしれない。

FaceBookApi

TwitterApi と同じ感じ。

GoogleApi

Gmail,GoogleCalendar,GoogleMap などのサービスにアクセスできる API。有料。

API が存在しないアノ Web サービスを使いたい時

例えば、エキサイト翻訳のサービスを自分のアプリケーションに合わせたいとする。しかし、エキサイト翻訳には Api が存在していない。実際に、Api が存在している Web サービスの方が稀だが、そんな時には、スクレイピングという方法がある。

Web スクレイピング

ホームページから自分のほしい情報を抽出する方法。
一般的なホームページはアクセスすると、html ファイルが渡され、ブラウザを通すことでホームページとして見れる。ならば、受け取った html ファイルからほしい情報だけ抽出してうまく表示すれば、Api のようなことができるのではないだろうかという話だ。
Java の場合は、指定した Url と http 通信を行い、html ファイルを受け取り、それを読み込み、文字列として様々な処理（主には html のタグを消したり、body 部分以外を切り取ったりする）をすることでスクレイピングができる。

イベント駆動型プログラミング

Swing,Javafx,Android ではアプリケーションを作る方法が似通っていた。何故かと言うと、全てイベント駆動型プログラミングに沿っているからだ。
イベント駆動型プログラミングでは、イベントが起きる度に処理をする。

競技プログラミング

ソフトウェア開発も楽しいが、他にも競技プログラミングも結構楽しい。
実際に、講座で一回やってみたいと思う。

問題

2016 年 7 月 22 日、話題ゲームポケモン Go が配信された。
ポケモン Go は GPS の情報を元に、公園や公共施設に設置されたポケストップと呼ばれるところに行き、モンスターボールを手に入れ、そのモンスターボールを使って歩いていると遭遇するポケモンと呼ばれる生物を捕まえるゲームである。
早速ダウンロードしたアイ沼君は、ハマりすぎて単位を落とすことを恐れて、一日 10 分しか歩かないことを決めた。(帰り時間は含まない)
自分家の周辺には、2 つのポケストップがある。アイ沼君は 10 分でポケストップを何個通過できるかを計算しようとしている。
アイ沼君は 2 つのポケストップからそれぞれに向かう時にかかる時間を調べた。
では、2 つのポケストップのそれぞれの時間から、どうポケストップを巡回すれば、より多くのポケストップを回れるかをプログラムで求めよう(できれば、早く巡回を終わりたい)。
以下の表にまとめた。

今いる所↓　向かう所→	A に行く	B に行く
家から	5	4
A から	0	6
B から	5	0

また、アイ沼くんの住んでる地域は、坂が多いので、A→B と B→A が同じ時間とは限らない。

Input1
5 4
0 6
5 0
OutPut1
B→A

Input2
2 3
0 0
1 0
OutPut2
A→B