Fragmenter, ActionBar og menyer

Mildrid Ljosland, Institutt for datateknologi og informatikk (IDI), NTNU
Lærestoffet er utviklet for faget IFUD1042 Applikasjonsutvikling for Android

Resymé: Denne gangen tar vi for oss to ting som er kommet med Android 3 og 4: Fragmenter og ActionBar. I tillegg ser vi også andre måter å lage menyer på enn å bruke ActionBar. I læreboka finner du dette stoffet i kapittel 6, 7 og 13.

4.1 Fragmenter

Fragmenter er en slags «mini-aktiviteter». Hensikten er at layouten skal bli mer fleksibel og tilpasse seg ulike skjermstørrelser. På små enheter kan man for eksempel vise ett fragment per skjermbilde, mens man på større enheter viser to eller flere fragmenter på et skjermbilde.

Her er et veldig enkelt eksempel der jeg plasserer to fragmenter ved siden av hverandre. Det gjør jeg ved å definere hovedlayouten til å inneholde to elementer av typen <fragment>. Hver av dem får et navn, henholdsvis Fragment1 og Fragment2.

```
<LinearLayout 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:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="horizontal"
    android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

    <fragment
        android:name="no.hist.itfag.fragment1.Fragment1"
        android:id="@+id/fragment1"
        android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent" />
    <fragment
        android:name="no.hist.itfag.fragment1.Fragment2"
        android:id="@+id/fragment2"
        android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent" />

</LinearLayout>
```

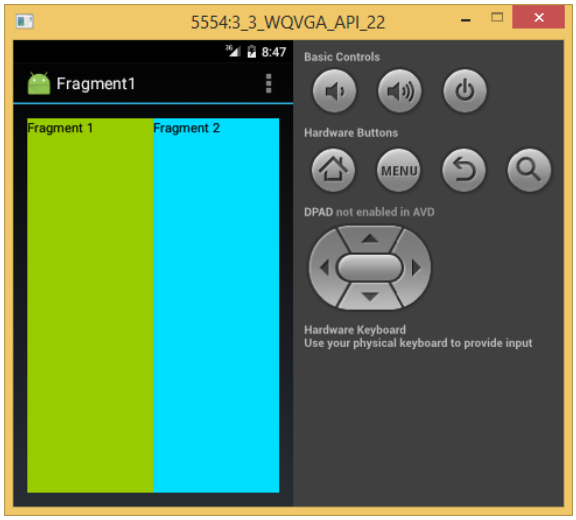
Så definerer jeg to tilhørende klasser, Fragment1.java og Fragment2.java, som begge er avledet av klassen Fragment. Slike Fragment-klasser bruker en metode som heter onCreateView(). Denne metoden gir oss tilgang til (blant annet) en LayoutInflater. Dette er et objekt som kan hente fram en layoutfil og lage en View av den (men ikke vise det fram slik som setContentView() gjør). I eksemplet sørger hver av fragment-klassene for hente fram hver sin layoutfil, henholdvis fragment1.xml og fragment2.xml. Under vises fragment1.java og fragment1.xml. fragment2.java og fragment2.xml er helt tilsvarende.

```
public class Fragment1 extends Fragment {
    public Fragment1() {
        // Required empty public constructor
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment1, container, false);
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="Fragment 1"
        android:id="@+id/textView"
        android:background="@android:color/holo_green_light"
        android:textColor="@android:color/black" />
</LinearLayout>
```

MainActivity har jeg ikke gjort noe med, den er beholdt slik den ble generert av AndroidStudio, bortsett fra at jeg har endret ActionBarActivity til Activity. Siden den inneholder fragmentene blir disse aktivert og deres innhold vist fram. (For å beholde ActionBar selv om jeg ikke bruker ActionBarActivity, har jeg fulgt oppskriften fra læreboka kapittel 1, «Getting Rid of the Support Library».)

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```



Men så veldig fleksibelt blir det ikke så lenge fragmentene er hardkodet i xml-filene. Sørger vi derimot for å styre dem i programmet vårt, kan vi få det så fleksibelt vi bare vil. Tenk deg at du har en applikasjon der du har en liste med oppslagsord, og ønsker å vise detaljer om et valgt ord. Da kan du la programmet teste om det er plass til både lista og detalj-beskrivelsen. Hvis det er det, vises begge deler, hvis ikke startes en ny aktivitet der detaljene vises fram.

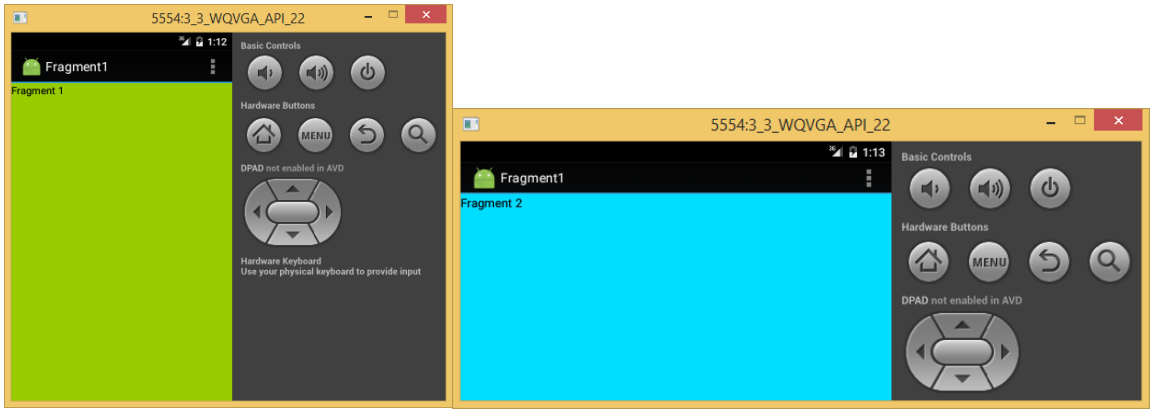
For å vise på programstyrt fragmentvalg, endrer jeg litt på eksemplet slik at fragment1 vises når vi er i portrettmodus og fragment2 når vi er i landskapsmodus. (I emulatoren kan du skifte mellom de to modiene ved å trykke Control F12.)

I activity_main fjerner (eller utkommenterer) jeg begge fragment-elementene, slik at jeg bare har en LinearLayout. Og jeg endrer MainActivity.onCreate() slik at den nå ser slik ut:

```
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
//      setContentView(R.layout.activity_main);
      FragmentManager fm = getFragmentManager();
      FragmentTransaction ft = fm.beginTransaction();
      WindowManager wm = getWindowManager();
      Display d = wm.getDefaultDisplay();
      int rotasjon = d.getRotation();
      if (rotasjon== Surface.ROTATION_0 ||
                rotasjon == Surface.ROTATION_180) {
          ft.replace(android.R.id.content, new Fragment1());
      }
      else {
          ft.replace(android.R.id.content, new Fragment2());
      }
      ft.commit();
  }
```

Vi må definere en transaksjon. Det gjøres ved å bruke klassene FragmentManager og FragmentTransaction. Først framskaffer vi en FragmentManager ved å kalle metoden getFragmentManager() og starter en transaksjon ved å kalle beginTransaction(). Så lages et nytt fragment, dette puttes inn i transaksjonen, og til slutt får man dette til å tre i funksjon ved å kalle commit(). Nå skal vi ikke ha noen fragment-elementer i xml-fila, siden de lages fra bunnen av i programmet.

Metoden replace() erstatter det som finnes fra før med det vi oppgir. Her erstatter vi android.R.id.content, som er det som finnes i aktivitetens view, med et fragment, enten Fragment1 eller Fragment2. For å finne ut hvilket vi ønsker, henter vi fram gjeldende rotasjon. Hvis den er 0 eller 180 grader (på høykant, riktig vei eller opp ned), velger vi det ene fragmentet, ellers, hvis den er 90 eller 270 grader (liggende den ene eller den andre veien), velger vi det andre fragmentet. På denne måten kan vi lage views som er tilpasset mobilens (eller nettbrettets) orientering, f.eks. ved å plassere knapper nedover i det ene og bortover i det andre tilfellet



.

I tillegg til replace(), har klassen FragmentTransaction også blant annet metodene add() og delete(), hide() og show(). En annen metode er addToBackStack(). Den gjør at brukeren kan gå bakover i applikasjonen ved å bruke tilbake-knappen.

Fragmenter har en livssyklus, på samme måte som aktiviteter. De har noen av de samme stadiene som aktiviteter, men også noen som aktiviteter ikke har, for eksempel onAttach, onCreateView, onDestroyView, onDetach. Foreløpig er det vel ikke så veldig aktuelt å ta i bruk disse, men hvis du trenger å lagre unna data før et fragment blir stoppet eller ødelagt, kan du bruke disse metodene til å gjøre det.

Mer om fragmenter finner du på [Fragments].

Samarbeid mellom fragmenter

Hvis vi ønsker at et fragment skal påvirke et annet fragment, er den beste måten å gjøre det på å bruke callback-metoder. Det fragmentet som har bruk for å endre noe i et annet fragment, definerer et interface med en metode, og kaller denne metoden når behovet oppstår. Aktiviteten som fragmentene er en del av, implementerer interfacet og dermed interfacets metode. Denne metoden kaller en ny metode i det andre fragmentet, som dermed sørger for å oppdatere seg. Altså: Det ene fragmentet gir beskjed til deres felles aktivitet om at endring er ønskelig, aktiviteten gir denne beskjeden videre til riktig fragment, og dette fragmentet sørger selv for endringen.

Ved å endre litt på det første eksemplet, kan jeg lage en applikasjon der jeg lar brukeren endre teksten i det grønne fragmentet, og da blir teksten i det blå fragmentet endret tilsvarende.

I fragment1.xml endrer jeg fra TextView til EditText.

I Fragment1.java legger jeg inn et interface (kan gjøres automatisk i Android Studio ved å velge File/New Fragment og hake av for «Include interface callbacks), og kaller det. For å holde rede på hvilket objekt som implementerer interfacet, lagres dette i medlemsvariabelen mListener. Den settes lik aktiviteten i onAttach-metoden. I onDetach() kuttes forbindelsen

igjen ved å sette den lik null. I onCreateView legger jeg inn en lytter på EditText-objektet. Jeg bruker en TextWatcher som krever at jeg implementerer de tre metodene afterTextChanged(), beforeTextChanged() og onTextChanged(). Jeg bruker bare den siste av disse, og det eneste den gjør, er å kalle interfacets metode, altså mListener.onFragmentInteraction().

```
public class Fragment1 extends Fragment {
    private OnFragmentInteractionListener mListener;
    public Fragment1() {
        // Required empty public constructor
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment1, container, false);
        EditText e = (EditText)v.findViewById(R.id.editText);
        e.addTextChangedListener(new TextWatcher() {
            public void afterTextChanged(Editable s){}
            public void beforeTextChanged(
                       CharSequence s, int start, int count, int after){}
            public void onTextChanged(
                       CharSequence s, int start, int before, int count){
                mListener.onFragmentInteraction(s.toString());
            }
        });
        return v;
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnFragmentInteractionListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString()
                    + " must implement OnFragmentInteractionListener");
        }
    }

    @Override
    public void onDetach() {
        super.onDetach();
        mListener = null;
    }

    public interface OnFragmentInteractionListener {
        public void onFragmentInteraction(String tekst);
    }
```

I MainActivity.java gir jeg beskjed om at aktiviteten skal implementere interfacen:

```
public class MainActivity extends Activity
        implements Fragment1.OnFragmentInteractionListener {
```

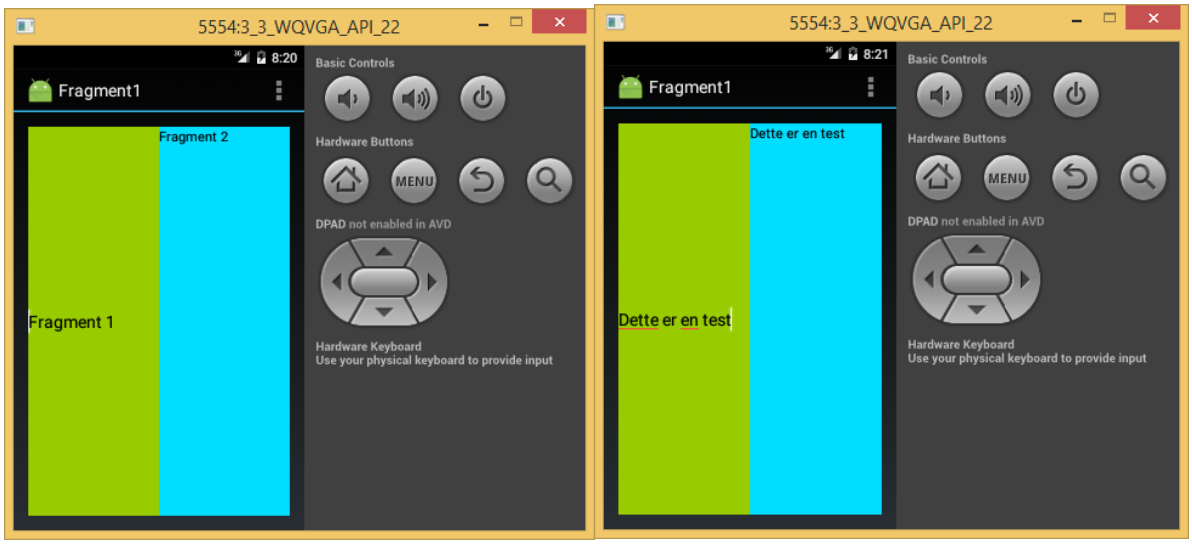
og definerer metoden onFragmentInteraction ved å kalle metoden endreTekst() i Fragment2.

```
public void onFragmentInteraction(String tekst) {
    Fragment2 f = (Fragment2)(getFragmentManager().
```

```
                findFragmentById(R.id.fragment2));
    f.endreTekst(tekst);
}
```

I Fragment2.java definerer jeg endreTekst slik:

```
public void endreTekst(String tekst) {
    TextView t = (TextView)(getView().findViewById(R.id.textView));
    t.setText(tekst);
}
```



Noen spesielle fragmenter

I forrige leksjon så vi på blant annet ListView og ListActivity. Vi har også ListFragment, som er et ListView plassert i et fragment og DialogFragment, som er en dialog plassert i et fragment, samt PreferenceFragment, som er et fragment spesialtilpasset til situasjonen at bruker skal sette eller hente fram verdiene på ulike brukerinnstillinger. PreferenceFragment skal vi komme tilbake til i neste leksjon, de andre to ser vi et eksempel på bruk av, i eksemplet i slutten av denne leksjonen.

4.2 ActionBar

Hvis du bruker Android Studio, har du sannsynligvis lagt merke til at du automatisk får laget metodene

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is
present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
```

```
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

Disse metodene brukes når man ønsker å ta i bruk meny-linja øverst i skjermbildet. onCreateOptionsMenu() bestemmer hvilke menyer som skal finnes, mens onOptionsItemSelected() definerer hva som skal skje i de ulike valgene.

Koden vist over, henter fram en meny definert som xml-fil. En slik meny må definert med et <meny>-element, og et eller flere <item>-elementer inni, for eksempel slik:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
      <item android:id="@+id/menu_solution"
          android:title="@string/solution"
          android:showAsAction="ifRoom" />
      <item android:id="@+id/menu_about"
          android:title="@string/about"
          android:showAsAction="ifRoom" />
      <item android:id="@+id/menu_exit"
          android:title="@string/exit"
          android:showAsAction="ifRoom" />
      <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          android:orderInCategory="100"
          android:showAsAction="never" />
</menu>
```

Her har jeg sagt at denne menyen skal settes inn i meny-linja hvis det er plass. Hvis det ikke er plass, vil den havne under ⋮, der det kommer fram en liste med de valgene som ikke vises direkte.

Et alternativ til å bruke xml-fil, er å legge inn menyene i programkoden. Da brukes metoden add, for eksempel slik:

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
       super.onCreateOptionsMenu(menu);
       MenuItem mi=menu.add(0,0,0, R.string.solution);
       mi.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
       mi=menu.add(0,1,0, R.string.about);
       mi.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
       mi=menu.add(0,2,0, R.string.exit);
       mi.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
       return true;
    }
```

Argument to i add() gir id'en til valget, så har man flere mulige valg, må de ha forskjellig verdi. Siste argument forteller hvilken tekst som skal vises fram. Det er også mulig å sette at valget skal vises som et bilde ved å bruke metoden setIcon().

I onOptionsItemSelected() må man teste på hvilket valg som er gjort ved å sjekke id'en.

```
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
      int id = item.getItemId();
      switch (id) {
          case R.id.menu_solution:
              showSolution(true);
              return true;
```

```
            case R.id.menu_about:
                showAbout();
                return true;
            case R.id.menu_exit:
                showSolution(false);
                finish();
                return true;
            case R.id.action_settings:
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
```

4.3 Eksempel: Quiz

Jeg skal lage en applikasjon som presenterer brukeren for diverse ja-nei-spørsmål for å teste brukerens kunnskaper om et tema.

Jeg lar spørsmålene og riktig svar lagres i to tabeller i strings.xml:

```
<resources>
    <string name="app_name">Quiz</string>
    <string name="action_settings">Settings</string>
    <string name="yes">Ja</string>
    <string name="no">Nei</string>
    <string name="solution">Fasit</string>
    <string name="about">Om Quiz</string>
    <string name="exit">Avslutt</string>
    <string name="aboutContent">Test dine geografikunnskaper</string>
    <string name="ok">OK</string>
    <string-array name="questions">
        <item >Heter hovedstaden i Belgia Brussel?</item>
        <item >Er Fansipan et fjell i Asia?</item>
        <item >Er Nederland en republikk?</item>
    </string-array>
    <string-array name ="answers">
        <item>@string/yes</item>
        <item>@string/yes</item>
        <item>@string/no</item>
    </string-array>
</resources>
```

I activity_main.xml definerer jeg to fragmenter:

```
<LinearLayout 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:orientation ="horizontal">

    <fragment
        android:name="no.hist.itfag.quiz.QuizFragment1"
        android:id="@+id/fragment1"
        android:layout_weight="2.0"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        tools:context=".QuizActivity" />
     <fragment
        android:name="no.hist.itfag.quiz.QuizFragment2"
```

```
        android:id="@+id/fragment2"
        android:layout_weight="0.5"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        tools:context=".QuizActivity" />

</LinearLayout>
```

Det ene fragmentet skal brukes til spørsmålene og det andre til svarene. Jeg lar begge ha vidde 0 slik at de kan tilpasses hva som helst. Vekta lar jeg være 2.0 på det første og 0.5 på det andre. Det gjør at det første blir mye bredere enn det andre – det skal jo inneholde spørsmålene, mens det andre bare skal inneholde ja eller nei.

Ved oppstart.

Etter å ha trykket på FASIT.

Siden vi lar både Fragment1 og Fragment2 være subklasser av ListView, og ListView har en ferdigdefinert xml-fil tilknyttet seg, trenger vi ikke å lage noen egne xml-filer for fragmentene. Men hvis vi vil (for eksempel fordi vi vil putte inn flere ting enn bare lista), kan vi lage dem selv. Da kan de for eksempel se slik ut:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <ListView
        android:id="@id/android:list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:drawSelectorOnTop="false"
        />

</LinearLayout>
```

Jeg lager også en meny (plassert i res/menu/menu_main.xml), og får blant annet menyvalget «Fasit»:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:id="@+id/menu_solution"
        android:title="@string/solution"
        android:showAsAction="ifRoom"
        />
      …
```

```
</menu>
```

Så går jeg løs på java-filene.

MainActivity har en onCreate()-metode som ser slik ut:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    showSolution(false);
    getActionBar().setDisplayHomeAsUpEnabled(true);
}
```

getActionBar().setDisplayHomeAsUpEnabled(true); gjør at applikasjonsikonet i menylinja blir et klikkbart valg. Vanligvis lar man brukeren gå opp eller ut av applikasjonen ved dette valget. I dette eksemplet har jeg valgt å la den avslutte applikasjonen.

Ved starten vil jeg ikke vise fram svarene, så da setter jeg showSolution(false). Metoden har jeg laget slik:

```
public void showSolution(boolean show) {
    FragmentManager fm = getFragmentManager();
    FragmentTransaction ft = fm.beginTransaction();
    Fragment2 fragment = (Fragment2)fm.findFragmentById(R.id.fragment2);
    if (show) ft.show(fragment);
    else ft.hide(fragment);
    ft.addToBackStack(null);
    ft.commit();
}
```

Jeg henter altså fram fragment2 og tar enten show() eller hide() på det, avhengig av om jeg vil at det skal vises fram eller ikke. Jeg tar også addToBackStack(), slik at brukeren kan gå bakover i utføringsrekkefølgen (på samme måte som bakoverknappen i nettlesere).

Menyhånteringa er som vist tidligere:

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
       getMenuInflater().inflate(R.menu.menu_main, menu);
       return true;
    }
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    switch (id) {
        case R.id.menu_solution:
            showSolution(true);
            return true;
        case R.id.menu_about:
            showAbout();
            return true;
        case android.R.id.home:
        case R.id.menu_exit:
            showSolution(false);
            finish();
            return true;
        case R.id.action_settings:
            return true;
    }
```

```
    return super.onOptionsItemSelected(item);
}
```

Når man velger «Fasit», kalles showSolution(true), så da kommer svarene på alle spørsmålene fram. Valgene android.R.id.home (applikasjonsikonet) og R.id.menu_exit gjør begge at applikasjonen avsluttes.

Klassen Fragment2 er ganske enkel. Den er et ListFragment med svarene:

```
public class Fragment2 extends ListFragment {
    /**
     * Mandatory empty constructor for the fragment manager to instantiate
     * the fragment (e.g. upon screen orientation changes).
     */
    public Fragment2() {
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Resources res= getResources();
        String[] solution = res.getStringArray(R.array.answers);
        setListAdapter(new ArrayAdapter<String>(getActivity(),
                android.R.layout.simple_list_item_1, solution));
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
    //    return inflater.inflate(R.layout.fragment2, container, false);
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

Det som skjer her er at vi først henter inn svarene fra tabellen answers i strings.xml og lar så setListAdapter() putte verdiene inn i listView-komponenten.

Så var det Fragment1. Her skal det være mulig å velge spørsmål, så jeg må implementere metoden onListItemClick(). Jeg velger å la det komme fram en dialog der brukeren kan velge Ja eller Nei. For å behandle svaret, lar jeg Fragment1 implementere en interface som jeg definerer i Dialog-klassen:

```
public class Fragment1 extends ListFragment implements
                                QuizDialogFragment.Callback {
    private String[] questions;
    private String[] answers;
    private int clicked = -1;

    public Fragment1() {
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Resources res= getResources();
        questions = res.getStringArray(R.array.questions);
        answers = res.getStringArray(R.array.answers);
        setListAdapter(new ArrayAdapter<String>(getActivity(),
                android.R.layout.simple_list_item_1, android.R.id.text1,
```

```
                questions));
    }
      @Override
      public void onListItemClick(ListView l, View v, int position,
                long id) {
          super.onListItemClick(l, v, position, id);
          QuizDialogFragment dialog =
                  QuizDialogFragment.newInstance(questions[position],
                       answers[position], this);
          dialog.show(getFragmentManager(), null);
          clicked = position;
      }
      public void showResponse(boolean correct) {
          ListView l = getListView();
          TextView v = (TextView)l.getChildAt(clicked);
          if (correct) v.setBackgroundColor(Color.GREEN);
          else v.setBackgroundColor(Color.RED);
      }
}
```

onListItemClick() sørger altså for at en dialogboks kommer opp, riktig spørsmål vises fram og riktig svar sendes med. Dermed kan dialog-klassen finne ut om brukeren svarer riktig, og sørge for at showResponse() kalles med enten true (riktig svar) eller false (feil svar).

Metoden showResponse() farger det valgte listeelementet grønt hvis brukeren svarte riktig, rødt hvis ikke.

Til slutt tar vi klassen QuizDialogFragment. Først har vi metoden newInstance():

```
static QuizDialogFragment newInstance(String title, String answer,
        Callback callback) {
    QuizDialogFragment fragment = new QuizDialogFragment();
    fragment.callback = callback;
    Bundle args = new Bundle();
    args.putString("title", title);
    args.putString("answer",answer);
    fragment.setArguments(args);
    return fragment;
}
```

newInstance() kan defineres akkurat som vi vil, bortesett fra at den må være statisk. Her har jeg definert den med tre parametre, title, answer og callback. Title og answer putter jeg inn i en Bundle som jeg igjen putter inn som argument i fragmentet. Da kan fragmentet hente dem fram når svaret skal behandles.

Callback er et interface som ser slik ut:

```
public interface Callback {
     public void showResponse(boolean correct);
}
```

I onCreateDialog() henter jeg fram argumentet title igjen via metoden getArguments().

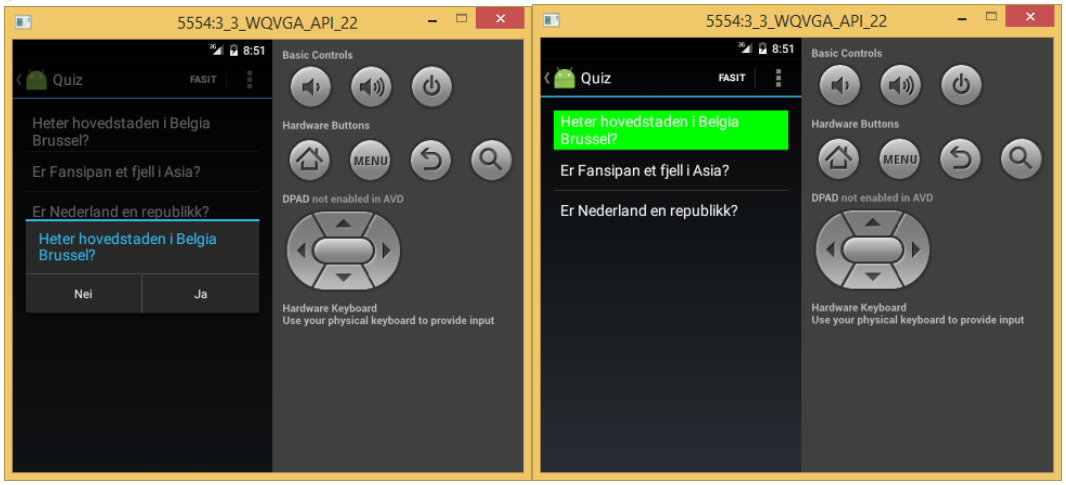
```
      @Override
      public Dialog onCreateDialog(Bundle savedInstanceState) {
            String tittel = getArguments().getString("title");
            AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
            builder.setTitle(tittel);
             builder.setPositiveButton(R.string.yes, new MyListener());
             builder.setNegativeButton(R.string.no, new MyListener());
```

```
            return builder.create();
        }
```

Jeg lager en AlertDialog ved å framskaffe et Builder-objekt, setter tittelen og de to knappene, før jeg lager selve dialogen.

MyListener() er en indre klasse som ser slik ut:

```
class MyListener implements DialogInterface.OnClickListener {
    String answer = getArguments().getString("answer");

    @Override
    public void onClick(DialogInterface dialog, int which) {
        int ans = Dialog.BUTTON_POSITIVE;
        if (answer.equals(getResources().getString(R.string.no)))
            ans = Dialog.BUTTON_NEGATIVE;
        if (callback != null) callback.showResponse(which == ans);
    }
}
```



Etter å ha klikket på spørsmål en. Ettet å ha klikket på Ja.

Parameteren which forteller hvilken knapp som ble trykket. callback er en objektvariabel i QuizDialogFragment-klassen, og ble satt i newInstance(). Når dialogen lages, sendes Fragment1 inn som argument til callback. Det kan vi gjøre fordi Fragment1 er definert ved implements QuizDialogFragment.Callback. Når svaret er avgitt, sjekkes det mot riktig svar, og callback (altså Fragment1) sin metode showResponse() kalles.

Dette eksemplet følger med leksjonen som zip-fil.

4.4 Litt om andre typer menyer

Før Android-versjon 3.0 hadde man ikke ActionBar. Men man hadde Options menyer, som kan aktiveres via meny-knappen på enheten (hvis en slik finnes). De lages på samme måte som ActionBar, via metoden onCreateOptionsMenu(), og aktivieres vanligvis via onOptionsItemSelected().

En alternativ måte å aktivere menyen på, er å bruke en lytter, for eksempel slik:

```
menuItem.setOnMenyItemClickListener(new OnMenuClickListener() {
      public boolean onMenuItemClick(MenuItem item) {
            …
            return true;
```

```
        }
});
```

Vi kan bare ha ett Menu-objekt i en aktivitet, dette lages når onCreateOptionsMenu blir kalt. Men vi kan bruke både onOptionsItemSelected() og setOnMenuItemClickListener() på den samme menyen. Da blir onMenuItemClick() kjørt først, og hvis den returnerer false kjøres onOptionsItemSelected() etterpå. Hvis den derimot returnerre true, kjøres ikke onOptionsItemSelected().

En annen type menyer er de såkalte Context-menyene. Dette er pop-opp-menyer som dukker opp hvis brukeren klikker og holder på en komponent som har fått en slik meny tilknyttert seg. Slike menyer brukes til ting som er nært knyttet til komponenten, for eksempel kan vi tenke oss et tegneprogram der man kan klikke på en figur for å velge farge på den.

For å få dette til, må vi lage metoden onCreateContextMenu(). Dessuten må den (eller de) komponenten(e) som skal få menyen, koples sammen med hva som skal gjøres, gjerne i form av en lytter.

Vi har også typen PopupMenu() som bruker en knapp til å få fram valgene.

Disse alternative måtene å lage menyer på, behandles i kapittel 7 i læreboka.

4.5 Referanser

[Fragments] http://developer.android.com/guide/components/fragments.html