TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

OPPIMISPÄIVÄKIRJA

PLA-32820 2018 Mobiiliohjelmointi

Iiro Loukkalahti

Tarkastaja: Mika Saari
Tarkastaja ja aihe hyväksytty
10. tammikuuta 2018

TIIVISTELMÄ

TOIMI KUNTA: Tampereen teknillisen yliopiston opinnäytepohja
Tampereen teknillinen yliopisto
Oppimispäiväkirja, XX sivua, YY liitesivua
Tammikuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: Mika Saari

Avainsanat: oppimispäiväkirja, mobiiliohjelmointi, java, android, älypuhelin

Tämä dokumentti on oppimispäiväkirja liittyen kurssin ’Mobiiliohjelmointi’ suorittamiseen. Sisältö on jaettu lukuihin harjoituskohtaisesti, jossa jokaiseen on erikseen listattu tekemiset ja suunnittelemiset päiväkohtaisesti.

Tässä pohjassa tiivistelmää varten 2 omaa tekstityyppiä: tunnistiedoille tyyli BibInfo ja tiivistelmätekstille *Abstract*, jossa riviväli on 1.0. Otsikkotyyppi on *Heading (no number)*, joka tekee automaattsesti sivunvaihdon (Page break before). Samaa otsikkotyyppiä käytetään mm. sisällysluettelossa. Lähdeluettelossa on identtinen tyyppi hieman eri nimellä, jolloin se voidaan poimia sisällysluetteloon. Sivunumeroja varten etusivun lopussa pitää olla *Section Break* ja tiivistelmän yläotsakkeen (header) asetus *Link to Previous* pois päältä, ja lisäksi sivunumeron muotoilusta *Start at i* (eikä *Continue*).

ABSTRACT

TOIMIKUNTA: Thesis template of Tampere University of Technology
Tampere University of Technology
Study journal, XX pages, YY Appendix pages
January 2018
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Mika Saari

Keywords: study journal, diary, mobile programming, java, android, smartphone

This document is a required study journal related to course 'Mobiiliohjelmointi'. All exercises have been separated to their own sections with daily documentation. The documentation will contain all steps taken to carry out the tasks from start to finish.

ALKUSANAT

Tämä dokumentti on laadittu TTY:n opinnäytetyöohjeen vuoden 2014 version mukaan edellistä pohjaa muokkaamalla.

Kiitokset kurssista.

Porissa, 5.9.2018

Iiro Loukkalahti

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

CC-lisenssi	Creative Commons -lisenssi
LaTeX	ladontajärjestelmä tieteelliseen kirjoittamiseen
TTY	Tampereen teknillinen yliopisto
URL	engl. Uniform Resource Locator, verkkosivun osoite

JOHDANTO

Päiväkirjani mobiili-ohjelmoinnin kurssiin liittyen. Mukana on opettajan antamien tehtävien, sekä Udacity ”Developing Android Apps”-verkkokurssin dokumentteja. Mukana on myös laajan harjoitustyön tiedot.

HARJOITUS 1 – TUTUSTUMINEN MOBIILIYMPÄRISTÖIHIN

Ensimmäisenä tehtävänä on tutustua haluamaansa kiinnostavaan mobiililaitteeseen. Tarkoitus on tutustua sen ominaisuuksiin ja tapoihin, miten laitetta voidaan ohjelmoida.

1.1 Aloitus (11.1.2018)

Aloitin tänään varsinaisen kurssin suorittamisen asentamalla yliopistolta saamillani tunnuksilla Microsoft Office 365 ProPlus-ohjelmiston. Latasin koulun sivuilta valmiin opinnäytetyöpohjan, jota käyttäisin tämän dokumentin runkona. Koitin aivan aluksi editoida sitä Microsoftin Word-ohjelman selainversiolla, siinä kuitenkaan onnistumatta. Päädyin täten asentamaan koko Office-paketin tietokoneelleni.

1.2 Laitteen valinta (12.1.2018)

Valitsin laitteekseni oman älypuhelimeni, Samsung Galaxy S8+. Tämä laite julkaistiin 29. maaliskuuta 2017. Se on Samsungin lippulaivapuhelin, joka edustaa Android-käyttöjärjestelmään pohjautuvia älypuhelimia. Samsungin sivuilta löytää laitteen tarkemmat ominaisuudet ja mm. käyttöoppaan.

Kuva 1. *Samsung Galaxy S8+*

1.3 Käyttöjärjestelmän ominaisuudet (13.1.2018)

Puhelimessa on tällä hetkellä asennettuna Android 7.0 ”Nougat”. Androidin omilla kotisivuilla on kattava esittely ja listaus sen kaikista ominaisuuksista. Androidista voi lukea myös yleispätevän ominaisuuslistauksen Wikipedian artikkelista.

1.4 Ohjelmointi (14.1.2018)

Laitteiden valmistajat sekä Androidin kehittäjä, Google, tarjoaa verkossa kattavat materiaalit ja ohjeet järjestelmän ohjelmointiin. Tämä lisää erityisesti ohjelmien määrää Googlen ja valmistajien omissa ohjelmakaupoissa, mikä taas lisää asiakkaiden kiinnostusta alustaa kohtaan. Materiaalien avulla pääsee nopeasti kehityksen alkuun. Viime kädessä tästä hyötyvät kaikki osapuolet.

Eri valmistajat tuottavat omia räätälöityjä versioitaan Androidista omiin tuotteisiinsa. Google on yksi valmistajista, joka käyttää ”referenssi”-laitteissaan ”puhdasta Androidia”. Esimerkiksi Samsungin laitteiden käyttöjärjestelmän pohjalla käytetään Androidia, mutta Samsung on itse tehnyt siihen mm. korjauksia, mitä virallisessa versiossa ei ole. Räätälöidyssä versiossa voi olla mukana täysin erilaiset oletussovellukset mm. tekstiviesteille, kuville tai vaikkapa internetin selaamiselle. Eri valmistajien tuottamat räätälöidyt käyttöliittymät voivat joskus näyttää jopa täysin eri käyttöjärjestelmiltä. Androidia käytettäessä tähän kaikkeen on mahdollisuus.

Android Developers sivu sisältää Googlen tuottamat viralliset materiaalit, jotka pätevät kaikkiin Android-pohjaisiin käyttöjärjestelmiin. Näitä ohjeita ja materiaaleja voidaan siis soveltaa valmistajasta riippumatta, jos alustan pohjalla on Android.

Samsung Developers on yksi esimerkki valmistaja- ja laitekohtaisista Android-kehitysmateriaaleista. Valmistaja on mm. listannut ohjeita Galaxy-sarjan mobiililaitteiden kameran ohjelmoinnista. Se tarjoaa käytettäväksi mm. Samsung Camera SDK-kirjastoa, jonka kautta voidaan paremmin ohjata ja käyttää laitteen kameraa ja sen ominaisuuksia. Tätä kirjastoa ei mitä luultavammin voi käyttää millään muilla kuin tämän valmistajan Galaxy-sarjan laitteilla.

1.5 Mahdolliset ohjelmointikielet (15.1.2018)

Androidin ohjelmointiin voidaan käyttää monia eri ohjelmointikieliä. Ohessa on katsaus suosituimpiin vaihtoehtoihin.

1.5.1 Java

Virallinen ohjelmointikieli, jolla suurin osa alustan ohjelmista on tehty. Tämä on turvallinen ja hyvä valinta mm. ensimmäisen oman ohjelman tekemiseen.

1.5.2 Kotlin

Melko uusi kieli, jota myös Google tukee. Koodi voidaan kääntää suoraan ajettavaksi Javan virtuaalikoneessa (JVM). Kieli kehitettiin olemaan ”parempi kuin Java”, kuitenkin niin, että yhteensopivuus Javaan säilytettiin. Tällä koitetaan edesauttaa kehittäjien siirtymistä Javasta Kotliniin helpommin. Lisätietoja täältä.

1.5.3 C ja C++

Kaksi vanhempaa ja erittäin käytettyä kieltä. Kielet antavat kehittäjälle vapaammat kädet mm. muistinhallintaan, jolloin laitteista voidaan ottaa ”kaikki mahdollinen mehu irti”. Kieliä voidaan käyttää ”Native Development Kit” (NDK) kautta Android-ohjelman osissa. Nämä osat voivat olla esimerkiksi vanhoja kirjastoja, joita halutaan käyttää mobiiliohjelman kehityksessä. Lisätietoja täältä.

1.5.4 Python ja Bash

Python on melko uusi alustariippumaton ja tulkattava ohjelmointikieli. Osa sen suosiosta perustuu myös sen helppoon syntaksiin, jonka myötä sitä myös suositellaan opeteltavaksi ensimmäisenä ohjelmointikielenä.

Bash on ohjelmointikieleksi miellettävä kieli sekä komentotulkki, jolla voidaan käyttää ja esimerkiksi ohjelmoida tietokonetta. Monissa Linux-jakeluissa Bash on oletuskomentotulkki.

Kumpaakin kieltä voidaan käyttää Androidissa ”Scripting Layer for Android (SL4A)”-avulla. Lisätietoja täältä.

1.5.5 LUA

Kevyt ja kooltaan pieni skriptikieli. Käytetään usein ohjelmien ”sisäisenä” sktriptikielenä (esim. pluginien kielenä jne.). Käytettävä koodi ajetaan LUA:n omassa virtuaalikoneessa. Nykyään kieli on myös erittäin suosittu mobiilipelien kehityksessä.

LUA:aa voidaan käyttää Androidilla “Corona SDK”-sovelluskehityspaketin kautta. Lisätietoja täältä.

1.5.6 HTML5, JavaScript, CSS:

Android-sovellus voidaan myös tehdä käyttäen apuna perinteisiä web-teknologioita. Adoben ”Phonegap”-teknologialla on mahdollista rakentaa mobiiliohjelma verkkosivuna, käyttäen kuitenkin hyödyksi mm. laitteen sensoreita jne. Lisätietoja täältä.

1.5.7 C#

Tällä Microsoftin kehittämällä kielellä on vahva jalansija Windows-työpöytäohjelmien ja verkkopalveluiden kehityksessä. Se on tyypiltään hyvin samantapainen kuin Java. Kieltä voidaan käyttää ”Xamarin”-työkalun avulla, jolla voidaan kääntää sama ja yksi ohjelma monelle eri alustalle (mm. Windows Phone, iOS, Android). Lisätietoja täältä.

1.6 Ohjelmointiin tarvittavat työkalut (18.1.2018)

Developer.com-sivusto on listannut kaikki parhaimmat kehitysympäristöt Android-kehitykseen. Voit lukea koko listan täältä.

Helpoin tapa toteuttaa Android-kehitystä on tehdä se esimerkiksi Windows-tietokoneella. Ohjelmien testaukseen ja ajamiseen voi käyttää oikean mobiililaitteen sijaan myös emulaattoreita. Emulaattori on ohjelma, joka ”matkii” oikeaa laitteen toimintaa. Voimme esimerkiksi asentaa Windowsiin muutamia eri valmistajien emulaattoreita, ja ajaa niitä ”ohjelmina”. Emulaattorit tukevat koodien debuggausta, mutta ne voivat olla erittäin raskaita. Täten nopein ja paras tapa on käyttää testaukseen oikeaa mobiililaitett (mahdollisesti muutamaa erilaista).

1.7 Laitteesta löytyvät ominaisuudet (24.1.2018)

Android Developer-sivulla on materiaalit ja ohjeet sensorien käyttöön. Koko aineiston voi lukea täältä. Materiaalissa oletetaan, että ohjelmistokehitys toteutetaan Java-kielellä, mutta sensoreita voi käyttää myös muilla kielillä.

Android-järjestelmässä ohjelmat joutuvat pyytämään käyttäjältä luvat tiettyihin asioihin. Yksi tällaisista asioista on esimerkiksi sijaintitiedot, jossa käytetään GPS-sensoria. Jos lupaa ei saada, on kehittäjän käsiteltävä tilanne. Ohjelmiin ei tällöin kannata lisätä turhia oikeusvaatimuksia (esim. sijaintitiedolle), jos niille ei ole oikeasti tarvetta. Loppukäyttäjä voi siis rajata ohjelman käytössä olevien API:en, eli rajapintojen, määrää.

HARJOITUS 2 – GIT VERSIONHALLINTA

1.8 Uusi repositorio (24.1 ja 9.2.2018 muokattu)

Tein uuden projektin GitHub-palveluun nimellä ”harj2”. Projekti on nähtävillä julkisesti profiilini kautta osoitteessa:

https://github.com/W0lfw00ds/harj2

Valitsin lisäasetuksen, joka lisää repositorioon suoraan oletusarvoisen ”README.md”-tiedoston, jonka sisältö näytetään etusivulla.

1.9 Projektin tuonti omalle koneelle

Kopioin projektin ”Clone or download”-valinnan kautta seuraavan osoitteen:

https://github.com/W0lfw00ds/harj2.git

Avasin tämän jälkeen Windows PowerShell-komentorivin. Ajoin seuraavat komennot:

```
PS C:\Users\iirol> cd C:\var\temp\
PS C:\var\temp> git clone https://github.com/W0lfw00ds/harj2.git
Cloning into 'harj2'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

Siirryin siis aluksi pohjakansioon, jonne haluan projektin kloonata (C:\var\temp). Tämän jälkeen ajoin vain Git-komennon ”clone”, joka hakee ja kloonaa projektin paikalliselle koneelle. Tiedot siirretään siihen kansioon, mikä on valittuna komentorivin työkansioksi.

1.10 Koodin lisäys repositorioon

Avasin sitten vanhan tutun työkalun, Microsoft Notepadin. Loin uuden tiedoston, johon kirjoitin lyhyen ”Hello world”-skriptin PowerShell-kielellä. Tiedoston nimi oli ”hello-world.ps1”. Tallensin tiedoston juuri kloonaamani projektin juureen, kansioon ”C:\var\temp\harj2\”.

Otin sitten PowerShell-komentorivityökalun jälleen esiin. Tällä kertaa ajoin sille seuraavat komennot:

```
PS C:\var\temp> cd harj2
PS C:\var\temp\harj2> git status
On branch master
Your branch is up-to-date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        helloworld.ps1

nothing added to commit but untracked files present (use "git add" to track)
PS C:\var\temp\harj2> git add *
PS C:\var\temp\harj2> git commit -m "Hello World-skripti lisätty!"
[master d24cdef] Hello World-skripti lisätty!
 1 file changed, 1 insertion(+)
 create mode 100644 helloworld.ps1
PS C:\var\temp\harj2> git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 329 bytes | 329.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/W0lfw00ds/harj2.git
   93a9301..d24cdef  master -> master
```

Vaihdoin siis aluksi projektin kansioon, jonka Git loi kloonauksessa. Tämän jälkeen tarkastin tilanteen, ja Git ilmoitti uudesta tiedostosta, jota ei ole vielä lisätty versionhallintaan. Lisäsin kaikki vielä versionhallinnassa olemattomat tiedostot komennolla ”git add *”, jonka jälkeen loin ensimmäisen commitin. Aivan lopuksi ajoin ”git push”, joka työnsi juuri luomani commitin GitHub-palvelimelle.

Harjoitus 2 oli nyt valmis.

HARJOITUS 3

1.11 Android-työkalujen asennus (6.1.2018)

Latasin uusimman version Windows 10 Pro 64bit käyttöjärjestelmälle. Kehitysympäristön asennukseen meni hetki, mutta ilman ongelmia. Asennus latasi automaattisesti puuttuvia komponentteja, ja laittoi asioita kuntoon.

1.12 Uuden projektin luominen (6.2.2018)

Loin uuden Android-projektin, jossa käytin valmista ja tyhjää ”Empty Activity”-pohjaa. Projekti käyttää työkalunaan Gradlea, ja jostakin syystä ympäristö valitti koko Gradlen puuttumisesta. Nopeasti netistä luettuani ymmärsin, että Gradlen pitäisi tulla Android Studion mukana valmiiksi säädettynä. Jouduin asettamaan asetuksista Gradlen ”home”-kansion, joka löytyi Android Studion asennuskansion sisältä. Tämän jälkeen ympäristö herjasi vielä muista ongelmista, mutta se osasi korjata ne itsekseen, lataamalla puuttuvia komponentteja verkosta.

Käytettäessä ”Empty Activity”-pohjaa, on ulkoasuun (activity_main.xml) jo valmiiksi lisätty yksi ”TextView”-elementti, joka tulostaa näytölle tekstin ”Hello World!”. Toisin sanoin, projekti tarvitsi pohjalle ainoastaan sijainnin sekä projektin nimen, ja ensimmäinen ohjelma oli valmiina ajettavaksi!

1.13 Ohjelman testaaminen Android-laitteella (6.2.2018)

Ennen projektin ajamista oikealla laitteella, oli laitteen asetuksista laitettava ”USB”-debuggaus päälle. Tämä onnistui seuraavasti:

Asetukset > Tietoja puhelimesta > Ohjelmiston tiedot

Tämän jälkeen kohtaa ”Koontiversio” oli napautettava sarjassa 7 kertaa. Tämän seurauksena asetuksiin tuli esille aiemmin piilotettu asetusryhmä, ”Sovelluskehittäjien asetukset”. Näistä asetuksista oli laitettava päälle valinta ”USB-virheenkorjaus”. Tämän jälkeen laite oli periaatteessa valmis yhdistettäväksi tietokoneeseen USB-kaapelilla.

Muistin aiemmin, että eri laitteille oli ladattava erillinen USB-ajuri, ennen kuin testattavia ohjelmia voitiin ajaa. Sitä ei kuitenkaan tällä kertaa tarvittu, vaan Android Studio osasi heti listata laitteen painettaessa ”Run..”-komentoa.

Puhelimen ruudulle tuli vielä erillinen valinta, jossa piti antaa oikeus USB-virheenkorjaukseen. Tämän jälkeen laite ja ympäristö oli viimein yhdistetty kokeilua varten.

Ajoin uudelleen ”Run…”-komennon, jonka jälkeen Android Studio avasi ohjelman puhelimeen ensimmäistä kertaa. Hello world!

Android Studio huomautti, että siihen pitää asentaa päivitys, jonka kautta ohjelman voi ”päivittää puhelimeen suoraan”, käyttäen komentoa ”Apply Changes”. Ilmeisesti tämä on nopeampi tapa muokata ja ajaa ohjelmaa uudelleen. En kuitenkaan vielä tässä vaiheessa asentanut pakettia, vaan ajoin ohjelman puhelimeen perinteisellä ”Run”-komennolla.

1.14 Lisäys Git-versionhallintaan (6.2. ja 9.2 muokattu)

Loin uuden projektin GitHubiin, nimellä ”harj3”. Se löytyy osoitteella:

https://github.com/W0lfw00ds/harj3

Siirryin sitten PowerShell-komentorivillä Android Studiolla luotuun projektikansioon, ja ajoin seuraavat Git-komennot:

```
PS C:\var\temp\harj2> cd C:\Users\iirol\StudioProjects
PS C:\Users\iirol\StudioProjects> cd .\Harjoitus3\
PS C:\Users\iirol\StudioProjects\Harjoitus3> echo "# harj3" >> README.md
PS C:\Users\iirol\StudioProjects\Harjoitus3> git init
PS C:\Users\iirol\StudioProjects\Harjoitus3> git status
PS C:\Users\iirol\StudioProjects\Harjoitus3> git add *
PS C:\Users\iirol\StudioProjects\Harjoitus3> git commit -m "eka committi!"
PS C:\Users\iirol\StudioProjects\Harjoitus3> git remote add origin
https://github.com/W0lfw00ds/harj3.git
PS C:\Users\iirol\StudioProjects\Harjoitus3> git push -u origin master
```

Ohjelman koodit oli nyt lisätty versionhallintaan onnistuneesti!

1.15 Versionhallinnan työkalut Android Studiossa

Android Studio itsessään vaatii GitHub-käyttäjätunnusten syöttämisen. Tämä ei kuitenkaan riittänyt, vaan GitHub-pyysi vielä omalla ikkunallaan tunnusten syöttämisen uudelleen. Palveluun oli nyt kuitenkin kirjauduttu Android Studion kautta, jolloin ”Version Control”-välilehti pääsi käytettäväksi.

En käyttänyt näitä työkaluja sen enempää, koska koen että komentorivipohjainen Git on paljon yksinkertaisempi, kuin Android Studion oma käyttöliittymällinen työkalu. Kokeilin tätä työkalua kuitenkin myöhemmin, ja sen napeista saa ajettua samoja toimintoja kuin mitä kirjoitetaan komentorivin kautta (esim. ”add”, ”commit”, ”push” jne). Ulkoasu on kuitenkin logiikkaansa nähden todella monimutkaisesti suunniteltu. Omasta mielestäni paras versionhallinta-työkalu on ollut Eclipse-kehitysympäristössä.

HARJOITUS 4 – LASKUKONE

1.16 Uuden projektin aloitus (8.2.2018)

Aloitin työn luomalla projektille uuden repositorion ”harj4” GitHubiin, osoitteeseen:

https://github.com/W0lfw00ds/harj4

1.17 Uusi projekti Android Studiossa (8.2.2018)

Tein uuden projektin Android Studiossa. Annoin projektin nimeksi ”Harjoitus4”, sekä käytin pohjana valmista tyhjää Acitivity-pohjaa. En tehnyt vielä mitään muuta tässä kohtaa, kuin tallensin projektin.

1.18 Lisäys versionhallintaan (8.2.2018)

Olin asentanut Git-ohjelman tietokoneelleni jo aiemmin, joten pääsin suoraan lisäämään projektia versionhallintaan.

Avasin Windows 10-käyttöjärjestelmän mukana tulevan ”Windows PowerShell”-komentorivityökalun. Tämä työkalu toimii miltei samoin kuin vanha tuttu komentokehote-ohjelma, mutta siinä on uusia tuettuja ominaisuuksia, esimerkiksi PowerShell-skriptojen ajaminen.

Siirryin aivan aluksi projektin sisältävään kansioon seuraavasti:

```
PS C:\Users\iirol>
PS C:\Users\iirol> cd .\StudioProjects\Harjoitus4\
```

Sitten ajoin PowerShellilla seuraavat komennot, jotka löytyvät GitHubin "Quick Setup”-sivulta. Tämä koodi luo uuden ”README.md” tiedoston, joka näytetään repositorion etusivulla, sekä tekee ensimmäisen commitin. Tämä osuus ei vielä lisää projektia:

```
PS C:\Users\iirol\StudioProjects\Harjoitus4> echo "# harj4" >> README.md
PS C:\Users\iirol\StudioProjects\Harjoitus4> git init
Initialized empty Git repository in C:/Users/iirol/StudioProjects/Harjo-
itus4/.git/
PS C:\Users\iirol\StudioProjects\Harjoitus4> git add README.md
PS C:\Users\iirol\StudioProjects\Harjoitus4> git commit -m "first commit"
[master (root-commit) beb5255] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
PS C:\Users\iirol\StudioProjects\Harjoitus4> git remote add origin
https://github.com/W0lfw00ds/harj4.git
PS C:\Users\iirol\StudioProjects\Harjoitus4> git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 237 bytes | 237.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
```

```
To https://github.com/W0lfw00ds/harj4.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

Lisäsin tämän jälkeen Android Studiossa luodut projektitiedostot versionhallintaan seuraavilla komennoilla:

```
PS C:\Users\iirol\StudioProjects\Harjoitus4> git add *
PS C:\Users\iirol\StudioProjects\Harjoitus4> git commit -m "added files"
PS C:\Users\iirol\StudioProjects\Harjoitus4> git push
```

Nyt projektin kaikki tiedostot oli lisätty, ja ne näkyivät repositorion kotisivuilla.

1.19 Laskuri-ohjelman kehitys (8.2.2018)

Avasin projektin nyt uudelleen Android Studiossa. Aloitin muokkaamalla ensin pohjaohjelmasta pois ylimääräisiä asioita.

1.19.1 Etusivun ulkoasu (8.2.2018)

Tyhjensin ensimmäisen sivun aktiviteetin (activity_main.cml) tiedoston sisältämään vain pohjaelementin, ”ConstraintLayout”. Lisäsin sen jälkeen sen sisälle yhden lapsen ”TableLayout”, jolla oli asetettuna ”ID”. Tämä melko simppeli pohjapaneeli on helppo tapa asetella napit ja muut sisällöt oikeille paikoilleen.

Koska jokainen rivi tulisi sisältämään miltei saman logiikan, olisi ulkoasun voinut rakentaa dynaamisesti, mutta olen oppinut kantapään kautta, että se tekee asioita hankalampia. Tämän takia rakensin ulkoasun perinteiseen tapaan, staattisesti XML-tiedoston sisälle.

Ideana oli luoda jokaiselle eri laskulle oma ”TableRow”, jolla ne rivitettäisiin omille riveilleen. Jokainen laskuoperaatiota tekevä rivi koostui seuraavista elementeistä:

1. **EditText:** normaali syötettävä tekstikenttä, jossa on oletus, ”hint”-, eli vihjeenä arvo ”0”. Tämä arvo katoaa heti, kun käyttäjä kirjoittaa kenttään jotain. Käyttäjän syöte on rajoitettu vain kokonaislukuihin, käyttämällä ominaisuutta ”inputType=’number’”. Tähän täytettiin ensimmäinen luku.
2. **TextView:** pelkkä teksti, jota ei voi valita. Tämä keskitettiin ”textAlignment”-ominaisuuden avulla. Tähän kirjoitettiin operaattorin merkki.
3. **EditText:** toinen samanlainen elementti kuin ensimmäinen. Tähän käyttäjä kirjoittaa toisen luvun.
4. **Button:** nappi, jota painamalla ohjelma suorittaa laskun operaattorin mukaisesti. Tulos asetettiin seuraavaksi esiteltävään elementtiin.
5. **EditText:** normaalista poiketen, tämä elementti on ”disabloitu”, joten siihen kirjoittaminen on estetty. Tekstin valinta ja kopiointi on kuitenkin mahdollista. Tähän listataan laskennan tulos, kun nappia painetaan.

Kaikkien rivien elementtien leveydet asetettiin suhteellisesti toisiinsa, käyttäen ”layout_weight”-ominaisuutta, ja asettamalla normaali leveys ”layout_width=’0dp’”. Tässä käytettiin suhteita: 25%, 5%, 25%, 25%, 20%.

Laskenta-rivien alapuolelle lisättiin vielä erillainen työkalurivi, jossa olivat napit ”Tyhjennä kaikki” ja ”Näytä logi”. Nämä luotiin samanlaisella menetelmällä kuin aiemmat rivit, käyttäen nyt leveyssuhteita 50% ja 50%.

1.20 Logi-sivun ulkoasu (8.2.2018)

Valinnainen lisätehtävä vaati toisen sivun, jossa näytettäisiin kaikki tehdyt laskuoperaatiot.

Aloitin tämän samalla tavalla kuin edellisen Activityn tekemisen. Lisäsin tällä kertaa ainoaksi lapseksi ”TableLayout”-elementin, jolla ei ollut itsellään yhtäkään lasta. Nämä lapset tultaisiin tekemään dynaamisesti.

1.21 Etusivun ohjelma (8.2.2018)

Etusivun ohjelman logiikka sijaitsee ”MainActivity”-luokassa. Aloitin tekemisen, listaamalla privaatit oliot jokaselle ulkoasun elementille (esim. TableLayout, TableRow, Button jne), jotka hain ”findViewById”-metodilla.

Olioiden listauksen jälkeen, asetin mahdolliset nappien kuuntelijat. Jokainen laskuoperaatioita tekevä rivi käyttäisi painettaessa samaa metodia:

```
private void laske(int luku1, int luku2, String operaattori, TextView tulos-
Kentta);
```

Metodin parametrit luetaan sen rivin oikeista kentistä, operaattori ”TextView”-elementin tekstistä, sekä tulosta varten annettiin viite rivin tuloksen listaavaan ”EditText”-elementtiin. Tämä oli helppoa ja mahdollista jo senkin takia, että jokainen rivi oli periaatteessa täysin samanlainen. Ainoastaan näytettävä operaattori vaihtui. Alla esimerkki napin toiminnallisen lisäämisestä, käyttäen apuna anonyymiä luokkaa:

```
this.plus_nappi.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {
    laske(
      Integer.valueOf(MainActivity.this.plus_luku1.getText().toString()),
      Integer.valueOf(MainActivity.this.plus_luku2.getText().toString()),
      MainActivity.this.plus_teksti.getText().toString(),
      MainActivity.this.plus_tulos
    );
  }
});
```

Työkalurivin napeille luotiin erilliset metodit ”tyhjenna()”, sekä ”naytalogi()”. Toiminnallisuus lisättiin samaan tapaan anonyymejä luokkia käyttäen, kutsuen ”onClick(View view)”-metodin sisällä vain vastaavaa metodia.

1.21.1 ”laske”-metodin logiikka

Metodi lukee parametreista molemmat luvut, sekä merkkijonona käytetyn operaattorin. Se valitsee sitten ”switch”-lohkon sisällä oikean logiikan operaattorin perustella, ja laskee tuloksen uuteen muuttujaan.

Tämän jälkeen haetaan nykyinen päivämäärä muodossa:

```
dd-MM-yyyy  HH:mm:ss
```

Sitten päivämäärän merkkijono ja laskettu tulos yhdistetään, jolloin saadaan seuraavanlainen merkkijono:

```
dd-MM-yyyy  HH:mm:ss: tulos
```

Tämä jono lisätään paikalliseen ”ArrayList<String>”-listaan, jossa on listattuna kaikki tehdyt operaatiot.

Lopuksi laskettu tulos asetetaan parametrina saadun ”EditText”-kentän tekstiksi.

1.21.2 ”tyhjenna”-metodin logiikka

Tämän metodin sisällä käydään läpi kaikki lukujen syöttämiseen luodut ”EditText”-elementit, joiden arvoksi asetetaan tyhjä. Tähän käytettiin apuna olion metodia ”.setText(””);”.

1.21.3 ”naytalogi”-metodin logiikka

Tämä metodi luo uuden ”Intent”-olion, joita käytetään siirryttäessä sivulta toiselle. Tämä on yksi normaaleista tavoista vaihtaa Activity-oliota Androidissa. ”Intent”-olion mukaan laitetaan ”Bundle”-olio, johon voidaan listata erilaisia parametreja. Tässä käytettiin metodia ”.putStringArrayList("logit", this.logi);”, jolla saatiin annettua mukaan ”laske”-metodissa täytetty lista merkkijonoja.

1.22 Logi-sivun ohjelma

Tämän sivun logiikkaan kuului ainoastaan käyttäjän tekemien laskujen tulostus. Logiikka alkaa ylikirjoitetusta metodista:

```
protected void onCreate(Bundle savedInstanceState)
```

Kuten edellisen sivun logiikassa mainittiin, saa tämä sivu avattaessa ”ArrayList<String>”-tyyppisen parametrin, joka sisältää laskuhistorian. Parametri kaivetaan tämän ylikirjoitetun metodin sisällä, hakemalla ”Intent”-olio, jolla sivulle tulliin. Tähän käytettiin seuraavia pätkiä:

```
// Lue lähetetyt logit
this.logit = getIntent().getStringArrayListExtra("logit");

// Luo logeista rivit
this.listaaLogit();
```

Alussa siis haetaan ”Intent”-olio metodilla ”getIntent()”, josta kaivetaan ”Bundle”-olion sisällä lähetetty merkkijono lista metodilla ”getStringArrayListExtra(”logit”)”. Tämän jälkeen lista talletetaan paikalliseen muuttujaan myöhempää käyttöä varten.

1.23 ”listaaLogit”-metodi

Kun rivit ollaan saatu tuotua onnistuneesti, ajetaan sen jälkeen tämä metodi. Metodi toimii kahdella tapaa:

- Tulosta erityisviesti, jos yhtään laskua ei ole vielä laskettu
- Tulosta kaikki laskut omina riveinään näytölle

Yksinkertaisuudessaan katsotaan siis parametrina saadun merkkijono listan koko. Jos lista on tyhjä, tulostetaan erityisviesti. Muussa tapauksessa taas tulostetaan rivit jokaista laskua kohden.

Pientä ulkoasun viilausta ajatellen, logiikkaan lisättiin oma ”boldaaTeksti”-Boolean muuttuja. Tämän muuttujan tila vaihdetaan jokaisen rivin kohdalla joko ”TRUE”- tai ”FALSE”-arvoiseksi vuorotellen. Varsinainen rivi luotiin käyttöliittymälle ”lisaaRivi”-metodilla, josta on lisää tietoja seuraavaksi.

1.24 ”lisaaRivi”-metodi

Tätä yhtä ja samaa metodia käytetään kaikkien rivien listaukseen (eli erityisviestin sekä laskettujen laskujen näyttämiseen käyttöliittymällä). Se luo uuden ”TextView”-olion, joka näyttää riville tulevan tekstin. Jos parametrina tullut ”boldaaTeksti”-muuttuja on arvoa ”TRUE”, muunnetaan teksti vielä lihavoituun muotuun.

Tämä tekstin esittävä olio lisätään uuden ”TableRow”-olion lapseksi, jolla on asetettuna kevyet paksunnukset, joka saa tekstin hieman irti näytön reunoilta.

Loppu viimeksi tämä ”TableRow” olio listataan koko käyttöliittymän pohjalla olevaan ”TableLayout” olioon, jolloin näytölle ilmestyy uusi rivi.

HARJOITUS 5-6 – TIETOKANTAA HYÖDYNTÄVÄ OHJELMA

1.25 Projektin pohjustus (19.2.2018)

Aloitin työn luomalla projektille uuden repositorion ”harj5-6” GitHubiin, osoitteeseen:

https://github.com/W0lfw00ds/harj5-6

Loin tämän jälkeen uuden projektin Android studiossa, täysin samaan tapaan kuin aiemmissakin harjoituksissa. Valitsin työkalun ehdottaman tyhjän aloituspohjan.

1.26 Idea (17.2.2018)

Halusin käyttää tehtävänannon teemaa omassa sovelluksessani. Idea oli siis toteuttaa yksinkertainen sovellus lehtien tietojen tallentamiseen mobiililaitteen sisäistä tietokantaa käyttäen. Sovellusta käytettäisiin tämän tiedon hallintaan ja esittämiseen.

Ajatukseni oli käyttää apuna harjoitustehtävässä olevan kuvan ulkoasua. Tiesin kuitenkin jo tässä kohtaa, ettei tämä yksi sivu tule välttämättä riittämään kaikkiin ominaisuuksiin, jota tulisin ohjelmassa tukemaan. Halusin lisätä ohjelmaan muita mahdollisia toimintoja, kuten tehtävänannossa ehdotettiin.

Lopullinen sovellus tulisi siis sisältämään kaksi erillistä näkymää:

1. Kirjojen listaus, lisäys sekä poisto
2. Lisätyn kirjan tietojen muokkaus

1.27 Tietokannan rajapinta (19.2.2018)

Sovellus käyttää tavanomaista rakennetta näkymille (luokka sekä XML-ulkoasu), mutta tietokanta on eroteltu omaksi rajapinnakseen.

1.27.1 ”Class.Database”-paketti (19.2.2018)

Tietokannalle on projektissa oma Java-paketti: ”Class.Database”. Tämä paketti sisältää kaikki sovelluksen tietokantaan liittyvät koodit, mm. rajapinnat tietojen hakuun ja manipulointiin.

Paketin sisältämä ”Database”-luokka on kaiken ydin. Se perii SQLite-tietokantaan liittyvän ”SQLiteOpenHelper”-luokan. Tämä luokka sisältää kaikki oleelliset asiat, joilla mobiililaitteen sisäistä SQLite-tietokantaa voidaan käsitellä.

Tärkeimmät asiat ”Database”-luokassa ovat:

- Tietokannan nimi
- Tietokannan versionumero
- ”SQLiteOpenHelper”-konstruktorin kutsu
- Ylikirjoitetut metodit joita kutsutaan tietokannan rakentamiseen tai päivittämiseen

Nämä on toteutettu lisäämällä aivan luokan yläosaan staattiset muuttujat ”DATABASE_VERSION” sekä ”DATABASE_NAME”.

Luokan konstruktorissa kutsutaan ”SQLiteOpenHelper”-luokan omaa konstruktoria ”super”-avainsanalla. Sille annetaan parametriksi aiemmin mainitut tietokannan nimi sekä versionumero, sekä konteksti (esim. näkymä jolla ollaan nyt).

Ajossa, ”SQLiteOpenHelper”-luokka tarkastaa perivän luokan antaman tietokantaversion ja vertaa sitä viimeisimpään versionumeroon, jolla sen konstruktoria kutsuttiin. Jos versio on muuttunut, kutsuu luokka ”onUpgrade”-metodia. Jos tietokantaa ei ole vielä luotu, se luodaan taas ”onCreate”-metodin kautta. Perivän luokan on tarkoitus siis ylikirjoittaa ja lisätä näiden metodien toiminnallisuus.

Sovellusta ajaessa on tarkoitus käyttää vain yhtä ”Database”-instanssia. Tämän vuoksi luokkaan on lisätty ”singleton pattern”-malli. Käyttäjä pääse käsiksi tietokannan instanssiin, kutsumalla staattista metodia ”getInstance”. Metodi vaatii kuitenkin parametriksi kontekstin (esim. näkymä josta tietokantaa halutaan muokata). Lyhyesti sanottuna metodi luo luokan sisälle ainoastaan yhden privaatin olion, jonka se palauttaa aina metodia kutsuttaessa. Olio luodaan ainoastaan silloin, kun se ei ole olemassa.

1.27.2 Repositoriot (20.2.2018)

Käytin tietokannan rakenteessa apuna ”repository pattern”-mallia, jossa mm. taulukohtaiset asiat on listattu omaan luokkaansa. Tässä projektissa oli käytössä vain yksi taulu: ”kirja”. Tällöin ”Class.Database.Repositories”-paketin alta löytyy ainoastaan yksi repositorio, ”Kirja”. Jokainen repositorio on myös listattu omaan pakettiinsa.

Repositoriolta vaaditut toiminnot löytyvät ”Repository”-käyttöliittymästä (interface). Tähän kuuluvat seuraavat ominaisuudet:

- **getCreateTableIfNotExistsSQL:** SQL-lauseen palautus, jolla taulu luodaan
- **deleteTableIfExists:** taulun poistaminen tietokannasta, jos se on olemassa
- **getTableName:** taulun nimen palautus
- **clearTable:** tyhjentää taulun kaikista riveistä
- **add:** lisää annetun olion tietokantaan ja palauttaa lisättyjen määrän
- **getAll:** palauttaa kaikki taulun rivit olioina
- **getByID:** palauttaa olion tietokantarivin ”id”-tietueen mukaan
- **getFirst:** palauttaa ensimmäisen tietokantarivin olion
- **modify:** muokkaa parametrin oliota tietokantaan
- **delete:** poistaa parametrin olion tietokannasta
- **deleteFirst:** poistaa ensimmäisen rivin taulusta

Nämä ominaisuudet vaaditaan siis kaikilta repositorioilta, joita tietokannan kautta voidaan käsitellä.

”Database”-luokkaan listataan kaikki käytettävät repositoriot, josta ne ovat myös käytettävissä jäsenmuuttujan kautta (esim. ”KirjaRepository”). Tämän lisäksi ne on listattu privaattiin ”repositories”-listaan. Repositorien käyttö kuitenkin edellyttää ”Database”-luokan instanssia, josta oli maininta aiemmassa kappaleessa.

”Database”-luokka käyttää repositorien ominaisuuksia apuna tietokannan rakentamiseen mm. taulujen osalta. Mm. tietokantaa päivitettäessä se pyytää repositorien SQL-koodit taulun poistoon, jonka jälkeen se luo taulut uudelleen pyytämällä repositoreilta SQL-luontikoodit.

Luokka käyttää hyväkseen yllä mainittuja ominaisuuksia mm. tietokannan luontiin tai päivittämiseen. Repositoriot listataan aivan aluksi ”repositories”-listaan, josta Repositoriot taas käyttävät ”Database”-luokkaa manipuloidakseen ”SQLite”-tietokantaa.

Lyhyesti selitettynä repositoriot ovat vastuussa vain omasta taulustaan. ”Database”-luokka toimii varsinaisena rajapintana ”SQLite”-tietokantaan, mutta vastaa myös tietokannan rakenteen (schema) päivittämisestä.

1.27.3 ”Kirja”-tietokantataulu (20.2.2018)

Sovelluksen kirjat tallennetaan ”Kirja”-nimiseen tauluun. Tässä taulussa on seuraavanlaiset kentät:

- Automaattisesti generoitava ID-tunniste (integer primary key)
- Numero (integer, not null)
- Nimi (text, not null)
- Painos (integer, not null)
- Hankinta-päivämäärä (text, not null)

ID-tunniste asetetaan automaattisesti, kun tauluun lisätään uusi rivi. Kaikki muut tiedot on annettava lisäyksen yhteydessä.

1.27.4 ”Kirja”-luokka (20.2.2018)

Repositorio-luokan lisäksi samaan pakettiin luodaan aina tietokantataulua vastaava luokka. ”Kirja”-luokka on siis ”KirjaRepository”-luokkaa varten samassa paketissa on siis myös ”Kirja”-luokka, joka vastaa taulun rakennetta.

1.28 Listaus-näkymä (19.2.2018)

1.28.1 Ulkoasu

Tämä listaus-näkymä näytetään ohjelmaa avattaessa. Se vastaa ulkoasultaan tehtävänannon tyyliä.

Ulkoasun pohja on toteutettu ”TableLayout”-elementillä, jonka sisälle asiat on lisätty rivikohtaisesti. Tätä samaa tapaa käytettiin myös aiemmissa tehtävissä.

Ensimmäiselle riville on listattu ”TableRow”-elementin sisälle kaksi ”EditText”-komponenttia. Toinen on varattu numeron ja toinen nimen syöttämiseen. Toisella rivillä on samaan tapaan listattuna kentät painosnumerolle ja hankintapäivämäärälle. Kolmannella rivillä on napit uuden kirjan lisäykseen sekä listan ensimmäisen kirjan poistoon.

Viimeiselle riville on lisättynä uusi ”TableLayout”-elementti, jonka sisälle kirja-kohtaiset rivit lisätään UI-komponentteina. Aivan aluksi tämä ”TableLayout”-elementti on täysin tyhjä. Sen sisältö haetaan koodissa.

1.28.2 Logiikka

Näkymän ”onCreate”-metodi on ylikirjoitettu, ja siihen lisätty muutama asia. Ensimmäinen asia on tietokannan hakeminen staattisen ”Database.getInstance”-metodin avulla. Viite olioon tallennetaan ”database”-jäsenmuuttujaan.

Seuraavaksi täytetään näkymän listaus tietokannasta löytyvillä kirjoilla. Tähän käytetään ”KirjaRepository”-luokan metodia ”getAll”. Se palauttaa ”ArrayList”-tyyppisen listan löytyneistä kirjoista.

Jokaista löytynyttä kirjaa varten luodaan kaksi oliota:

1. TextView
2. TableRow

Ensimmäiseen elementtiin asetetaan kirjan nimi. Tämän jälkeen luodaan uusi ”TableRow”-elementti, jonka sisälle näytettävä teksti lisätään. Tämän jälkeen koko hoito lisätään näkymän ”TableLayout” lapseksi.

Muokkausta varten ”TableRow”-oliolle lisätään myös ”OnClick”-kuuntelija. Käyttäjän painaessa kirjan nimeä, siirrytään muokkausnäkymään, jossa kirjan tietoja voidaan muokata. Muokkaus-näkymä avataan ”edit_click”-metodissa, jota kuuntelija on asetettu kutsumaan. Se antaa parametrina ”View”-tyyppisen olion, joka vastaa käyttäjän napauttamaa elementtiä käyttöliittymällä.

Listauksen jälkeen ohjelma on valmis ja odottaa käyttäjän toimintoja. Käyttäjä voi lisätä tietokantaan uuden kirjan, lisäämällä näkymän yläosaan kirjan tiedot, ja painamalla tämän jälkeen ”add new”. Napin painaminen ajaa ”addnew_click”-metodin, joka parsii kaikki käyttäjän syöttämät tiedot. Jos jokin tieto on väärin, esimerkiksi ”numero”-kenttään onkin laitettu kirjaimia, tulostaa ohjelma virheilmoituksen ”Toast”-dialogina. Vasta kun kaikki tiedot menevät parsinnan läpi, yritetään tietokantaan lisätä uusi kirja.

Uuden kirjan luominen onnistuu luomalla uuden ”Kirja”-olion, jonka jälkeen olio annetaan tietokannan ”KirjaRepository”-luokan metodille ”add”. Metodi palauttaa paluuarvona ”boolean” arvon, joka ilmaisee lisäyksen onnistumisen. Lisäyksen jälkeen ajetaan ”listaaKirjat”-metodi, joka tyhjentää listatut kirjat ja täyttää ne tietokannan mukaisesti uudelleen.

1.29 Muokkaus-näkymä (19.2.2018)

1.29.1 Ulkoasu

Kirjan muokkaussivu on suurimmalta osalta kopioitu aiemmasta listaus-näkymästä. Tällä kertaa kaikki kirjan tiedot on kuitenkin jaettu omille riveilleen, jotta tietoja olisi helpompi lukea ja muokata. Kosta kirjojen ”id”-tieto on vain tietokannan käyttöön, se ei ole käyttäjän muokattavissa.

Käyttäjä voi tallentaa tehneet muokkauksensa ”tallenna”-napilla. Jos muokkauksia ei halua tallentaa, voi painaa laitteen ”takaisin”-näppäintä tai käyttöliittymän ”peruuta”-nappia.

1.29.2 Logiikka

Käyttäjän painaessa listattua kirjaa, se avataan tälle erilliselle muokkaus-näkymälle. Listaus-näkymässä listattua kirjaa esittävä UI-elementti ja kirja niputetaan yhteen ”setTag”-metodilla. Tällä voidaan tallentaa jokin ”object”-tyyppinen olio mihin tahansa UI-elementtiin.

Ennen muokkaus-näkymän avausta, listaus-näkymä lukee painetun rivin ”Kirja”-olion. Se lukee sen kaikki kentät ja lisää ne ”Bundle”-tyyppisen olion sisälle. Tämä paketti taas lisätään ”Intent”-olion sisälle, jota käytetään siirtymiseen sivulta toiselle.

En käyttänyt tässä tapauksessa ”Parcable”-rajapintaa olion lähettämiselle toiselle sivulle, vaan kaikki tiedot listattiin eriteltyinä avainpareina ”Bundle”-olion sisälle. Käytin avaimina ”KirjaRepository”-luokassa olevia taulun kenttien nimiä (esim. COLUMN_ID, COLUMN_NUMERO).

Vastaavasti muokkaus-sivua ladattaessa, kirjan tiedot luetaan samaisesta ”Bundle”-oliosta oikeilla avaimilla. Tiedot luetaan ylikirjoitetussa ”onCreate”-metodissa. Ensin on haettava ”Intent”-olio, jonka avulla sivulle tultiin. Tämän jälkeen on luettava sen mukana lähetetyn ”Bundle”-olion tietoja ”getIntExta” ja ”getStringExtra”-metodien kautta. Kun kaikki tiedot on parsittu ulos, voidaan luoda uusi ”Kirja”-olio ja antaa sille lähetetyt tiedot parametreina.

Aivan lopuksi uudelleen rakennettu kirja lisätään privaatiksi jäsenmuuttujaksi. Kirjan tiedot listataan käyttöliittymälle ”listaaKirja”-metodin avulla.

Näkymä käyttää tietojen validointiin ”Kirja”-olion settereitä. Jos jokin niistä heittää poikkeuksen tietoja tallennettaessa, tietoja ei kirjoiteta tietokantaan.

HARJOITUS 7-8 – FIREBASE

1.30 Projektin pohjustus (16.3.2018)

Aloitin työn luomalla projektille uuden repositorion ”harj7-8” GitHubiin, osoitteeseen:

https://github.com/W0lfw00ds/harj7-8

Loin tämän jälkeen uuden projektin Android studiossa, käyttämällä tyhjää pohjaa. Tehtävän mukaisesti tulen kopioimaan aiemmasta harjoituksesta kaiken tarpeellisen. Muutaman ohjevideon saattelemana, poistin paketista myös aiemmassa harjoituksessa luomani ”Database”-paketin. Tämän sijaan tultaisiin käyttämään Firebasen rajapintaa.

1.31 Firebase (17.3.2018)

Aloitin tutustumisen kirjautumalla Google-tililleni, sekä siirtymällä Firebasen kotisivuille osoitteeseen:

https://firebase.google.com

Painoin sen jälkeen sivulta ”Get started”, jonka kautta pääsin Firebase console-sivustolle. Tällä sivulla oli mahdollista luoda uusi Firebase projekti. Painoinkin heti ”Add project”-linkkiä.

1.31.1 Uuden Firebase-projektin luominen (17.3.2018)

Annoin uuden projektin nimeksi ”firebaseHarj78” sekä valitsin alue-asetukseen Suomen. Projektin ID:ksi listautui annetuilla tiedoilla ”fir-harj78”. Lyhyen latauksen jälkeen uusi projekti oli luotu. Firebasen kotisivu listasi tämän jälkeen liudan erilaisia komponentteja, mitä projektiin voisi lisätä.

1.31.2 Firebase Database-palvelun lisäys (17.3.2018)

Harjoitus 7:n tehtävänannon mukaisesti siirryin ensiksi lisäämään Firebase-projektiin tietokantaa. Valitsin projektin kotisivuilta ”Database”-ikkunan alta ”Get started”.

Firebase tarjosi minulle valittavaksi ”Realtime database”- tai ”Cloud firestore Beta”-tyyppisen tietokannan. Jälkimmäinen vaihtoehto kuulosti houkuttelevalta, mutta pidättäydin varmuuden vuoksi ensimmäisessä.

Tämän jälkeen eteeni aukeni valkea, konsoli-ruutu. Ruudun yläreunassa oli projektini ID ”fir-harj78”. Ajattelin että tämän ID esittää MySQL-tietokannan tavoin yhtä ”tietokantaa”. Minun tulisi nyt lisätä tietokannan alle taulut, joiden alle tulisi sitten tietokentät.

Kulutin tässä välissä todella paljon aikaa Firebasen tietokannan tutkimiseen. Aloitin opiskelun katsomalla opetusvideoita Youtubesta sekä selaamalla Firebasen virallisia tutoriaaleja. Tutustuin yleisimpiin käyttötapauksiin liittyen mm. tiedon hakemiseen Javalla. Lopulta olin valmis ohjelman tietokantarakenteen suunnitteluun.

1.31.3 Firebase-tietokannan rakenne (17.3.2018, 27.3.2018)

Pitkän prosessin jälkeen päädyin seuraavanlaiseen tietokantarakenteeseen. Firebasen tietokantamalli on hyvinkin erilainen, verrattuna mm. tavanomaiseen relaatiotietokantaan (esim. MySQL, Progress). Firebasen tietokanta rakentuu jättimäisenä, JSON-tyyppisenä tekstipuuna. Puun rakennetta on mahdollista tarkastella ja muokata palvelun verkkosivujen kautta.

Osa ohjeista kertoo, ettei tietokantaa ole välttämättä järkevä lisätä lainkaan tietoja itse verkkosivujen kautta. Oikeammaksi tavaksi mainitaan tietojen lisäys suoraan omasta ohjelmasta, joka rakentaa puun. Käytin verkkosivun työkalua kuitenkin apuna opiskelussa, ja rakensin sen avulla erilaisia JSON-puita kokeilumielessä.

Päädyin lopuksi listaamaan jokaisen käyttäjän tiedot erillisesti oman käyttäjätunnuksensa (uid) alle omina puinaan. Tämä mahdollistaa mm. sen, että kaikki käyttäjän tiedon ovat helposti poistettavissa, kun pelkkä käyttäjätunnuksellinen ”juuri” puussa poistetaan. Vaihtoehtoisia tapoja olisi myös ollut mm. listata erilaiset asiat omiin puihinsa, ja käyttäjätunnus olisi tallennettuna erillisenä kenttänä, esimerkiksi seuraavasti:

```
"Kirja" : [
    {
        "uid" : "tamaOnUniikkiKayttajatunnus",
        "hankintapvm" : "21.02.2017"
    }
]
```

Ohjelma 1. *Esimerkki toisenlaisesta tietorakenteesta, jossa käyttäjä on osa tietoa.*

Firebasen tietokanta ja autentikointi on valmiina linkitetty toisiinsa. Jokaiselle rekisteröityneelle käyttäjälle generoidaan oma ”uid”, eli uniikki ID. Tämä tunnus on ohjelmakohtainen. Toisin sanottuna, tähän Firebase-projektiin luotujen käyttäjien ”uid” toimii ainoastaan tämän projektin sisällä. Viralliset dokumentit suosittelevat käyttämään tätä käyttäjän tunnistamiseen. Sisällöltään ”uid” on satunnainen merkkijono (esim. ”pGC6sgeNnWUPtr69TZAHc3xTbU2”). Firebasen ”Authentication”-palvelusta on lisää tietoja myöhemmässä luvussa.

```
{
    "pGC6sgeNnWUPtr69TZAHc3xTbU2" : {
        "Kirja" : {
            "-L8PajU_5yCSvgjLKjN8" : {
                "hankintapvm" : "23.07.2005",
                "nimi" : "asd456",
                "numero" : 10,
                "painos" : 4
            },
            "-L8Pak9T0ZUXf83Rxt90" : {
                "hankintapvm" : "23.07.2005",
                "nimi" : "asd456tjh",
                "numero" : 2,
                "painos" : 4
            }
        }
    }
}
```

Ohjelma 2. *Lopullisen Firebase-tietokannan rakenne*

Yllä olevassa koodissa on esimerkki lopullisesta tietokantarakenteesta, jota käytin harjoituksen ratkaisussa. JSON-koodi on otettu ulos Firebasen konsolin ”Export JSON”-työkalun avulla.

Koodista voidaan huomata, että ”Kirja”-puun alle lisätyt kokonaisuudet on listattu erikoisten avainten alle (esimerkiksi "-L8PajU_5yCSvgjLKjN8”). Nämä ovat Firebasen automaattisesti luomia, uniikkeja avaimia. Kaikki ”Kirja”-puun alle lisätyt kokonaisuudet on luotu Android-ohjelmassa kutsumalla Firebasen ”push”-komentoa, joka luo nämä avaimet automaattisesti. Avaimen alle on listattuna varsinaisen kirjan tiedot, jotka lisättiin antamalla Firebaselle suoraan Java-olioita.

1.31.4 Firebase-tietokannan säännöt

Ilman minkäänlaisia sääntöjä tai ”oikeuksia”, ei tietokannan käyttämisestä tulisi oikein mitään. Esimerkiksi jos kaikki voivat lisätä ja poistaa tietoa täysin vapaasti, ei tietokantaa käyttävä ohjelma voi toimia. Tämän vuoksi on erittäin tärkeää, että jokaiselle tietokannalle olisi määritelty ainakin jonkinlaiset säännöt (rules). Säännöt koostumat mm. siitä, kuka saa tehdä mitä ja minne.

Firebase-tietokanta ei tue MySQL-tietokannan tavoin ”schemaa”, eli ns. ”mallia”, mikä määrittäisi eri tietojen tietotyypit ja taulujen rakenteet. Tietokantaa on kuitenkin mahdollista ”mallintaa” säännöillä. Nämä säännöt ovat kuitenkin rakenteeltaan ja luonteeltaan täysin erilaisia kuin perinteisten tietokantojen ”schemat”, enkä menisi niitä edes keskenään vertaamaan.

Tietokannan säännöt saa auki Firebase-konsolin ”rules”-välilehdeltä. Hauskan säännöistä tekee se, että ne kirjoitetaan tiedon tapaan JSON-kielellä. Tämän ohella JSON-elementteihin on mahdollista lisätä tekstimuodossa myös ”JavaScript”-kieltä. JavaScriptillä voidaan mm. tehdä dynaamisia ”if”-lauseita. Kaikkein yksinkertaisin tällaisesta voisi olla pelkkä ”true”.

Seuraavalla sivulla on koodi, jota käytetään tämän harjoituksen tietokannan säännöissä.

```
{
  "rules": {

    "$uid": {
      ".read": "$uid === auth.uid",
      ".write": "$uid === auth.uid",
      "$unknown": { ".validate": false }, // Lock fields

      "Kirja": {
        "$kirjaid": {
          "$unknown": { ".validate": false }, // Lock fields

          "hankintapvm": {
            ".validate": "newData.isString() &&
                          newData.val().length>0"
          },
          "nimi": {
            ".validate": "newData.isString() &&
                          newData.val().length>0"
          },
          "numero": {
            ".validate": "newData.isNumber() &&
                          newData.val()>0 &&
                          newData.val()<90000000"
          },
          "painos": {
            ".validate": "newData.isNumber() &&
                          newData.val()>0 &&
                          newData.val()<90000000"
          }
        }
      }
    }
  }
}
```

Ohjelma 3. *Harjoituksen tietokannan säännöstö.*

Yllä oleva säännöstö rajoittaa käyttäjän oikeudet vain muokkaamaan oman ”uid”-tunnuksensa alla olevia tietoja. Käyttäjän on myös oltava kirjautunut, jotta ”uid”-tunnus olisi luettavissa.

Dynaamisiin solmuihin (esim. ”uid”) viitataan säännöissä $-merkillä. Toisin sanottuna, juuren alla voi olla minkä nimisiä solmuja tahansa, kunhan tämän dynaamisen ”uid”-solmun säännöt pätevät.

Käyttäjää voidaan estää luomasta tuntemattomia dynaamisia kenttiä, käyttämällä yksinkertaista ”.validate: false”-koodia. Tämä tarkoittaa, että aina kun käyttäjä on lisäämässä ”uid”-solmun alle jotakin dynaamista lapsisolmua, se ei läpäise tätä sääntöä. Tässä säännön määrittämiseen käytettiin ”$unknown”-nimistä dynaamista solmua. Nimi olisi voinut olla kuitenkin mikä tahansa muukin. Tämä yksinkertainen sääntö antaa meille puitteet rakentaa kiinteitä rakenteita, kuten olen yläpuolella tehnyt. Lyhyesti tämä tarkoittaa sitä, että käyttäjä voi rakentaa itselleen ainoastaan tietynlaisen puun.

Dynaamisen ”uid”-solmun alle on listattuna ”Kirja”-solmu, jonka alle kirjat listataan uniikkeiden tunnusten alle. Kirjan uniikkia ja dynaamista solmua merkataan säännöissä nimellä ”$kirjaid”.

Varsinaisen kirjan tietoja esittävät kentät (esim. ”hankintapvm”) sisältävät omat tarkastuksensa. Mm. tässä ”hankintapvm” sisältää säännön, että asetettavan arvon on oltava merkkijono (String). Toinen ehto se, että merkkijonon on oltava väh. 1 merkin pituinen.

Säännöistä riittäisi kerrottavaa vaikka vuosiksi, mutta en ala tässä kohtaa enää avaamaan niitä sen enempää.

1.31.5 Firebase Authenticate-palvelun lisäys

Ennen kuin Firebase-tietokantaa voidaan käyttää järkevästi, kannattaa lisätä tuki Firebasen ”Authenticate”-ominaisuudelle. Android Studiossa ”Firebase Assistant”-työkalu myös kehottaa palvelun lisäämistä, ennen kuin tietokanta otetaan käyttöön. Autentikoinnin kautta tietokantaan tehtäviä hakuja voidaan rajata mm. käyttäjätilin mukaisesti. Voidaan esimerkiksi luoda säännöt, jonka perusteella käyttäjä voi muokata tai poistaa ainoastaan tietoja, vaikka hänen käyttäjätunnuksensa perusteella. Autentikointi on aktivoitavissa saman verkkosivun kautta kuin tietokantakin, linkillä ”Authentication”.

Sivun kautta alussa tärkein välilehti on ”Sign-in method”. Tämän sivun kautta listataan, mitä kirjautumismenetelmiä Firebase-projekti tukee. Valitsin käytettäväksi Google- sekä ”Email/Password”-tyyppiset tilit. Google-tili on esimerkiksi Googlen sähköposti, muotoa ”etunimi.sukunimi@gmail.com”. Jälkimmäinen vaihtoehto taas antaa käyttäjän käyttää mitä tahansa sähköpostia ja valittua salasanaa tunnuksen luomiseen. Firebasen Authenticate-palvelu on nyt valmis käytettäväksi.

Firebasen uudemmat versiot tukevat Googlen valmiiksi kehittämää kirjautumislogiikka, joka on lisättävissä ”FirebaseUI”-paketista. Osa ohjeista oli vanhoja, ja ehdottivat täysin oman kirjautumislogiikan lisäystä. Tähän ei onneksi ollut tarve, vaan sain tehtyä kirjautumisen suoraan tuon valmiin paketin avulla.

1.31.6 Firebase-tuen lisäys Android-ohjelmaan

Katsoin internetistä ohjeita Firebasen lisäämiseen, ja asia vaikutti hieman työläältä. Ohjeissa neuvottiin lisäämään tietoja mm. sovelluksen manifestiin ja mm. Gradlen tiedostoihin. Tähän oli kuitenkin tullut muutos, koska Android Studion kehittäjät olivat lisänneet kehitysympäristöön ”Firebase Assistant”-ominaisuuden.

Tämä työkalu sisältää kaikki oleellisimmat ominaisuudet Firebase-palveluiden lisäämiseen todella helposti. Aloitin asennuksen valitsemalla ”Tools -> Firebase”, jonka jälkeen työkalu listasi minulle liudan Firebasen tukemia ominaisuuksia. Valitsin näistä reaaliaikaisen tietokannan: ”Save and retrieve data”. Asennus alkaa kirjautumalla omaan Firebase-tiliin. Kirjautumisten jälkeen Android Studio ja Firebase-tili on yhdistetty toisiinsa.

Jokaiselle Firebasen ominaisuudelle oli oma osionsa ”Firebase Assistant”-työkalussa. Tietokannan konfigurointi ehdotettiin tehtäväksi vasta autentikoinnin jälkeen, jonka myös tein. Lopulta olin ajanut molempien ominaisuuksien asennukset läpi, ja asennuksien kaikki vaiheet näyttivät vihreää ruksia.

Tässä vaiheessa kohtasin ensimmäisiä ongelmia. Projektin vanhimmaksi SDK:si oli asetettu taso 15, mutta Firebase vaati vähintään tasoa 16. Tein muutoksen ohjelmatason ”build.gradle”-tiedostoon. Muutoksen jälkeen studio herjasi kuitenkin yhteensopivuusongelmasta ”Firebase”-kirjastojen kohdalla. Kun nostin ”min SDK”-tasoa, se antoi myös käytettäväksi uudemman version Firebasen kirjastoista. Nostin tietokannan ja autentikoinnin kirjastojen versionumeron ”11.8.0”. Vielä tämänkin jälkeen editori ehdotti nostamaan version ”12.0.0”, mikä ei kuitenkaan ollut enää yhteensopiva kaikkien muiden asioiden kanssa, joten pidättäydyin jo muuttamassani arvossa.

1.32 Jatkokehitetty Database-paketti

Käytin tuhottoman paljon aikaa aiemman harjoituksen ”Database”-paketin jatkokehitykseen. Kun oivalsin Firebasen toimintarakenteen, halusin jalostaa sen toimimaan ”Repository”-patterin kautta. Tietokantaa siis käytettäisiin samalla idealla kuin aiemmin: jokaiselle ”taululle” olisi oma repositorio-luokka, jonka kautta tietoa käsiteltäisiin. Idea olisi luoda yhtenäinen ja ”geneerinen” rajapinta sekä SQLite- että Firebase-tietokannalle.

Kulutin aikaa sen takia niin paljon, koska uskon voivani hyödyntää pakettia esimerkiksi omissa Android-ohjelmissani.

1.32.1 Firebasen ja SQLiten tietokantaluokat

Kummallekin tekniikalle on luotu omat ”pohjaluokkansa”, jotka eivät ole keskenään mitenkään sidoksissa. ”FirebaseDatabase”-luokka löytyy oman ”Firebase”-pakettinsa alta, kun taas ”SQLiteDatabase”-luokka ”SQLite”-paketin alta.

Näiden luokkien yksinkertainen tarkoitus on tarjota kehittäjälle referenssit repositoreihin, joiden kautta tietoa voidaan muokata. Molempien rakenteessa on käytetty ”Singleton”-patternia, eli käyttäjä voi luoda vain yhden käytettävän instanssin. Instanssi saadaan kutsumalla luokkien staattista metodia ”getInstance”.

Kuten aiemmassa harjoituksessa, SQLiteDatabase-luokka on peritty SQLiteOpenHelper-luokasta. Rakenne on hyvin samanlainen kuin aiempi versio. FirebaseDatabase-luokkaa ei periytetä mistään.

SQLiteDatabase-luokka viittaa ”SQLiteRepository”-tyyppisiin repositoreihin. FirebaseDatabase-luokka taas ”FirebaseRepository”-tyyppisiin repositoreihin. Kaikki repositoriot pohjautuvat yhteen ja samaan, geneeriseen ”Repository”-luokkaan. Tästä on lisää tietoa seuraavissa kappaleissa.

1.32.2 Entity-luokka

Tässä tietokantakirjastossa muokataan tietoja olioiden perusteella. Jotta tämä olisi mahdollista, on kaikki tietokannassa käytettävät oliot periytettävä tästä ”Entity”-luokasta. Luokka takaa, että kaikilla tietokannan kanssa käytettävillä olioilla on uniikki, merkkijonotyyppinen avain: ”key”. Avain on uniikki tunniste, jolla olio on mahdollistaa erottaa ja sen tietoja voidaan päivittää.

Firebase-tietokanta luo automaattisesti merkkijono-tyyppisiä avaimia ”push”-ominaisuutta käytettäessä. Tämän takia ”Entity”-luokan avain on tyypiltään merkkijono, eikä esimerkiksi kokonaisluku. SQLite-tietokannassa on kuitenkin käytössä automaattinen avaimen generointi, joka on tyypiltään kokonaisluku. Käytettäessä SQLiteRepository-

luokkaa, on entiteetin avaimen oltava arvoltaan numeerinen. Tällöin avain muutetaan kokonasluvuksi ennen tietokantaan lisäystä. Tietokannasta luettaessa avain muutetaan taas merkkijonksi. Tämä on harmillista, mutta loppukäyttäjän ei ole periaatteessa edes tarkoitus muokata olioiden avaimia. Luultavasti muokkaan rakennetta vielä myöhemmin, jos keksin tilalle jotakin parempaa.

Avaimen lisäksi tietokanta käyttää hyväksi ”Entity”-luokasta perityn luokan nimeä. Harjoituksessa käytettiin ainoastaan yhtä entiteettiä ”Kirja”. Tällöin SQLite-tietokannassa taulun nimeksi tulee ”Kirja”, sekä Firebasen tietokannassa kirjat sisältävän solmun avaimeksi ”Kirja”.

Tietokantaan tallennettavien ”taulujen kenttien nimet” luetaan dynaamisesti ”Entity”-olion luokan rakenteesta Javan ”Reflection”-kirjastolla. Mm. harjoituksen ”Kirja”-entiteetillä on yksityinen ”numero”-kokonaislukumuuttuja. Kaikki luokan yksityiset muuttujat luetaan läpi, ja samoja muuttujien nimiä tullaan käyttämään tietokannan kentissä. SQLite-tietokannassa nimiä käytetään taulun kolumnina, Firebasessa taas avainlukuparin avaimena.

”Entity”-luokka implementoi Androidin ”Parcelable”-rajapinnan. Tämän vuoksi kaikkien periytettyjen entiteettien, esim. ”Kirja”, on implementoitava tämä rajapinta. ” Parcelable”-rajapinnan avulla olio voidaan muuttaa ”avain-arvopari”-olioksi, jossa sen jäsenmuuttujien arvot on tallennettuna yksinkertaisina Java-tyyppeinä (esim. String, int jne). Aiemmassa harjoituksessa olio jouduttiin pilkkomaan ”Bundle”-pakettiin, kun näkymästä siirryttiin toiseen näkymään. Nyt tämä logiikka on toteutettu suoraan siirrettävän olion sisällä. ”Parcelable”-rajapinnan toteuttavat oliot voidaan suoraan lisätä sellaisenaan ”Intent”-olion mukaan.

1.32.3 Repository-luokka

Kaiken yhteen niputtavana pohjana toimii tietokantapaketin abstrakti ”Repository”-luokka. Sen idea pohjautuu Firebasen tapaan, jossa tietoja päivitetään suoraan olioiden avulla. Kehittäjän ei siis tarvitse käyttää esimerkiksi perinteistä SQL-kieltä tietojen päivittämiseen, vaan hän voi suoraan antaa muokatun olion repositorion ”modify”-metodille. Luokkaan on listattu kaikki yleisimmät ”CRUD”-operaatiot (create, read, update, delete). Nämä operaatiot antavat kehittäjän siis lisätä, hakea, päivittää ja poistaa tietoja olioiden perusteella.

Tietojen haku voi olla rajapinnoissa usein asynkroonista tai synkroonista. SQLite on toteutettu Androidissa toimimaan synkroonisesti, kun taas Firebase toimii asynkroonisesti. ”Repository”-luokan rakenne on tehty niin, että se tukee sekä asynkroonisia että synkroonisia tietokantoja. Idea on käyttää yksinkertaisia ”callback”-metodeita, jotka tietokantaa kutsuva ohjelma antaa. ”Repository”-luokan pohjalle on määritelty muutama erilainen rajapinta (interface), joiden avulla kyselyjä tehdään.

Aivan pohjimmaisena rajapintana toimii ”ErrorListener”-rajapinta. Sillä on ainoastaan yksi metodi ”onError”, jonka parametrina on ”DatabaseException”-tyyppinen olio. Tästä rajapinnasta on sen jälkeen periytetty kolme muuta rajapintaa:

- ResultListener
- ResultItemListener<T>
- ResultItemsListener<T>

Geneerinen tyyppiparametri “T” on oltava ”Entity”-luokasta periytetty luokka. Tässä harjoituksessa se voisi siis olla ”Kirja”. Kaikki kolme rajapintaa sisältävät saman metodin ”onSuccess”. Ainoa ero on sen parametreissa. ”ResultListener” ei vaadi parametria lainkaan, ”ResultItemListener” vaatii yhden entiteetin, sekä viimeinen ”ResultItemsListener” vaatii listan entiteettejä.

”Repository”-luokassa on toteutettu osa kyselyistä niin, että osa vaatii ”ResultListener”-rajapinnan, osa ei. Esimerkiksi ”getAll”-metodi vaatii tällaisen olion, koska käyttäjä saa rajapinnan kautta tietokannasta löydetyt oliot ”onSuccess”-metodin parametrina. Ilman tätä oliota, käyttäjä ei voisi saada tuloksia mitenkään. Jos jokin menee kuitenkin pieleen, ajetaan rajapinnan ”onError”-metodi. Idea on, että kehittäjä loisi kumpaankin tapaukseen omaan logiikan – mitä tehdään kun kysely palauttaa listan olioita, tai mitä jos kysely päättyy virheeseen?

Aiemmin mainitun ”getAll”-metodin ”onSuccess” saa parametrina listan löytyneitä olioita. ”getFirst”-metodin ”onSuccess” taas saa parametrina vain yhden entiteetti-olion. ”delete”-metodin ”onSuccess” ei saa mitään parametreja. Erilaisia ”callback”-tapauksia on siis kolme: ei palautettavaa oliota, yksi palautettava olio, tai lista palautettavia olioita.

1.32.4 SQLiteRepository-luokka

Tämä luokka perii aiemmin mainitun, abstraktin ”Repository”-luokan. Tämä on myös tyypiltään abstrakti luokka. Luokka toteuttaa geneerisesti ja dynaamisesti kaikki ”Repository”-luokan vaatimat kyselyt.

Kuten jo aiemmin mainittiin, SQLite-tietokantaa oli tarkoitus käyttää samoin kuin Firebase-tietokantaa. Tämä ei kuitenkaan ollut kovin yksinkertainen tai helppo tehtävä. Vaikeutena oli ainakin nämä muutama asia:

- SQL-kielellä toteutetut kyselyt
- ”Entity”-luokasta perityn olion yksityisten jäsenmuuttujien lukeminen ja asetus
- Java-tietotyyppien muuttaminen SQLite-tietotyypeiksi ja toisinpäin.
- ”Entity”-luokasta perityn olion muuttaminen ”ContentValues”-olioksi
- ”Entity”-luokasta perityn olion luominen ”Cursor”-olion perusteella dynaamisesti

Helpoimmin yllä mainitut asiat olisi voinut tehdä luomalla jokaiselle ”Entity”-luokasta peritylle oliolle vahvasti tyypitetyt repositoriot, jossa jokainen asia olisi erikseen toteutettu. Tämä veisi kuitenkin todella paljon aikaa. Esimerkiksi 20 erilaista tietokantataulua vaatisi 20 implementoitua repositoriota, josta jokainen sisältäisi esimerkiksi 15-20 metodia. Myös suurin osa koodista olisi duplikaattikoodia.

Käyttämällä dynaamista ja geneeristä rakennetta, meille riittää kuitenkin tämä yksi toteutus. Se toimii kaikentyyppisille entiteeteille. Sen ainoa huono puoli on Javan ”Reflection”-kirjaston hitaus.

Kolmantena ja viimeisenä tasona on jokaista entiteettiä varten luotu repositorio-luokka, joka perii ”SQLiteRepository”-luokan. Harjoituksessa on ainoastaan yksi tällainen, ja se on nimeltään ”KirjaSQLiteRepository”. Tämä luokka sisältää siis kaikki aiemmin mainitun ”SQLiteRepository”-luokan kyselyt. Tämän jälkeen luokan sisälle voidaan lisätä kaikki entiteetti-kohtaiset kyselyt. Esimerkiksi ”KirjaSQLiteReposity”-luokkaan voitaisiin lisätä omat ”get”-metodit, jollai haettaisiin tiettyä kirjaa mm. kirjan nimen tai hankintapäivän perusteella. Harjoituksessa tällaisia entiteetti-kohtaisia kyselyitä ei tarvittu, vaan ”SQLiteRepository”-luokan kyselyt olivat tarpeeksi.

1.32.5 FirebaseRepository-luokka

Tämän luokan toiminnallisuus on täysin sama kuin ”SQLiteRepository”-luokan. Luokka sisältää oletustoteutuksen geneerisestä ja abstraktista ”Repository”-luokasta. Tämän toteutus oli paljon yksinkertaisempi kuin ”SQLiteRepository”-luokan. Osa syynä oli se, että Firebasen kirjasto tuki suoraan olioiden käyttöä tietokannan päivittämisessä.

```
com.google.firebase.database.FirebaseDatabase.getInstance().getReference()
  .child(FirebaseDatabase.getUserRootNodeName())
  .child(this.getTableName()) // Use inherited class’ name as table name
  .push() // Generates unique string key for each entity
  .setValue(entity, new DatabaseReference.CompletionListener() {

    @Override public void onComplete(
      DatabaseError databaseError,
      DatabaseReference databaseReference) {

      if (databaseError != null) {
        if (resultListener != null) {
          resultListener.onError(new DatabaseException("Entity wasn't added to
the database!", databaseError));
        }
       } else {
         if (resultListener != null) {
           resultListener.onSuccess();
       }
     }
   }
});
```

Ohjelma 4. *”add”-metodin toteutuksessa käytettiin ”Firebase”-kirjaston valmiita ratkaisuja.*

1.33 Periytetty ”Application”-luokka

Koska en henkilökohtaisesti tykkää kovin paljon staattisista muuttujista, halusin muokata Android-ohjelman aloittavaa ”Application”-luokkaa. Loin projektiin uuden Java-luokan harjoituksen nimellä ”harjoitus78”. Tämä luokka periytyy ”Application”-luokasta, sekä ylikirjoittaa tämän ”onCreate”-metodin.

Android luo jokaista ohjelmaa varten oman ”Application”-instanssin. Se on siis oiva paikka lisätä mm. referenssit Firebase- ja SQLite-tietokannoille. Vaikka ”Database”-paketin luokat tukevat jo singleton-patternia, halusin siirtää ne tänne.

Ylikirjoitetussa ”onCreate”-metodissa otetaan talteen ohjelman ”Context”-olio. Mm. SQLite vaatii ”Context”-olion toimiakseen. Olio tallennetaan yksityiseen ja staattiseen jäsenmuuttujaan ”context”. Tämän avulla voimme kutsua ”Context”-olion vaativia metodeja myös paikoista, joista se ei ole saatavissa. Olio on mahdollista saada esimerkiksi Androidin ”Activity”- tai ”Application”-olion sisällä.

```
this.getApplicationContext();
```

Ohjelma 5. *”Context”-olion haku ”Application”-olion sisällä.*

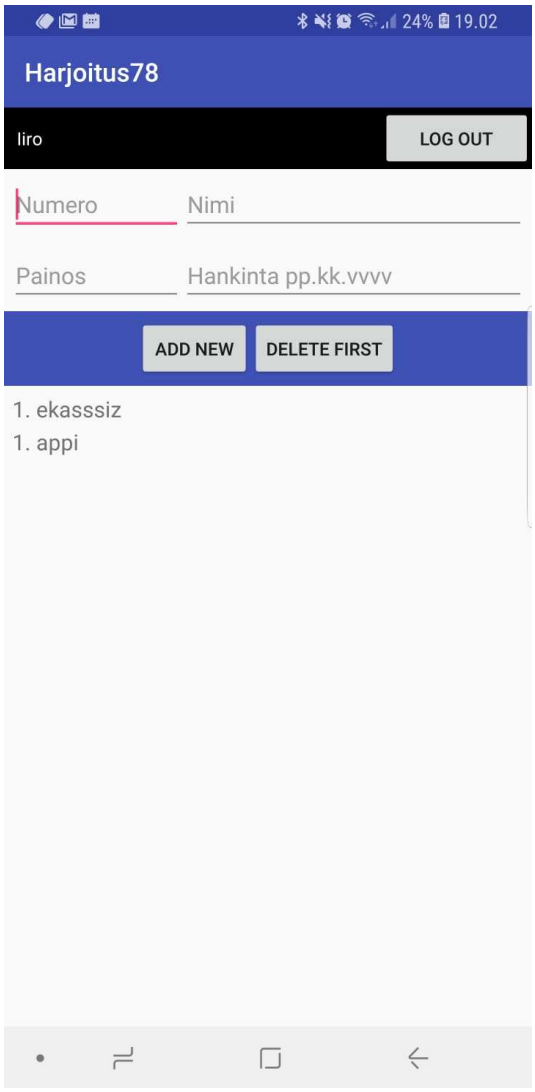
1.34 Päänäkymä

Tämän näkymän pääidea pysyy samanlaisena kuin aiemmassakin harjoituksessa. Käytin pohjana aiemman harjoituksen käyttöliittymän ja logiikan koodeja.

Käyttöliittymän suurin ulkoinen muutos on kirjautumispalkki, jonka kautta käyttäjä voi suorittaa ”autentikoinnin”. Autentikointi suoritetaan aiemmin mainitun ”FirebaseUI”-paketin avulla. Kirjautumisen tilaa voidaan seurata ”FirebaseAuth.getInstance().getCurrentUser()”-metodin kautta. Ohjelmassa olisi myös voinut käyttää kuuntelijaa, joka seuraa käyttäjän kirjautumistilaa. Kirjautuneen käyttäjän nimi kirjataan palkin etupäähän, sekä kirjautumiseen käytettävä nappi muuttaa tilaansa kirjautumistilan mukaisesti.

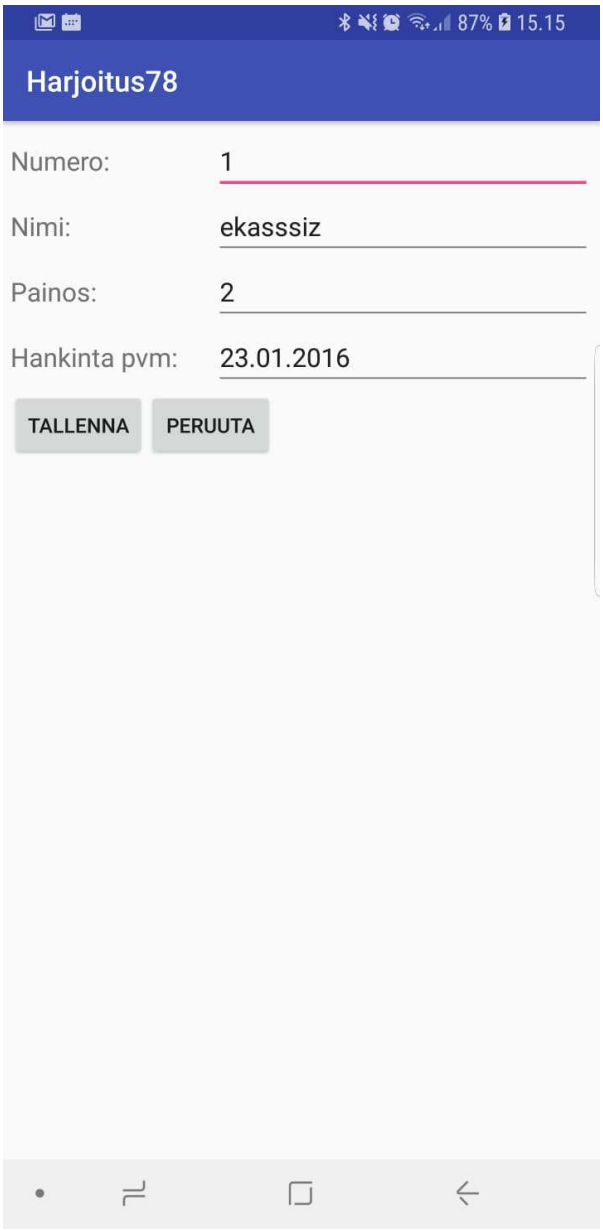
Kirjautunut käyttäjä määrittää, mitä tietoja ohjelma näyttää. Ilman kirjautumista, Firebase-tietokannasta ei voi oikeastaan lukea mitään tietoja.

Ohjelma käyttää aiemman harjoituksen tapaan päivitettyä ”Database”-pakettia tiedon hakemiseen. Tämän näkymän puolella on vain määritelty, mitä tapahtuu jos esimerkiksi tiedonhaku päättyy virheeseen.



Kuva 2. *Päänäkymä, jossa on listattuna kaksi kirjaa.*

1.35 Kirjan editointinäkymä



Kuva 3. *Kirjan editointinäkymä.*

Aiemman harjoituksen mukaisesti tämä näkymä on pysynyt samanlaisena. Pinnan alla suurin muutos oli kuitenkin ”Parcelable”-rajanpinnan tuki. Aiemmassa harjoituksessa olio pilkottiin ”Bundle”-pakettiin, jonka avulla se siirrettiin tähän näkymään. Tässä näkymässä olio taas luotiin ”Bundle”-paketin kautta uudelleen. Tässä uudessa versiossa ”Kirja”-olio voidaan suoraan lisätä tai hakea siirtymissä käytetyistä ”Intent”-olioista ilman ylimääräistä parsintaa tai uudelleenkokoamista.

HARJOITUS 9-10 – GPS JA GOOGLE MAPS

1.36 Projektin pohjustus (5.3.2018)

Aloitin työn luomalla projektille uuden repositorion ”harj9-10” GitHubiin, osoitteeseen:

https://github.com/W0lfw00ds/harj9-10

Loin tämän jälkeen uuden projektin Android studiossa, täysin samaan tapaan kuin aiemmissakin harjoituksissa. Valitsin työkalun ehdottaman tyhjän aloituspohjan. Rajoitin tällä kertaa ”min SDK” eli viimeisimmäksi Android-versioksi ”API 21” (”Android 5.0 Lollipop”). Tein tämän osittain siksi, että haluan saada käyttööni uudempia versioita mm. Google Maps-kirjastosta.

1.37 Idea (5.3.2018)

Halusin tehdä yhden yksinkertaisen mutta toimivan ohjelman, jossa pääsisin tutustumaan Google Maps-komponentin ja GPS-rajapinnan saloihin.

Ohjelman ulkoasu listaa yläreunassa GPS-signaalin arvot (mm. ”altitude” (korkeus), ”latitude” (leveysaste), ”longitude” (pituusaste)). Listauksen alapuolella on toinen paneeli, jossa on Google Map-komponentti sekä siihen liittyvät säätimet.

1.38 GPS-rajapinnan käyttöönotto

Googlen rajapinnan käyttöönotto sai päähäni harmaita hiuksia. Palvelu vaati ”billing”- eli laskutustietojen antamista. Koitin kiertää asian muutamaan otteeseen, mutta jouduin lopulta antautumaan.

Google Maps-rajapinnan palvelun saa auki osoitteella:

https://console.cloud.google.com/google/maps-apis/

Nopean rekisteröitymisen ja laskutustietojen antamisen jälkeen, sain rajapinnalle oman API-avaimeni. Ilman tätä avainta, rajapintaa ei voi käyttää.

1.39 Applikaation aloitus

Ennen oman applikaationi kehitystä, tein Android Studiolla hetkellisen ”Map Activity”-projektin. Käytin siinä pohjana Android Studion valmista projektia, jossa käytetään Google Map-karttaa sekä GPS-antureita.

Kopioin projektista varsinkin ”AndroidManifest.xml”-tiedostossa määritetyt käyttöoikeudet (”ACCESS_FINE_LOCATION”, ”ACCESS_COARSE_LOCATION sekä ”INTERNET”). Tämän lisäksi samaan tiedostoon listataan ”meta-data”-tyyppinen elementti, johon annetaan aiemmin saatu GPS-rajapinnan API-avain. Listasin varsinaisen avaimen ”google_maps_api.xml”-tiedostoon, jonka jälkeen asetin sen ”AndroidManifest.xml”-tiedostossa olevan ”meta-data”-elementin arvoksi referenssillä.

Googlen virallisia ohjeita seuraamalla, varsinaisen Google Maps-komponentin lisäys käyttöliittymään oli myös varsin vaivatonta. Komponentti lisätään näkymään ”fragment”-tyyppisenä elementtinä esimerkiksi seuraavalla koodilla:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
                    android:id="@+id/SupportMapFragment_map"
                    android:layout_width="match_parent"
                    android:layout_height="200dip"
                    tools:context=".MainActivity"
                    android:name="com.google.android.gms.maps.SupportMapFrag-
ment"/>
```

Ohjelma 6. *Google-Maps-karttakomponentin lisäys näkymään.*

1.40 GPS-rajapinnan käyttöoikeuksien kysely

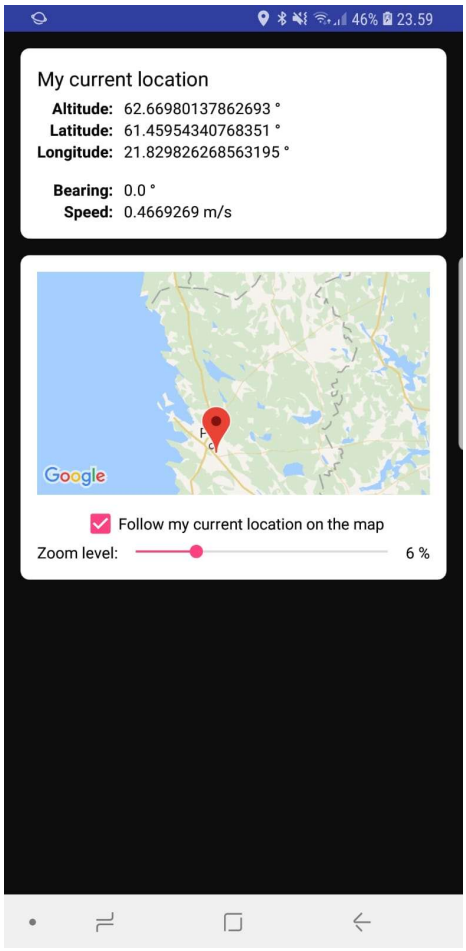
Android haluaa, että käyttäjä pyytää koodissaan vielä erikseen lupaa GPS-anturien käyttöön. Ilman erillistä pyyntöä, GPS-antureita ei voi käyttää. Toteutin pyynnöt oletusnäkymän ”onCreate”-metodissa seuraavasti:

```
ActivityCompat.requestPermissions(
  this,
  new String[] {
    Manifest.permission.ACCESS_FINE_LOCATION,
    Manifest.permission.ACCESS_COARSE_LOCATION
  },
  1
);
```

Ohjelma 7. *Sijaintitietojen käyttöoikeuksien pyytäminen.*

1.41 Lopullinen ulkoasu

Päädyin lopullisessa versiossa seuraavanlaiseen ulkoasuun:



Kuva 4. *Ohjelman lopullinen ulkoasu*

1.42 GPS-anturitietojen ja kartan yhdistäminen

Oletusnäkymän luokka implementoi ”OnMapReadyCallback”-rajapinnan sekä ”LocationListener”-rajapinnan.

”OnMapReadyCallback”-rajapinta toteuttaa ”onMapReady”-metodin, joka ajetaan sitten, kun kartta on ladattu käyttöliittymässä olevaan ”fragment”-elementtiin. Kartan lataus aloitetaan asynkronisen ”getMapAsync”-metodin ajamisella. Metodi saa parametriksi ”GoogleMap”-tyyppisen olion, jonka kautta karttaa on mahdollista manipuloida. Talletan tämän olion luokan paikalliseen muuttujaan.

GPS-signaalien kuuntelu tapahtuu ”LocationListener”-rajapinnan kautta. Sen toteutus vaatii neljän eri metodin määrittelyä. Näistä tärkein on ”onLocationChanged”-jota kutsutaan aina, kun mobiillilaitteen sijainti muuttuu tarpeeksi. Tämän metodin sisällä on myös kaikki ohjelman toiminnan kannalta tärkein logiikka. Sen sisällä päivitetään yläreunan

paneelin GPS-parametrit sekä liikutetaan kartta-komponentin kuvaa. Viimeisin paikkatieto tallennetaan myös luokan jäsenmuuttujaan ”Location” sekä ”LatLng”-tyyppisinä olioina. Näistä jälkimmäinen on rakennettu ”Location”-olion pohjalta.

Kartan manipulointi on siirretty oman ”updateMarker”-metodin sisälle. Jos luokan ”GoogleMap”- tai ”LatLng”-oliot ovat alustamattomia, metodi ei tee mitään. Toisin sanottuna, metodi odottaa niin kauan, kunnes käyttöliittymän kartta sekä anturin kautta saatu sijaintitieto ovat molemmat saatavilla. Tämä vastaa harjoituksen kysymykseen, miten nämä asiat saa toimimaan keskenään.

”updateMarker”-metodi luo karttaan ”neulan”, joka sijoitetaan saatujen GPS-koordinaattien mukaisesti. Neulaan on myös lisätty teksti ”Me”, joka näkyy sitä napautettaessa. Jos käyttäjä on valinnut ”Follow my current location on the map”-option, neulaa siirretään mobiililaitteen sijainnin mukaisesti. Kartan kamera myös animoidaan siirtymään niin, että neula sijoittuu kartan keskiosaan.

Neula tallennetaan luokan jäsenmuuttujaan, jota lisäyksen jälkeen vain siirretään. Ensimmäisellä kerralla neula siis luodaan GPS-signaalien mukaiseen positioon, mutta aina sen jälkeen sitä vain siirretään.

UD851-EXERCISES-STUDENT MOOC-HARJOITUKSET

Valitsin tehtäväkseni Udacityn Android-kurssin, johon oli linkki materiaaleissa. Rekisteröidyin palveluun, latasin harjoitukset ja aloitin niiden tekemisen.

Tähän pakettiin on koottuna kaikki harjoituspuolen tehtävät. Sunshine-applikaation kokonaisuus on jaettu erikseen.

1.43 Lesson01-Favorite-Toys

1.43.1 T01.01-Exercise-CreateLayout (1h)

Avasin tämän ensimmäisen harjoitustehtävän projektin Android Studiossa (import). Applikaation kääntäminen ei aluksi onnistunut, koska Studio herjasi puuttuvista komponenteista.

Yksi näistä komponenteista oli ilmeisesti vanhempi koodien kääntöön vaikuttava ”gradle build”-komponentti. Toinen oli ”Install ConstraintLayout for Android 1.0.0-beta4 (revision: 1)”. Studio auttoi ongelmien korjaamisessa aika hyvin, koska virheilmoitukset tulivat alaosan ”Messages”-paneeliin. Viesteissä oli linkit, joiden kautta Studio asensi puuttuvat komponentit.

Asennusten jälkeen Studio kehotti päivittämään tietyt komponentit uudempiin versioihin, mikä viittaisi siihen, että virheet korjattiin asentamalla vanhemmat komponentit uusien tilalle. En kuitenkaan enää päivittänyt komponentteja.

Simppeli tehtävä. Muutettiin XML-layoutista elementti, poistettiin muutamia attribuutteja, sekä poistettiin turha ConstraitLayout-riippuvuus gradlesta (eli ei tule koodin käännökseen mukaan turhaan).

1.43.2 T01.02-Exercise-DisplayToyList (0.5h)

Tällä kertaa annettiin TextView-elementille id, sekä se haettiin Java-puolella TextView-olioon. Sitten haettiin lista lelujen nimiä staattisella metodilla. Nimet listattiin tähän tekstikenttään. Simppeli ja nopea harjoitus.

1.43.3 T01.03-Exercise-AddScrolling (0.5h)

Yksinkertainen ScrollView-elementin lisäys TextView-elementin ympärille. ScrollView-elementin leveyksiä ja korkeutta muokattiin myös samalla. Todella yksinkertainen harjoitus.

1.44 Lesson02-GitHub-Repo-Search

1.44.1 T02.01-Exercise-CreateLayout (1h)

Harjoitus aloitettiin taas muokkaamalla ”ConstrainLayout”-elementti ”LinearLayout”-elementiksi. Poistin taas turhaksi jääneen viitauksen gradlen tiedostosta. ”LinearLayout”-elementti asetettiin vertikaaliin orientaatioon, eli sen lapset piirretään päällekkäin.

Hakuja varten luotiin oma ”EditText”, johon asetettiin vihje syöttää hakusana. Tämän alle lisättiin ”TextView”, johon haettu URL-osoite kirjataan. Aivan alas luotiin oma ”ScrollView”-elementti, jonka sisälle haun tulokset listataan. Elementit haettiin koodissa tuttuun tapaan id-kenttien perusteella ja talletettiin jäsenmuuttujiin.

1.44.2 T02.02-Exercise-AddMenu (1h)

Harjoitus aloitetaan lisäämällä otsikkoteksti ”strings.xml”-resurssitiedostoon. Tämän jälkeen resursseihin lisätään uusi kansio ”menu”-elementeille. Tänne luodaan yksi ”main.xml”-tiedosto, johon määritellään käytettävä ”menu”-elementti. Menuun lisätään vain yksi ”item”-elementti (nappi), jonka otsikkona käytetään aiemmin ”strings.xml”-tiedostoon lisättyä tekstiä.

MainActivity-luokassa ylikirjoitetaan ”onCreateOptionsMenu”-metodi, jonka sisällä aiemmin lisätty ”menu”-elementti laajennetaan ja asetetaan käytettäväksi menuksi. Laajennus tehdään ”getMenuInflater()”-metodin kautta.

Lopuksi ylikirjoitetaan ”onOptionsItemSelected”-metodi, joka ajetaan, kun käyttäjä valitsee menusta jonkin itemin. Tässä painallus näyttää ”Toast”-ilmoituksen, jossa lukee ”Search clicked”. Painettu itemi selvitetään sen id:n kautta, käyttämällä ”getItemId”-metodia.

1.44.3 T02.03-Exercise-DisplayUrl (1h)

Harjoituksessa käytetään apuna Androidin omaa ”URI.builder”-luokkaa, jonka avulla voidaan käsitellä erilaisia URI:a. Esimerkiksi ”URL”-verkkosoitteet ovat pohjimmiltaan ”URI”-osoitteita. Tämä kirjasto tuottaa kuitenkin ”Android”-tyyppisen ”URL”-osoitteen. Se on muunnettava verkkomuotoon, käyttämällä Javan ”URL”-luokkaa, ja antamalla sille parametriksi tämä luotu ”Android”-tyyppinen ”URL”-osoite ”String”-tyyppisenä.

Harjoituksessa oli valmiina mm. staattiset ”String”-oliot, joissa oli listattuna Github-palvelun perusosoite, sekä mm. merkit, joita käytetään tehdessä sivustolle erilaisia hakuja.

Aiemmassa harjoituksessa toteutettu ”Toast”-viesti korvattiin nyt näyttämään rakennettu Github-haku, kun käyttäjä painaa ”Search”-itemiä menusta.

1.44.4 T02.04-Exercise-ConnectingToTheInternet (1h)

Harjoitus aloitetaan lisäämässä ”permission”- eli lupa internet-yhteyden käyttämiseen.

Aiempaa ”makeGithubSearchQuery”-metodia laajennetaan nyt kutsumaan ”NetworkUtils.getResponseFromHttpUrl”-metodia. Tämä metodi saa parametrikseen aiemman harjoituksen lopputuloksena saadun ”url”-verkkoosoitteen. Metodi tekee haun verkosta ja koittaa palauttaa löytetyt Github-palvelun tulokset ”String”-muuttujana. Tämä tulos asetetaan näkymän ”TextView”-elementtiin.

1.44.5 T02.05-Exercise-CreateAsyncTask (1.5h)

Tässä tehtävässä luodaan oma ”AsyncTask”-luokka, jonka sisälle aiemman tehtävän verkkohaku siirretään. Aiemmin verkkohaku ajettiin ”main”-säikeessä, joka on todella huono tapa. Tällöin yli 5s kestävä haku jäädyttää käyttöliittymän, ja Android ilmoittaa ohjelman pysähtyneen. Tämän vuoksi verkkohaut olisi aina tehtävä erillisessä ”thread”-, eli säikeessä.

”doInBackground”-metodille annetaan parametriksi aiemmin selvitetty ”URL”-olio, ja verkkohaun koodit siirretään sen sisälle. Tämän metodin sisältö ajetaan asynkroonisesti.

”onPostExecute”-metodi ajetaan heti, kun ”doInBackground”-metodista poistutaan. Tämä metodi ajetaan ”main”-säikeessä, ja se toimii synkroonisesti muun ohjelman kanssa. Metodi saa nyt parametriksi ”String”-muotoisen olion, joka sisältää verkkohaun tuloksen tekstimuodossa. Se asetetaan näkymän ”TextView”-elementtiin tekstiksi.

Lopuksi ”makeGithubSearchQuery”-metodi muutetaan luomaan jokaista hakua varten uusi ”AsyncTask”-olio (luokka, joka luotiin juuri aiemmin). Nyt haut hoidetaan asynkronisesti taustalla, eikä käyttöliittymä enää jäädy.

1.44.6 T02.06-Exercise-AddPolish (2h)

Tämän harjoituksen tarkoituksena on viimeistellä aiemmin tehdyt muutokset. Meidän on mm. käsiteltävä mahdolliset virhetilanteet. Yksi virhetilanne voi tulla esimerkiksi aikakatkaisusta, jos käyttäjällä on hidas internet-yhteys. Toinen virhe voisi olla tilanne, jossa käytetty verkkopalvelin ei vastaa, tai se on pois käytöstä.

Aiempiin harjoituksiin verrattuna, tässä oli melko paljon tehtäviä muutoksia.

Ajatus on luoda käyttöliittymälle piilotettu ”TextView”-elementti, johon tapahtunut virhe kirjoitetaan. Virheen sattuessa piilotettu ”TextView”-elementti näytetään, kun taas tuloksen näyttävä ”TextView”-elementti vastavuoroisesti piilotetaan. Virheviesti lisätään normaalisti ”strings.xml”-resurssitiedostoon.

Näkymään lisättiin myös ”ProgressBar”-elementti, joka valottaa verkkohaun tilannetta. Sen näkyvyyttä ja toimintaa ohjattiin ”AsyncTask”-luokan sisältä (esim. ”onPreExecute”).

1.45 Lesson03-Green-Recycler-View

1.45.1 T03.01-Exercise-RecyclerViewLayout (0.5h)

Tässä harjoituksessa aloitetaan ”RecyclerView”-elementin käyttäminen. Työ aloitetaan lisäämällä ”build.gradle”-tiedostoon riippuvuus ”recyclerview”-kirjastolle, käyttämällä ”compile”-komentoa ”dependencies”-listauksen sisällä.

”RecyclerView”-elementti muokataan ”activity_main.xml”-näkymässä olevan “TextView”-elementin tilalle. Elementti asetetaan täyttämään koko laitteen ruutu ”match_parent”-arvoilla. ”id”-attribuutti on asetettava harjoituksen vaatimusten mukaan.

1.45.2 T03.02-Exercise-ViewHolder (3.5h)

Tässä harjoituksessa luodaan ulkoasutiedosto (number_list_item.xml), joka esittää yhtä itemiä rullattavassa listassa. Ulkoasu pohjautuu ”FrameLayout”-elementtiin, jonka sisälle lisätään yksi ”TextView”-elementti näytettävää numeroa varten.

Harjoituksesta löytyy ”GreenAdapter”-luokan pohja, joka periytyy ”RecyclerView.Adapter<GreenAdapter.NumberViewHolder>”-luokasta. Luokan sisälle luodaan uusi sisäinen, joka periytyy ”RecyclerView.ViewHolder”-luokasta. Luokan jäsenmuuttujaksi lisätään yksi ”TextView”-elementti, joka on näytettävän ”number_list_item”-ulkoasun tekstiä varten. Luokan konstruktori saa parametrikseen ”View”-olion, joka on annettava eteenpäin luokalle josta on periydytty (super class) ”super”-kutsulla. Tämän jälkeen haetaan normaaliin tapaan käytettävät näkymät jäsenmuuttujiin (tässä ainoa”TextView”-elementti). Tämä luokka toimii välikätenä, tallentaen viittaukset luotuihin ”View”-näkymiin, joita käytetään uudelleen. Emme siis aja ”findViewByID”-metodia satoja kertoja, vaan ajatus on uudelleen käyttää listauksessa jo näkyvistä poistunutta näkymää. Kun näkymä näytetään uudelleen, sen tiedot vain ylikirjoitetaan (tässä ”TextView”-elementin sisältö). Tietojen ylikirjoitusta varten luodaan erillinen metodi ”bind”, joka saa parametrikseen päivitettävät tiedot.

Mielestäni tämä harjoitus oli yksi haastavammista sisäistää.

1.45.3 T03.03-Exercise-RecyclerViewAdapter (2h)

Tässä harjoituksessa viimeistellään ”GreenAdapter”-luokan rakenne, jotta sitä voitaisiin käyttää aiemmin luotujen komponenttien kanssa.

Luokka merkataan aluksi periytyvän ” RecyclerView.Adapter<GreenAdapter.NumberViewHolder>”-luokasta. Tämä geneerinen luokka saa parametrinaan aiemmin luodun ”GreenAdapter.NumberViewHolder”-luokan.

Adapteri vaatii ylikirjoitetuksi kolme metodia. Näistä ensimmäinen on ”onCreateViewHolder”, jonka on palautettava tässä tapauksessa ” NumberViewHolder”-tyyppinen olio, sekä laajennettava (inflate) aiemmista tehtävistä tuttu ”number_list_item”-ulkoasu.

Toinen ylikirjoitettava metodi on ”onBindViewHolder”. Se saa parametrina ”NumberViewHolder”-tyyppisen olion, jonka luokan loimme adapteri-luokan sisälle. Loimme sille ”bind”-metodin. Ohjaamme adapterin siis kutsumaan suoraan ”NumberViewHolder”-olion metodia, kun on aika päivittää itemin tiedot. Ylikirjoitettavana tietona käytetään parametrina saatavaa ”position”- eli itemin sijaintinumeroa listalla.

Viimeinen ylikirjoitettava metodi on ”getItemCount”- joka palauttaa kokonaislukuna adapterilla olevien itemien kokonaismäärän.

1.45.4 T03.04-Exercise-WiringUpRecyclerView (1h)

Tässä tehtävässä on tarkoitus nitoa yhteen aiemmin luodut komponentit (GreenAdapter, RecyclerView).

Harjoitus aloitetaan hakemalla ulkoasusta aiemmin lisäämämme ”rv_numbers”-tunnisteella merkattu ”RecyclerView”-näkymä. Tämän jälkeen sille luodaan uusi ”LinearLayoutManager”, joka hoitaa ladattavien näkymien asettelun. Tämän jälkeen luodaan uusi instanssi luomastamme ”GreenAdapter”-luokasta, joka asetetaan haetun ”RecyclerView”-näkymän adapteriksi.

1.45.5 T03.07-Exercise-RecyclerViewClickHandling (2h)

Tässä harjoituksessa listan toimintaan lisättiin tuki itemin klikkaukselle. Kun käyttäjä painaa jotakin listan itemiä, näytetään käyttöliittymällä ”Toast”-viesti.

Homma aloitetaan luomalla ”GreenAdapter”-luokan sisälle ”ListItemClickListener”-rajapinta (interface). Tällä rajapinnalla on vain yksi metodi ”onListItemClick”, joka saa parametrikseen klikatun listaitemin indeksin. Ajatus on, että kun käyttäjä klikkaa listalla olevaa itemiä, tämä metodi ajetaan.

”GreenAdapter”-luokan konstruktoria muokataan siten, että sille annetaan parametriksi ”ListItemClickListener”-rajapinta. Adapterin käyttäjä voi siis ulkopuolelta helposti asettaa, mitä tehdään, kun itemiä klikataan.

Aiemmin luotu ”NumberViewHolder”-luokka asetetaan implementoimaan ”View.OnClickListener”-rajapinta, jonka kautta sille on määriteltävä ”onClick”-metodi, joka ajetaan, kun itemiä klikataan. Tämän metodin loppuun lisätään pala koodia, joka ajaa ”GreenBack”-luokan jäsenmuuttujaan tallennetun ”ListItemClickListener”-rajapinnan ”onListItemClick”-metodin. Ohjaamme siis itemin klikkauksen takaisin kehittäjän konstruktorissa antamaan rajapintaan (kätevää, eikö?).

Lopullinen ”ListItemClickListener”-rajapinta implementoidaan ”MainActivity”-luokassa. Voimme tämän jälkeen antaa ”MainActivity”-luokan instanssin toisena ja uutena parametrina ”GreenBack”-luokan konstruktorille. Lopullinen klikkauksen jälkeinen logiikka on siis määritelty ”MainActivity”-luokan sisällä, jonne se myös loogisesti sopii.

1.46 Lesson04a-Starting-New-Activities

1.46.1 T04a.01-Exercise-AddNewActivity (0.5h)

Tässä tehtävässä harjoitellaan uusien ”Activity”-elementtien lisäystä projektiin. Tehtävässä luodaan yksi ”activity_child.xml”-niminen ”Activity”, johon lisätään yksi ”TextView”-elementti. Tämä ainoa lapsielementti haetaan jäsenmuuttujaan normaaliin tapaan ”ChildActivity”-luokan ”onCreate”-metodissa. Erittäin simppeli tehtävä.

1.46.2 T04a.02-Exercise-StartNewActivity (1h)

Tämä tehtävä oli ”Intent”-olioiden opiskelua varten. Tarkoitus on nyt yhdistää edellisen tehtävän ”ChildActivity”- sekä ”MainActivity”-elementit toisiinsa. Painamalla nappulaa ”MainActivity”-näkymässä, halutaan avata ”ChildActivity”. Tämä korvaa aiemman toiminnan, jossa näytettiin ”Toast”-viesti.

Siirtymistä varten luodaan uusi ”Intent”-olio, joka saa parametrikis ”MainActivity”-luokan instanssin (periytyy ”Context”-luokasta) sekä ”ChildActivity”-luokan. Siirtyminen aloitetaan ”startActivity”-metodilla, joka saa tämän ”Intent”-olion parametrina.

1.46.3 T04a.03-Exercise-PassingDataBetweenActivities (1.5h)

Tässä edellistä harjoitusta jatketaan, lisäämällä jo luodun ”Intent”-olion mukaan dataa. Se lisätään kutsumalla ”Intent”-olion ”putExtra”-metodia. Tässä esimerkissä haluttiin lisätä vain tekstiä, joten metodille annettiin parametriksi ”Intent.EXTRA_TEXT” sekä toiseksi parametriksi varsinainen lähetettävä teksti.

Data on luettava erikseen sen vastaanottavassa ”ChildActivity”-luokassa. Luku lisätään ylikirjoitettuun ”onCreate”-metodiin, käyttämällä ”getIntent”-metodia. Tämän jälkeen tarkastetaan, lähetettiinkö ”Intent”-olion mukana parametri, käyttämällä ”hasExtra”-metodia. Jos metodi palauttaa ”true” (boolean), parsitaan teksti ”getStringExtra”-metodin avulla.

1.47 Lesson04b-Webpages-Maps-and-Sharing

1.47.1 T04b.01-Exercise-OpenWebpage (3h)

Tässä tehtävässä harjoitellaan ”Implicit Intent”-olioiden käyttöä. Esimerkiksi puhelinnumeroon soitettaessa, meitä ei välttämättä kiinnosta, mitä applikaatiota käyttäjä sen toteuttamiseen käyttää. Toinen esimerkki voisi olla vaikka selain. Käyttäjä voi siis itse valita, mitä applikaatiota hän haluaa käyttää sisällön näyttämiseen. Esimerkiksi sähköpostin lähettämiseen, käyttäjällä voi olla montakin eri applikaatiota.

Tehtävä aloitetaan luomalla nappiin logiikka, jonka tarkoitus on avata verkkoselaimeen sivu. Tällä kertaa ”Intent”-olio luodaan käyttämällä ”Intent.ACTION_VIEW”-parametria sekä ”Uri”-oliota. Tämän jälkeen kutsutaan luodun ”Intent”-olion ”resolveActivity”-metodia, joka pyrkii selvittämään applikaation, joka olisi kykenevä avaamaan parametrina annetun ”Uri”-olion. Tällainen voisi olla esimerkiksi asennettu ”Chrome”-selain. Jos yhtäkään applikaatiota ei löytynyt, palauttaa metodi ”null”. Muussa tapauksessa voidaan ajaa ”startActivity”-metodi, ja antaa sille tämä ”Intent”-olio parametriksi. Tämä käynnistää ”resolveActivity”-metodin löytämän applikaation ja antaa sille ”Uri”-olion parametriksi.

Loppujen lopuksi homma on aika simppeli, vaikka taustalla tapahtuva logiikka on melko monimutkainen.

1.47.2 T04b.02-Exercise-OpenMap (2.5h)

Tässä tehtävässä harjoitellaan verkkoselaimen sijaan karttojen avaamista. Aiemman harjoituksen mukaisesti, meidän on luotava luotava ”Uri”-olio, joka annetaan ”Intent”-olille parametriksi. ”Uri”-olion sisältö luodaan Android-ohjeiden mukaisesti, jonka ohjeet löytyvät ”Maps”-kohdasta täältä:

https://developer.android.com/guide/components/intents-common

”Uri”-oliota ei luoda ”manuaalisesti”, vaan apuna käytetään ”Uri.Builder”-oliota. Kartoissa ”Uri”-olion ”scheme”-arvoksi asetetaan ”geo”. Tässä harjoituksessa paikka haetaan nimen perusteella, joten ”latitude”- ja ”longitude”-parametreiksi asetetaan nolliksi (0,0). Paikan nimi asetetaan ”String”-tyyppisenä muuttujana ”Uri.Builder”-olion ”query”-metodilla.

Tämän jälkeen luomme ”Intent”-olion samaan tapaan kuin aiemmin, mutta nyt parametrina on kartan paikkaa pyytävä ”Uri”-olio. Meidän on jälleen tarkastettava, että laitteeseen on asennettu karttoja tukeva applikaatio, ”resolveActivity”-metodilla.

1.47.3 T04b.03-Exercise-ShareText (1.5h)

Tämän kategorian viimeisessä harjoituksessa luodaan ”jaa”-toiminto tekstipohjaiselle sisällölle.

Projektiin luodaan uusi ”shareText”-metodi, joka saa parametrina jaettavan tekstin. Muita tarvittavia parametreja ovat ”mimeType” sekä valintaikkunan otsikkoteksti. ”mimeType”-parametrin avulla filteröidään valittavaksi vain sellaiset ohjelmat, jotka osaavat avata sen tyyppistä dataa. Tekstipohjainen sisältö merkataan ”text/plain”-tyyppiseksi.

Lopulta käytetään ”ShareCombat.IntentBuilder”-luokaa ”Intent”-olion luontiin ja ajamiseen seuraavasti:

```
ShareCompat.IntentBuilder.from(this)
.setType("text/plain")
.setChooserTitle(“Otsikko valintaikkunaan” )
.setText(“Jaettava teksti”)
.startChooser();
```

Ohjelma 8. *Tekstipohjaisen sisällön jakaminen Androidissa*

1.48 Lesson05a-Android-Lifecycle

1.48.1 T05a.01-Exercise-LogLifecycle (0.5h)

Tässä harjoituksessa aloitellaan Android-applikaatioiden elinkiertoa. Elinkierrolla tarkoitetaan tapaa, jossa Android-järjestelmä voi sulkea ohjelmat milloin tahansa. Tämän vuoksi ohjelmat on suunniteltava hieman eri tavalla kuin esim. perinteiset työpöytäohjelmat.

Harjoituksessa lisätään log-viestit jokaiseen elinkierron metodiin (esim. ”onCreate”, ”onStart”, ”onStop”). Tehtävän pohjalta pääteltiin, missä järjestyksessä kaikki elinvaiheet ajetaan.

Erittäin yksinkertainen, mutta valaiseva tehtävä.

1.48.2 T05a.02-Exercise-PersistData (2.5h)

Tässä harjoituksessa ylikirjoitetaan ”onSaveInstanceState”-metodi, jossa sen parametrin ”Bundle”-olioon lisätään käyttöliittymässä olevan ”TextView”-elementin teksti. Sama

teksti koitetaan lukea ylikirjoitetussa ”onCreate”-metodissa, joka saa mahdolliseksi parametrikseen ”Bundle”-olion. Teksti saadaan luettua mm. silloin, kun laite käännetään. Kuten ”Intent”-olioiden kanssa, parametri saadaan ”Bundle”-oliosta samalla ”String”-tyyppisellä avaimella, kuin se on siihen lisättykin.

Harjoituksessa siis valaistaan sitä, miten voi itse hoitaa käännön jälkeisen tilan palautuksen (esim. käyttäjän syöttämät tekstit kirjoitetaan takaisin kenttiin jne.). Hyvä tehtävä.

1.48.3 T05a.03-Exercise-FixLifecycleDisplayBug (2h)

Tässä jatketaan ”Lifecycle”-, eli applikaation elinkierron, harjoittelemista. Eri elinvaihdeiden ylikirjoitetut metodit kirjoittava ajaessaan oman otsikkonsa staattiseen listaan. Tämä lista tulee siis lopulta sisältämään kaikki elinvaiheet, jotka on ajettu.

Kun ohjelma pistetään taustalle, painamalla kotinäppäintä, huomataan, että harvinaisempi ”onRestart”-elinvaihe ajetaan. Tällöin näkymää ei ole siis luotu ”onCreate” asti, vaan vanha näkymä oli laitteen muistissa valmiina ajoon.

1.49 Lesson05b-Smarter-GitHub-Repo-Search

1.49.1 T05b.01-Exercise-SaveResults (2h)

Tässä pähkäillään, miten ”EditText”-elementin tieto tallennetaan ”Android”-käyttöjärjestelmän tasolla, mutta ”TextView”-elementin ei. Tämä liittyy siihen, että käyttäjä syöttää tiedot ”EditText”-elementtiin, muttei ”TextView”-elementtiin.

Harjoituksessa halutaan tallentaa ”TextView”-elementin teksti samaan tapaan, kuin ”Android”-järjestelmä tallentaa ”EditText”-elementin tekstin.

Kuten aiemmissa harjoituksissa, syötetyt tiedot tallennetaan ”Bundle”-olioon, joka täytetään ylikirjoitetussa ”onSaveInstanceState”-metodissa. Olion sisältämien tietojen avaimet lisätään ”constant”-tyyppisiksi, staattisiksi muuttujiksi.

Kuten aiemminkin, mahdolliset tallennetut tiedot luetaan ylikirjoitetun ”onCreate”-metodin sisällä. Se saa parametrikseen mahdollisesti asetetun ”savedInstanceState”-olion, joka siis voi sisältää aiemmin ”onSaveInstanceState”-metodissa lisätyt asiat (tässä harjoituksessa mm. ”TextView”-elementin tekstin).

1.49.2 T05b.02-Exercise-AddAsyncTaskLoader (3h)

Tässä korvataan aiemmin käytetty ”AsyncTask”-olio uudemmalla, ”AsyncTaskLoader”-oliolla. Tämän olion elinkaari eroaa ”Activity”-olion elinkaaresta, ja se pysyy elossa kauemmin. Mm. laitteen kääntäminen ei pysäytä sen kautta aloitettua verkkosivun hakua.

Tämä tehtävä oli erittäin työläs. ”MainActivity”-luokka muokattiin implementoimaan ” LoaderManager.LoaderCallbacks<String>”-rajapinta joka vaati mm. ” onCreateLoader”, ”loadInBackground”, ” onLoadFinished” sekä ” onLoaderReset”-metodien lisäystä. ”LoaderManager”-olio käyttää tuttua ”Bundle”-oliota alustamiseensa. Sille annetaan nyt aiemmin ”AsyncTask”-oliossa tehdyn sivulatauksen ”URL”-teksti. Parametrit ”parsitaan” ulos ylikirjoitetussa ”loadInBackground”-metodissa. Samassa paikassa suoritetaan myös varsinainen lataus, käyttämällä ”NetworkUtils.getResponseFromHttpUrl”-metodia.

Minulla ei ollut aiemmin tietoa näistä ”Loader”-asioista. Tulen luultavasti käyttämään niitä omissa applikaatioissani jatkossa, kun lataan verkon kautta asioita. Niitä voi luultavasti käyttää myös moneen muuhunkin asiaan. Varsinainen etu on juuri erilainen elinkaari verrattuna ”Activity”-komponentteihin.

1.49.3 T05b.03-Exercise-PolishAsyncTask (1.5h)

”Loader”-komponentti käsittelee automaattisesti orientaatiomuutokset, mutta haluamme, että se ei aloita hakuaan uudelleen, jos käyttäjä poistuu näkymästä. Haluamme, että jo aloitettu tiedonhaku jatkuu niin, että tulos tallennetaan paikalliseen muuttujaan talteen (”cache”, eli suomalaisittain ”välimuisti”). Tämän jälkeen, ohjelma aina tarkastaa ensin välimuistin, ennen uuden sivulatauksen tekemistä.

Välimuistin tarkastus lisätään ylikirjoitetun ”onStartLoading”-metodiin logiikkaan. Samaan paikkaan lisätään kutsu ”deliverResult”-metodilla, jonka kautta saatu tulos palautetaan. Tämän metodin sisällä hoidetaan myös tiedot tallennus ”välimuisti”-muuttujaan.

Tämäkin harjoitus on erittäin hyvä lisäys normaalin ”Loader”-komponentin käytössä. Muutama peukku tälle!

1.50 Lesson06-Visualizer-Preferences

1.50.1 T06.01-Exercise-SetupTheActivity (2h)

Tässä harjoitustehtävässä muokataan uutta ”Visualizer”-applikaatita. Editointi aloitetaan lisäämällä siihen oma ”Settings”-näkymä, joka avataan lisätyn menun napista. Nappi ja menu lisätään aivan samaan tapaan kuin aiemmissakin harjoituksissa.

“Settings”-näkymä lisätään myös normaaliin tapaan, käyttämällä ”Empty”, eli tyhjää, näkymäpohjaa. ”AndroidManifest.xml”-tiedostossa asetetaan ”VisualizerActivity”-näkymän ”launchMode”-attribuutti asetetaan ”singleTop”-tilaan, jolloin näkymää ei luoda uudelleen, kun siihen palataan uudesta ”Settings”-näkymästä. Vastaavasti ”SettingsActivity”-näkymälle lisätään ”parentActivityName”-attribuutti, joka asetetaan viittaamaan ”VisualizerActivity”-näkymään.

”Settings”-näkymän ylikirjoitetussa ”onCreate”-metodissa muokataan ”ActionBar”-elementissä näytettävä kotinäppäin. Kotinäppäin muutetaan näyttämään nuolelta, ajamalla ”ActionBar”-olion metodi ”setDisplayHomeAsUpEnabled”.

1.50.2 T06.02-Exercise-MakeAPreferenceFragment (3h)

Aivan aluksi luotiin ”SettingsFragment”-näkymä, joka asetetaan periytyväksi ”PreferenceFragmentCompat”-luokasta. Tämä vaatii ”onCreatePreferences”-metodin toteuttamista. Tämän metodin sisällä on tarkoitus ladata ”XML”-tiedosto, jossa määritellään laitteen kovalevylle tallennettavat asetukset (”Preference”). Tiedosto ladataan kutsumalla metodia ”addPreferencesFromResource”, johon annetaan parametriksi ”XML”-tiedoston resurssi (esim. muotoa ”R.xml.pref_visualizer”).

”Preference”-arvot listaava ”pref_visualizer.xml”-tiedosto luodaan projektin ”xml”-kansioon. Listauksessa määritetään mm. oletusarvo ja tiedon tyyppi (esim. ”CheckBoxPreference”-elementti on ”boolean”-tyyppisiä tietoja varten).

Koska tehtävässä käytetään ”Preference Fragment Compact Library”-riippuvuutta, meidän on myös lisättävä ”styles.xml”-tiedostoon tyylimääritelmä. Ilman tätä määritystä applikaatio kaatuu, kun ”Settings”-näkymää koitetaan avata.

1.50.3 T06.03-Exercise-ReadingFromSharedPreferences (1h)

Tässä harjoituksessa luetaan laitteen kovalevylle tallennettavia ”Preference”-tietoja. ”Preference”-tiedostolle ei annetta erillistä nimeä, joten apuna käytetään ”oletustiedostoa”. Oletustiedosto saadaan kutsumalla metodia ”PreferenceManager.getDefaultSharedPreferences”. Tämän jälkeen tietoja voidaan lukea normaaliin tapaan, käyttämällä tietojen avainta ja tietotyyppiä. Harjoituksessa luetaan esim. ”show_bass”-avaimella merkatun tiedon ”boolean”-arvo kutsumalla ”SharefPrefrence”-olion ”getBoolean(”show_bass”, true)”-metodia. Jälkimmäinen parametri on tiedon oletusarvo, jota käytetään, jos arvoa ei ole vielä koskaan kirjoitettu.

1.50.4 T06.04-Exercise-UseResources (1h)

Tässä jatkokehitettiin edellisen tehtävän toteutusta, siirtämällä ”Preference”-tietojen ”String”-pohjaiset avaimet sekä oletusarvot omiin ”xml”-resurssitiedostoihinsa.

Avaimet lisättiin normaaliin tapaan ”strings.xml”-resurssitiedostoon. Erona normaaliin tapaan, lisätyt tekstit merkattiin ”translatable=’false’”-attribuutilla. Tällä kerrotaan ohjelman tekstien kääntäjälle, että näitä tekstejä ei tarvitse kääntää, jolla säästetään turhaa työtä.

”boolean”-oletusarvo tallennetaan uuteen ”bools.xml”-tiedostoon, joka tulee sisältämään siis pelkkiä ”boolean”-arvoja.

Käyttämällä tietoja näiden resurssitiedostojen kautta, mahdolliset kirjoitusvirheet koodissa tulevat myös helpommin ilmi. Mahdolliset arvojen muutokset voidaan korjata myös yhteen paikkaan.

1.50.5 T06.05-Exercise-PreferenceChangeListener (2h)

Tässä lisättiin aiemmin luotuun ”Preference”-logiikkaan kuuntelija, joka ajetaan aina, kun jokin ”Preference”-tiedoista muuttuu.

”OnSharedPreferenceChangeListener”-rajapinta toteutetaan ”VisualizerActivity”-luokassa, jossa se vaatii ”onSharedPreferenceChanged”-metodin lisäystä. Tämän metodin sisälle lisätään logiikka, joka lukee päivittyneen tietueen arvon, ja ajaa sen mukaan oman logiikkansa. Tässä harjoituksessa asetetaan bassotaajuksia esittävän viivan näkyvyys ”setShowBass”-metodilla. Sille annetaan tässä kohtaa parametriksi muuttuneen tietueen uusi arvo.

Kuuntelija asetetaan ylikirjoitetussa ”onCreate”-metodissa. Ohjelman toiminnan kannalta on myös tärkeää, että kuuntelija poistetaan ylikirjoitetussa ”onDestroy”-metodissa.

1.50.6 T06.06-Exercise-AddTwoMoreCheckboxes (0.5h)

Tässä lisättiin ainoastaan kaksi muuta ”boolean”-tyyppistä ”Preference”-tietoa verrattuna aiempaan tehtävään.

1.50.7 T06.07-Exercise-ListPreference (3h)

Tässä harjoituksessa luodaan uusi ”list preference”-lista, joka lisätään aiemmin luotuun ”Settings”-näkymään. Lista itsessään esittää yhtä laitteeseen tallennettavaa tietoa, mutta sen mahdolliset arvot on listattu. Lisätty lista näkyy siis ”Settings”-näkymässä nappina, jota painamalla avautuu uusi valikko, joka listaa kaikki käytettävät arvot. Valitsemalla jonkin listan arvoista, arvo tallennetaan laitteen muistiin (”SharedPreferences”).

Tässä harjoituksessa luodaan uusi ”list preference” värin valitsemiseen. Listalle lisätään kolme eri väriä: punainen, sininen ja vihreä.

Varsinainen ”list preference”-valinta lisätään aiemmin tuttuun ”pref_visualizer.xml”-tiedostoon. Kaikki tekstit listataan tuttuun tapaan ”strings.xml”-tiedostoon resursseina (mm. ”title”- ja ”key”-attribuuttien arvot). ”list preference”-valinnan mahdollisia arvoja varten luodaan uusi ”arrays.xml”-tiedosto. Tämä tiedoston sisälle määritellään kaksi taulukkoa:

yksi taulukko esittämään valittavien arvojen otsikot, sekä toinen määrittelemään varsinaiset arvot. Kaikki nämä tiedot ovat kuitenkin ”String”-tyyppisiä, joten ne on määriteltävä ”strings.xml”-tiedostossa.

Lopulta päänäkymän ”onCreate”-metodiin on vielä lisättävä logiikka, joka asettaa valitun värin ”SharedPreference”-komponentista.

1.50.8 T06.08-Exercise-PreferenceSummary (1.5h)

Aiemmassa ”list preference”-toteutuksessa jää epäselväksi, mikä valinta oli valittuna. ”Settings”-näkymässä näkyy ainoastaan nappi ”Shape Color”. Meidän on implementoitava näkymään ”preference summary”-komponentti, joka esittää valitun arvon tekstimuotoisena otsikkona. Teksti kirjoitetaan valinnan otsikon alle. Aiemmissa ”Show bass”- ja ”Show Mid Range”-asetuksissa se on implementoitu jo valmiina (esim. ”Show bass” alla lukee ”Shown”, jos valinta on päällä), mutta ”list preference”-tyyppisen valinnan kohdalla, se on implementoitava itse.

Tehtävän toteutuksessa käytetään jälleen apuna aiemmissa tehtävissä tutuksi tullutta ”OnSharedPreferenceChangeListener”-rajapintaa, jonka avulla valinnan ”preference summary”-päivitetään. Kuten aiemminkin, kuuntelija on asetettava ”onCreate”-metodissa, mutta myös poistettava ”onDestroy”-metodissa.

Ylikirjoitettuun ”onCreatePreferences”-metodiin lisätään logiikka, joka käy läpi kaikki listatut asetukset (”Preference”). Luokkaan lisätään uusi metodi ”setPreferenceSummary”. Idea on siis käydä läpi kaikki asetukset, sekä ”ListPreference”-olion kohdalla, ylikirjoittaa ”preference summary”-kentän arvo vastaamaan valittua väriä.

1.50.9 T06.09-Exercise-EditTextPreference (2h)

Tässä harjoitellaan ”EditText”-tyyppisen asetuksen lisäämistä ”Settings”-näkymään. Kentällä toteutetaan asetus, jolla käyttäjä voi muokata ulkoasussa olevien palikoiden kokoa arvovälillä 1-3.

Tekstikenttä lisätään samoin kuin aiempi ”ListPreference”-elementti ”pref_visualizer.xml”-tiedostoon. Se lisätään käyttämällä elementtiä ”EditTextPreference”. Sille annetaan attribuuteiksi oletusarvo, avain sekä otsikkoteksti. Tekstit lisätään normaaliin tapaan ”strings.xml”-tiedostoon.

”VisualizerActivity”-luokan koodiin lisätään uusi metodi, joka lataa parametrina annetusta ”SharedPreferences”-oliosta tallennetun koon. Koko tallennetaan tekstipohjaisena, mutta tämä metodi muuttaa arvon ”float”-tyyppiseksi. Lopulta metodi kutsuu koon asettavaa, toista metodia.

Kuten aiemmassa harjoituksessa, meidän on taas lisättävä oma ”if”-haara ”setPreferenceSummary”-metodin sisälle. Jos parametrina syötetty ”Preference”-olio on tyypiltään ”EditTextPreference”, asetetaan näytettäväksi ”preference summary”-otsikoksi tallennettu koko (esim. ”1.2”). Tässä harjoituksessa ei vielä lisätä kokotiedon validointia.

1.50.10 T06.10-Exercise-EditTextPreferenceConstraints (2h)

Tässä harjoitellaan tallennettavan ”Preference”-tiedon validointia, ennen kuin se varsinaisesti tallennetaan. Apuna käytetään ”Preference.OnPreferenceChangeListener”-rajapintaa, jonka toteutettava ”onPreferenceChange”-metodi ajetaan, ennen ”Preference”-tiedon tallentamista.

Kuuntelija on ”Preference”-kohtainen, ja tässä harjoituksessa haluamme validoida aiemmin lisätyn ”EditTextPreference”-kentän arvon. Kuuntelija rekisteröidään ajamalla valitun ”Preference”-olion metodi ”setOnPreferenceChangeListener”. ”Preference.OnPreferenceChangeListener”-rajapinta on toteutettu ”SettingsFragment”-luokassa, joten annamme metodille parametriksi ”this”.

”onPreferenceChange”-metodi saa parametrikseen ”Preference”-olion (johon uusi arvo ollaan kirjoittamassa), sekä ”Object”-tyyppisen olion, joka on uusi tallennettava arvo. Tämä ”Object”-tyyppinen olio on muokattava tekstimuotoiseksi, jonka jälkeen se koitetaan muuttaa ”float”-tyyppiseksi. Virheiden kannalta koko touhu ympäröidään ”try-catch”-rakenteella, joka virheen sattuessa näyttää käyttälle ”Toast”-viestin, sekä estää tiedon tallentamisen (palauttamalla ”false”).

1.51 Lesson08-Quiz-Example

1.51.1 T08.01-Exercise-AddTheContentProviderPermission (0.5h)

Tämä on ensimmäinen tehtävä, jossa harjoitellaan ”Content Provider”-komponenttien käyttöä.

Homma aloitetaan lisäämällä lukuoikeudet harjoituksessa käytettävään pakettiin. Oikeudet lisätään normaaliin tapaan ”AndroidManifest.xml”-tiedostoon:

```
 <uses-permission
android:name="com.example.udacity.droidtermsexample.TERMS_READ" />
```

Ohjelma 9. *”Content-Provider”-komponentin sisällön lukuoikeuksien lisäys.*

1.51.2 T08.02-Exercise-AddAsyncTaskToRetrieveCursor (3h)

Tässä tehtävässä tehtiin ensimmäinen asynkrooninen kutsu ulkopuolella olevaan ”DroidTermsExample”-pakettiin. Kutsu toteutettiin perityssä ”AsyncTask”-luokan ”doInBackground”-metodissa.

Kyselyssä käytetään apuna ”ContentResolver”-luokkaa, jonka kautta kaikki kyselyt tehdään erilaisille ”Content Provider”-komponenteille. Tässä harjoituksessa ”Content Provider” (”DroidTermsExampleContract”) palauttaa kyselyn tuloksena ”Cursor”-tyyppisen olion, joka on samanlainen, kuin mitä käytetään esim. ”SQLite”-tietokannan kyselyissä.

Kyselyn ”URI”-saadaan ”DroidTermsExampleContract.CONTENT_URI”-muuttujan kautta. Sille annetaan parametriksi neljä ”null”-arvoa. Saatu ”Cursor”-olio tallennetaan ”MainActivity”-luokan jäsenmuuttujaan.

1.51.3 T08.03-Exercise-FinishQuizExample (2h)

Viime tehtävässä saatiin onnistuneesti ”Cursor”-olio kyselyn tuloksena ”Content Provider”-komponentista. Tässä oli tarkoitus kaivaa tulokset ulos saadusta ”Cursor”-oliosta. Olio sisältää tulokset riveinä, jotka on kaivettava silmukan avulla. Riviä vaihdetaan ”moveToNext()”-metodin kutsulla, joka palauttaa ”true” niin kauan, kuin uusi rivi on saatavilla.

Haluttujen tietojen kolumnien indeksit selvitetään ensin ”Cursor”-olion ”getColumnIndex”-metodilla. Indeksejä käytetään sen jälkeen parametrina ”getString”-metodille, joka palauttaa sen hetkisen rivin tiedot.

Parsitut tiedot näytetään lopulta applikaation käyttöliittymällä.

1.52 Lesson09b-ToDo-List-AAC

1.52.1 T09b.01-Exercise-CreateEntity (2h)

Tämä on ensimmäinen harjoitus, jossa aloitetaan ”room”-kirjaston käyttäminen. Tehvässä merkataan annotaatioilla ”TaskEntry”-luokan osia.

Luokan ”class”-määritelmään lisätään ”@Entity(tableName = "task")”-annotaatio, jolla luokka merkataan entiteetiksi. Parametri ”tableName”-asetetaan tietokantataulun nimeksi tälle luokalle ”task”.

”id”-kenttä merkataan ”@PrimaryKey(autoGenerate = true)”-annotaatiolla, joka siis asettaa jäsenmuuttujan taulun avainkentäksi. Parametri takaa sen, että kentän arvot luodaan automaattisesti.

Koska luokassa on enemmän kuin yksi konstruktori, täytyy muut kuin ”room”-kirjaston käyttämä konstruktori merkata ”@Ignore”-annotaatiolla.

1.52.2 T09b.02-Exercise-SaveTaskInDatabaseFromAddTaskActivity (1h)

Tässä harjoituksessa otetaan käyttöön SQLite-tietokantaa käyttäviä ”Data Access Objects”-olioita. Yksinkertainen mobiiliohjelma antaa käyttäjän kirjoittaa SQLite-pohjaiseen tietokantaan uusia rivejä.

Tehtävä aloitetaan lisäämällä ”Activity”-luokkaan uusi jäsenmuuttuja, johon tietokanta-olio luodaan.

Tallennettavat tiedot luetaan osittain käyttöliittymän komponenteista, jonka jälkeen niiden pohjalta luodaan uusi ”TaskEntry”-olio. Tämä olio voidaan tämän jälkeen lisätä tietokantaan, käyttämällä ”insertTask”-metodia.

Harjoituksessa otetaan väliaikaisesti käyttöön kyselyjen ajaminen ”UI”-säikeellä, jolla vain todennetaan ohjelman toimintaa. Normaalissa käytössä tämä ei todellakaan ole suositeltavaa.

1.52.3 T09b.03-Exercise-RetrieveTasksFromDatabaseAtMainActivity (1.5h)

Tässä harjoitellaan tietojen lukua tietokannasta, sekä niiden näyttämistä käyttöliittymällä.

Työ aloitetaan jälleen samalla tavalla kuin edellinenkin harjoitus. ”Activity”-luokan ”onCreate”-metodissa haetaan viittaus tietokanta-olioon, sekä tallennetaan se paikalliseen jäsenmuuttujaan.

Jotta näkymä päivitettäisiin aina, kun se näytetään, lisätään tietokantarivien listaava logiikka ”onResume”-metodin sisälle. Tämä ajetaan mm. silloin, kun palaamme toisesta näkymästä tähän näkymään.

Tiedot haetaan ”Task”-repositoriosta, käyttämällä ”loadAllTasks”-metodia. Löydetyt oliot annetaan käyttöliittymän listauksesta vastaavalle adapterille, joka muokkaa ne näytettävään muotoon.

1.52.4 T09b.04-Exercise-Executors (2h)

Tässä perehdytään ”Executor”-olioiden käyttöön. Niiden avulla voidaan luoda helposti rajapinta, joka ajaa tietokantakyselyt omassa säikeessään. Harjoituksessa käytettävä

”Executor”-olio asetetaan toimimaan yhden säikeen avulla, jolloin ”race conditions”-ehtoja ei tarvita. Tämä tarkoittaa sitä, että kaikki tehtävät kyselyt ajetaan tulevassa järjestyksessä. Aiemmin käytetyt ”Runnable”-oliot ajetaan nyt tämän uuden ”Executor”-olion kautta.

1.52.5 T09b.05-Exercise-DeleteTask (1.5h)

Tässä käytetään edellisen harjoituksen tapaan ”Executor”-oliota tietokantakyselyn ajamiseen. Harjoituksessa lisätään logiikka, joka poistaa pyyhkäisy-eleellä (swipe) muokatun rivin käyttöliittymästä sekä laitteen sisäisestä tietokannasta.

Poistettavaan riviin liittyvä ”TaskEntry”-olio saadaan ”RecyclerView”-olion adapterin kautta. Pyyhkäisy-eleen toteuttava rajapinta antaa meille parametrina numeron, joka kertoo, kuinka mones ”TaskEntry”-olio on kyseessä. Lisäämme adapteri-luokan toteutukseen erillisen metodin, joka antaa meille alkuperäisen listan ”TaskEntry”-olioista. Tämän jälkeen voimme rivinumeron perusteella hakea poistettavan ”TaskEntry”-olion tästä listasta.

1.52.6 T09b.06-Exercise-UpdateTask (1.5h)

Tässä harjoituksessa hyödynnetään räätälöityä tietokantakysyä, kun tietokannasta haetaan ”TaskEntry”-olio tietyn ”ID”-arvon perusteella.

Ohjelmaan lisätään logiikka, jossa ”TaskEntry”-olio lisätään tietokantaan, jos sillä ei ole vielä omaa ”ID”-arvoa (tai se on oletusarvollinen). Muussa tapauksessa olion tiedot päivitetään tietokantaan ”UPDATE”-tietokantakyselyllä. Logiikassa käytetään apuna edellisistä tehtävistä tuttuja ”Executor”-olioita. Käyttäjä voi ohjelmassa tallentaa muokatun tehtävän tiedot ”update”-nappia painamalla.

1.52.7 T09b.07-Exercise-AddLiveData (3h)

Tässä muutetaan osa tietokantaluokan kyselyistä palauttamaan halutut tiedot ”LiveData”-olioon ”pakattuina”. ”LiveData” on Androidin virallinen komponentti, jolla voidaan toteuttaa ”observer pattern”-arkkitehtuuria.

Aiemmin löydetyt tietokantarivit palautettiin ”List”-oliossa. Nyt lista on pakattuna ”LiveData”-olion sisään. ”LiveData”-paketointi mahdollistaa tiedon muuttumisen seuraamisen. Tämän avulla voimme suorittaa haluttuja asioita ainoastaan silloin, kun on tarve. Tässä harjoituksessa haluamme päivittää tehtävien listaa vain, jos tehtävät ovat muuttuneet.

Uusi ”Observer”-olio, eli seuraaja, rekisteröidään ”LiveData”-olion ”observer”-metodilla. Sille määritellään ”onChanged”-metodi, joka ajetaan aina, kun tiedot ovat muuttuneet. Harjoituksessa näytettävien tehtävien lista päivitetään aina, kun tietokannassa olevat tehtävät muuttuvat.

1.52.8 T09b.08-Exercise-AddLiveDataToAddTaskActivity (1.5h)

Tässä ”LiveData”-ominaisuus lisätään myös toiseen hakuun, jossa tehtäviä haetaan ”ID”-arvon perusteella. Muutamme siis samaan tapaan tietokantahaun toteuttavan metodin paluutyypin ”LiveData<TaskEntry>”-muotoon.

Voimme nyt myös poistaa aiemmin käyttämämme ”Executor”-logiikat, ja korvata ne ”LiveData”-rajapinnan ominaisuuksilla. Meidän ei enää tarvitse ajaa tietokantahakuja erillisessä säikeessä, vaan voimme hyödyntää aiemmin mainittuja ”Observer”-ominaisuuksia.

Erikoisuutena edelliseen tehtävään, tässä haluamme poistaa muutosten kuuntelijan (”Observer”) heti, kun rajapinta on ajanut ”onChanged”-metodin yhden kerran. Emme siis jätä kuuntelijaa ajamaan käyttöliittymän päivitystä uudelleen ja uudelleen muutosten tapahtuessa.

1.52.9 T09b.09-Exercise-AddTheViewModel (2h)

Aiemmat ratkaisut johtivat siihen, että laitetta käännettäessä, tietokantahaut suoritettiin uudelleen. Haluamme välttää tämän, jonka takia lisäämme ohjelmaamme ”ViewModel”-tuen. Vaikka ”Activity”-olio tuhotaan, jää ”ViewModel”-olio siitä riippumattomasti ”eloon”. ”OnSaveInstanceState”-metodia ei tässä harjoituksessa käytetty, koska se on tarkoitettu pienelle datalle, joka voidaan helposti serialisoida.

Yksinkertaisuudessaan lisäsimme projektiin uuden luokan, joka periytyy ”AndroidViewModel”-luokasta. Luomamme ”ViewModel”-olion instanssi saadaan staattisen ”ViewModelProviders”-luokan kautta.

Tietokantahaun toteuttavat logiikat on siirretty ”Activity”-luokastamme uuteen ”AndroidViewModel”-pohjaiseen luokkaan.

1.52.10 T09b.10-Exercise-AddViewModelToAddTaskActivity (2h)

Tässä haluamme lisätä ”ViewModel”-tuen myös ”AddTask”-näkymään. Tässä erona on se, että tehtävän hakemiseen tarvitaan ”ID”-arvo. Tämän vuoksi käytämme ”ViewModel”-olion luontiin erillistä ”Factory”-kaavaa, jossa ”olio tuotetaan tehtaan kautta”.

Tehdasolio periytetään ”ViewModelProvider.NewInstanceFactory”-luokasta, ja sen on toteutettava ”create”-metodi, joka palauttaa varsinaisen ”ViewModel”-oliomme.

Varsinainen ”ViewModel”-luokka vaatii konstruktorissa viittauksen ”AppDatabase”-olioon, sekä tehtävän ”ID”-arvon ”int”-tyyppisenä arvona. Aiemmin luomamme tehdasluokka osaa antaa nämä parametrit ”create”-metodissaan, sillä tehdas saa itse nämä arvot parametrikseen konstruktorissa.

Viimeinen vaihe oli ”ViewModel”-olion käyttöönotto, joka sujui miltei samalla tavalla kuin aiemminkin. Tällä kertaa loimme aluksi uuden instanssin tehdasluokasta, jonka jälkeen se annettiin parametrina staattisen ”ViewModelProviders”-luokan ”of”-metodille.

1.53 Lesson10-Hydration-Reminder

1.53.1 T10.01-Exercise-IntentServices (3h)

Tämä on ensimmäinen harjoitus, jossa kokeillaan Androidin ”Service”-ominaisuutta. Tarkoituksena on muokata valmista vedenjuonnista huolehtivaa applikaatiota. Käyttäjä pystyy merkkaamaan juomansa vesilasit, painamalla ohjelman käyttöliittymällä olevaa juomalasin kuvaa. Määrät tallennetaan laitteeseen ”SharedPreferences”-ominaisuuden kautta. Tällä hetkellä määrät tallennetaan suoraan ”Activity”-näkymän kautta. Haluamme kuitenkin muuttaa tätä niin, että muutokset tehdään ”Service”-palvelun avulla. Ideana on se, että tulevaisuudessa ohjelma voisi toimia niin, että juotujen vesien määrät voidaankin tallentaa esimerkiksi verkon välityksellä vaikka jollekin palvelimelle. ”Service”-rajapinta on sopiva työkalu kumpaankin tilanteeseen.

Ratkaisuun tarvitaan monta tiedostoa (mm. luokat ”ReminderTasks”, sekä ”WaterReminderIntentSevice”). Näiden lisäksi luotu ”Service”, eli ”WaterReminderService” on merkattava manifestiin. Tehtävässä käytetään ainoastaan yhtä ”Action”-parametria ”Service”-palvelun kanssa, joka on kovakoodattu ”ACTION_INCREMENT_WATER_COUNT”-lukujono. Luotu ”IntentSevice” siis aloitetaan tällä parametrilla, kun se käynnistetään ”MainActivity”-näkymästä. Tällä identifioidaan, mikä toiminto käynnistetyssä palvelussa toteutetaan.

1.53.2 T10.02-Exercise-CreateNotification (2.5h)

Tässä harjoitellaan ilmoitusten (”Notification”) luomista. Tarkoitus on muistuttaa laitteen käyttäjää tietyin väliajoin juomaan vettä. Ennen ajoitettua logiikkaa, testaamme ilmoitusten toiminnan aluksi napin avulla. Ilmoitusta varten projektissa koodataan erillinen ”NotificationUtils”-apuluokka, jota on tarkoitus käyttää ilmoitusten luontiin ja näyttämiseen.

Ilmoitusta klikkaamalla, ajetaan ”PendingIntent”-olio. Tällä voidaan avata ulkopuolinen ohjelma, joka on tässä kontekstissa meidän ”Hydration Reminder App”. ”PendingIntent”-

olio asetetaan myös avaamaan ohjelmamme ”MainActivity”-näkymä. ”PendingIntent”-olion luonti ja palautus toteutettiin ”contentIntent”-metodiin.

Toinen apuluokkaan kuuluva osa on mm. metodi, joka palauttaa ”Bitmap”-tyyppisen olio-kuvan, joka näytetään ilmoituksessa. Näytettävä ikoni ladataan projektin sisäisistä resursseista ”BitmapFactory”-luokan avulla.

Viimeinen vaihe on ”remindUserBecauseCharging”-metodi. Tämän sisällä meidän on tarkoitus luodu uusi ”NotificationChannel”, eli ilmoituskanava, jonka kautta ilmoituksemme näytetään. Kanava lisätään ”NotificationManager”-järjestelmäpalveluun, ja sille annetaan uniikki ”Channel ID”-avainarvo.

Ilmoituksen luomista varten käytämme ”NotificationCompat.Builder”-luokkaa. joka on ns. ”ilmoitusten rakentaja”. Rakentajan avulla voimme mm. asettaa ilmoituksen kanavakoodin (”Channel ID”), värin, pienen ikonin, ison ikonin, tekstin sekä tyylin. Lisänä on myös mahdollisuus asettaa ilmoitus värisyttämään laitetta, mikä vaatii kuitenkin luvan (”android.permission.VIBRATE”) lisäyksen projektin manifestiin (”AndroidManifest.xml”).

Viimeisenä osana voimme nyt kutsua ”NotificationManager”-olion ”notify”-metodia, jolla luomamme ilmoitus lähetetään.

1.53.3 T10.03-Exercise-NotificationActions (2h)

Tässä harjoituksessa on tarkoitus muokata edellisessä tehtävässä luotua ilmoitusta (”Notification”) niin, että se sisältää mahdollisia toimintoja. Käyttäjä voi painaa ilmoituksen alareunasta kahta erilaista valintaa: ”join vettä” sekä ”ei kiitos”. Valintaa painamalla käyttäjä voi siis suoraan kasvattaa juomaansa vesimäärää, avaamatta ohjelmaa kokonaan lainkaan. Käyttäjä voi myös ohittaa juonnin toisella valinnalla, joka myös poistaa kaikki aiheeseen liittyvät ilmoitukset.

Toimintoja varten luodaan vanhaan tapaan uniikit lukujonot, joita käytetään avaimina. Niiden avulla tiedämme, mitä valintaa käyttäjä painoi. Tämän lisäksi luomme jokaista valintaa kohden oman staattisen metodin, jolla ”NotificationCompat.Action”-olio luodaan. Nämä valinta-oliot rekisteröidään ilmoitukseen aiemmasta tehtävästä tutulla ” NotificationCompat.Builder”-oliolla.

Painettuun valintaan liittyvä toiminnallisuus ajetaan staattisen ”executeTask”-metodin sisällä. Se saa parametriksi valinnan uniikin avainarvon, jonka perusteella oikea logiikka suoritetaan.

1.53.4 T10.04-Exercise-PeriodicSyncWithJobDispatcher (3h)

Nyt on aika lisätä vedenjuonnista muistuttavaan applikaatioon ajoitettu ilmoituksen luominen. Android ”L” tarjosi tähän uuden rajapinnan, mutta se toimii ainoastaan Androidin ”API 21”-tasosta lähtien. Meillä on kuitenkin toinen kirjasto, ”FirebaseJobDispatcher”. Tämä kirjasto toimii jopa ”API 9”-tasolta asti. Se on siis heti paljon parempi vaihtoehto, koska tukee moninkertaisen määrän laitteita. Käytämme tätä ”FirebaseJobDispatcher”-kirjastoa ajoitettujen ilmoitusten näyttämiseen.

Homma aloitetaan riippuvuuden lisäyksellä, joka lisätään projektin ”build.gradle”-tiedostoon (firebase-jobdispatcher). Seuraavaksi lisäämme ”ReminderTasks”-luokan staattiseen ”executeTask”-metodiin uuden vaihtoehdon, jos ohjelmaa kutsutaan uudella uniikilla merkkijonolla (luokassa vakio ”ACTION_CHARGING_REMINDER”). Tällöin ohjelma luo uuden ilmoituksen veden juomisesta, sekä päivittää näytettyjen ilmoitusten lukumäärän.

Ajoitusta varten, meidän on luotava uusi luokka (”WaterReminderFirebaseJobService”), joka periytyy ”JobService”-luokasta. Tämän luokan tehtävänä on hoitaa uuden ilmoituksen näyttäminen. Luokka vaatii vähintään kahden metodin implementointia ”onStartJob”- sekä ”onStopJob”. Näistä ensimmäinen koodataan ajamaan varsinainen tehtävä, joka halutaan suorittaa. Toinen metodi tulee ajetuksi silloin, kun tehtävän suorittamiseen asetetut vaatimukset eivät enää täyty. Tässä harjoituksessa mm. laitteen irrottaminen latauksesta kesken ajon, laukaisee ”onStopJob”-metodin, ja tehtävä peruuntuu. Tällöin ilmoituksen luonti peruuntuu.

Varsinainen logiikka ajetaan ”AsyncTask”-luokan sisällä, koska ”onStartJob”-metodin koodi ajetaan normaalisti pääsäikeessä. Tämä lisää harjoituksen vaikeutta, mutta se on tähän tarkoitukseen paras tapa. Kutsumme asynkronisesti aiemmin muokkaamaamme, staattista ”executeTask”-metodia.

Viimeinen vaihe on lisätä luomamme ”WaterReminderFirebaseJobService”-luokan määrittely ”AndroidManifest.xml”-tiedostoon, käyttäen ”service”-tagia.

Lopuksi poistamme vanhan testinappulan, jolla ilmoituksia luotiin. Nyt ohjelma osaa luoda ilmoitukset ajoitetusti!

1.53.5 T10.05-Exercise-ChargingBroadcastReceiver (2.5h)

Tämä on ensimmäinen tehtävä, jossa harjoitellaan ”BroadcastReceiver”-rajapinnan käyttöä. Tämä rajapintaa antaa työkalut, joiden avulla voimme ohjata ohjelmaamme tiettyjen tapahtumien mukaan. Näitä tapahtumia voivat esimerkiksi laturin tai vaikka kuulokkeiden liittäminen laitteeseen. Yksinkertaisuudessaan rekisteröimme applikaatiomme kuuntelemaan näitä muutoksia, käyttämällä ”BroadcastReceiver”-kirjastoa.

Päänäkymään lisätään uusi metodi (”showCharging”), joka muuttaa käyttöliittymällä olevan pistokkeen väriä.

Luomme ylikirjoitetussa ”onCreate”-metodissa uuden ”IntentFilter”-olion dynaamisesti, joka asetetaan päästämään läpi ainoastaan laturin kytkemistä sekä irroittamista (”Intent.ACTION_POWER_CONNECTED” ja ”Intent.ACTION_POWER_DISCONNECTED”) koskevat tapahtumat (”Action”).

Tapahtuman vastaanottamista varten luomme erillisen luokan, joka periytyy ”BroadcastReceiver”-luokasta. Tälle on implementoitava ”onReceiver”-metodi, jota kutsutaan, kun aiemmin luomamme ”IntentFilter”-olio päästää läpi tapahtuman. Asetamme tämän metodin kutsumaan alussa luomaamme ”showCharging”-metodia. Varsinaisen tapahtuman tiedon saamme parametrin ”Intent”-olion ”getAction”-metodista.

Tarvitsemme tiedot tapahtumista vain silloin, kun päänäkymä on avattuna, koska silloin käyttöliittymää on päivitettävä. Rekisteröimmekin ”BroadcastReceiver”-oliomme kuuntelemaan tapahtumia ”onResume”-metodissa, sekä lopettamaan kuuntelun ”onPause”-metodissa. Rekisteröinti saa parametrikseen ”BroadcastReceiver”- sekä ”IntentFilter”-oliomme.

1.53.6 T10.06-Exercise-StickyBroadcastForCharging (1.5h)

Aiemmassa ratkaisussa oli puute, joka saattoi jättää käyttöliittymälle väärän latauskuvan. Tämän vuoksi meidän on lisättävä ”onResume”-metodiin lisäys, joka ajaa myös ”showCharging”-metodin.

Tällä kertaa emme saa kuitenkaan laturin tilaa samalla tavalla kuin aiemmin, vaan se on selvitettävä muulla tavalla.

Tehtävässä käytetään laitteen ”API”-tason mukaan kahta erilaista ratkaisua. Jos laite on ”API 23”-tasoinen tai korkeampi, on paras tapa käyttää ”BatteryManager”-luokkaa. Vanhemmille laitteille taas ”Sticky Intent”-olion kautta.

”BatteryManager”-olio antaa latauksen tilan ”isCharging”-metodin paluuarvona. ”Sticky Intent” tapauksessa meidän on pyydettävä tila ”Context”-oliolta, kutsumalla ”registerReceiver”-metodia, jolle annetaan ”IntentFilter”-olio, joka rajaa tapahtumat akun muutoksiin (”Intent.ACTION_BATTERY_CHANGED”).

1.54 Lesson11-Completeing-The-UI

1.54.1 T11.01-Exercise-ConstraintLayout (3h)

Tämä oppituntiosio tähtää käyttöliittymän opiskeluun. Tämä on ensimmäinen tehtävä, jossa luodaan uudelleen esimerkin mukainen ”boarding pass”, eli koneeseennousukortti.

Ulkoasu rakennetaan kokonaan ”ConstraintLayout”-näkymän päälle. Tämä ulkoasu on uusi sekä rakenteeltaan muita tyyppejä (esim. ”FrameLayout” tai ”LinearLayout”) kevyempi. Tämän vuoksi sen käyttö on suositeltavaa aina, kun se on mahdollista.

Ulkoasu rakennetaan niin, että ”ConstraitLayout”-näkymään lisätyt elementit (”View” tai ”ViewGroup”) piirretään suhteessa toisiinsa. Esimerkiksi piirrettävä nappi asetetaan suhteelliseen kohtaan pohjan ”ConstraitLayout”-näkymään nähden. Sen alapuolelle voidaan lisätä erikseen esim. ”Label”-näkymä, jonka paikka on suhteutettu nappulaan. Tekniikka tukee tietenkin myös esim. ”Padding”- ja ”Margin”-ominaisuuksia, joten mahdollisuuksia on paljon.

Tehtävän ratkaisussa oli apuna hyvä video, jossa koko maihinnousukortti koottiin alusta loppuun. Mielestäni kaikkein vaikein kohta oli kortin keskivaiheilla, johon lentokone lisätään vektorikuvana. Siinä lähtöpaikan ja määränpään väli asetetaan venyväksi, sekä niiden keskelle piirretään lentokone. Sanoisin että ”ConstraintLayout”-pohjaisen käyttöliittymän teko kannattaa aina aloittaa Android Studion käyttöliittymätyökalulla. Valmista käyttöliittymää on sitten helppo hienosäätää XML-koodista.

1.54.2 T11.02-Exercise-DataBinding (2h)

Viime tehtävässä käyttöliittymän tekstit ja arvot luettiin kovakoodattuina ”res”-kansion resursseista (esim. ”strings.xml”-tiedostosta). Tässä harjoituksessa on päämääränä ottaa käyttöön ”Data Binding Library”-kirjasto. Se antaa meille työkalut linkittää tietoa käyttöliittymän komponentteihin ilman, että joudumme käymään jokaisen elementin läpi yksi kerrallaan (kutsuen ”findViewByID”-metodia).

Homma aloitetaan käyttöönottamalla kirjasto, muokkaamalla ”gradle.build”-tiedostoa. Se tapahtuu yksinkertaisesti lisäämällä asetuksen: ”dataBinding.enabled = true;” (tämän jälkeen projekti on synkronoitava). Asetus listataan ”android”-osion sisälle.

Liitettävä tieto luetaan harjoituksen mukana tulevasta ”BoardingPassInfo”-luokasta. Harjoitus sisältää toisen luokan, ”FakeDataUtils”, joka generoi ”BoardingPassInfo”-olion satunaisilla tiedoilla. Tämän olion jäsenmuuttujat on tarkoitus linkittää käyttöliittymän komponentteihin.

Android luo automaattisesti tuen tietojen liitoksille, kun käyttöliittymän juurena käytetään ”layout”-elementtiä. Muokkaamme siis käyttöliittymää tämän mukaisesti. Tämän jälkeen ”Android Studio” on luonut meille ”ActivityMainBinding”-olion, jonka kautta tietojen liittäminen voidaan tehdä (Olio luodaan dynaamisesti, ja se noudattaa tietynlaisia nimeämissääntöjä). Voimme nyt lisätä ”MainActivity”-luokkaamme uuden jäsenmuuttujan, johon ”ActivityMainBinding”-olio alustetaan.

Saamme ”ActivityMainBinding”-olion ”DataBindingUtil”-luokan kautta, kun kutsumme ”setContentView”-metodia. Se saa parametrikseen kyseisen näkymän (”this”), sekä näkymän ulkoasutiedoston resurssina. Olion avulla asetamme jokaisen käyttöliittymän komponentin tiedot yksi kerrallaan. Olio sisältää julkiset jäsenmuuttujat jokaista komponenttia kohden, joiden kautta tieto asetetaan.

1.54.3 T11.03-Exercise-LandscapeLayout (2.5h)

Android tukee ominaisuutta, jossa sama näkymä voidaan toteuttaa eri tavalla laitteen orientaation mukaisesti. Voimme siis esimerkiksi määritellä kaksi täysin omaa tiedostoa, joissa päänäkymä on aivan erilainen. Tällä tavoin voimme käyttää mm. horisontaalisessa orientaatiossa olevan laitteen näytön tilaa hyödyllisemmin.

Jotta kahden ulkoasutiedoston käyttäminen olisi järkevää, voimme myös jakaa käyttöliittymän osia omiin tiedostoihinsa. Voimme esimerkiksi luoda värikkään otsikkorivin, jota käytetään monissa näkymissä. Tämä onnistuu yksinkertaisesti luomalla uuden ulkoasutiedoston projektin ”layout”-kansioon. Lisäämme ulkoasun pohjalle vaikka ”ConstraintLayout”-elementin, jonka sisälle kopioimme ositettavasta ulkoasusta alkuperäisen koodin.

Ulkoistetut osat voidaan lisätä takaisin alkuperäisiin näyttöihin ”include”-elementin avulla. Viittaus ulkoistettuun osaan asetetaan elementin ”layout”-attribuutilla.

Edellisessä harjoituksessa luotu käyttöliittymä jaetaan kahteen eri tiedostoon (pysty- sekä vaakatasoiseen versioon). Tämän lisäksi käyttöliittymältä eriytetään omiin tiedostoihinsa mm. koneeseen nousemiseen liittyvät tiedot (sinisellä taustalla) sekä lennon tiedot (lentokoneen ikoni, mistä mihin jne).

Osituksessa on otettava huomioon, että viittaukset siirrettyihin komponentteihin on korjattava. Käsittelemmekin eriytettyjä komponenttikokonaisuuksia nyt ”yhtenä komponenttina” (Esim. lennon tiedot). Muokkaamme ulkoasun toimimaan nyt suhteellisesti tähän komponenttikokonaisuuteen nähden, sekä poistamme mahdolliset viittaukset kokonaisuuden ulkopuolelta sen sisällä oleviin komponentteihin.

1.55 Lesson12-Visual-Polish

1.55.1 T12.01-Exercise-ColorsAndFonts (0.5h)

Tämä on ensimmäinen harjoitus, jolla aloitetaan ulkoasun muokkaamisen opettelu.

Harjoitus aloitetaan listaamalla uusia värejä. Ne lisätään miltei samalla tavoin kuin esim. kovakoodatut merkkijonot (”strings.xml”). Tässä tehtävässä värit listataan ”colors.xml”-tiedostoon. Väri määritetään ”color”-elementillä.

Kun värit on muokattu tehtävänannon mukaisesti, lisätään näkymään myös uusi teksti (joka on määritelty ”strings.xml”-tiedostoon). Teksti esitetään ”TextView”-elementin avulla. Viimeinen vaihe edellyttää, että tekstin ”textSize”-attribuutti on oltava kokoa ”16sp”, sekä ”fontFamily”-attribuutin on oltava ”sans-serif-smallcaps”.

1.55.2 T12.02-Exercise-CreateNewStyles (0.5h)

Android sisältää tuen ”tyyleille” (”style”). Niiden avulla voimme niputtaa ulkoasussa käytettävien elementtien attribuutteja. Tietyllä tavalla siis ”ulkoistamme” osan attribuuteista erillisen tyylin sisälle. Sen jälkeen sisällytämme tyylin elementtiin, jolta se saa tarvittavat attribuutit. Tämä on hyödyllistä mm. silloin, jos ulkoasussamme toistuvat samat attribuutit samoilla arvoilla.

Tyylien seuraava taso on niiden ”periminen”. Voimme siis luoda tyylin, joka pohjautuu johonkin toiseen tyyliin. Uudelleen määriteltävät attribuutit ylikirjoittavat pohjatyylissä olevat attribuutit.

Harjoituksessa luodaan ensiksi yksi tyyli, joka sisältää kaikissa ulkoasun ”TextView”-elementeissä toistuvat attribuutit ja arvot. Kun ne on korvattu yhdellä tyylillä (”folderStyle”), luomme vielä ylimälle ”Inbox”-riville oman tyylin (”inboxStyle”). Tyyli perii aiemmin luodun tyylin, sekä lisää siihen vielä ”textStyle”-attribuutin. Näin saamme ylimmän rivin korostettua, samalla, kun se pohjautuu jo aiemmin luotuun tyyliin (ilman duplikaatti koodia!).

1.55.3 T12.03-Exercise-TabletLayout (1.5h)

Tässä harjoitellaan responsiivisen ulkoasun luontia tabletille. Käytämme apunamme jo aiemmin opeteltuja tekniikoita, joilla mm. luomme erilliset ulkoasut pysty- ja vaakatason orientaatioille.

Orientoinnin avulla eriytetyt ulkoasut saatiin ”-port”- ja ”-land”-päätteillä. Tablettia varten meidän on käytettävä uutta ”smallest width”-nimeämistä. Sillä voidaan määrittää

”pienin sopiva leveys”, jolla ulkoasu valitaan. Tavallaan kansion nimestä tulee siis oma ”if-lauseensa”.

Tablettia varten luomme uuden resurssi-kansion, nimeltä ”res/layout-sw600dp”. Kopioimme tämän sisälle ”layout”-kansiosta ”responsive_activity.xml”-ulkoasun. Tämän jälkeen, kun applikaatio ajetaan tabletilta, käytetään ulkosuun ”layout-sw600dp”-kansion versiota. Viimeisessä vaiheessa muokkaamme tämän ulkoasun tehtävän mukaisesti (mm. kuva tekstin vasemmalle puolelle).

1.55.4 T12.04-Exercise-TouchSelector (0.5h)

Android tukee tietynlaisia ”valitsimia” (”selector”), joilla voimme muokata elementtien ulkoasua. Yksi tällaisista on esimerkiksi nappi (”Button”). Valitsin voidaan määrittää sisältämään erilaisia värejä tilojen mukaisesti. Nappi voi olla tilaltaan esim. painettu tai ei-painettu. Voisimme siis asettaa napin esimerkiksi muuttumaan eriväriseksi, kun se on painettuna sormen alla. ”Selector”-määrittely asetetaan elementin ”background”-attribuutin arvoksi.

Tehtävässä luodaan yksinkertainen valitsin (”list_item_selector.xml”), jolla näkymän listasta saadaan interaktiivisempi, kun se vaihtaa väriään painettaessa.

Tämä oli tehtäväpuolen viimeinen harjoitus!

UD851-SUNSHINE-STUDENT MOOC-HARJOITUKSET

Tämä oli toinen ”paketti” harjoituksia. Idea on ilmeisesti rakentaa harjoituksen vastauksista lopulta toimiva ”Sunshine”-applikaatio.

1.56 S01.01-Exercise-CreateLayout (1.25h)

Koottuna tässä oli samoja asioita korjattavana kuin aiemmassa ”Lesson01-Favorite-Toys”-paketissa. Gradlesta poistettiin ConstrainLayout-riippuvuus. Oletusnäkymän juurielementti muutettiin FrameLayout-tyyppiseksi, teksti muutettiin ”ScrollView”-elementin sisälle jne.

1.57 S01.02-Exercise-AddWeatherList (1h)

Näkymästä haettiin TextView-elementti, jonka tekstiksi asetettiin riveittäin päästä keksittyä ”säädataa”. Tekstit lisättiin käymällä koko lista läpi, ja lisäämällä tekstien väliin rivinvaihtomerkit. Käytin itse ”setText”-metodia tekstin asettamiseen.

1.58 S02.01-Exercise-Networking (2h)

Tässä harjoituksessa oli jälleen tarkoituksena lisätä ”Sunshine”-applikaatioon uusia ominaisuuksia, joita käytiin läpi ”Excercises”-paketin puolella. Osana oli mm. ”URI”-osoitteen rakentaminen, lupa internet-yhteyden käyttämiseen ja oikean säädatan hakeminen internetistä. Paikka säätietojen hakuun luettiin ”SharedPreferences”-rajapinnan kautta. Saatu säätieto saapui verkosta ”JSON”-formaatissa. Verkkohakuun käytettiin aiemmista harjoituksista tuttua ”AsyncTask”-luokkaa.

1.59 S02.02-Exercise-Menus (1h)

Tässä ideana oli lisätä ”Sunshine”-applikaatioon kustomoitu ”menu”, jossa on nappi säätietojen päivittämiselle. Menun lisäys on jo harjoituspuolelta tuttua puuhaa. Menu itemin teksti lisättiin normaaliin tapaan ”strings.xml”-tiedostoon. Resurssikansioon (”res”) lisättiin oma alikansio ”menu”, johon uusi ”forecast”-menu lisättiin. Menu laajennettiin ylikirjoitetussa ”onCreateOptionsMenu”-metodissa. Tietojen päivitys hoidettiin ylikirjoitetussa ”onOptionItemSelected”-metodissa.

1.60 S02.03-Exercise-Polish (1h)

Samoin kuin harjoituspuolella, tässä ”kiillotettiin” toiminta kuntoon. Latausta varten näkymässä näytetään ”ProgressBar”-elementti. Näkymässä on piilotettu ”TextView”-elementti, joka asetetaan näkyväksi virheen sattuessa (ja haun tulokset piilotetaan). Näytettävä virheviesti kirjoitettiin normaalisti ”strings.xml”-tiedostoon. Latauspalkkia ja virheitä ohjataan aiemman harjoituksen mukaisesti ”AsyncTask”-luokan sisältä.

1.61 S03.01-Exercise-RecyclerView (2h)

Tämän harjoituksen tarkoitus on lisätä aiemmassa harjoituspaketin ”Lesson03-Green-Recycler-View”-harjoituksessa opitut asiat osaksi ”Sunshine”-ohjelmaan. Tästä on kuitenkin luettuna pois klikkauksien käsittely.

Kävin vaiheet tarkasti läpi, ja otin mallia harjoituspaketin tehtävästä.

Lopuksi minun on pakko sanoa, että tämän kierrättävän komponentin käyttöönotto voi olla yhtä tuskaa. Se koostuu niin monista eri palasista ja liitoksista. Tehtävässä käytetty rakenne pitäisi tallettaa johonkin turvaan malliksi, jolloin sitä voisi käyttää pohjana mm. omissa projekteissa. Kaikkien asioiden muistaminen ulkoa vaikuttaa mahdottomalta.

1.62 S03.02-Exercise-RecyclerViewClickHandling (0.5h)

Tässä lisätään samaan tapaan, kuin harjoituspaketin tehtävässä, klikkauksen käsittely listan itemeille. Samat työvaiheet löytyvät ”T03.07-Exercise-RecyclerViewClickHandling”-kohdasta, josta ne voi katsoa läpi. Sama juttu.

1.63 S04.01-Exercise-LaunchNewActivity (1.5h)

Harjoituksessa lisätään uusi tyhjä ”Activity” nimeltä ”DetailActivity”. Kuten harjoituspuolen tehtävässä, luodaan uusi ”Intent”-olio käyttämällä ”MainActivity”-luokkaa kontekstina, sekä päämäärän parametrina ”DetailsActivity.class”. Siirtyminen aloitetaan vanhaan ja tuttuun tapaan ”startActivity”-metodin avulla.

Tehtävässä ylikirjoitettiin aimmin ”Toast”-viestin näyttävä metodi siirtymään näkymältä toiselle.

Uusi ”DetailsActivity” pitää olla lisättynä myös ”AndroidManifest.xml”-tiedostossa.

1.64 S04.02-Exercise-DisplayDayForecast (1.5h)

Kuten harjoituspuolen tehtävässä, tässä ”Intent”-olion mukaan lisätään lähetettävää dataa. Apuna käytetään ”Intent”-olion tuttua ”putExtra”-metodia, jolle annetaan parametriksi ”Intent.EXTRA_TEXT” (”avain”) sekä ”String”-tyyppisenä päivän säätiedot (”luku”). Data lisätään ”Intent”-olioon ennen ”startActivity”-metodin ajoa.

”DetailsActivity”-olion puolella mahdollisesti lähetetty paketti vastaanotetaan. Mahdollisesti lähetetty ”Intent”-olio luetaan ”getIntent”-metodin kautta (kuten harjoituspuolen tehtävässäkin). Jos tämä metodi palauttaa ”null”, tai palautetun ”Intent”-olion metodi ”hasExtra(Intent.EXTRA_TEXT) palauttaa ”false”, ei parametria lähetetty lainkaan. Muussa tapauksessa lähetetty parametri saadaan luettua tuttuun tapaan ”getStringExtra(Intent.EXTRA_TEXT)”-metodin kutsulla.

Jos parametri saadaan luettua, se asetetaan käyttöliittymän ”TextView”-elementtiin.

1.65 S04.03-Exercise-AddMapAndSharing (2.5h)

Tämä harjoitus oli tehtäviltään hieman työläämpi. Tässä lisätään kahden eri näkymän yläreunan menu-palkkiin uusi ”item”, joista toisella jaetaan säätietoja ja toisella paikkatietoja . Tehtävässä tehdyt asiat käydään harjoittelupaketin puolellakin läpi.

”DetailsActivity”-näkymän menu asetetaan tutulla ja ylikirjoitetulla ”onCreateOptionsMenu”-metodilla. Menun on määritelty ”res/menu”-kansion tiedostoon ”detail.xml”. Siinä on vain yksi ”item”, jolle annetaan ”id”-arvoksi ”action_share”. Tarvitsemme tätä ”id”-arvoa siinä kohtaa, kun linkitämme tämän ”Intent”-olion ja menun itemin toisiinsa. Nappia painamalla, säätiedot jaetaan tekstipohjaisesti (kuten aiemmassa harjoituksessa).

Toinen nappi lisätään ”MainActivity”-näkymän menuun. Tätä nappia painamalla lähetetään ”Uri”-olio, jonka sisältö koostuu paikkatiedoista (kuten harjoituksissa: ”geo”-scheme, latitude, longitude).

”setIntent”-metodi, jolla ”Intent”-olio sekä menun ”item”-elementti linkitetään toisiinsa, tehdään ”onCreateOptionsMenu”-metodin sisällä. Samassa paikassa ”menu”-palkit laajennetaan ”xml”-tiedostoista. Kun menun nappia painaa, se laukaisee siihen linkitetyn ”Intent”-olion ja siirtymisen.

Käytettävät tekstit on listattu normaaliin tapaan ”strings.xml”-tiedostoon.

1.66 S05.01-Exercise-AsyncTaskLoader (1.5h)

Tässä lisättiin harjoituspaketin puolella tutuksi tullut ”AsyncTaskLoader”-komponentti ohjelmassa aiemmin käytetyn ”FetchWeatherTask”-komponentin tilalle.

Miltei täysin samat työvaiheet löytyvät harjoitustehtävästä ”Lesson05b-Smarter-GitHub-Repo-Search”. Tässä muunnettiin vielä aiemmin käytetty ”refresh”-menu napin logiikka käynnistämään ”Loader”-komponentti uudelleen.

Logiikkaan lisättiin myös kohta, jossa käyttäjälle näytetään tyhjä hakutulos, ennen kuin uudet tiedot on ladattu. Tämä lisää käyttäjäystävällisyyttä, koska käyttäjä näkee, että haku on aloitettu.

1.67 S06.01-Exercise-LaunchSettingsActivity (2h)

Tässä ”Sunshine”-applikaatioon lisätään harjoituspuolella opittu ”Settings”-näkymä. Tämä näkymä on vielä toistaiseksi tyhjä, mutta harjoituksessa lisätään ainoastaan siirtymislogiikat.

”AndroidManifest.xml”-tiedostossa on jälleen merkattava ”MainActivity”-näkymälle attribuutti ”android:launchMode=’singleTop’”, Tämän lisäksi ”application”-tagien sisälle on lisättävä uusi ”activity”-elementti, uutta ”SettingsActivity”-näkymää varten.

”DetailActivity”-näkymälle on lisättävä ylikirjoitettu ”onOptionsItemSelected”-metodi, jossa ”menu”-palkin painike linkitetään siirtymään ”SettingsActivity”-näkymään. Myös ” MainActivity”-luokan ”onOptionsItemSelected”-metodi on ylikirjoitettava samalla tavalla.

”SettingsActivity”-näkymään lisätään koodia sen verran, että sen ylikirjoitetussa ”onOptionsItemSelected”-metodissa suoritetaan ”onBackPresset”-metodi, kun ”menu”-palkista painetaan kotinäppäintä (nuoli). Napin painallus siis vastaa ”takaisin”-napin painamista laitteessa.

Varsinainen ”Settings”-nappi lisätään applikaatioon ”menu”-kansiossa olevaan ”forecast.xml”-tiedostoon. Tiedostoon on listattava uusi ”item”-elementti.

1.68 S06.02-Exercise-SettingsFragment (2h)

Tässä lisäämme harjoituspuolella toteutetun ”Settings”-sivun sisällön, käyttämällä ”PreferenceFragmentCompat”-luokkaa, sekä asetukset listaavaa ”pref_general.xml”-tiedostoa. Idea on listata asetukset täysin samaan tapaan kuin aiemmin, paitsi tässä valittavia asetuksia on vain kaksi (”EditTextPreference”- sekä ”ListPreference”-elementeillä asetettavat arvot). Tekstipohjainen asetus on paikkatietoa varten (esim. ”Pori, FI 28100”). Lista-pohjainen asetus on taas mittajärjestelmän valintaa varten (celcius/fahrenheit).

Aiemmin luodun (tyhjän) ”SettingsActivity”-näkymän ”XML”-tiedoston sisältö ylikirjoitetaan sisältämään ”fragment”-määritys, jonka jälkeen sen sisällä voidaan esittää ”Fragment”-komponentteja.

Viimeinen silaus on ”preference summary”-otsikoitten lisäys. Aimmista harjoitustehtävistä viisaana, ”ListPreference”-valita vaati jälleen hieman enemmän työtä kuin tekstipohjainen.

Tämä oli hyvää kertausta!

1.69 S06.03-Exercise-PolishingPreferences (1.5h)

Tässä vaiheessa aiemmin listatut ”Preference”-arvot otetaan varsinaiseen käyttöön. Homma aloitetaan muokkaamalla applikaation paketissa olevaa ”SunshinePreferences”-luokkaa, jonka sisälle ohjelman asetuksia on listattu. Mm. ”getPreferredWeatherLocation”-metodiin on lisättävä tallennetun paikkatiedon lukeminen ”SharedPreferences”-komponentin avulla. Mm. ”isMetric”-metodissa luetaan käyttäjän tallentama mittajärjestelmän valinta, joka muutetaan ”boolean”-tyyppiseksi paluuarvoksi.

Tämän jälkeen ”MainActivity”-luokkaan on lisättävä ”OnSharedPreferenceChangeListener”, jonka avulla voimme ilmoittaa ohjelmalle, jos tallennetut asetukset ovat muuttuneet. Kuuntelija rekisteröidään tuttuun tapaan ”onCreate”-metodissa, sekä poistetaan ”onDestroy”-metodissa (muistivuodon vuoksi). Kun tallennetut tiedot muuttuvat, asetetaan paikallinen ja staattinen ”PREFERENCES_HAVE_BEEN_UPDATED”-muuttuja arvoon ”true”.

”MainActivity”-luokan ”onStart”-metodi on vielä ylikirjoitettava. Sen sisälle lisätään logiikka, joka tarkastaa aiemmin mahdollisesti asetetun ”PREFERENCES_HAVE_BEEN_UPDATED”-muuttujan arvon. Jos tiedot ovat muuttuneet, käynnistetään ”Loader”-komponentti uudelleen (eli logiikka, joka hakee tiedot verkosta) sekä nollataan muuttujan arvo.

”openLocationInMap”-metodi muutetaan käyttämään aiemmin muokattua ”SunshinePreferences”-luokkaa paikkatiedon selvittämiseen.

1.70 S09.04-Exercise-UsingCursorLoader (2h)

Tässä ”Sunshine”-applikaation ”AsyncTaskLoader”-komponentti korvattiin “CursorLoader”-komponentilla. Dataa ei myöskään enää haeta itse, vaan sen hakemiseen käytetään ”WeatherProvider”-komponenttia (”Content Provider”).

Tulokset saadaan tuttuun tapaan ”Cursor”-olion sisällä, jolla myös korvataan aiemmin käytetty ”String[]”-taulukko. Vanhentunutta ja päivitettyä ”Cursor”-oliota varten luotiin myös uusi ”swapCursor”-metodi, jolla ylikirjoitetaan paikallinen ”Cursor”-jäsenmuuttuja. Rivien laskenta muutettiin käyttämään ”Cursor”-olion ”getCount()”.

”Cursor”-olion käyttö tapahtui samaan tapaan kuin aiemmin harjoituspaketin tehtävissä. Kolumnien indeksit ovat tallennettuna ”MainActivity”-luokaan muuttujina, josta niitä käytetään ”Cursor”-olion kanssa.

1.71 S09.05-Exercise-MoreDetails (1.5h)

Tässä ”CursorLoader”-komponentti lisätään myös ”Details”-näkymän puoelle, näyttämään lisää säähän liittyviä tietoja.

Ulkoasua muutettiin siten, että jokainen tieto on omassa ”TextView”-elementissään, ”LinearLayout”-juurielementin sisällä. Näytettäviä tietoja ovat mm. valitun päivän päivämäärä ja ilmanpaine.

”DetailsActivity”-luokka asetettiin toteuttavan ” LoaderManager.LoaderCallbacks<Cursor>”-rajapinta, jonka myötä lisättiin ”onCreateLoader”, ”onLoadFinished” sekä ”onLoaderReset”-metodit toteutuksineen. Varsinainen ”CursorLoader”-olio alustettiin ”DetailActivity”-luokan ylikirjoitetussa “onCreate”-metodissa.

”MainActivity”-luokka muokattiin sen verran, että se lähettää nyt siirryttäessä ”Intent”-olion ”Bundle”-paketissa ”Uri”-tyyppisen olion. Ajatus on siis se, että ”DetailsActivity”-saa pelkän osoitteen päivän säätietojen avaamiseen. Se vastaanottaa osoitteen ja kysyy tarvitsemansa tiedot sen jälkeen ”Content Provider”-komponentin kautta.

”Cursor”-oliosta parsitaan tarvittavat tiedot kuten aiemmissakin harjoitustehtävissä, jonka jälkeen ne kirjoitetaan lisättyihin ”TextView”-elementteihin.

1.72 S10.01-Exercise-SynchronizingTheWeather (2h)

Käytämme tässä tehtässä apuna aiemmin harjoituspuolelta opittuja menetelmiä applikaation tietokannan päivittämiseen.

Aloitamme muutoksen lisäämällä uuden luokan, ”SunshineSyncTask”, jonka tehtävänä on suorittaa varsinainen tietokannan päivitys. Siirsimme aiemman logiikan ”AsyncTaskLoader”-luokasta tänne, ”syncWeather”-nimiseen metodiin. Uudet tiedot asetetaan ylikirjoittamaan tietokannassa olevat vanhat tiedot.

Seuraavaksi luomme toisen uuden luokan, ”SunshineSyncIntentService”, joka periytyy ”IntentService”-luokasta. ”IntentService”-luokan konstruktoria kutsutaan käyttämällä tämän luokan nimeä parametrina. Viimeinen vaihe on ylikirjoittaa ”onHandleIntent”-metodi, joka asetetaan kutsumaan aiemmin luomaamme ”syncWeather”-metodia. Palvelu on tämän jälkeen lisättävä ”AndroidManifest.xml”-tiedostoon.

Kolmas vaihe on tietokannan päivitysten varsinainen käyttöönotto. Tätä varten luomme uuden apuluokan, nimeltä ”SunshineSyncUtils”. Tämän sisälle luomme yhden metodin

(”startImmediateSync”), joka aloittaa ”IntentService”-olion (”SunshineSyncIntentService”), ja aloittaa päivityksen heti.

Lopuksi asetamme ”SettingsFragment”-näkymän kutsumaan tätä ”startImmediateSync”-metodia aina, kun laitteen sijainti muuttuu.

”Sunshine”-applikaatio alkaa olemaan tässä kohtaa jo sen verran laaja, että sen rakenne alkaa olemaan monimutkaisempi.

1.73 S10.02-Exercise-SmarterSyncing (2h)

Tässä tehtävässä päivitämme edellisen harjoituksen ratkaisua, tekemällä päivittämisestä järkevämpää. Haluamme varmistaa, että ”startImmediateSync”-metodi ajetaan vain kerran, kun applikaatio ajetaan. Tahdomme myös ajaa päivityksen ainoastaan silloin, kun laitteen sisäinen tietokanta on tyhjä.

Lisäämme jo olemassa olevaan ”SunshineSyncUtils”-luokkaan uuden ”boolean”-tyyppisen jäsenmuuttujan nimeltä ”sInitialized”. Käytämme tätä muuttujaa ns. ”lippuna”, joka nostetaan, kun tietokanta on päivitetty.

Seuraava tehtävä on lisätä uusi ”initialize”-metodi, joka käyttää aimmin lisättyä ”sInitialized”-muuttujaa hyväkseen. Se kutsuu ”startImmediateSync”-metodia ainoastaan silloin, kun ”sInitialized” on epätosi (”false”), jonka jälkeen se asettaa sen arvoksi tosi (”true”). Tietokannan sisältö luetaan aiemmin lisäämämme ”ContentProvider”-rajapinnan kautta. Päivitystä tehdään ainoastaan silloin, kun mitään tietoja ei ole saatavilla. Tällainen tilanne on mm. silloin, kun ohjelma on juuri asennettu laitteeseen. Tietokantakyselyt tehdään samaan tapaan kuin aiemmissakin harjoituksissa, ja ne on ajettava eri kuin pääsäikeellä. ”ContentProvider”-rajapinnan kyselyt voidaan suorittaa taustalla käyttämällä ”AsyncTask”-luokkaa.

Lopuksi poistamme päänäkymästä ”startImmediateSync”-metodin kutsun, ja korvaamme sen uudella ”initialize”-metodilla.

1.74 S10.03-Exercise-FirebaseJobDispatcher (2.5h)

Seuraava vaihe on lisätä ajoitetut päivitykset ”Sunshine”-applikaation. Haluamme, että ohjelma osaa päivittää tarvitsemansa tiedot taustalla. Käyttäjän ei siis tarvitse odottaa päivitysten latausta silloin, kun ohjelma on avattuna, ja sään tila pitäisi tarkastaa.

Käytämme ajoitusten toteutukseen samaa kirjastoa kuin harjoituspuolella, eli ”FirebaseJobService”-luokkaa. Tämä kirjasto toimi ”API 9”-tasoon asti. Käyttöä varten meidän on lisättävä ”FirebaseJobDispatcher”-kirjasto projektin ”build.gradle”-tiedostoon (com.firebase:firebase-jobdispatcher). Tämän lisäksi käynnistettävä palvelu (Service), on merkattava ”AndroidManifest.xml”-tiedostoon (tässä ”SunshineFirebaseJobService”).

Luomme uuden ”JobService”-luokasta periytyvän luokan nimeltä ”SunshineFirebaseJobService”. Tämän palvelun tehtävä on hoitaa varsinainen tietojen päivittäminen. Kuten harjoituspuolella, ylikirjoitamme uudessa luokassa ”onStartJob”- sekä ”onStopJob”-metodit. Asetamme ”onStartJob”-metodin kutsumaan tietokannan tiedot päivittävää ”syncWeather”-metodia taustalla (käyttäen tutuksi tullutta ”AsyncTask”-luokkaa). Onnistuneen päivityksen jälkeen metodin lopussa kutsutaan ”jobFinished”-metodia. Kuten harjoituksessa, meidän on putsattava taustalla ajettava ”AsyncTask”-olio, kun ”onStartJob”-metodia kutsutaan. Se merkitsee sitä, että ajon aikana tuli sen estäviä ongelmia. ”AsyncTask”-olion tausta-ajo voi olla edelleen käynnissä, kun keskeytys tapahtuu, joten se on peruutettava.

Seuraava vaihe on tehtävän ”ajoittaja” asetus, joka siis käynnistää aiemmin luomamme ”SunshineFirebaseJobService”-olion. Tätä varten käytämme ”FirebaseJobDispatcher”-oliota, joka saadaan ”GooglePlayDriver”-olion kautta. Päivitämme viime vaiheessa luotua ”SunshineSyncUtils”-luokkaa uudella metodilla, joka määrittelee uuden ajoitetun tehtävän. Varsinainen ajoitus tehdään samaan tapaan ”rakentajan” avulla kuin harjoituspuolellakin, kutsumalla ”FirebaseJobDispatcher”-olion ”newJobBuilder”-metodia.

Luomme ajoitusta varten oman uniikin tunnisteen, joka määritellään vakiona merkkijonona nimeltä ”SUNSHINE_SYNC_TAG”. Se toimii suoritettavan palvelun tunnisteena. Muita lisättäviä vakioita ovat mm. muuttujat, jotka määrittelevät, kuinka usein päivitys ajetaan. Käytämme näitä parametreina palvelun rakentajalle, jolta saamme aivan lopuksi varsinaisen ”Job”-olion (”build”-metodilta).

Aivan viimeinen vaihe on ajoittaa palvelun suoritus, antamalla luotu ”Job”-olio parametrina ”FirebaseJobDispatcher”-olion ”schedule”-metodille. Palvelu on nyt valmis, sekä se on ajoitettu!

1.75 S10.04-Exercise-Notifications (2.5h)

Seuraava vaihe on tuen lisääminen ilmoitusten näyttämiseen, jolla ilmoitamme käyttäjälle tietojen päivittyneen. Tehtäväpohjassa on valmis ”NotificationUtils”-luokka, joka on tarkoitus täydentää toimivaksi.

Kuten harjoituksissa, aloitamme homman luomalla uudelle ilmoitukselle uniikin tunnisteen, jolla voimme vaikuttaa näytettyyn ilmoitukseen myöhemmin (vakio kokonaisluku). Liitämme ilmoitukseen ”Intent”-olion, jolla voidaan avata applikaation ”DetailActivity”-näkymä. Haluamme ”DetailActivity”-näkymän toimivan niin, että se siirtyy paluunapilla ”MainActivity”-näkymään. Tämä on tehtävä ”TaskStackBuilder”-olion avulla, jolla muokataan navigoinnin ”kasaa” (”stack”). Ilman kasan muokkausta ohjelma poistuisi paluunapilla kokonaan applikaatiosta (tässä tapauksessa).

Ilmoitusten näyttämisen oletusarvoa (”boolean”) ei kirjoiteta koodin sekaan, vaan siihen käytetään resursseja (”res/values”-kansio). Arvo merkataan ”bools.xml”-nimiseen tiedostoon, samaan tapaan kuin esim. vakio merkkijonot tulisi kirjoittaa ”strings.xml”-tiedostoon. Tiedolle asetetaan nimi (”name”) sekä arvo (XML-elementin sisältö). Tiedon arvo luetaan viittaamalla siihen nimen kautta (esim. ”nayta_ilmoitukset_oletusarvo”).

Käyttäjän asetukset listataan ”pref_general.xml”-tiedostoon, joka on ”res/xml”-kansiossa. Se, tulisiko ilmoituksia näyttää, merkataan tähän tiedostoon uutena ”CheckBoxPreference”-elementtinä. Sen oletusarvo luetaan aiemmin mainitusta ”bools.xml”-tiedostosta. Asetukseen viittaava avain (”key”), asetetun arvon selite (”summaryOff” sekä ”summaryOn”) sekä asetuksen nimi (”title”), luetaan kaikki ”strings.xml”-tiedostosta (resursseina). Asetuksen nykyinen arvo luetaan ”SharedPreferences”-olion kautta (saadaan ”PreferenceManager”-olion avulla).

Varsinainen ilmoitus lähetetään harjoituksista tuttuun tapaan, käyttämällä ”NotificationManager”-järjestelmäpalvelua. Se saadaan ”Context”-olion kautta, kutsumalla ”getSystemService”-metodia. Ilmoitus lähetetään sen jälkeen ”NotificationManager”-olion ”notify”-metodilla.

Nämä vaiheet harjoiteltiin juuri vähän aikaa sitten harjoituspuolen applikaatiossa.

1.76 S11.01-Exercise-NewListItemLayout (2h)

Tässä vaiheessa päivitämme Sunshine-applikaation ulkoasua. Aiemmassa versiossa säät ilmoitettiin yksinkertaisina, tekstimuotoisina riveinä. Nyt haluamme, että rivin alussa on säätä kuvaava ikoni, sekä lämpötilat ovat suuremmalla fontilla aivan oikeassa reunassa. Päivää ja säätä kuvaava teksti sijoitetaan allekkain rivin keskiosaan.

Rivin ulkoasu ositettiin jo aiemmin omaan tiedostoonsa: ”forecast_list_item.xml”. Nyt muutamme sen pohjautumaan ”ConstraintLayout”-elementtiin.

Varsinainen rivin luonti ja näyttäminen suoritetaan listan adapterissa, luokassa ”ForecastAdaptor”. Viimeinen vaihe on muokata adapteri toimimaan uuden ulkoasun kanssa. Ulkoasu on nyt jaettu mm. ”ImageView”- ja ”TextView”-komponentteihin.

1.77 S11.02-Exercise-TodayListItem (2h)

Tässä on ideana muokata listausta niin, että tämän päivän rivi esitetään korostettuna. Muiden päivien rivit pysyvät kuitenkin samanlaisina kuin aiemminkin. Yksinkertaisuudessaan korostetun rivin reunoille jääviä välejä, ikonia ja kaikkia fontteja suurennetaan.

Korostettua riviä varten luodaan uusi ulkoasutiedosto, ”list_item_forecast_today.xml”. Sen pohjalla käytetään aiemmista harjoituksista tuttua ”ConstraintLayout”-ulkoasua.

Korostetun rivin käyttöä ohjataan ”boolean”-tyyppisellä muuttujalla, jonka oletusarvo on kovakoodattu. Lisäämme projektiin omat resurssit pysty- sekä vaakatilalle. Vaakatilan resurssi kirjoitetaan tiedostoon ”res/values/bools.xml”, kun taas tämän ylikirjoittava tiedosto menee paikkaan ”res/values-port/bools.xml”. Kummissakin tiedostoissa on sama ”boolean”-tyyppinen arvo, avaimella ”use_today_layout”. Oletuksena asetamme tämän arvon pystytilassa arvoon ”false”, eli korostettu rivi on pois päältä.

Käytettävän rivin tyyppi palautetaan ”getItemViewType”-metodista. Haemme resursseista aiemmin lisäämämme ”use_today_layout”-arvon. Jos piirrettävä rivi on ensimmäinen, sekä korostetut rivit ovat käytössä, pyytää metodi piirtämään korostetun rivin. Kaikissa muissa tapauksissa pyydetään piirtämään tavanomainen, pienempi rivi.

Viimeinen vaihe on päivittää ”onCreateViewHolder”-metodi. Meidän on muokattava se niin, että se osaa nyt ”getItemViewType”-metodin perusteella ”laajentaa” (”inflate”) oikean rivipohjan (ulkoasutiedostoista).

1.78 S11.03-Exercise-DetailLayoutAndDataBinding (2h)

Nyt on aika päivittää myös ”Details”-näkymän ulkoasu komeampaan muotoon. Tarkoitus on myös muokata se käyttämään ”DataBinding”-kirjastoa tietojen esittämiseen.

”DataBinding”-kirjaston käyttöönotto tehtiin jo aiemmin harjoituspuolen tehtävissä. Homma aloitetaan tässä samaan tapaan, enkä kirjoita siitä uudelleen.

Ulkoasu rakennetaan hyödyntäen ulkoasutiedoston ositusta. Tämä tekniikka harjoiteltiin harjoituspuolen tehtävissä. Sivun yläosa määritellään ”primary_weather_info.xml”-tiedostoon ja alaosan ”extra_weather_detail.xml”-tiedostoon. Muokkaamme vanhan ulkoasun pohjautumaan ”LinearLayout”-elementtiin, jonka sisälle molemmat osat sisällytetään. ”DataBinding”-tuen lisäyksessä poistamme koodista myös kaikki aiemmin käytetyt ”findViewByID”-metodin kutsut.

Viimeinen vaihe on lisätä jokaiseen elementtiin sen sisältöä kuvaava selite (”content description”). Android tukee erilaisia tekniikoita, joilla se auttaa mm. vammaisia ihmisiä käyttämään applikaatioita. Selitteellä voidaan kuvata mm. kuvan sisältöä tai merkitystä, ilman, että kuva on nähtävissä.

1.79 S12.01-Exercise-DimensionsColorsAndFonts (1.5h)

Tämä tehtävä aloittaa Sunshine-applikaation viimeisen osion, jossa päivitetään ulkoasua.

Työ alkaa värien määrittelyllä, jotka lisätään ”res/values/colors.xml”-tiedostoon (kuten harjoituspuolella). Tänne määritellään mm. yleisvärit sekä erilaisille teksteille suunnatut värit (esim. normaali ja korostettu teksti).

Seuraava vaihe on dimensioiden määrittely (mittasuhteet). Ne määritellään samaan tapaan kuin värit, mutta omaan tiedostoonsa (”res/values/dimens.xml”). Kaikki pituudet ja koot ovat kopioitavissa tehtävänannon kautta.

Kolmanneksi muokkaamme kaikkien selitteitä näyttävien ”TextView”-elementtien ”fontFamily”-attribuutin arvoksi ”sans-serif”. Vastaavasti lämpötilaa kuvaavat tekstit tulostetaan ”sans-serif-light”-tyylillä.

Viimeinen ja kaikkein työläin vaihe on osittamiemme arvojen (”colors.xml” ja ”dimens.xml”) käyttöönotto. Muokkaamme näkymistä mm. kovakoodatut välien ja marginaalien arvot käyttämään ”dimens.xml”-tiedostossa määrittämiämme arvoja. Eri osien värit muokataan vastaavasti käyttämään ”colors.xml”-tiedoston vakioita.

1.80 S12.02-Exercise-Styles (2h)

Tällä hetkellä ”Sunshine”-applikaation otsikko rivi käyttää oletustyylittelyä. Sen pohjalta riviin tulostetaan ainoastaan applikaation nimi. Haluamme nyt muokata tämän rivin tyylikkäämmäksi.

Otsikon muokkaus tapahtuu kahden muutoksen kautta. Meidän on ylikirjoitettava ”MainActivity”-näkymässä käytettävä tyyli, joka käyttää ”ActionBar”-osiolle muokattua tyyliä. ”Activity”-näkymän tyyli peritään ”@style/AppTheme”-tyylistä. Otsikko rivi taas pohjautuu ”@style/Widget.AppCompat.Light.ActionBar.Solid.Inverse”-tyyliin.

”MainActivity”-näkymä muokataan käyttämään uutta ”AppTheme.Forecast”-tyyliä ”AndroidManifest.xml”-tiedoston avulla. Tämä tyyli muokattiin käyttämään uutta otsikkorivin tyyliä nimeltä ”ActionBar.Solid.Sunshine.NoTitle”. Nyt ”MainActivity”-näkymä käyttää muokkaamaamme otsikkoriviä.

”Extra weather information”-osion tekstejä varten luotiin oma tyyli (”DetailLabel”), jolla tekstit näytetään korostusvärillä ja ”sans-serif”-tyylisenä. Tyyli periytyi Androidin omaan, otsikoiden ”@style/TextAppearance.AppCompat.Title”-tyyliin.

Viimeisenä vaiheena on vielä erillisen tyylin luominen ”DetailsActivity”-näkymän teksteille. Pohja perittiin tällä kertaa ”@style/TextAppearance.AppCompat.Headline”-tyylistä. Osa teksteistä asetettiin mm. valkoiseksi.

1.81 S12.03-Exercise-TouchSelectors (1h)

Seuraava vaihe on muutaman valitsimen lisäys (”Selector”), joita juuri harjoittelimme harjoituspuolen tehtävissä.

Ensimmäinen on nimeltään ”today_touch_selector”, joka muuttuu ”@color/colorPrimary”-väristä ”colorPrimaryDark”-vakion väriseksi silloin, kun näkymä on valittuna

(”selected”). Tätä valitsinta käytetään ainoastaan tämän päivän rivillä, joka on tyyliltään poikkeava.

Toinen valitsin on nimeltään ”touch_selector”, jota käytetään muilla kuin tämän päivän riveillä. Se pohjautuu Androidin valmiiseen valitsijaan.

Viimeinen vaihe on uusien valitsimien käyttöönotto. Kuten aiemminkin, asetamme valitsimet ulkoasujen ”background”-attribuutin arvoksi. Lista käytti rivien piirtoon ”list_item_forecast_today.xml”- sekä ”forecast_list_item.xml”-ulkoasuja. Asetamme valitsimet näiden ulkoasutiedostojen juurielementtiin (”ConstraintLayout”).

Nyt listan rivit toimivat interaktiivisesti ulkoasunsa perusteella!

1.82 S12.04-Exercise-ResourceQualifiers (1h)

On aika tehdä Sunshine-applikaation viimeinen päivitys!

Tehtävä aloitetaan luomalla omat versiot applikaation ulkoasusta, riippuen laitteen orientaatiosta. Haluamme, että ”Details”-näkymä on vaakatason tilassa (”landscape”) erilainen. Kuten kurssin edetessä neuvottiin, tulisi näytön tila käyttää aina hyvin ja harkiten. Muokkaammekin näkymää nyt niin, että perustiedot listataan näytön vasemmalle puolelle, ja lisätiedot oikealla puolelle. Tämä tehdään yksinkertaisesti lisäämällä projektiin tutulla tavalla uusi ”res/layout-land”-kansio, johon kopioimme alkuperäisen ” activity_detail.xml”-ulkoasun. Muokkaamme kopion vastaamaan uutta ulkoasuideaa.

Ulkoasun lisäksi haluamme tehdä omat versiot käytettävistä mittasuhteista (”dimensions”). Ensiksi luomme uuden ”res/values-land/dimens.xml”-tiedoston, jossa muokkaamme vaaka- ja pystytason välejä (”padding”). Näitä arvoja käytetään ainoastaan silloin, kun laite on vaakatasossa. Toinen tiedosto on ”res/values-sw600dp/dimens.xml”. Tämän tiedoston arvoja käytetään ainostaan silloin, kun käytettävä laite on tabletti.

Sunshine-applikaatio on nyt valmis!

LAAJA HARJOITUSTYÖ

1.83 Idea

Ajatuksenani on luoda yksinkertainen valuuttakurssien muunnon mahdollistava ohjelma. Käyttäjä valitsisi ensin oletusvaluutan, johon muita valuuttoja voisi verrata.

Käyttäisin tässä apuna avointa valuuttakurssien rajapintaa, jonka kautta koittaisin päivittää ohjelman paikallisen tietokannan (jos mahdollista) verkon kautta. Kurssi haettaisiin ainoastaan kerran päivissä, jonka jälkeen se luettaisiin päivän loppuun asti laitteen tietokannasta. Yksinkertaisuudessaan rajapintaan tehdään kysely, joka palauttaa JSON-tyyppisen listan kaikista sen tarjoamista valuutoista ja niiden arvosta suhteessa johonkin valittuun valuuttaan (esim. $).

Ohjelmaa avattaessa se tarkastaa tietokannan tilanteen, ja hakee mahdolliset päivitykset verkon kautta. Tämän jälkeen se avaa uuden sivun, jossa on listattuna oletus- tai käyttäjän viimeksi valitsema valuutta. Tämän alapuolella on kaikki muut valuutat ja niiden arvo suhteessa käyttäjän valitsemaan valuuttaan.

Käyttäjä voi ainoastaan vaihtaa oman oletusvaluuttaansa, eikä muita asetuksia ole.

1.84 Rajapinta valuuttakurssien päivittämiseen

Kulutin jonkin verran aikaa valuuttakursseja jakavien rajapintojen vertailuun. Isoksi haasteeksi muodostui se, että moni palvelu vaati käyttäjätilin rekisteröitymisen. Jokainen käyttäjä saa palvelulta salaisen avaimen, jonka avulla tietoja voidaan ladata. Palvelut siis halusivat jakaa tietylle käyttäjälle vain rajallisen määrän tietoja. Tällä tavoin palvelut estävät mm. rajapinnan väärinkäytön, koska datan käyttö voidaan liittää suoraan tiettyyn käyttäjätiliin.

Koska mobiiliapplikaatio ei saa vaatia käyttäjältä rekisteröitymistä, ei rekisteröintiä vaativaa rajapintaa voida käyttää. Vaikka applikaatiota varten tehtäisiin oma ”jaettu käyttäjätili” (jaettu avain), tulisivat rajapinnan tilikohtaiset siirtorajat vastaan. Tämän vuoksi täysin avoimen rajapinnan löytäminen oli välttämätöntä.

Päädyin etsinnöissäni käyttämään Euroopan Keskuspankin avointa rajapintaa. Se ei vaadi erillistä rekisteröitymistä, eikä nopeiden etsintöjen perusteella rajoita datan lataamista millään tavalla.

1.85 Euroopan keskuspankin rajapinta

Euroopan keskuspankki (ECB) tarjoaa valuuttakursseja muutamassa eri formaatissa. Tarjonta on esiteltävissä tällä kotisivulla:

http://www.ecb.europa.eu/stats/policy_and_exchange_rates/euro_reference_exchange_rates

Aivan sivun alareunassa on osio ”Current reference rates”. Tästä voidaan valita valuuttakurssien lataus eri formaateissa. Tarjolla on:

- PDF (Portable Document Format)
- CSV .zip-tiedostossa (Comma-separated values)
- XML (Extensible markup language)

Valitsin näistä omaan käyttööni mielestäni helpoimman, eli XML-formaatin. Valuuttakurssit saa ladattua XML-formaatissa seuraavasta linkistä:

http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml

1.86 Aloitus

Aloitin projektin tavanomaiseen tapaan, käyttämällä tyhjää applikaatiopohjaa. Annoin ohjelmalleni nimeksi ”Currency Converter”. En kuitenkaan lisännyt ohjelmaa julkiseen versionhallintaan, kuten aiemmissa tehtävissä. Innostuin tämän työn ideoinnista niin paljon, että ajattelin tehdä siitä ensimmäisen vapaa-aikanani tuotetun Android-applikaation. Palautan projektin lähdekoodit erillisesti zip-pakettina sähköpostitse, koska en halua jakaa projektin koodeja ulkopuolisille.

Aiemmista harjoituksista otan tähän projektiin mukaan SQLite-tietokantakirjastoni (”sqlitedatabase”).

1.87 Preferences-kirjasto

Halusin kehittää Androidin ”SharedPreference”-kirjaston ympärille oman pienen kirjastoni, joka jäljittelee edelleen vanhaa ja tuttua, repositorio-kaavaa. Halusin tehdä sen osittain siksi, että pidän alkuperäistä kirjastoa hieman alhaisella tasolla toteutettuna. Esimerkiksi tiedon hakeminen käyttämällä ”String”-tyyppistä avainta joka paikassa on mielestäni hölmöä.

Lyhyesti selitettynä, ”SharedPreference”-kirjastolla on mahdollista tallentaa tietoja käyttäjän laitteeseen ilman tietokantaa. Jokaiselle tallennettavalle tiedolle valitaan tietotyyppi sekä ”String”-tyyppinen avain/nimi. Kirjasto antaa myös syöttää valinnaisen oletusarvon, jota käytetään, kun tietueeseen ei ole vielä koskaan kirjoitettu mitään.

Kirjasto koostuu pääosin kahdesta pääluokasta:

- **PreferenceRepository**
- **PreferenceValue**

Arkkitehtuuri on sellainen, että käyttäjä perii ”PreferenceRepository”-luokasta oman repositorio-luokan. Perityssä luokan konstruktorissa sitten alustetaan kaikki repositorioon kuuluvat kentät (”PreferenceValue”). Kaikki käytettävät kentät on listattuna kansiossa ”preferences\values”. Näistä yksi esimerkkejä on esim. ”StringPreferenceValue”-luokka, jonka avulla tallennetaan merkkijonotyyppistä tietoa käyttäjän laitteeseen. Loin kaikille ”SharedPreference”-kirjaston tukemille tyypeille on omat luokkansa, jonka kautta erityyppiset tiedot voidaan tallentaa. Mahdollisia muita tuettuja ja tallennettavia tyyppejä ovat esimerkiksi ”int”, ”long” ja ”boolean”.

Idea on siis ottaa tämä kirjasto osaksi Android-applikaatiota, ja luoda sitten minimissään yksi repositorio, missä on ainakin yksi kenttä. ”examples”-kansiosta löytyy esimerkit, joissa on luotu repositorio-pankki (”ExamplePreferenceRepositoryBank”), joka voi sisältää monta eri repositoriota. Esimerkin repositorion (”ExamplePreferenceRepository”) sisään on taas listattuna kolme eri kenttää, joista ensimmäinen on ”BooleanPreferenceValue”-tyyppinen kenttä nimeltä ”RememberSelection”. Tämän kentän avulla käyttäjän mobiililaitteeseen voidaan tallentaa ”boolean”-tyyppinen tieto.

Tässä harjoitustyössä kirjastoa käytettiin luomalla projektiin ”Preferences”-luokka, joka toimi ns. ”repositorio-pankkina”. Sen sisällä on listattuna yksi ”User”-niminen repositorio. Tämän repositorion sisälle on listattuna kaikki applikaatiossa käytettävät tietueet. Esimerkiksi ”StringPreferenceValue”-tyyppiseen ”LastInputValue”-kenttään tallennetaan käyttäjän viimeisin syöte. Näin se pysyy käytettävissä, vaikka applikaatio tai koko laite suljettaisiin.

”Preferences”-luokan instanssi asetetaan saataville luokassa ”CurrencyConverterApp”. Aiemman harjoituksen mukaisesti, tämä luokka on siis peritty Androidin alkuperäisestä

”Application”-luokasta, josta luodaan aina ohjelman ensikäynnistyksessä instanssi. Myös muiden kirjastojen instanssit, mm. ”sqlitedatabase”, listataan tähän samaan luokkaan.

1.88 Fontfaces-luokka

Aiempien kokemuksieni perusteella, halusin välttää kuvien käyttämistä applikaatiossani. Syy tähän on se, että Android-käyttöjärjestelmää käytetään niin monella erilaisella laitteella. Osalla laitteista ruudun tarkkuus voi olla hyvinkin huono, kun taas paremmissa malleissa tarkkuus voi olla yli kolminkertainen. Tämä johtaa siihen, että kaikista kuvista on lisättävä monta erikokoista versiota. Matalamman resoluution puhelimet käyttävät sitten pienempiä kuvia, ja tarkemmat mallit taas suurempia.

Perinteisistä kuvista saa siis hyvin nopeasti itselleen varsin kookkaan työmaan, jonka halusin välttää. Mielestäni paljon parempi vaihtoehto kuville on käyttää vektoripohjaista grafiikkaa. Yksi helpoin keino on luoda mahdolliset logot ja kuvakkeet esim. fontin avulla. Uutisten mukaan myös Androidin uusimmat versiot tukisivat ”SVG”-tyyppisiä vektorikuvia. Halusin kuitenkin taata yhteensopivuuden vanhemmille versioille, joten päädyin fonttien käyttämiseen.

Latasin käyttööni ilmaisen ”Font Awesome”-fontin, joka on yleisesti hyvin suosittu ja käytetty. Se sisältää liudan erilaisia logoja, joita voi käyttää aivan vapaasti myös maksullisissa sovelluksissa. Fontista voi lukea lisää sen kotisivuilta:

https://fontawesome.com

Kopioin fontin ”tff”-tyyppiset tiedostot oman ”assets\fonts”-kansion alle. Jouduin myös uudelleennimeämään ne, koska ”assets”-kansion tiedostot eivät saa sisältää mm. välilyöntejä tai viivoja.

Vaikka fontit oli nyt tuotu osaksi projektia, ei niitä voida suoraan käyttää. Tämän vuoksi loin uuden apu-luokan ”Fontfaces”, jonka tehtävä on tarjota fontit ”Typeface”-tyyppisinä olioina. Kaikki käytettävät ”Typeface”- oliot luodaan luokan konstruktorissa, jonka jälkeen ne voidaan lukea ”gettereillä”. Esim. ”fa-regular-400”-fontti voidaan lukea metodilla ”fa_regular_400()”.

Luokan instanssi listataan normaaliin tapaan ”CurrencyConverterApp”-luokan sisälle, josta se on saatavilla mistä tahansa.

Esimerkiksi ”TextView”-olio voidaan nyt vaihtaa käyttämään ”fa-regular-400”-fonttia, kun se asetetaan aktiviteetin ”onCreate”-metodissa seuraavasti:

```
this.TextView_logo = this.findViewById(R.id.TextView_logo);
this.TextView_logo.setTypeface(CurrencyConvert-
erApp.Fontfaces.fa_solid_400());
```

1.89 Retrofit-kirjasto

Koska ECB jakaa valuuttakurssit XML-formaatissa, piti minun löytää sopiva kirjasto tämä tiedon lataamiselle ja parsimiselle. Nopealla googletuksella vastaani tuli ”Retrofit”-niminen kirjasto. Sen toiminta perustuu pääosin tiedon lataamiseen, jonka jälkeen tieto parsitaan haluttuun olio-muotoon. Aivan oletuksena sitä käytetään JSON-tyyppisten merkkijonojen parsintaan, josta myös löytyi liuta tutoriaaleja ja ohjeita. XML-tyyppisen tiedon parsimisesta tietoa oli kuitenkin tarjolla niukemmin.

Jokaista XML-kyselyä varten on luotava oma ”Interface”-rajapintansa, jota vasten kirjasto luo varsinaisen olion. Tässä projektissa oli vain yksi kysely, joka palauttaa monen valuuttakurssin tiedot. Tätä kyselyä vastaava rajapintamääritelmä löytyy tiedostosta ” IEuropeanCentralBank”. Tämän tiedoston laatimiseen meni todella kauan, koska ohjeita sen luontiin ei tuntunut löytyvän mistään. Sain kuitenkin haalittua tiedon pienistä osista ja eri paikoista, jonka pohjalta sain sen valmiiksi.

Lyhyesti määriteltynä, rajapintaan on listattu luokat jokaista XML-tiedoston elementtiä varten. Valuutat annetaan esimerkiksi ”Cube”-nimisen elementin sisällä seuraavassa formaatissa:

```
<Cube currency="USD" rate="1.1870"/>
```

Tämän pohjalta rajapinnasta on siis löydyttävä luokka, joka on merkattu annotaatiolla:

```
@Element(name = "Cube")
```

Tämän lisäksi määritellyn luokan sisältä on löydyttävä ”currency” ja ”rate”-attribuuteille omat muuttujat. Ne merkataan seuraavasti:

```
@Attribute(name="time", required=false)
public String time;
@Attribute(name="currency", required=false)
public String currency;
@Attribute(name="rate", required=false)
public double rate;
```

Kokonaisuudessaan koko parsittavan XML-tiedon rakenne on oltava täysin samanlainen tämän rajapinnan kanssa. Muussa tapauksessa kirjasto ilmoittaa virheellisestä määritelmästä. Tarkasteltaessa voidaan huomata, kuinka sisäkkäiset XML-elementit on määritelty rajapinnassa sisäisinä ja staattisina Java-luokkina. Jokaiselle elementille on määritelty oma luokka sekä attribuutille jäsenmuuttuja.

Tätä rajapintaa nyt hyödyntäen, saamme lopulta ohjelman ajossa käyttöömme ”Envelope”-olion, joka sisältää viitaukset kaikkiin sen sisältämiin asioihin (mm. attribuutit, elementit. elementtien sisältämät elementit jne). Parsittu olio on siis ns. puu, joka esittää XML-tiedon olio-muodossa.

1.90 SQLiteDatabase-kirjasto

Kuten aimmin mainitsin, käytin tässä harjoituksessa jo aiemmassa harjoituksessa kehittämääni tietokantakirjastoa. Jatkoin kuitenkin sen kehitystä, koska en lopulta ollut tyytyväinen sen ”callback”-rakenteeseen. Rakensin siis rungon uudelleen ”SQLite”-tyyppiseen tapaan ”synkroonisesti”. Muutin myös avaimen tyypin merkkijonosta ”long”-tyyppiseksi. Alun perin idea oli siis ylläpitää yhtenäistä ”rajapintaa” sekä ”Firebase”- että ”SQLite”-tietokannan käytössä.

Kirjasto toimii muuten suurilta osin täysin samoin kuin aiemmin. Tämän projektin tietokannassa on vain yksi repositorio ”CurrencyReferenceRepository”, jolla tehdään hakuja vain yhdestä taulusta ”CurrencyReference”. Tähän ainoaan tauluun tallennetaan kaikki valuuttakurssit, kun ne päivitetään internetistä.

”CurrencyReference”-oliossa on ylikirjoitettuna ”toString()”-metodi, joka palauttaa valuutan koodinimen (esim. ”EUR”). Tätä käytetään hyödyksi mm. silloin, kun valuutat listataan ja näytetään alasvetovalikkossa.

Toinen lisätty ominaisuus on ”Comparable<CurrencyReference>”-rajapinnan implementointi, jolla eri valuutat voidaan järjestää listassa aakkosjärjestykseen. Järjestämiseen käytetään Javan perinteistä ”Collections.sort()”-metodia.

Viimeisimpänä on tottakai ”Parcelable”-rajapinnan toteutus, joka takaa olioiden helpomman siirron aktiviteetista toiseen. Käytin tässä apuna ”Android Studio”-ohjelman laajennusta nimeltä ”Android Parcelable code generator”. Se generoi ja implementoi ”Parcelable”-rajapinnan ”boilerplate”-koodit muutamalla klikkauksella.

1.91 GetCurrencyReferences-luokka

Halusin erottaa valuuttakurssien hakemisen omaan luokkaansa. Idea on, että valuuttakurssien lukeminen tapahtuu aina tämän rajapinnan kautta. Varsinainen ohjelma ei siis tiedä, mistä tiedot on oikeasti haettu. Se vain vastaanottaa listan valuuttakursseja.

Yksinkertaistettuna rajapinnalle annetaan parametrina ”ResultListener”-tyyppinen ”callback”-olio. Sillä on yksi toteutettava metodi ”onComplete”, joka saa parametrina ”OperationResult”-tyyppisen olion. Olio sisältää mm. tiedon, onnistuiko haku vai ei. Tämä tieto ilmaistaan ”Result”-tyyppisellä enumeraatio-jäsenmuuttujalla:

- **Result.CANNOT_GET_ANYTHING:** Valuuttakursseja ei saada mitenkään. Internet-yhteys on ainoa vaihtoehto.

- **Result.USING_OLD_INFORMATION:** Valuuttakurssit ovat vanhoja, ja ne ladattiin sisäisestä tietokannasta. Internet-yhteys tarvitaan kurssien päivittämiseen.

- **Result.USING_LATEST:** Viimeisimmät valuuttakurssit ovat käytössä. Internet-yhteyttä ei tarvita.

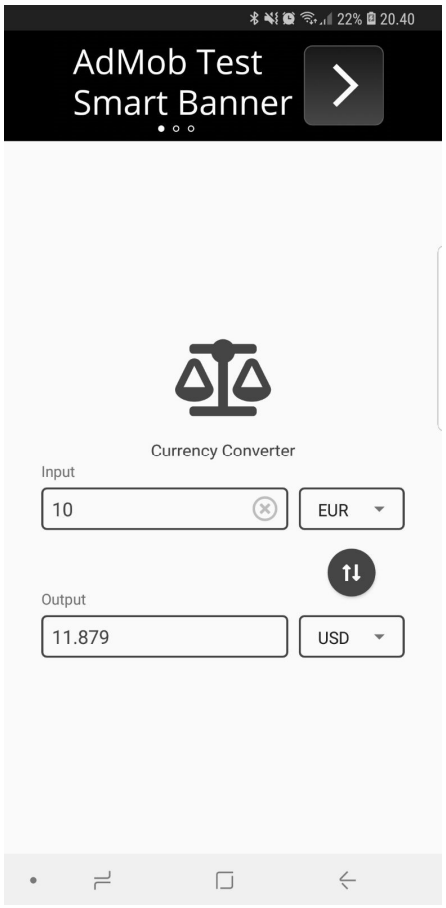
Rajapinnan logiikka vertaa haettaessa viimeistä päivitysaikaa ja nykyhetkeä toisiinsa. Viimeisin päivitysaika tallennetaan laitteen muistiin aiemmin mainitun ”Preferences”-kirjaston avulla. Jos tietoja ei ole koskaan päivitetty, on internet-yhteys pakollinen ohjelman toiminnan kannalta. Jos tiedot ovat taas vanhentuneita, siitä ilmoitetaan käyttäjälle varoituksena. Applikaatio on siis ainoastaan silloin käyttökelvoton, kun se ei ole koskaan saanut päivittää valuuttakursseja.

Jos sisäiset tiedot katsotaan puuttuviksi tai vanhentuneiksi, rajapinta yrittää hakea uusimmat kurssit ECB:n rajapinnan kautta. Onnistuessaan, sisäisen tietokannan ”CurrencyReference”-taulu ylikirjoitetaan uusilla kursseilla. Epäonnistuessaan rajapinta palauttaa taas listan vanhoja valuuttakursseja (tai pahimmassa tapauksessa tyhjän listan, jolloin applikaatiota ei voi käyttää).

”GetCurrencyReferences”-luokka periytyy ”AsyncTask”-luokasta, jonka avulla voidaan suorittaa koodin ajoa asynkroonisesti taustalla. Varsinainen logiikka on listattuna ylikirjoitettuun ”doInBackground”-metodiin, ja varsinainen parametrina annettu ”callback”-olio ajetaan ylikirjoitetussa ”onPostExecute”-metodissa.

Varsin ongelmallisen rakenteesta tekee se, että varsinainen logiikka ajetaan asynkroonisesti. Tämän lisäksi ”Retrofit”-rajapinnan kautta tehtävät web-kyselyt ajetaan asynkroonisesti. Tähän lisättynä kaikki mahdolliset virhetilanteet, ei ole ihme, että palikan kehittämiseen meni reilusti yli viikko.

1.92 Oletusnäkymä



Kuva 5. *Oletusnäkymä on tyypiltään yksinkertainen mutta voimakas. Sen on tarkoitus olla niin helppokäyttöinen, että erillisiä ohjeita ei tarvita.*

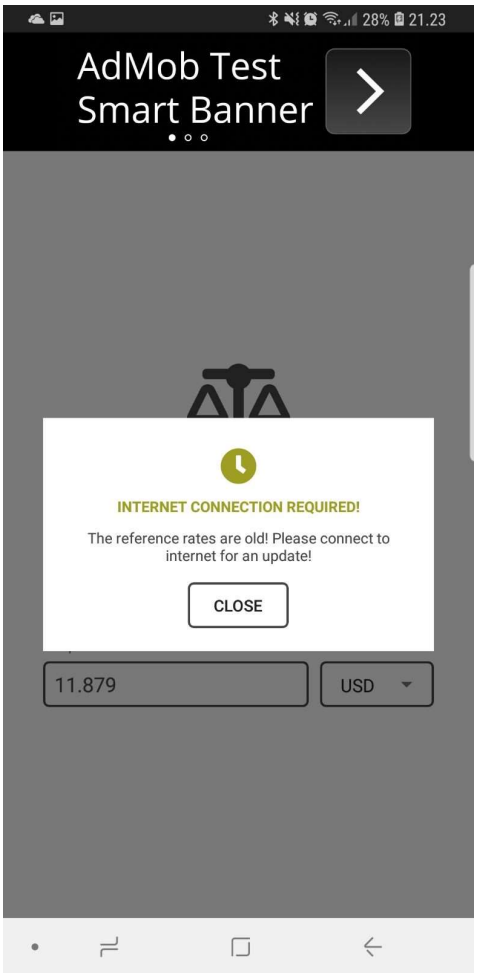
Ulkoasu koostuu pääosin kolmesta osasta: syötekentästä, vaihtonapista sekä tulostuskentästä. Aivan näkymän yläreunaan on varattu pieni alue Googlen AdMob-mainoksia varten. Tekstikenttien vieressä oikealla on valuuttojen valitaan tarkoitetut alasvetovalikot.

Käyttäjän on tarkoitus ohjelman avauksen jälkeen valita käyttämänsä valuutat. Kuvassa käyttöön on valittu lähtövaluutaksi ”EUR”, eli euro. Muuntovaluutaksi on valittu taas ”USD”, eli Yhdysvaltain dollari. Kuvassa on muunnettu annetut 10 euroa 11.879 dollariksi.

Painamalla tummaa vaihtonappia alasvetovalikoiden välissä, ohjelma vaihtaa yllä sekä alla olevan valuutan keskenään. Eli tässä tapauksessa nappia painettaessa muunnettaisiin 10 dollaria euroiksi.

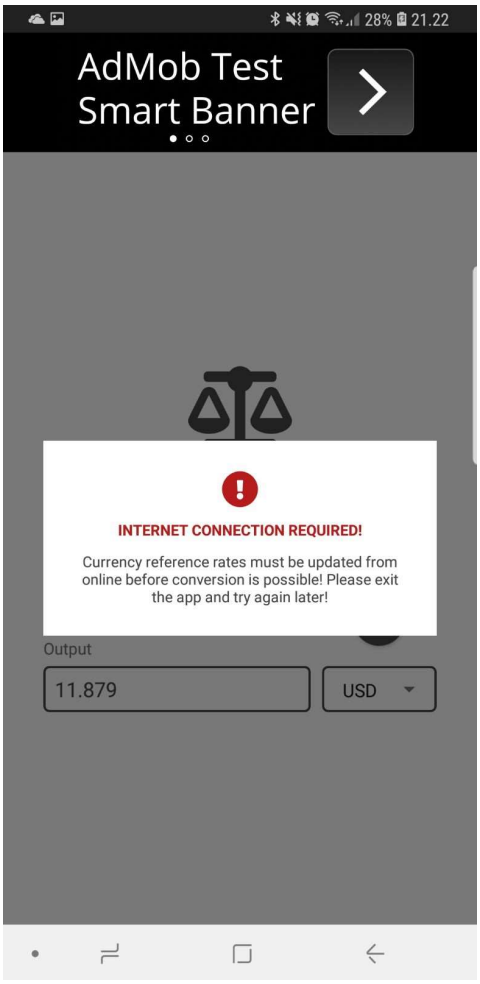
Alempi tekstikenttä, johon muunnettu luku kirjoitetaan, on ns. ”read-only”-kenttä. Käyttäjä ei siis voi kirjoittaa kenttään mitään. Ainoastaan yllä oleva kenttä on syöttöä varten.

1.93 Virhenäkymät



Kuva 6. *Käyttäjälle näytetään varoitus, jos uusimmat valuuttakurssit eivät ole käytettävissä.*

Jos laitteessa ei ole käytössä internet-yhteyttä, tai uusimpia valuuttakursseja ei saada päivitettyä, näytetään varoitus-dialogi. Tämän dialogin on tarkoitus varoittaa käyttäjää, koska applikaation valuuttakurssit voivat olla vanhentuneita, eikä niihin voi luottaa. Käyttäjällä on kuitenkin mahdollisuus sulkea dialogi, sekä jatkaa applikaation käyttöä vanhojen valuuttakurssien mukaisesti.



Kuva 7. *Jos valuuttakursseja ei saada mistään, käyttäjä ei voi käyttää applikaatiota.*

Harvinaisin virhetilanne on se, että käyttäjä avaa ohjelman ensimmäisen kerran niin, että applikaatio ei ole koskaan pystynyt päivittämään valuuttakursseja. Tällöin ohjelmaa ei voi käyttää lainkaan. Silloin käyttäjälle näytetään yllä oleva varoitus, jota ei ole mahdollista ohittaa. Sovellus on tässä tapauksessa vain suljettava.

1.94 Google AdMob

Halusin applikaation tuottavan mahdollista pikkurahaa, joten lisäsin oletusnäkymän yläreunaan keskikokoisen bannerin. Valitsin mainosten käyttämiseen Googlen AdMob-palvelun, joka on erittäin helppokäyttöinen.

AdMob-palvelun kotisivu löytyy täältä:

https://www.google.com/admob/

1.94.1 Rekisteröityminen

Palvelun käyttö aloitetaan normaaliin tapaan rekisteröitymällä. Käytin tässä omaa henkilökohtaista Google-tiliäni, jota tulen käyttämään myös applikaation lisäämiseen Googlen Play-kauppaan.

Palvelun käyttöönotto vaatii rekisteröitymisen, jossa annetaan mm. omat henkilötiedot sekä mahdollinen luottokortin numero. Luottokortin lisäys on mielestäni valinnaista, kunnes mahdolliset tuotot haluaa nostaa käteen. Alimpana nostorajana on kuitenkin 70$. Itse lisään omat luottotietoni vasta sitten, kun tienaamani summa ylittävää nostorajan.

Ennen käyttöönottoa, on myös listattava oman applikaation nimi ja alusta. Valitsin näihin siis ”Android” ja ”Currency Converter”.

1.94.2 Sovellustunnus

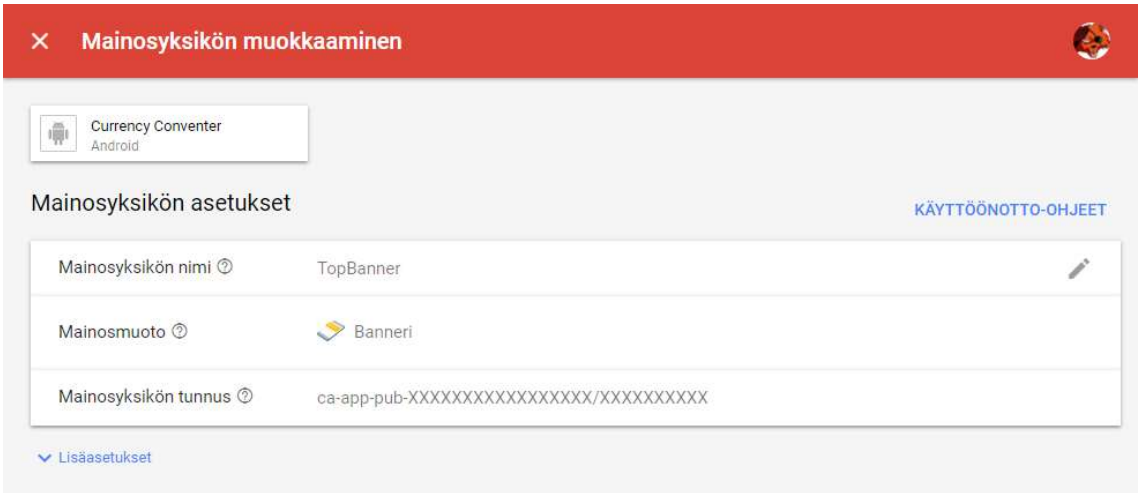
Käyttöönotto alkaa AdMob-sovellustunnuksen luomisella. Tunnus on ohjelma- ja alustakohtainen. Loin tässä kohtaa siis sovellustunnuksen aiemmin listaamalleni applikaatiolle (”Android” ja ”Currency Converter”).

```
ca-app-pub-XXXXXXXXXXXXXXXX~XXXXXXXXXX
```

Ohjelma 10. *Esimerkki AdMob-sovellustunnuksesta*

1.94.3 Mainosyksiköt

Sovellustunnuksen saamisen jälkeen, voidaan aloittaa mainosten lisäys. Palvelu antaa valittavaksi monia erilaisia mainostyyppejä, joista otin käytettäväkseni ”Banneri”-mainosmuodon.



Kuva 8. *AdMob-palvelun ohjauspaaneli, jossa muokataan yksittäistä mainosyksikköä. Applikaatiossa on vain yksi mainos, ja sitä hallitaan tämän sivun kautta.*

Mainosyksikön lisäämisen jälkeen saadaan uniikki ”mainosyksikön tunnus”. Tämä tunnus tulee liittää varsinaiseen mainoselementtiin, joka lisätään näkymän ulkoasutiedostoon.

1.94.4 Käyttöönotto

Mainoksen käyttöönotto aloitetaan AdMob-palvelun alustuksella, jossa annetaan parametriksi oma sovellustunnus.

```
// Google ADs
MobileAds.initialize(this, "ca-app-pub-XXXXXXXXXXXXXXXX~XXXXXXXXXX");
```

Ohjelma 11. *AdMob-palvelun alustus oletusnäkymän “onCreate”-metodissa.*

Tämän jälkeen XML-ulkoasusta on haettava lisätty ”AdView”-näkymä, johon mainos ladataan. Mainos ladataan näkymään seuraavasti:

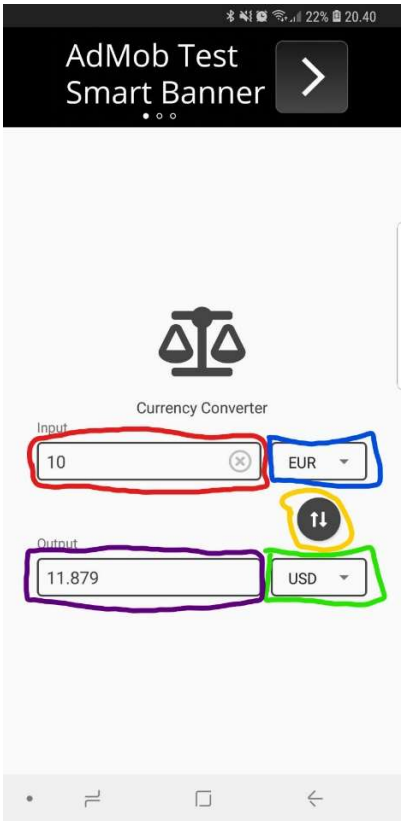
```
this.mAdView = findViewById(R.id.adView);
mAdView.loadAd(new AdRequest.Builder().build());
```

Ohjelma 12. *Mainoksen lataus “AdView”-elementtiin.*

XML-ulkoasun “AdView”-elementissä on muistettava määrittää ” ads:adUnitId”-attribuutti, johon laitetaan aiemmin saatu ”mainosyksikön tunnus”.

1.95 Applikaation käyttöohjeet

Aiemmissa kappaleissa on jo mainittuna applikaation ainoan oletusnäkymän toiminta kokonaisuudessaan. Listasin tähän kuitenkin vielä todella yksinkertaiset ohjeet applikaation käyttöön.



Kuva 9. *Applikaation toimminnot korostettuna.*

1. Varmista että puhelimessasi on toimiva internet-yhteys
2. Avaa ”Currency Converter” sovellus ensimmäisen kerran
3. Odota ja anna applikaation ladata ja päivittää käytettävät valuuttakurssit
4. Valitse Kuvassa 8. sinisellä merkattuun alasvetovalikkoon lähdevaluutta (esim. ”EUR”).
5. Valitse Kuvassa 8. vihreällä merkattuun alasvetovalikkoon vientivaluutta, mihin lähdevaluutta muunnetaan (esim. ”USD”).
6. Kirjoita kuvassa 8. punaisella merkattuun tekstikenttään rahamäärä, jonka haluat muuntaa. Voit tyhjentää tekstin painamalla oikeaan reunaan ilmestyvää ”x”-symbolia.
7. Raha muunnetaan suoraan kirjoittaessa kuvan 8. alapuolella violetilla merkattuun tekstikenttään.
8. Painamalla alasvetovalikoiden välissä olevaa nappia (kuva 8. merkattu keltaisella), voit vaihtaa valitut valuuttatyypit keskenään (esim. ”EUR” <-> ”USD”)
9. Applikaatio tallentaa automaattisesti tekemäsi valinnat. Tiedot säilyvät, vaikka sulkisitkin applikaation käytön välissä.

1.96 Loppuajatukset

Innostuin tosiaan tästä laajemmasta harjoitustyöstä enemmän, kuin olisi ehkä pitänyt. Käytin sen tekemiseen yhteensä varmaan 3-4kk aikaa. Loppujen lopuksi siitä jäi käteen kuitenkin erittäin paljon uusia oppeja ja juttuja. Koin, että se kehitti itseäni hyvin paljon Android-kehittäjänä. Työstä teki mielenkiintoisen varsinkin se, että se koostuu niin monista eri palasista (mm. ”ECB”-rajapinta, ”AdMob”, ”Retrofit” jne). Käytetyt kirjastot ovat sen verran suosittuja, että niiden osaaminen on erittäin järkevää.

Loppusanoinani haluan kiittää tätä mobiiliohjelmoinnin kurssia. Eri tehtävissä tuli koluttua Androidin perusasioita erittäin mukavasti. Kiitos!

YHTEENVETO

Tämä dokumentti sisältää kaiken dokumentaation mobiili-ohjelmointi kurssiini liittyen. Mukana ovat niin opettajan antamat harjoitukset, kuin myös Udacity ”Developing Android Apps”-verkkokurssi.

LÄHTEET

[1] Stack Overflow. (11.5.2018). Saatavissa: https://stackoverflow.com/

[2] Retrofit. (11.5.2018). Saatavissa: http://square.github.io/retrofit/

[3] Vogella, Using Retrofit 2.x as REST client. (11.5.2018). Saatavissa: http://www.vogella.com/tutorials/Retrofit/article.html

[4] European Central Bank (ECB). (11.5.2018). Saatavissa: http://www.ecb.europa.eu/stats/policy_and_exchange_rates/euro_reference_exchange_rates/

[5] Google AdMob. (11.5.2018). Saatavissa: https://www.google.com/admob/

[6] Google Mobile Ads SDK for Android, Get Started. (11.5.2018). Saatavissa: https://developers.google.com/admob/android/quick-start

[7] Fonticons Inc, Font Awesome. (11.5.2018). Saatavissa: https://fontawesome.com

[8] Developer Guides (Android). (11.5.2018). Saatavissa: https://developer.android.com/guide/

[9] Tutorials Point, Android. (11.5.2018). Saatavissa: https://www.tutorialspoint.com/android/index.htm

[10] Ray Wenderlich, Android Development Tutorials. (11.5.2018). Saatavissa: https://www.raywenderlich.com/category/android

[11] Oracle, The Java Tutorials. (11.5.2018). Saatavissa: https://docs.oracle.com/javase/tutorial/

[12] W3Schools, XML Tutorial. (11.5.2018). Saatavissa: https://www.w3schools.com/xml/

[13] Ravi Tamada, Android Layouts: Linear Layout, Relative Layout and Table Layout. (11.5.2018). Saatavissa: https://www.androidhive.info/2011/07/android-layouts-linear-layout-relative-layout-and-table-layout/

[14] Simple XML serialization. (11.5.2018). Saatavissa: http://simple.sourceforge.net

[15] Google Developers, Google I/O 2010 - Android REST client applications. (11.5.2018). Saatavissa: https://www.youtube.com/watch?v=xHXn3Kg2IQE

[16] Lars Vogel, Introduction to background processing in Android – Tutorial, Version 3.5, 14.9.2017. (11.5.2018). Saatavissa: http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html

[17] Ali Muzaffar, 8 ways to do asynchronous processing in Android and counting, 28.5.2015. (11.5.2018). Saatavilla: https://android.jlelse.eu/8-ways-to-do-asynchronous-processing-in-android-and-counting-f634dc6fae4e

[18] Udhay, Android Restful Webservice Tutorial – Introduction to RESTful webservice – Part 1, 1.5.2014. (11.5.2018). Saatavilla: http://programmer-guru.com/android-tutorial/android-restful-webservice-tutorial-part-1/

[19] Debut Infotech, Top 8 Tips & Tricks for Android App Development. (11.5.2018). Saatavilla: http://www.businessofapps.com/top-8-tips-tricks-for-android-app-development/

[20] Aritra Roy, How to become a better Android developer: 30+ bite-sized pro tips. (11.5.2018). Saatavissa: https://techbeacon.com/how-become-better-android-developer-30-bite-sized-pro-tips

[21] Wikipedia, ISO 4217. (11.5.2018). Saatavilla: https://en.wikipedia.org/wiki/ISO_4217

[22] XE.com Inc, XE Currency Converter. (11.5.2018). Saatavilla: https://www.xe.com/currencyconverter/

[23] XE.com Inc, ISO 4217 Currency Codes. (11.5.2018). Saatavilla: https://www.xe.com/iso4217.php

[24] Oanda, Currency Converter. (11.5.2018). Saatavilla: https://www.oanda.com/currency/converter/

[25] Travelex, Currency converter. (11.5.2018). Saatavilla: https://www.travelex.com/currency-converter

[26] Lars Vogel, Google Maps Android API v2 – Tutorial, 30.6.2016. (15.5.2018). Saatavilla: http://www.vogella.com/tutorials/AndroidGoogleMaps/article.html

[27] Google Developers, Get Started. (15.5.2018). Saatavilla: https://console.cloud.google.com/google/maps-apis