OULUN AMMATTIKORKEAKOULU

Valtteri Juutilainen

MOBIILI- JA HYBRIDIOHJELMOINTI

Mobiili- ja hybridiohjelmointi ja niiden vertailu

MOBIILI- JA HYBRIDIOHJELMOINTI

Mobiili- ja hybridiohjelmointi ja niiden vertailu

Valtteri Juutilainen
Opinnäytetyö
Syksy 2021
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Valtteri Juutilainen
Opinnäytetyön nimi: Mobiili- ja hybridiohjelmointi. Mobiili- ja hybridiohjelmointi ja niiden vertailu
Työn ohjaaja: Eino Niemi
Työn valmistumislukukausi ja -vuosi: Syksy 2021 Sivumäärä: 29

Ohjelmistokehitys on jatkuvasti muuttuva ala ja yhtenä sen alaosana on mobiiliohjelmistokehitys. Yhtenä uutena muutoksena mobiiliohjelmistokehityksessä on hybridiohjelmointi. Sillä tarkoitetaan ohjelmistokehitystä, jossa yhdellä koodikannalla luodaan sovellus, joka kääntyy usealle eri alustalle. Hyötynä on se, että säästetään paljon resursseja, kun huolettavana on vain yksi koodikanta usean sijasta. Osana tätä opinnäytetyötä käsitellään mobiiliohjelmointia sekä tutustutaan hybridiohjelmointiin. Käydään hieman läpi niiden eroavaisuuksia, hyötyjä, haittoja sekä hieman sitä, kuinka hyödyllinen hybridiohjelmointi on natiiviin mobiiliohjelmointiin verrattuna sekä yksittäiselle henkilölle että suuremmillekin yrityksille.

Opinnäytetyö tehtiin Oulun ammattikorkeakoululle (Oamk) ja sen ensimmäisenä tavoitteena olikin Oamkin mobiiliohjelmointikurssin opetusmateriaalin kääntö Java-kielestä Kotlin-kieleksi, jonka mukana syvennettiin mobiiliohjelmoinnin tietoja ja taitoja. Java-kielen käyttö Android-ympäristössä ei ole enää Googlen suosittelemaa, vaan suositellaan Kotlin-kielen käyttöä, sillä sen tarkoituksena on korvata Java.

Opetusmateriaalin käännön jälkeen käytiin läpi Flutter-hybridiohjelmointiin liittyvä opetusmateriaali, jonka avulla opeteltiin Flutterin käyttäminen hybridiohjelmoinnissa. Sitä seurasi valmiina olevan harjoitusmateriaalin tarkistus, jonka jälkeen pohdittiin lisämateriaalin luomista.

Opetusmateriaalin valmistuttua, tutustuttiin hybridiohjelmointiin lisää useita eri lähteitä käyttäen, tarkoituksena oppia hybridiohjelmoinnin (Flutterin) hyödyistä ja haitoista sekä yleisesti siitä, kenelle Flutter on tarkoitettu.

Asiasanat: mobiiliohjelmointi, hybridiohjelmointi, Kotlin, Flutter

ABSTRACT

Oulu University of Applied Sciences
Information and Communication Technologies, Software development orientation option

Author: Valtteri Juutilainen
Title of thesis: Mobile and Hybrid App Development. Mobile and Hybrid App Development and their differences
Supervisor: Eino Niemi
Term and year when the thesis was submitted: Fall 2021
Number of pages: 29

Software development is a continuously changing industry. In mobile software development one of the large changes in recent times has been hybrid software development, in which only one codebase is created. Then that one codebase is compiled to be used for multiple platforms. As a gained benefit, resources are saved, when there is only one codebase to be taken care of. As a part of this thesis, we process things about mobile development and learn more about hybrid programming. The thesis processes differences, positives and negatives of both native mobile programming and hybrid programming.

This thesis was created for OAMK and it's first objective was to recreate the Java mobile programming course with the Kotlin language. The second objective was to learn more about Flutter hybrid programming and to go through the course material and check if changes are necessary. Further learning was conducted afterwards to learn more about hybrid programming (Flutter) and when and to who it is good for.

Keywords: mobile programming, hybrid programming, Kotlin, Flutter

SISÄLLYS

1 JOHDANTO

Ohjelmistokehityksessä on tärkeää, että opetus pysyy mukana alan kehityksessä, joten Oulun ammattikorkeakoulu jatkuvasti uudistaa opiskelijoille suunnattuja kursseja. Osana tätä opinnäytetyötä uudistetaan vanhaa sekä luodaan uutta opetusmateriaalia nykyisille ja tuleville ohjelmistokehityskursseille.

Opinnäytetyön tarkoituksena on tarkastella ja oppia lisää nykyisistä Oamkin opetetuista mobiiliohjelmointikielistä ja yhdestä uudesta kielestä sekä hieman arvioida niiden hyödyllisyyttä ja tulevaisuutta. Oamkin mobiilikehitykseen kuuluu tällä hetkellä Swift-ohjelmointikieli iOS-pohjaisille laitteille sekä Java-ohjelmointikieli Android-pohjaisille laitteille. Opinnäytetyön yhtenä tarkoituksena uudistetaan nykyinen Java-kurssin opetusmateriaali korvaavalle Kotlin-ohjelmointikielelle, jonka jälkeen tutustutaan ja luodaan mahdollista opetusmateriaalia hybridiohjelmointiin kuuluvaan ohjelmistoalustaan nimeltään Flutter, jossa käytössä on ohjelmointikielenä Dart.

Tässä työssä käydään yleisesti läpi, mitä mobiiliohjelmoinnilla tarkoitetaan, ja siihen liittyviä ohjelmointikieliä sekä niiden tämänhetkinen ja tulevaisuuden tila sekä kerrotaan niihin luoduista opetusmateriaaleista. Ohjelmistokehitys on jatkuvasti kehittyvä ala, joten osana tätä opinnäytetyötä tarkastellaan myös jo tällä hetkellä kasvavaa muutosta, hybridiohjelmointia, jossa kehitetään sovelluksia useille eri alustoille yhdellä ohjelmointikielellä, jolloin säästetään resursseja. Hybridiohjelmoinnilla voi olla tulevaisuudessa vahva vaikutus ohjelmistokehityksen tehokkuuteen, kun on tarkoituksena kehittää sovelluksia useammille alustoille.

On huomioitava, että koska opinnäytetyötä ei tehty tilauksena yritykselle, työhön ei saatu mukaan täysin varmistettua tietoa esimerkiksi hybridiohjelmoinnin taloudellisista hyödyistä, joten kyseiset tiedot perustuvat lähteisiin tai omaan mielipiteeseeni ja käsitykseeni asiasta.

2 MOBIILIOHJELMOINTI

Opinnäytetyön mobiiliohjelmointiosio keskittyy Android-alustan puhelinten mobiilikehitykseen, joten tässä työssä ei käydä läpi kovinkaan tarkasti muita alustoja, kuten esimerkiksi iOS ja sen ohjelmointikieltä Swiftiä. Tässä osiossa esitellään lyhyesti muutamia ohjelmointikieliä, jonka jälkeen seuraavassa osiossa jatketaan tehtyyn työhön liittyvien ohjelmointikielten käsittelyä.

2.1 Mitä mobiiliohjelmoinnilla tarkoitetaan?

Mobiiliohjelmoinnilla tarkoitetaan mobiilialustoille eli helposti mukaan otettaville laitteille, kuten puhelimille, tableteille tai älykelloille, suunnattua ohjelmistokehitystä. Yleensä useat eri mobiililaitteet, vaikka ne olisivat käyttötarkoituksiltaan erilaisia, käyttävät samoja käyttöjärjestelmiä, kuten Googlen Android, jota käytetään tableteissa, puhelimissa sekä älykelloissa, jotta niiden toimivuus toistensa kanssa olisi sujuvampaa ja jotta niiden luominen ja ylläpito olisi helpompaa ohjelmistokehittäjille.

Tämänhetkinen tilanne mobiiliohjelmoinnissa käyttöjärjestelmien kannalta on varsin selkeä. Lähes jokainen mobiililaite käyttää joko Android- tai iOS-käyttöjärjestelmää. Sen takia ohjelmistokehittäjät sekä yritykset voivat keskittyä joko yhteen tai molempiin kyseisistä käyttöjärjestelmistä, jonka ansiosta he voivat tavoittaa yli 99 % mobiilikäyttäjistä. (Statcounter 2021; Statista 2021a).
Mobiiliohjelmoinnissa voidaan käyttää useita eri ohjelmointikieliä, mutta eräinä tärkeimpinä ohjelmointikielinä on Androidilla Kotlin ja Java sekä iOS:llä Swift.

2.2 Java

Java on Sun Microsystemsin (nyt Oracle Corporationin omistama) vuonna 1995 kehittämä olio-ohjelmointikieli, joka kääntyy JVM-tavukoodiksi (Java Virtual Machine -tavukoodiksi), jonka suorittaminen onnistuu millä tahansa laitteella, joka tukee JVM:ää. Java Virtual Machinella tarkoitetaan virtuaalikonetta, joka suorittaa sille käännettyjä ohjelmia. Kyseinen virtuaalikone toimii käytetyn laitteiston käyttöjärjestelmän päällä, jonka ansiosta käytetyllä laitteella ei ole mitään väliä, kunhan kyseinen laite tukee JVM:ää. (Hartman 2021).

Vaikka Javan suosio on laskenut kovasti jatkuvien tietoturvariskien takia, se on edelleen maailman käytetyin ohjelmointikieli laajan tukevan laitevalikoiman takia, joka on Oraclen viimeisimmän luvun mukaan 15 miljardia Javaa tukevaa laitetta (Oracle 2016). Tämän lisäksi Oraclen hieman uudemman luvun mukaan maailmassa on 51 miljardia aktiivista JVM:ää (Gupta 2020).
Java oli ja toki edelleen on tärkeä osa Android-ohjelmistokehitystä, sillä aluksi Java oli Android-alustan ainoa ohjelmointikieli, jolla voitiin luoda uusia käyttäjien tekemiä sovelluksia. Javan käyttämistä Android-sovellusten kehittämisessä voidaan edelleen toteuttaa, mutta vuodesta 2019 eteenpäin Google on suositellut Javan korvaavaa ohjelmointikielen, Kotlinin käyttöä ohjelmistokehityksessä (Lardinois 2019).

2.3 Kotlin

Kotlin on JetBrainsin kehittämä monialustainen ohjelmointikieli, joka kääntyy Javan kaltaisesti JVM-tavukoodiksi. Se on suunniteltu toimimaan, kuten muutkin JVM-tavukoodiksi kääntyvät ohjelmointikielet, kaikissa Javaa tukevissa laitteissa. Kotlinin ensimmäinen julkaisu tapahtui heinäkuussa 2011 JetBrainsin nettisivuilla (Makarkin 2011), jossa lyhyesti kerrottiin Kotlinista sekä jaettiin aiheeseen liittyvät dokumentoinnit. Kotlinin ensimmäinen vakaa julkaisu (versio 1.0) tapahtui vasta helmikuussa 2016.

Vuonna 2017 Google alokoi kehittää Kotlinin tukemista Android-ohjelmointiympäristössä ja 2019 Kotlin korvasi Javan Android-ohjelmointikehityksessä suositeltuna ohjelmointikielenä, sillä Kotlin oli noussut suosituksi ohjelmointikieleksi Android-alustalla sekä se nähtiin kaikkiaan vahvempana ohjelmointikielenä kuin Java. Tämän seurauksena Kotlinin tukeminen ja käyttö vahvistui entisestään, kuten Googlen kehittämässä Android Studiossa, jossa aiemmin Kotlin-ohjelmia ei voitu luoda ilman erikseen hankittuja liitännäisiä. Sovellusten luominen ilman liitännäisten hankkimista tehtiin mahdolliseksi ja Android Studio alkoi myös oletuksena suosittelemaan uusien projektien luomista Kotlinilla (Google 2019).

2.4 Flutter

Flutter on Googlen kehittämä avoimen lähdekoodin ohjelmistokehitysalusta, jonka tarkoituksena on luoda monialustaisia sovelluksia eli sovelluksia, jotka luodaan käytettäviksi useille erilaisille alustoille. Monialustaiset sovellukset saattavat vaatia oman koontiversionsa jokaiselle tuetulle alustalle,

mutta Flutterin kanssa koodi käännetään vain kerran, jolloin se toimii suoraan kaikilla halutuilla alustoilla.

Flutter esiteltiin ensimmäisen kerran huhtikuussa 2015 Dart Developer Summit -tapahtumassa nimellä ”Sky”, jolloin se oli toiminnassa vain Android-alustalla (Google 2015). Vasta joulukuussa 2018 Google esitteli Flutterin ensimmäisen merkittävän version, Flutter 1.0, jolloin aloitettiin tuki iOS-alustalle (Google 2018). Maaliskuussa 2021 Flutter 2.0 julkaistiin, jolloin tuki levisi myös web- ja työpöytäsovelluksiin (Flutter 2021c).

Flutterissa käytetään ohjelmointikielenä Googlen kehittämää Dartia, joka on olio-ohjelmointiin perustuva ohjelmointikieli, joka soveltuu hyvin asiakasohjelmien eli loppukäyttäjien sovelluksien kehittämiseen, kuten web- tai mobiilisovelluksiin. Dart esiteltiin ensimmäisen kerran lokakuussa 2011 GOTO-konferenssissa (GOTO 2011), jolloin se sai sekalaisen vastaanoton, sillä sen nähtiin jakavan internetiä suunnitellulla Chrome-selaimeen tarkoitetulla virtuaalikoneella, eli sen seurauksena Dart toimisi vain Chrome-selaimilla. Myöhemmin suunnitelmat muuttuivat ja Dartin kääntö onnistuisi JavaScript-kielelle, jolloin sen suorittaminen onnistuisi useimmilla moderneilla selaimilla.
Windowsilla, macOS:llä sekä Linuxilla Flutter suoritetaan Dart-virtuaalikoneella, jossa käytettään JIT-suoritusmoottoria (Just In Time), jonka ansiosta voidaan tehdä muutoksia lähdetiedostoihin ja heti ladata muutoksen sovelluksen käyttöön ilman koodin uudelleenkääntöä.
Android- ja iOS-alustoille koodi käännetään käyttämällä AOT-menetelmää (Ahead Of Time), joka takaa paremman suorituskyvyn sovelluksen suorittamisen aikana.

Flutter on todennäköisesti järkevämpi vaihtoehto natiivien ohjelmistokielien sijaan, jos tarkoituksena on luoda sovellus usealle eri alustalle. Varsinkin pienet yritykset voivat säästää mittavan määrän rajoitetuista resursseista, kun huollettavana on vain yksi ainoa koodikanta. Vaikka tarkoituksena olisikin luoda sovellus pelkästään yhdelle alustalle, voi olla kehittäjän kannalta järkevää käyttää Flutteria, jos tulevaisuudessa on tarvetta laajentaa uusille alustoille.

2.5 Swift

Swift on Applen kehittämä yleiskäyttöinen iOS-alustalle suunnattu ohjelmointikieli, joka julkaistiin ensimmäisen kerran kesäkuussa 2014 (Apple 2014). Sen tarkoituksena oli korvata Applen aiemmin käyttämä Objective-C-ohjelmointikieli. Koska Swiftin tarkoituksena oli korvata Objective-C, sen piti

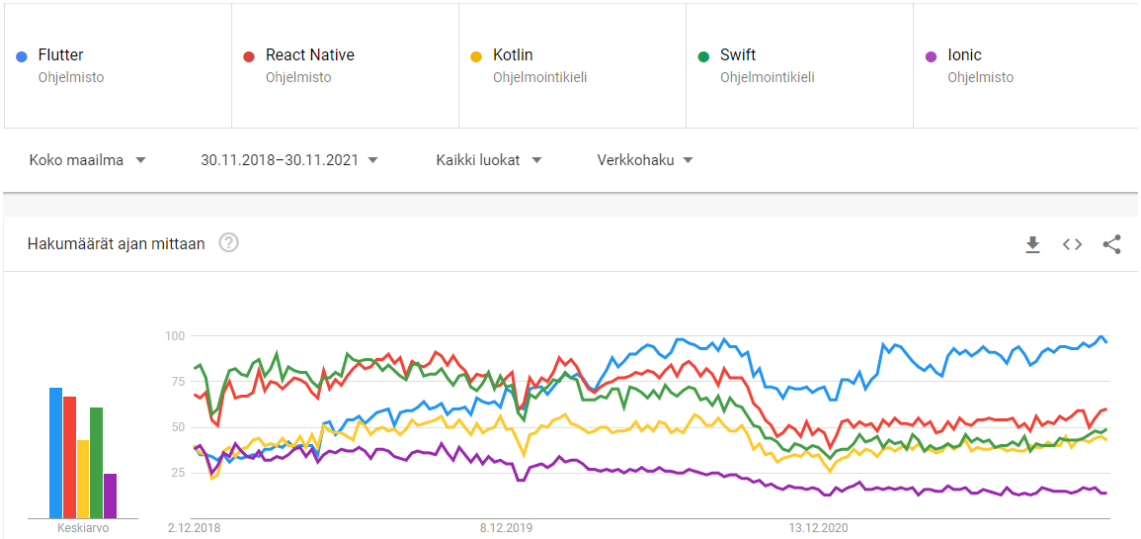
olla samankaltainen toiminnaltaan, jotta ohjelmistokehittäjät voivat helpommin vaihtaa Swiftiin, sillä eihän Apple halunnut luoda liikaa ongelmia ohjelmistokehittäjilleen ohjelmistokielen vaihtuessa. Osana tätä muutosta oli Xcode eli iOS-alustojen ohjelmistokehitysympäristö, jossa voitiin luoda sovelluksia, jotka tukevat C-, Objective-C-, C++- sekä Swift koodia sovelluksissa.

Swiftin tarkoitus on olla monipuolinen ja tehokas ohjelmointikieli, kun se samalla myös toimii turvallisesti. Se tukee dynaamista sidontaa, laajennettavaa ohjelmointia sekä muita samankaltaisia ominaisuuksia, jonka ansiosta ohjelmistovikojen löytäminen on helpompaa. Swift on käynyt läpi useita laajoja päivityksiä, ja se on nyt versiossa 5.4.2.

2.6 Hybridiohjelmoinnin (Flutter) vertailu natiiviin mobiiliohjelmointiin

Tällä hetkellä natiivi mobiilikehitys on huomattavasti suositumpaa kuin hybridiohjelmointi. Statistan teettämän kyselyn mukaan noin kolmasosa mobiilikehittäjistä käyttävät jotain hybridikehitysteknologiaa. Heistä 42 % käyttävät ainakin Flutteria, joka on kasvanut viimeisen 3 vuoden aikana 30 %:n käytöstä 42 %:iin ja kyseinen luku on todennäköisesti edelleen kasvussa (Statista 2021). Koska kehittäjät ja yritykset haluavat sovellusten tavoittavan mahdollisimman suuren henkilömäärän, kehitys on keskittynyt hieman enemmän hybridiohjelmointiin, joten hybridiohjelmointi on vienyt osan natiivikehityksen osuudesta.

Hybridiohjelmoinnin ja mobiiliohjelmoinnin käytöstä on vaikea löytää tarkkoja lukuja, mutta yhtenä lähteenä voidaan käyttää Googlen tarjoamaa Google Trends -nettipalvelua, jossa voidaan vertailla eri hakutermejä ja -aiheita. Google Trends ei näytä raakoja lukuja hakumääristä, vaan se vertailee annettujen hakutermien ja -aiheiden hakuja prosentteina toisiinsa vertailtuna. 100 on korkein hakumäärä yhdelle aiheelle, 50 on puolet kyseisen aiheen suurimmasta hakumäärästä valitulla aikajaksolla. Kun aiheita on useimpia, verrataan suurimman aiheen korkeimpaan hakumäärään.
Kuvan 1 mukaisesti nähdään, että Flutter on kyseisistä aiheista haetuin, joten voitaisiin olettaa, että se on todennäköisesti myös kaikista käytetyin. Kyseisessä haussa ei näytetä mm. Androidissa käytettävää Javaa, sillä Javan käyttö on Androidin ulkopuolella erittäin suuri, joten on vaikeaa arvioida Javan käyttöä pelkästään mobiilikehityksessä. Kuvasta voidaan myös havaita Flutterin kasvu viimeisen kolmen vuoden aikana sekä muiden hybriditeknologioiden lasku, jonka takana on todennäköisesti Flutterin suosion kasvu. Voidaan havaita myös natiivissa iOS-ohjelmoinnissa käytettävän Swiftin lasku sekä Android-ohjelmoinnissa käytettävän Kotlinin olematon kasvu.



KUVA 1: Google Trendsin näyttämä kaavio viimeisen kolmen vuoden ajalta useiden eri ohjelmistojen/ohjelmointikielten vertailusta. (Google Trends 2021)

Yrityskäytössä hybridiohjelmointi (Flutter) on saanut valtavasti ylistystä, sillä sovelluksia luodaan nopeammin ja halvemmalla kuin saman sovelluksen luominen useille eri alustoille usealla eri kielellä. Flutteria käyttänyt iRobot on kertonut, että heidän sovelluskehityksensä tuottavuus kasvoi 400 %:lla, kun he vaihtoivat Flutteriin (iRobot 2021 ; Flutter 2021a). Tämä selittyi sillä, että heidän kehittäjänsä tunsivat Googlen alustat ja Flutterin dokumentointi oli heidän mielestään hyvä. He saivat luotua monialustaisen sovelluksensa nopeammin kuin heidän luomansa alkuperäisen iOS-sovelluksen, jonka ansiosta heidän käyttäjäkantansa myös kasvoi 300 %:lla, kun yhdellä koodikannalla pidettiin yllä kolmella alustalla löytyvää sovellusta. On siis selvää, että varsinkin pienet yritykset, joilla ei ole varaa suuriin kehittäjätiimeihin, hyötyvät Flutterista suuresti, kun muutama kehittäjä saa aikaiseksi kymmenen kehittäjän työn. Myös eBayn Flutter-sovellus oli onnistunut, sillä 100 % kehittäjistä piti Flutterista enemmän kuin iOS:stä tai Androidista. 70 % heidän kehittäjistään uskoi, että sovelluskehitys olisi kaksi kertaa nopeampaa kuin natiivi sovelluskehitys (Flutter 2021b).

Ei siis tule yllätyksenä, kun yritykset siirtyvät natiivista mobiiliohjelmoinnista hybridiohjelmointiin. Hybridiohjelmoinnilla on useita hyötyjä natiiviin mobiiliohjelmointiin verrattuna. Kun sovelluksella on yksi ainoa koodikanta, kaikki kehitykseen liittyvä on nopeampaa, kuten koodin kirjoitus, testaus ja ylläpito. Kun kirjoitettavana on vain yksi koodikanta, vaikka sitä ei tarvittaisi, voidaan Flutterilla luotu sovellus helposti laajentaa mukaan toisille alustoille ilman suurempia muutoksia koodikantaan, jolloin sovellus tavoittaa useampia käyttäjiä ja sovellus tuottaa enemmän rahaa yritykselle tai kehittäjälle.

3 MATERIAALIN UUDISTAMINEN MOBIILIKÄYTTÖLIITTYMÄT-KURSSILLE

Opinnäytetyö jaettiin seuraaviin kolmeen osaan: Mobiilikäyttöliittymät-kurssin materiaalin uudistus, Flutter-kurssin materiaalin tarkastaminen sekä itse opinnäytetyö-dokumentin kirjoittaminen.
Opinnäytetyön ensimmäisessä osassa käytiin läpi Mobiilikäyttöliittymät-kurssin opetusmateriaali, joka koostuu laajasta ohjaavasta PowerPoint-esityksestä, sekä kyseiseen esitykseen liittyvistä Java-harjoituksista ja -esimerkeistä. Osana tätä työtä opetusmateriaali sekä siihen liittyvien harjoitusten ja esimerkkien koodit käännettiin Java-kielestä Kotlin-kielelle.

Opinnäytetyön toisessa osiossa käytiin läpi valmiina ollut mahdollisen Flutter-kurssin materiaali, jonka avulla opeteltiin Flutterin perusteet. Kun Flutter oli hallussa, käytiin materiaali uudestaan läpi ajatellen, miten sitä voisi mahdollisesti parantaa.

3.1 Mobiilikäyttöliittymät-kurssin materiaalin uudistus

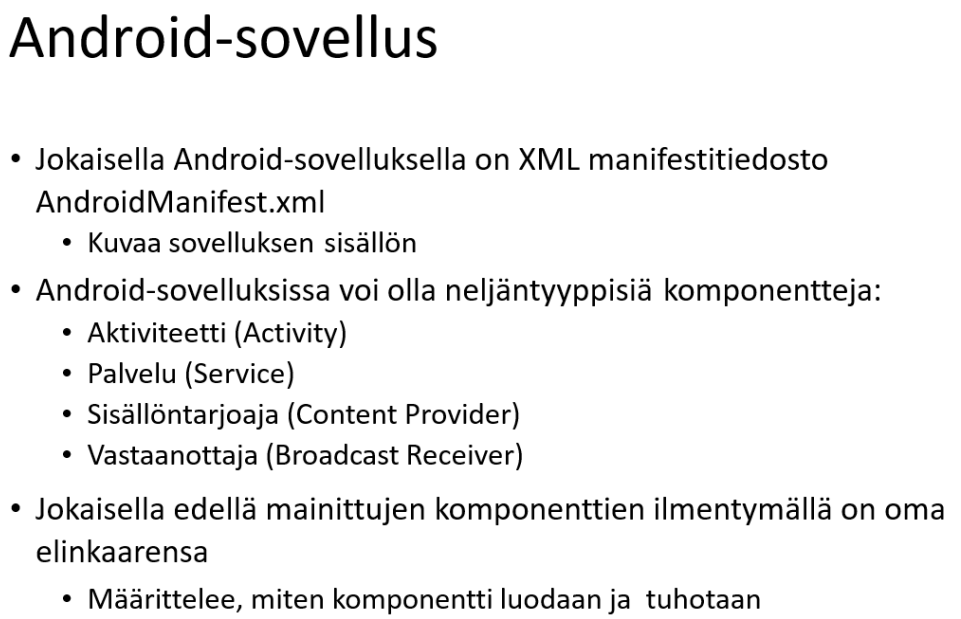
Mobiilikäyttöliittymät-kurssi käsittelee ohjelmistokehitystä Android-alustalle käyttäen Java-ohjelmointikieltä. Nykyisin Google suosittelee Javan käytön vaihtoa sen korvaajaan Kotliniin, joka on vahvasti Javan kaltainen ohjelmointikieli ja olikin luotu Javan korvaajaksi. Tässä luvussa käydään läpi materiaalin esimerkkejä ja harjoituksia lyhyin selostuksin.

Mobiilikäyttöliittymät-kurssin materiaali käännettiin Java-kielestä Kotlin-kielelle, jolloin myös pyrittiin pitämään esimerkkien ja sovellusten koodien rakenne mahdollisimman samankaltaisina. Tämä onnistui loppuen lopuksi erittäin hyvin, sillä Java ja Kotlin ovat hyvin samankaltaisia kieliä ja niillä molemmilla löytyvät samat aliohjelmat ja luokat niiden standardikirjastosta, joten sovellusten kääntö kielestä toiseen on melko yksinkertaista.
Ennen satunnaisten aiheiden käsittelyä listataan kaikki luodun materiaalin käsittelemät aiheet:

- Android-ohjelmoinnin kehitysympäristö
- Android-sovellus
- Muuttujat
- Konditionaalilausekkeet
- Aktiviteetti (Activity) ja sen elinkaari

- Palvelu (Service)
- Sisällöntarjoaja (Content Provider)
- Vastaanottaja (Broadcast Receiver)
- Internet-käyttölupa
- Yleiset View-komponentit
- Aikeet (Intent), Explicit Intent ja Implicit Intent
- Tilan tallennus (Save State)
- Asettelut (Layout)
- Tapahtuman käsittely
- Resurssit
- Kuva
- Ääni
- Video
- Toast
- Valikot ja toimintapalkki
- Säikeet (Threads)
- Fragmentit.

Ensimmäisenä opiskelijalle esitellään muutamia sovelluksien sisältämiä perusosia kuvan 2 mukaisesti.



KUVA 2: Android sovelluksen sisältämiä osia. Kuva pitää paikkansa sekä Javalle että Kotlinille.

Ensimmäiset koodit seuraavat heti perässä. Kuvien 3 ja 4 mukaisesti opiskelijalle opetetaan ohjelmoinnin yksi tärkeimmistä asioista, eli muuttujat.

Muuttujat (Variables)

- https://developer.android.com/kotlin/learn
- Kotlinissa muuttujat luodaan etuliittellä var (variable), val (value) tai const (constant)
  - Muuttuja voi olla joko numero (bit,short,long,int,float,double), boolean (true/false), merkki (yksittäinen merkki, a-z/A-Z/0-9/!/?...), merkkijono (string) tai taulukko joka sisältää useita yhden muuttujatyypin arvoja.
  - var on muuttujatyyppi, jonka arvot asetetaan joskus ohjelman suoriutuessa, ja sen arvoit voidaan asettaa useita kertoja.
  - const on muuttujatyyppi joka nimensä mukaisesti pysyy vakiona. Sen arvo asetetaan koodin kääntyessä jonka takia sen arvoksi voidaan asettaa pelkästään yksinkertaisia arvoja (luvut, tekstit ym., Ei voida hakea arvoa esim. funktion kautta)
  - val on muuttujatyyppi jonka toimii kuten constant, eli sen arvo voidaan asettaa vain kerran. Toisin kuin const, sen arvo voidaan asettaa milloin tahansa koodin suoriutuessa.

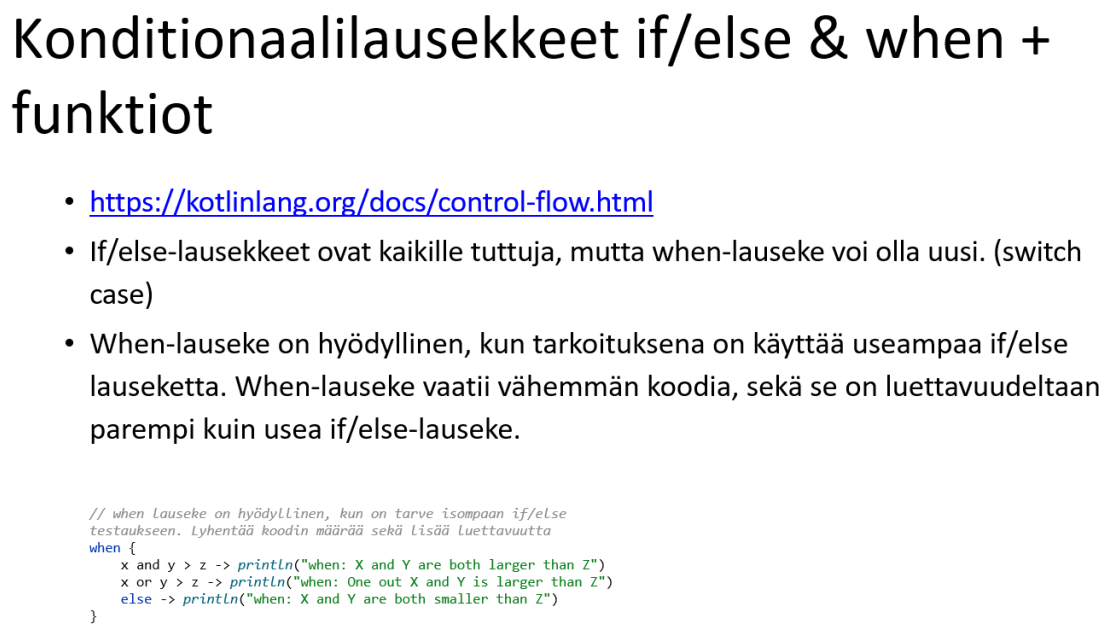
KUVA 3: Muuttujien selittäminen sanallisesti.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        var byteVariable: Byte = 25 // Byten arvo on -128 ja 127 välillä
        var shortVariable: Short = 250 // Shortin arvo on -32768 ja 32767 välillä
        var intVariable: Int = 2500 // Intin arvo on -2147483648 ja 2147483647 välillä
        var longVariable: Long = 25000 // Longin arvo on -9,223,372,036,854,775,808 ja 9,223,372,036,854,775,807 välillä
        // Kun luodaan numeromuuttujia, ei ole välttämätöntä käyttää esim. Byte-datatyyppiä vaikka olisikin tiedossa, että numero pysyy -128 ja 127 välillä, sillä
        // ero kyseisten muuttujien välillä on pienemmässä mittakaavassa olematon
        // Kun luodaan numeroita sisältävä muuttuja, jos mahdollista, oletuksena käytetään int-datatyyppiä. "var numero = 2500" on int

        var floatVariable = 2.5f // Floatin arvo on 1.4E-45 ja 3.4028235E38 välillä
        var doubleVariable = 2.5 // Doublen arvo on 4.94065645841246544e-324 ja 1.79769313486231570e+308 välillä
        // Float ja Double voivat sisältää desimaaliarvoja, eli ne ovat tarkempia arvoja kuin int jos tarkoituksena on laskea jotain
        // Kun luodaan float-datatyyppiä käyttävä muuttuja, yksinkertaisesti luodaan numeromuuttuja jonka loppuun lisätään "f" loppuun
        // Kun luodaan double-datatyyppiä käyttävä muuttuja, muuttujan arvoksi annetaan desimaaliarvo, ilman mitään muita merkkejä
        // floatilla voidaan ilmaista 7 ja doublella 16 desimaalin tarkkuudella

        var booleanVariable: Boolean = true // Booleanin arvo on joko true tai false

        var charVariable: Char = 'A' // Char sisältää vain yhden merkin

        var stringVariable: String = "Hello" // String on pidempi tekstijono

        // Voidaan myös luoda val (value) ja const (constant) muuttujia, joiden arvo voidaan asettaa vain kerran
        // Niillä on kuitenkin eroavaisuuksia, const muuttujan arvo pitää olla täysin tiedossa koodin kääntäessä eli sen arvoksi asetetaan esim. string tai numeroarvo
        // const muuttujan pitää myös olla joko ylätasolla eli kuten tässä tiedostossa, luokan ulkopuolella tai sen pitää olla luokan esittelyssä
        // val muuttuja on kuin const joka voidaan luoda ja sen arvo voidaan asettaa milloin tahansa koodin suorittuessa joten sen arvoksi voidaan asettaa esimerkiksi
        // toisia muuttujia vaikkapa nappia painaessa
        val valueStringVariable: String = "Hello" // Toimii täysin samoin kuin normaali string, mutta tämän muuttujan arvoa ei voida enää muokata
    }
}
```

KUVA 4: Muuttujien selittäminen koodiesimerkkinä.

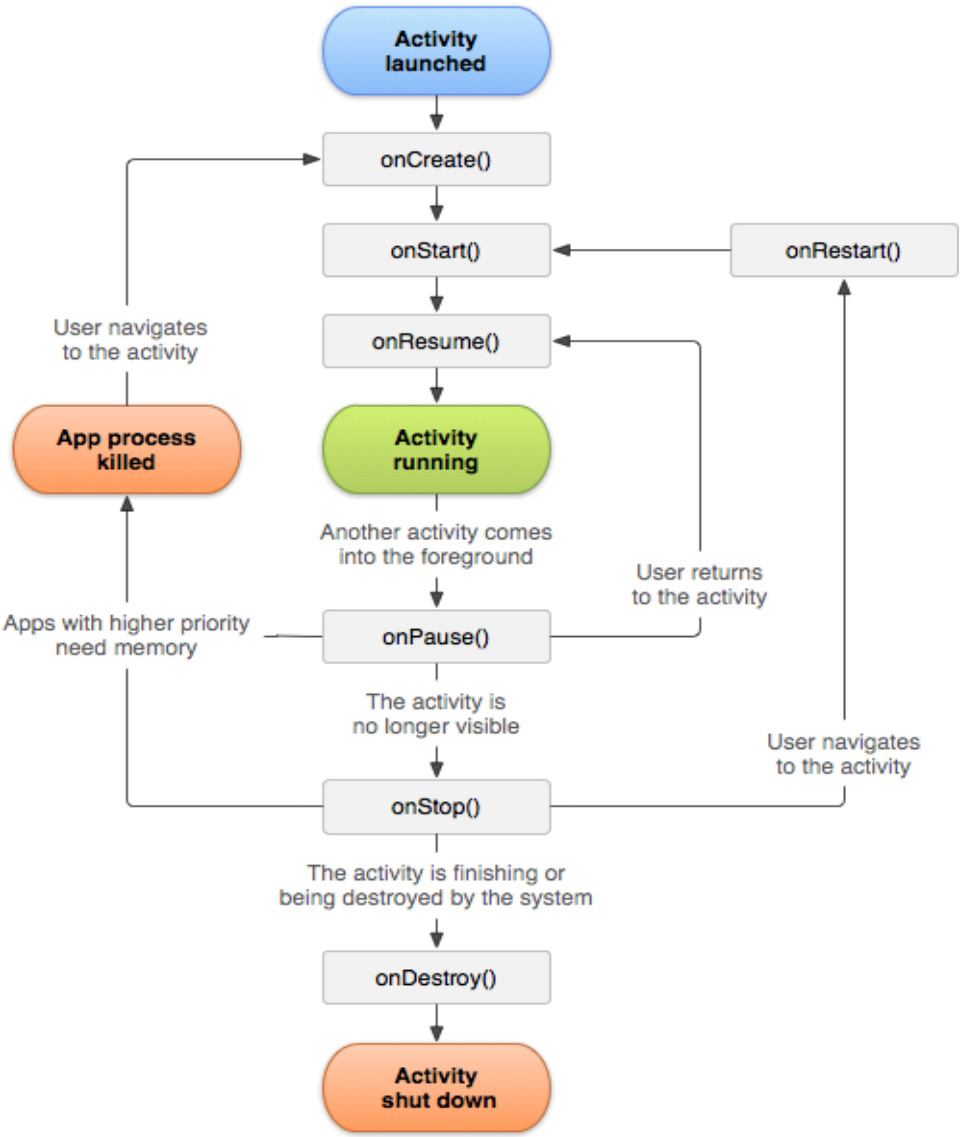
Alkuperäisessä, Java-materiaalissa muuttujia ei varsinaisesti käyty läpi omana aiheenaan, joten päätin luoda niille oman osionsa. Muuttujat opetetaan kahdella dialla. Ensin selitetään muuttujatyypit ja toisella dialla näytetään koodiesimerkit jokaisesta muuttujatyypistä.

Seuraavaksi opiskelija käy läpi konditionaalilausekkeet, eli if/else- ja when-lausekkeet, jotka ohjaavat sovelluksen toimintaa tiettyjen ehtojen mukaisesti. Koska if- ja else-lausekkeet ovat erittäin tuttuja muistakin ohjelmointikielistä, niitä ei käsitellä, vaan oletetaan, että opiskelija jo ymmärtää niiden toiminnan. Jokaisessa osiossa annetaan aiheeseen liittyvä linkki, josta saa tarvittavaa lisätietoa, jos opiskelija sitä tarvitsee.



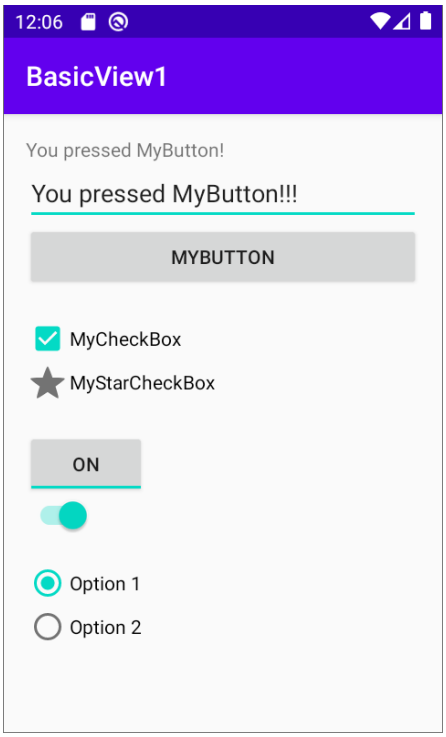
KUVA 5: Konditionaalilausekkeen ”when” selitys.

Konditionaalilausekkeiden jälkeen opiskelija alkaa käymään läpi erilaisia sovelluksen toimintaan liittyviä aiheita, kuten aktiviteetit, palvelut, sisällöntarjoajat ja vastaanottajat. Aiheet vahvistavat sovelluksen toimintaa esim. uusien ikkunoiden avaamisella tai tiettyjen tapahtumien kuuntelemista ja siihen tapahtumaan reagoimista. Käydään myös läpi kuva 6 avulla aktiviteetin eli sovellusten ikkunoiden elinkaari, joka on tärkeä ymmärtää, jotta sovelluksen toiminnan kannalta kehittäjällä ei tule mitään yllätyksiä.



KUVA 6: Aktiviteetin elinkaari. (Android Developers 2021)

Seuraavaksi opiskelija käy läpi useita eri sovelluksen View-komponentteja eli erilaisia sovelluksen osia, joita sovellus voi käyttää tiedon esittämiseen sekä sovelluksen ja käyttäjän välisen vuorovaikutuksen vahvistamiseen. Kuvassa 7 näytetään esimerkkinä tekstikenttä, nappi ja valintaruutu, jotka ovat yksinkertaisia mutta tärkeitä käyttöliittymän osia.



KUVA 7: BasicView1-esimerkki, jossa käydään läpi muutamia View-komponentteja.

Kuvista 8 ja 9 voidaan nähdä, että Javan ja Kotlinin koodirakenne on samankaltaisia. Se ei olekaan yllätys, kun Kotlinin tarkoituksena onkin olla Javan kaltainen mutta toiminnaltaan parempi ohjelmointikieli.

Explicit Intent

Tiedon vastaanotto aktiviteetista, kun edellinen-painiketta painetaan:

```java
public class MainActivity extends AppCompatActivity {

    private static final int REQUEST_CODE = 12;
    private TextView textView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView = findViewById(R.id.text);
    }

    public void onClick(View view) {
        Intent intent = new Intent(this, SecondActivity.class);
        startActivityForResult(intent, REQUEST_CODE);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,  Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == REQUEST_CODE) {
            if (resultCode == RESULT_OK) {
                Bundle bundle = data.getExtras();
                String str = bundle.getString("email1");
                textView.setText(str);
            }
        }
    }
}
```

```java
public class SecondActivity extends AppCompatActivity {

    EditText editText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        editText = findViewById(R.id.emailEdit);
    }

    @Override
    public void onBackPressed() {
        Intent intent = new Intent();
        String str1 = editText.getText().toString();
        intent.putExtra("email1", str1);
        setResult(RESULT_OK, intent);
        super.onBackPressed();
    }
}
```

Harjoitus 2

KUVA 8: Explicit Intent Java-kielellä. Explicit Intentillä tarkoitetaan Android-ohjelmoinnissa sovelluksen tietyn toisen aktiviteetin (käyttöliittymän ikkuna) avaamista.

Explicit Intent

Tiedon vastaanotto aktiviteetista, kun edellinen-painiketta painetaan:

```
class MainActivity : AppCompatActivity() {

    private val REQUEST_CODE = 12
    private var textView: TextView? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        textView = findViewById(R.id.text)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == REQUEST_CODE) {
            if (resultCode == RESULT_OK) {
                var bundle = data?.extras
                var str = bundle!!.getString("name")
                textView!!.text = str
            }
        }
    }

    fun onClick(view: View?) {
        val intent = Intent(this, SecondActivity::class.java)
        startActivityForResult(intent, REQUEST_CODE)
    }
}
```

```
class SecondActivity : AppCompatActivity() {

    var editText: EditText? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_second)
        editText = findViewById(R.id.editText);
    }

    override fun onBackPressed() {
        var intent = Intent()
        val str = editText!!.text.toString()
        intent.putExtra("name", str)
        setResult(RESULT_OK, intent)
        super.onBackPressed()
    }
}
```

Harjoitus 2

KUVA 9: Explicit Intent Kotlin-kielellä. Implicit Intentillä tarkoitetaan android-ohjelmoinnissa suoritettavan toiminnan määritystä.

Luotua opiskelumateriaalia on enemmänkin, mutta sitä ei tässä työssä enempää käsitellä. Materiaali koostuu samankaltaisista ja monimutkaisemmistakin esimerkeistä ja harjoituksista, joita opiskelijoiden pitää käydä läpi ja ratkaista.

3.2 Flutter-kurssin materiaalin tarkistus

Työn toisessa osiossa tutustuttiin vuoden 2020 lopulla luotuun Flutter-opetusmateriaaliin, jonka jälkeen myös tutustuttiin lisää Flutteriin muita lähteitä käyttäen. Kun Flutter oli perusteiltaan tuttu, käytiin opetusmateriaali uudestaan läpi ja tarkasteltiin materiaalin luottettavuutta ja mietittiin, olisiko jotain uutta lisättävää. Materiaali oli tärkeää tarkistaa, sillä Flutter oli vielä melko uusi asia ja olikin vasta saanut suuren päivityksen. Kun käytiin Flutterin materiaali uudestaan läpi, yllätyksenä materiaali olikin edelleen hyvin käytettävä. Se oli edelleen täysin ajanmukainen, eikä se vaatinut suurempia muutoksia. Materiaali vaati vain pari muutosta. Luotiin vinkki koodin suorittamiseen liittyen, jonka kanssa itselläni oli ollut ongelmia, sekä luotiin kuvan 10 mukainen harjoitus.

Harjoitustehtävä 4

Tee Calculator–sovellus, jolla voidaan laskea kahden luvun summa, erotus, tulo sekä osamäärä

Tee sovelluksesta fyysisen laskimen kaltainen, eli napit jokaiselle luvulle sekä operaattoreille

KUVA 10: Luotu harjoitus vaatii opiskelijaa luomaan kuvan kaltaisen laskimen.

Seuraavaksi käydään kuvassa 10 olevan harjoituksen esimerkkikoodia läpi.

```
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(Build Context context) {
    return MaterialApp(
      title: 'Calculator',
      theme: ThemeData.dark(),
      home: Scaffold(
        body: SafeArea(
          child: Container(
            alignment: Alignment.center,
            child: Calculator(),
          ),
        ),
      ),
    );
  }
}
```

main()-funktiossa käynnistetään sovellus käyttämällä MyApp()-luokkaa pohjana. Kyseinen luokka on hyvin yksinkertainen. Se sisältää pelkästään Scaffold-tyyppisen Widgetin luomisen, joka on siis yksinkertaisesti laatikko, jolle valitaan tyyliksi dark, joka tekee laatikosta tummanharmaan. Laatikon sisälle asetetaan Calculator-luokan olio, joka selitetään seuraavaksi.

```
class Calculator extends StatefulWidget {
  @override
  _CalculatorState createState() => _CalculatorState();
}
```

Yllä luodaan Calculator-luokka, joka ei sisällä paljoa. Siinä ulotetaan luokka StatefulWidgetiksi, jolloin luokan käyttöliittymään voidaan tehdä muutoksia ja näyttää kyseiset muutokset käyttäjälle muutoksien tapahtuessa. Toisin kun Javassa ja Kotlinissa, Flutterissa (Dartissa) muuttujat voidaan sijoittaa käyttöliittymien objektien sisälle, kun taas Javassa ja Kotlinissa käyttöliittymän objektien arvoiksi asetetaan muuttujan sisältämä arvo.

Java & Kotlin tehdään tyyliin: “object.value(variable)”. Kun muuttujan arvo asetetaan objektin arvoksi, objektin piirtäminen (päivittäminen) tapahtuu käyttöliittymässä uudelleen. Jos arvoa käytetään useassa objektissa, ne täytyy asettaa jokaiselle objektille erikseen aina, kun muuttujan arvo vaihtuu.

Flutter (Dart) tehdään tyyliin: ”object(child: Text(variable))”. Tässä tapauksessa muuttujan arvoa ei aseteta objektin arvoksi, vaan asetetaan objektin lapsen(tekstilaatikko) arvoksi kyseinen muuttuja, eli itse muuttuja kokonaisuutenaan on arvona eikä muuttujan arvo. Kun arvo muuttuu, kaikki objektit, joissa kyseinen muuttuja on lisättynä, päivittyvät uuden muuttujan arvon mukaan.

Käytetään createState()-funktiota CalculatorState()-luokan luomiseen. Uutta luokkaa käytetään Calculator-luokan muokkaamiseen.

```
class _CalculatorState extends State<Calculator> {
  var num1 = ""; // ensimmäinen valittu luku
  var num2 = ""; // toinen valittu luku
  var num3 = ""; // laskutoimitus tekstimuodossa
  var value = ""; // summa
  var state = 0; // muokataanko num1 vai num2 (muuttuu +/- nappia painaessa)
  var chosenCalc = ""; // valittu laskutapa ("add" tai "subtract")
```

Yllä on sovelluksen käyttämiä muuttujia, selitykset kommenteissa. Seuraavaksi luodaan sovelluksen käyttöliittymä, joka sisältää muutaman tekstin sekä useita nappeja.

```
  @override
  Widget build(BuildContext context) {
    return Container(
      color: Colors.black45,
      child: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Column(
          mainAxisSize: MainAxisSize.min,
```

```
crossAxisAlignment: CrossAxisAlignment.center,
children: [
```

Toisin kuin Javassa, Dartissa objektin (tekstilaatikon) arvoksi voidaan asettaa muuttuja, jonka arvo päivittyy aina, kun muuttujan arvo päivittyy. Javassa joudutaan erikseen päivittämään objektin arvo, kun muutetaan muuttujan arvoa.

```
Text(num3),
Text(value),
```

Luodaan seuraavaksi kaikki napit numeroita sekä operaattoreita varten.

```
Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    myElevatedButton("7"),
    SizedBox(width: 5),
    myElevatedButton("8"),
    SizedBox(width: 5),
    myElevatedButton("9")
  ],
),
SizedBox(height: 5),
Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    myElevatedButton("4"),
    SizedBox(width: 5),
    myElevatedButton("5"),
    SizedBox(width: 5),
    myElevatedButton("6")
  ],
),
SizedBox(height: 5),
Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    myElevatedButton("1"),
    SizedBox(width: 5),
    myElevatedButton("2"),
    SizedBox(width: 5),
    myElevatedButton("3")
  ],
),
SizedBox(height: 5),
Row(
  mainAxisSize: MainAxisSize.min,
  children: [myElevatedButton("0")],
),
SizedBox(height: 15),
Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    ElevatedButton(
      child: Text('+'),
      onPressed: () {
```

```
                setState(() {
                  add();
                });
              },
            ),
            SizedBox(width: 5),
            ElevatedButton(
              child: Text('-'),
              onPressed: () {
                setState(() {
                  subtract();
                });
              },
            ),
          ],
        ),
        SizedBox(height: 5),
        Row(
          mainAxisSize: MainAxisSize.min,
          children: [
            ElevatedButton(
              child: Text('*'),
              onPressed: () {
                setState(() {
                  multiply();
                });
              },
            ),
            SizedBox(width: 5),
            ElevatedButton(
              child: Text('/'),
              onPressed: () {
                setState(() {
                  divide();
                });
              },
            ),
          ],
        ),
        SizedBox(height: 5),
        ElevatedButton(
          child: Text('='),
          onPressed: () {
            setState(() {
              equals();
            });
          },
        )
      ],
    ),
  ),
);
}
```

Sovelluksen toiminta tapahtuu seuraavien funktioiden kautta. Ensimmäisenä ovat funktiot add(), substract(), multiply() sekä divide(), joiden toiminnan kautta valitaan laskutoiminnan tyyppi sekä lisätään jo valmiiksi taululle kirjoitettu luku muuttujaan num3, jota käytetään tapahtuvan laskun näyttämisessä.

```
void add() {
  if (num1 != "" && chosenCalc == "") {
    state = 1;
    chosenCalc = "add";
    num3 = num3 + "+";
  }
}
void subtract() {
  if (num1 != "" && chosenCalc == "") {
    state = 1;
    chosenCalc = "subtract";
    num3 = num3 + "-";
  }
}
void multiply() {
  if (num1 != "" && chosenCalc == "") {
    state = 1;
    chosenCalc = "multiply";
    num3 = num3 + "*";
  }
}
void divide() {
  if (num1 != "" && chosenCalc == "") {
    state = 1;
    chosenCalc = "divide";
    num3 = num3 + "/";
  }
}
```

equals()-funktiossa suoritetaan varsinainen laskutoimitus. Testataan valittu laskutoimitustyyppi (chosenCalc-muuttuja) ja suoritetaan kyseinen laskutoimitus laskemalla muuttujat num1 ja num2 valitun laskutoimituksen mukaisesti.

```
void equals() {
  if (chosenCalc == "add") {
    value = (int.parse(num1) + int.parse(num2)).toString();
  } else if (chosenCalc == "subtract") {
    value = (int.parse(num1) - int.parse(num2)).toString();
  } else if (chosenCalc == "multiply") {
    value = (int.parse(num1) * int.parse(num2)).toString();
  } else if (chosenCalc == "divide") {
    value = (int.parse(num1) / int.parse(num2)).toString();
  }
```

```
    print("total value: " + value.toString());
    Future.delayed(Duration(milliseconds: 100), () {
      resetCalculator();
    });
  }
```

writeDigits(int digit)-funktiossa kirjoitetaan sovelluksen käyttäjän valitsemat luvut. Tätä funktiota kutsutaan aina, kun jotain napeista 0–9 painetaan.

state-muuttujan mukaisesti valitaan, muokataanko muuttujaa num1 vai num2. Jos state-muuttujan arvo on 0, se tarkoittaa sitä, että käyttäjä ei ole vielä kirjoittanut ensimmäistä laskettavaa lukua eli laskutapaa ei ole vielä valittu.

```
  void writeDigits(int digit) {
    if (state == 0) {
      if (num1 == "" && digit == 0) {
        // Ei kirjoiteta nollia jos num1 on tyhjä
      } else {
        num1 = num1 + digit.toString();
        num3 = num3 + digit.toString();
        print("num1: " + num1);
      }
    } else if (state == 1) {
      if (num2 == "" && digit == 0) {
        // Ei kirjoiteta nollia jos num2 on tyhjä
      } else {
        num2 = num2 + digit.toString();
        num3 = num3 + digit.toString();
        print("num2: " + num2);
      }
    }
  }
```

resetCalculator()-funktiossa asetetaan kaikki käytetyt arvot tyhjiksi, jonka jälkeen käyttäjä voi jatkaa laskujen laskemista ilman sovelluksen uudelleenkäynnistämistä.

```
  void resetCalculator() {
    num1 = "";
    num2 = "";
    num3 = "";
    value = "";
    state = 0;
    chosenCalc = "";
  }
```

myElevatedButton on sovelluksessa käytettu mukautettu nappi, joka toimii kuten normaali nappi. Painaessa se aina kutsuu writeDigits()-funktiota.

```
  ElevatedButton myElevatedButton(String buttonDigit) {
    return ElevatedButton(
        child: Text(buttonDigit),
```

```
        onPressed: () {
          setState(() {
            writeDigits(int.parse(buttonDigit));
          });
        });
  }
}
```

4 POHDINTA

Opinnäytetyössä perehdyin Mobiili- ja Hybridiohjelmointiin. Tavoitteena oli oppia lisää mobiili- ja hybridiohjelmoinnista ohjelmistokehittäjänä sekä luoda lisää parempaa oppimateriaalia Oamkin tulevaisuuden opiskelijoille.

Mielestäni opinnäytetyö onnistui hyvin, mutta ongelmia oli aikataulun ja opinnäytetyön kirjoituksen kanssa. Ongelmia syntyi yksittäisten pienten asioiden yhdistämisessä yhdeksi suureksi kokonaisuudeksi. Opin paljon uutta sovelluskehityksestä sekä tiedon etsimisessä. Yhtenä ongelmana tämän opinnäytetyön kanssa oli luotettavan tiedon löytäminen, sillä on vaikeaa löytää tarkempia luotettavia lukuja esimerkiksi ohjelmointikielten käytöstä. Oli helppoa löytää muiden henkilöiden tai laajempien organisaatioiden mielipiteitä ohjelmointikielistä, mutta niissä ei yleensä ollut minkäänlaista tietoa tai todisteita takaamassa esim. ohjelmointikielen tehokkuudesta, joten erittäin laaja määrä löytyvästä tiedosta voidaan laskea pelkästään mielipiteinä.
Flutteriin liittyvästä tiedosta kuitenkin löytyi mukavasti yritysesimerkkejä, joissa oli muutamia käyttökelpoisia lukuja.
Koska opinnäytetyössä ei ollut varsinaista yritysasiakasta, ei voitu tarkemmin tutustua esimerkiksi hybridiohjelmoinnin vaikutukseen sovelluskehityksessä tunnetussa ympäristössä.

Mobiiliohjelmointi on erittäin laaja osa ohjelmistokehitystä, sillä mobiililaitteita löytyy jokaisella. Tulevaisuudessa näen mobiilikehityksen siirtyvän entistä enemmän hybriditeknologioihin, sillä yritykset säästävät valtavasti resursseja, kun ylläpidettävänä on ainoastaan yksi koodikanta usean sijaan. On toki mahdollista, että monet kehittäjät pysyvät natiivissa mobiilikehityksessä, jos heillä ei esimerkiksi ole halua laajentaa usealle eri alustalle. On toki huomioitava, että Flutteriin siirtyminen ei tietenkään ole täysin yksinkertaista, kun käytössä on todennäköisesti uusi kieli, jolloin valmiiksi luotu sovellus joudutaan luomaan täysin uudestaan. Tämän takia uskonkin, että tulevaisuuden sovellukset tulevat olemaan enemmän hybriditeknologiaan pohjautuvia ja nykyiset natiivisovellukset hiljalleen joko mukautuvat hybriditeknologiaa käyttäviksi tai ne putoavat käytöstä, kun kilpailevat hybriditeknologiaan pohjautuvat sovellukset kehittyvät nopeammin ja edullisemmin, jonka takia voittomarginaalit ovat paremmat.

LÄHDELUETTELO

Android Developers 2021. Google. The Activity Lifecycle. Hakupäivä 16.2021. https://developer.android.com/guide/components/activities/activity-lifecycle.

Apple 2014. Apple – WWDC 2014. Hakupäivä 6.12.2021. https://www.youtube.com/watch?v=w87fOAG8fjk&t=6234s.

Flutter 2021a. iRobot Uses Flutter to Expand Access to Coding. Hakupäivä 6.12.2021. https://flutter.dev/showcase/irobot.

Flutter 2021b. Delighting engineers at eBay with Flutter. Hakupäivä 6.12.2021. https://flutter.dev/showcase/ebay.

Flutter 2021c. March 3, 2021, Flutter Engage Edition: 2.0 release. Hakupäivä 14.12.2021. https://docs.flutter.dev/whats-new#march-3-2021-flutter-engage-edition-20-release.

Google 2015. Sky: An Experiment Writing Dart for Mobile (Dart Developer Summit 2015). Hakupäivä 6.12.2021. https://www.youtube.com/watch?v=PnIWI33YMwA.

Google 2018. Flutter 1.0: Google's Portable UI Toolkit. Hakupäivä 14.12.2021. https://developers.googleblog.com/2018/12/flutter-10-googles-portable-ui-toolkit.html.

Google 2019. Android's commitment to Kotlin. Hakupäivä 14.12.2021. https://developers.googleblog.com/2018/12/flutter-10-googles-portable-ui-toolkit.html.

Google Trends 2021. Flutter, React Native, Kotlin, Swift & Ionic hakumäärät. Hakupäivä 14.12.2021. https://trends.google.com/trends/explore?date=2018-11-30%202021-11-30&q=%2Fg%2F11f03_rzbg,%2Fg%2F11h03gfxy9,%2Fm%2F0_lcrx4,%2Fm%2F010sd4y3,%2Fg%2F1q6l_n0n0.

GOTO 2011. Dart, a new programming language for structured web programming. Hakupäivä 6.12.2021. https://gotocon.com/aarhus-2011/presentation/Opening%20Keynote:%20Dart,%20a%20new%20programming%20language%20for%20structured%20web%20programming.

Gupta, Manish 2020. Java in 2020. Oracle Blogs. Hakupäivä 6.12.2021. https://blogs.oracle.com/java/post/java-in-2020.

Hartman, James 2021. JVM | What is Java Virtual Machine & its Architecture. Guru99. Hakupäivä 6.12.2021. https://www.guru99.com/java-virtual-machine-jvm.html.

iRobot 2021. Building a Coding Experience for All. Hakupäivä 6.12.2021. https://edu.irobot.com/the-latest/building-a-coding-experience-for-all.

Lardinois, Frederic 7.5.2019. TechCrunch. Kotlin is now Google's preferred language for Android app development. Hakupäivä 6.12.2021. https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/.

Makarkin, Alexey 2011. Welcome. Hakupäivä 6.12.2021. https://web.archive.org/web/20110924201052/http:/confluence.jetbrains.net/display/Kotlin/Welcome. Wayback-machine näyttää sivun päivältä 24.9.2011. Kyseistä sivua ei enää sellaisenaan ole olemassa.

Oracle 2016. Timeline of key Java milestones. Hakupäivä 6.12.2021. https://www.oracle.com/java/moved-by-java/timeline/.

Statista 2021a. Mobile operating systems' market share worldwide from January 2012 to June 2021. Luettu toukokuussa 2021, Ei enää saatavilla ilmaiseksi. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

Statista 2021b. Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021. Hakupäivä 6.12.2021. https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/.

Statcounter 2021. Mobile Operating System Market Share Worldwide. Hakupäivä 6.12.2021. https://gs.statcounter.com/os-market-share/mobile/worldwide.