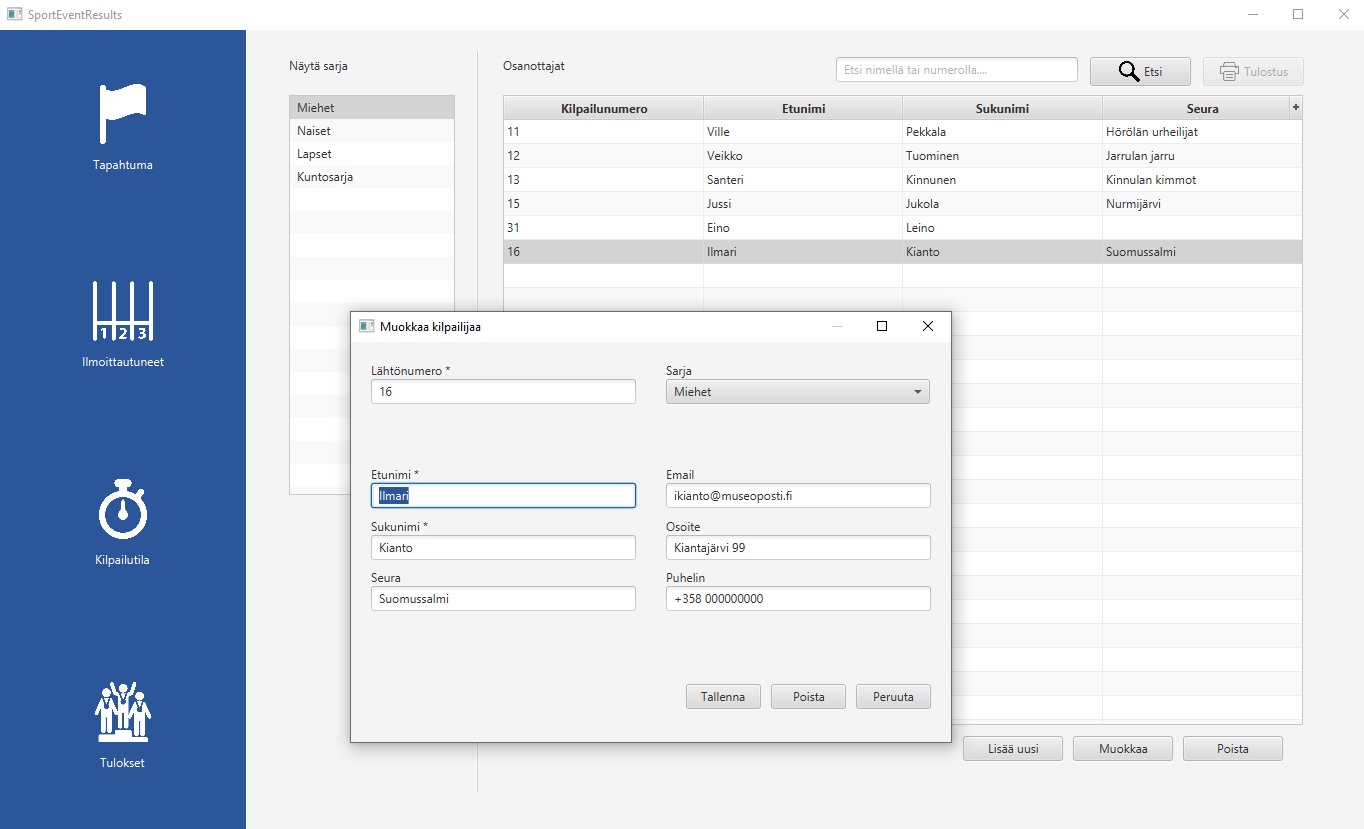


****

Ohjelmistotekniikka Kevät 2019 - Loppudokumentti

Olavi Hartonen

Sisällys

[Päivitetty vaatimusmäärittely 2](#_Toc7376288)

[Kuvaus 2](#_Toc7376289)

[Toiminalliset vaatimukset 2](#_Toc7376290)

[Projektisuunnitelma 3](#_Toc7376291)

[Käyttöohjeet 4](#_Toc7376292)

[Tapahtuman ja sarjojen hallinta 5](#_Toc7376293)

[Ilmoittautuneet 6](#_Toc7376294)

[Kilpailutila 7](#_Toc7376295)

[Tulostila 8](#_Toc7376296)

[Ohjelmiston rakenteen ja toiminnan kuvaus yleisellä tasolla 9](#_Toc7376297)

[Ohjelman ominaisuudet ja käsitteet 9](#_Toc7376298)

[Ohjelmiston rakenne 10](#_Toc7376299)

[Tarkempi kuvaus sovelluslogiikasta 11](#_Toc7376300)

[Keskeisten toimintojen tarkempi kuvaus 13](#_Toc7376301)

[Testaus 16](#_Toc7376302)

[Kehitys ja itsereflektio 16](#_Toc7376303)

[Työnkulku ja pohdintaa 17](#_Toc7376304)

[Työaikakirjanpito 19](#_Toc7376305)

[Vertaisarviot 20](#_Toc7376306)

[Vertaisarvio Budjetointiohjelma 20](#_Toc7376307)

[Vertaisarvio Jari Kuivaniemen Kirjanpito-projektista 20](#_Toc7376308)

## Päivitetty vaatimusmäärittely

### Kuvaus

SportEventResults on tulospalveluohjelma urheilutapahtumiin. Ohjelma on tarkoitettu käytettäväksi lähinnä massakuntotapahtumien osanottajien ja tulostietojen hallintaan. Ohjelmassa on kaksi tilaa. Valmistelutilaa käytetään, kun kilpailut eivät ole käynnissä. Tällöin voidaan syöttää ja tarkastella kilpailuiden tietoja, kuten osanottajatietoja. Kilpailutilassa taas ohjelmaan syötetään maaliin tulevien kilpailijoiden numeroita ja heille saatuja aikoja. Maalintulleista osallistujista voidaan muodostaa tuloslistoja sarjoittain.

### Toiminalliset vaatimukset

Ohjelmassa täytyisi voida tehdä seuraavia asioita.

Valmistelutila

* Kilpailuiden lisääminen järjestelmään
* Sarjojen lisääminen kilpailuihin
  + Sarjaa ei pidä voida poistaa, jos siinä on osanottajia
  + Sarjoihin liittyvä tilastointi. Montako osanottajaa sarjassa ja jotain sellaista
* Osanottajien lisääminen ja poistaminen
  + Tarkastus onko sama kilpailija jo järjestelmässä.
  + Onko joku jo samalla numerolla
  + Pakolliset kentät
* Kilpailun osanottajien tarkastelu sarjoittain
  + Osanottajatietojen muokkaus
  + Osanottajien etsiminen nimen tai numeron perusteella

Kilpailutila

* Tuloksen lisääminen kilpailijalle numeron perusteella (myös keskeyttämiset)
* Väärinsyötettyjen rivien korjaaminen ja poistaminen.
* Tulosten tarkastelu sarjoittain (myös keskeneräisten), tulosten järjestäminen.
* Maaliin saapumattomien näyttö.

Suunnitellut lisäominaisuudet

* Osanottolistojen ja tulosten tulostaminen.
* Osanottolistojen ja tulosten muuntaminen html-muotoon ja reaaliaikainen julkaisu netissä.
* Ohjelmaan integroitu ajanotto.
* Tulosten syöttö käsin syöttämisen sijaan RFID-lukijan avulla.
* Ratakartta
* Tietojen tuominen ohjelmaan esim. Excelistä
* Ohjelman käyttäjä saisi itse määrittää esim. pakolliset kentät

### Projektisuunnitelma

Jokaisessa vaiheessa lisätään toiminnallisuuksia ja tehdään niihin liittyvät testit.

Ensimmäinen vaihe

* Tietokanta-ratkaisun miettiminen
* Kilpailuiden, sarjojen ja osanottajien lisääminen valmistelutilassa

Toinen vaihe

* Alustavan käyttöliittymän rakentaminen
* Valmistelutilan perustoiminnallisuuksien viimeistely
  + Lähtölistat sarjoittain
  + Kilpailun yhteenveto: sarjat + osanottajat

Kolmas vaihe

* Tulosten syöttömahdollisuus kilpailutilassa

Neljäs vaihe

* Kilpailutilan perustoiminnallisuuksien viimeistely

Viimeinen vaihe

* Perustoimintojen viimeistely
* Käyttöliittymän hiominen
* Mahdollisesti lisäominaisuuksien lisääminen

## Käyttöohjeet

SportEventResults:lla voit hallita järjestämäsi urheilutapahtuman osanottaja- ja tulostietoja. Ohjelma on suunniteltu helpottamaan lähinnä massakuntotapahtumien järjestämistä, mutta sitä voi käyttää minkä tahansa urheilutapahtuman hallintaan, missä suoritusten tuloksena on aika.

**Ohjelmassa on neljä erilaista tilaa:**

1. **Tapahtuma. Tapahtuman ja sarjojen hallinta.**
2. **Ilmoittautuneet. Ilmoittautuneiden hallinta.**
3. **Kilpailutila**
4. **Tulokset**

Tapahtuman ja sarjojen hallinta:Tässä tilassa voit muokata tapahtuman tietoja, kuten tapahtuman nimi ja aika, sekä lisätä ja poistaa sarjoja.

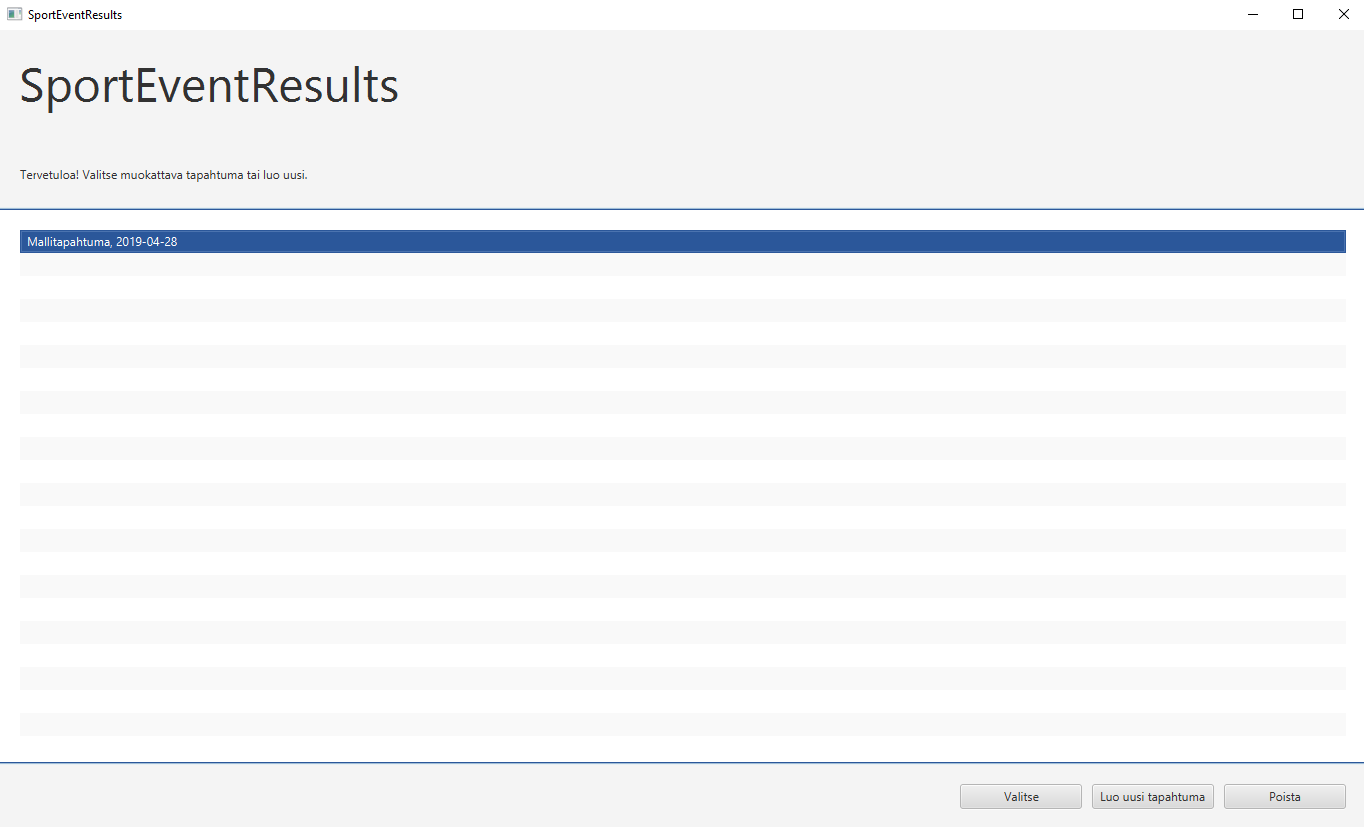
Ilmoittautuneiden hallinta: Tässä tilassa voit lisätä ja poistaa kilpailijoita sarjoihin.

Kilpailutilassa voit syöttää maaliin tuleville aikoja ja tarkistella maaliin tulleita ja vielä maaliin saapumattomia

Tulostilassa voit tarkastella ja tulostaa tuloslistan.

Ohjelman mukana tulee mallitapahtuma, jota voi muokata ja kokeilla. Mallitapahtumassa tapahtumaan on syötetty muutama sarja ja osanottajia. Ketään ei kuitenkaan ole vielä merkitty maaliin, eikä tuloslistassa näy kuin tapahtuman ja sarjojen tiedot.

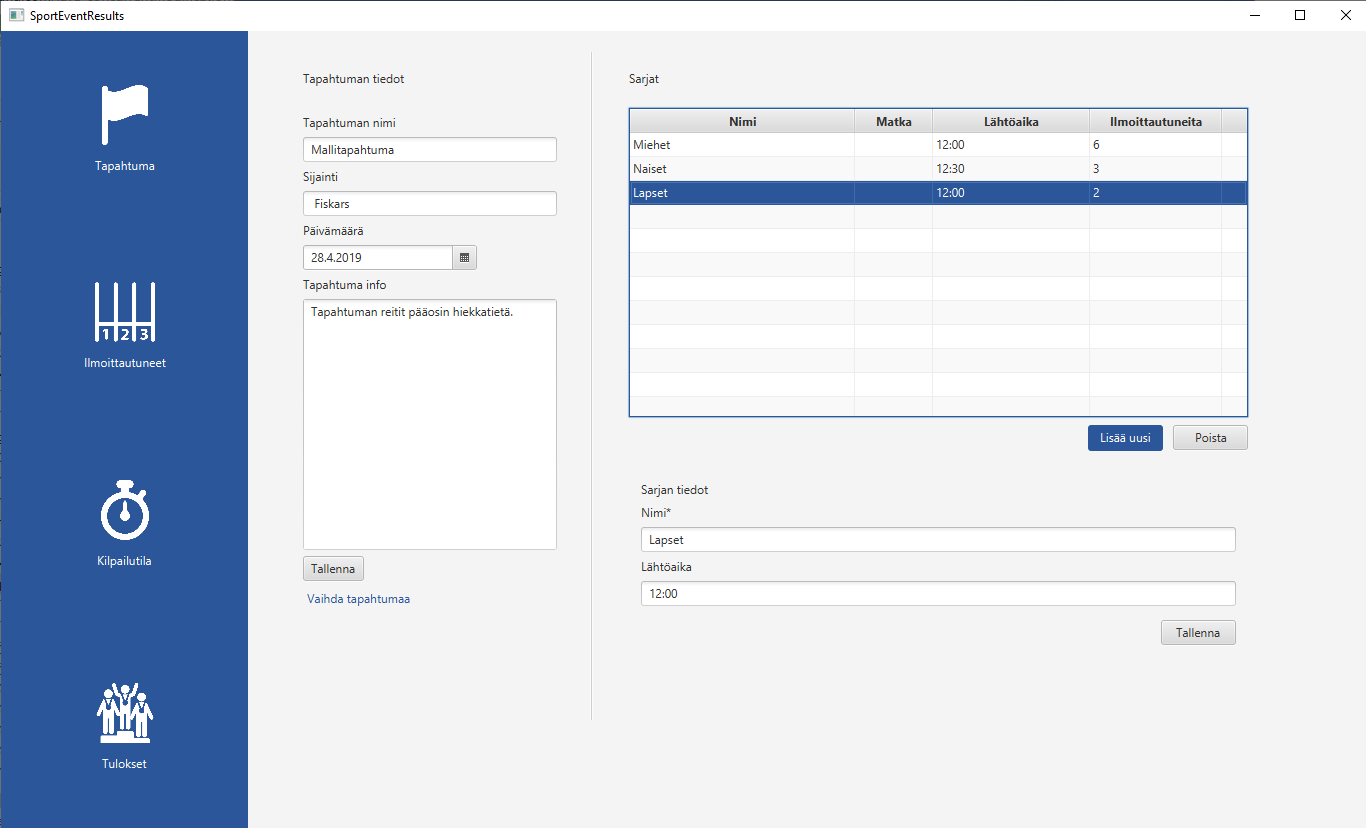
Ohjelma sisältää kaksi tiedostoa: SportEventResults.jar ja eventresultdb.mv.db. Siirrä nämä tiedostot haluamaasi sijaintiin koneellasi, esimerkiksi Programm Files -kansion alle omaan kansioonsa. Ohjelma käynnistyy kaksoisklikkaamalla SportEventResults.jar -tiedostoa.



Kun ohjelma käynnistetään, tulee ensimmäisenä vastaan ruutu, josta voi valita käsiteltävän tapahtuma. Listassa näkyy kaikki ohjelmaan tallennetut tapahtumat. Mikäli yhtään tapahtumaa ei ole, on lista tyhjä. Tapahtuman voi valita muokattavaksi ”muokkaa”-napista tai kaksoisnäpäyttämällä tapahtuman nimeä. Voit myös luoda kokonaan uuden tapahtuman painamalla ”luo uusi” tai poistaa listan tapahtuman klikkaamalla ”poista”.

Tämän jälkeen päästään varsinaiseen tapahtuman hallintaan, jossa navigoidaan ikkunan vasemmassa reunassa olevan valikon avulla. Valikon kautta pääsee ohjelman neljään eri tilaan. Tapahtuma ja Ilmoittautuneet tiloissa valmistellaan tapahtumaa ennen sen varsinaista alkua ja kilpailutilassa syötetään aikoja kilpailun aikana.

### Tapahtuman ja sarjojen hallinta



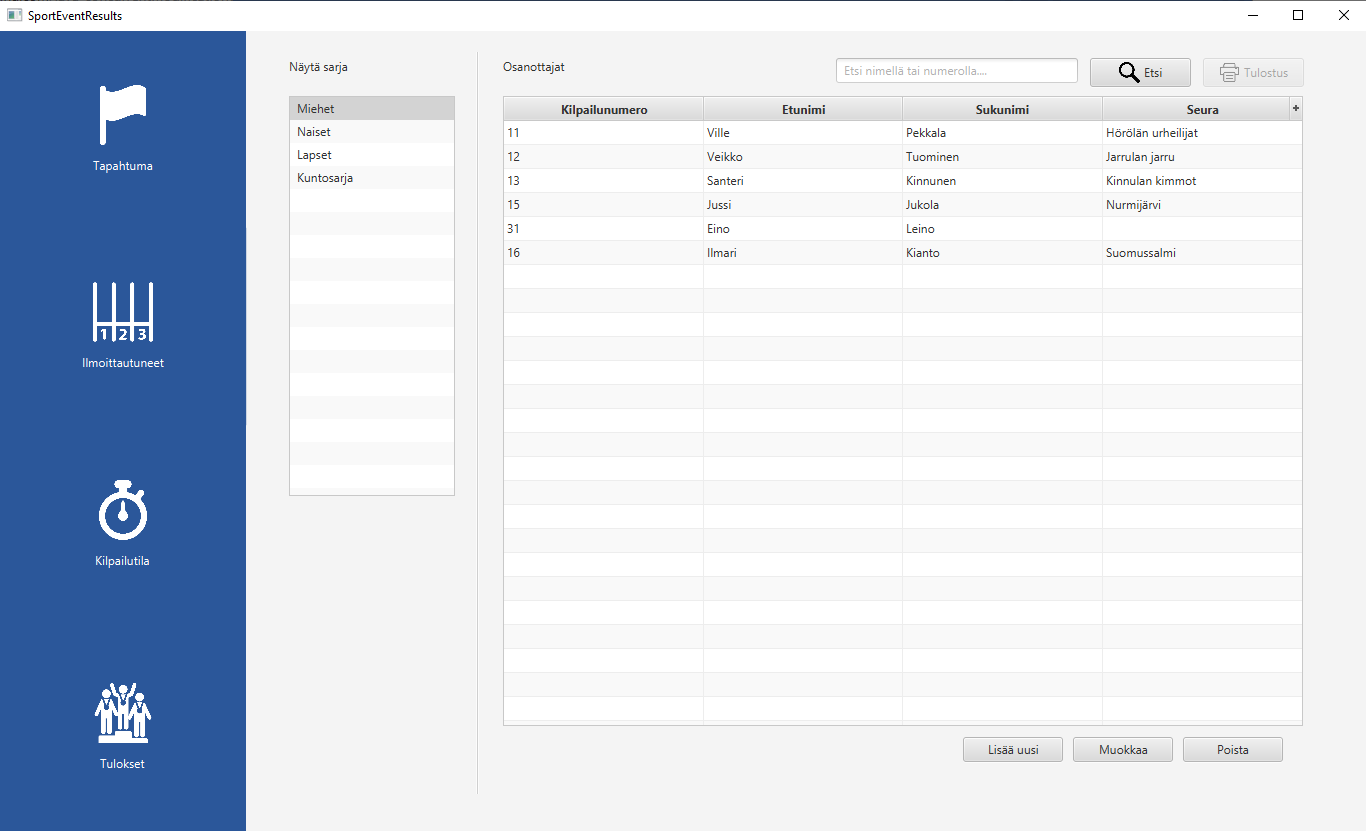
Tapahtuman hallinta tilassa voit muokata tapahtuman tietoja, kuten tapahtuman nimi- ja aikatietoja, sekä lisätä ja poistaa sarjoja.

Näkymä on jaettu kahteen osaan. Vasemmalla puoliskolla on tapahtuman yleiset tiedot ja oikealla tapahtumaan liittyvien sarjojen tiedot. Muutokset tallennetaan painamalla tallenna nappia. Huomaa, erilliset napit tapahtuman tiedoille ja sarjan tiedoille.

Voit lisätä uusia sarjoja ”Lisää uusi”- napista. Sarjan nimen ja lähtöajan pääsee muokkaamaan sarjalistan alapuolella olevista kentistä. Näissä kentissä näkyy kulloinkin valittuna olevan sarjan tiedot. Sarjan nimi voi sisältää esimerkiksi sarjan nimen, kuten ”Naiset yleinen” ja esimerkiksi tiedon matkasta esim. ”Miehet kunto 20km”.

Huom! Et voi poistaa kilpailusta sarjoja, joihin liittyy osanottajia. Poista ensin sarjan osanottajat ja vasta sen jälkeen sarjan poistaminen on mahdollista.

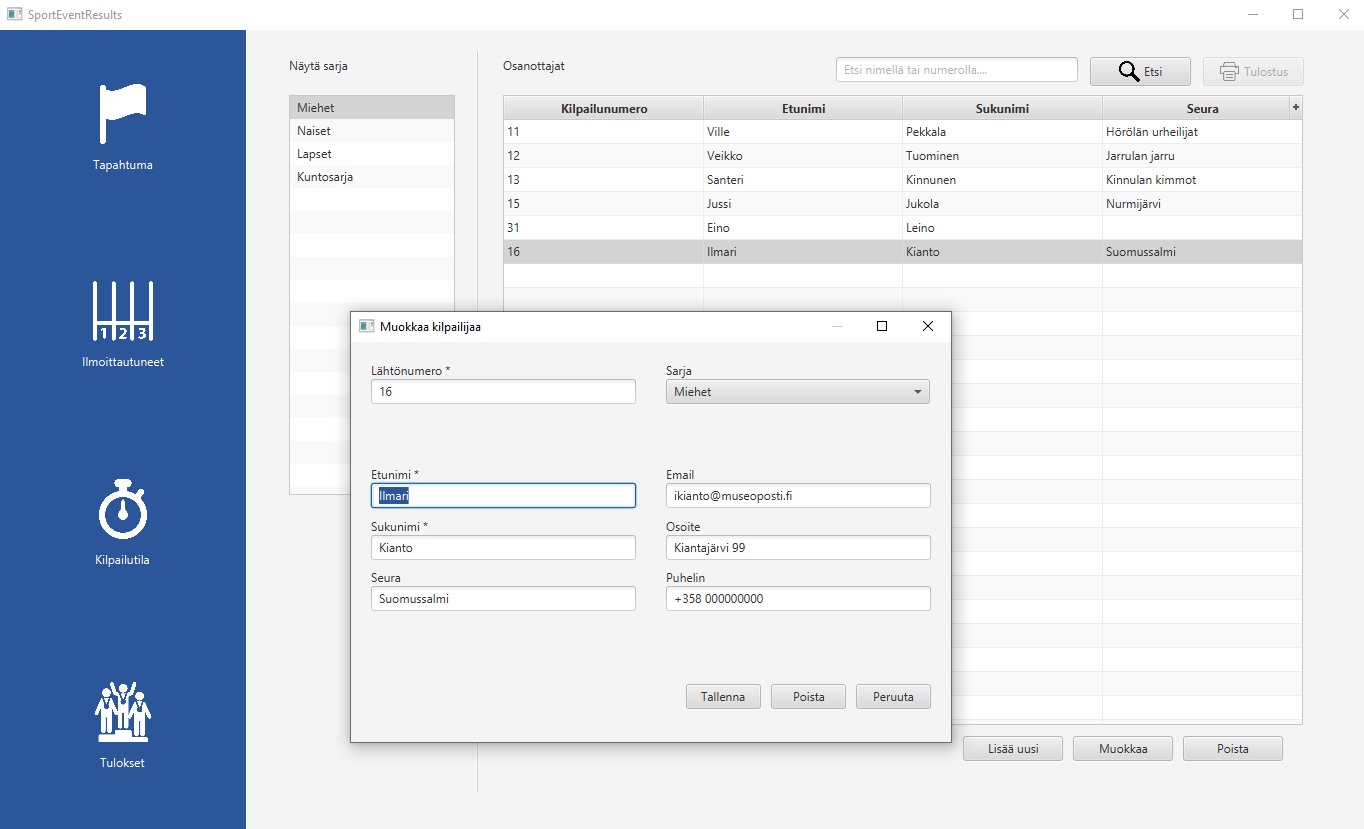
### Ilmoittautuneet



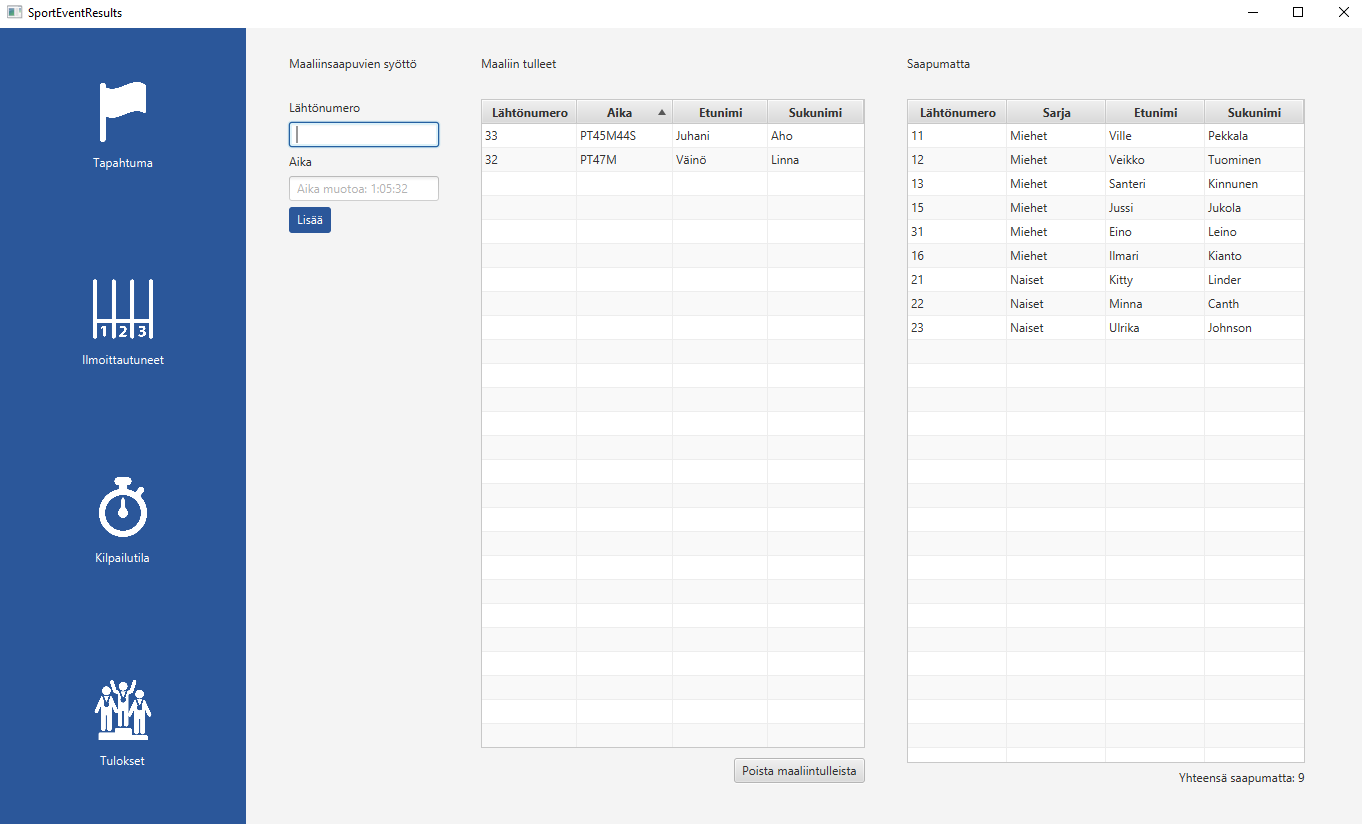
Kun tapahtumaan on lisätty sarjoja, pääsee niihin lisäämään osanottajia Ilmoittautuneet tilassa. Näkymä on jaettu kahteen osaan. Vasemmalla on lista, josta valitaan minkä sarjan osanottajia tarkastellaan ja oikealla näkyy tähän sarjaan kuuluvat osanottajat.

Voit lisätä uusia kilpailijoita valitsemalla ”Lisää uusi” tai voit muokata jo olemassa olevia kilpailijoita valitsemalla ”muokkaa” tai kaksoisnäpäyttämällä kilpailijan riviä. Myös kilpailijoiden poistaminen tapahtuu tässä tilassa.

Kilpailijatietojen muokkaamiseen avautuu uusi ikkuna. Jokaiselle kilpailijalle on annettava vähintään nimi, kilpailunumero ja sarja. Huom! Kussakin tapahtumassa voi olla sama kilpailunumero käytössä vain yhdellä kilpailijalla.

****

### Kilpailutila



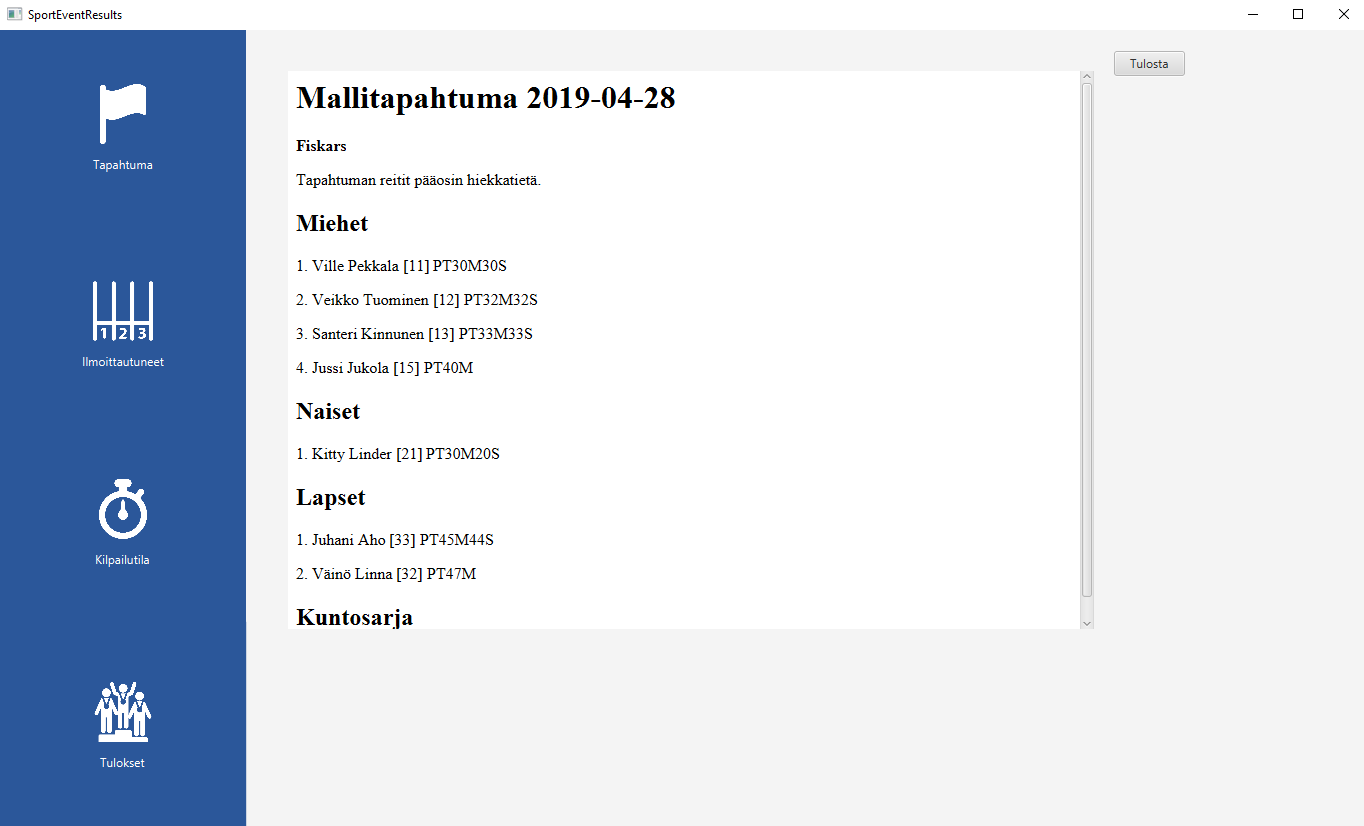
Kilpailutilassa näytetään kaikki tapahtumaan osallistuvat osanottajat. Osanottajat on lajiteltu kahteen taulukkoon. Toisessa taulukossa näytetään kilpailijat, jotka eivät vielä ole saapuneet maaliin ja toisessa taulukossa maaliin tulleet. Lisäksi näytetään maaliin saapumattomien määrä, jotta järjestäjät pysyvät tapahtuman kulusta helpommin selvillä.

Maaliin saapuvien syöttö: Kilpailujen ajanotto on järjestettävä manuaalisesti esimerkiksi sekuntikellolla, joka laitetaan käyntiin sarjan startatessa. Kun kilpailijoita saapuu maaliin, täytyy maalipaikalla olla tapahtuman järjestäjiä seuraamassa saapuvien osanottajien kilpailunumeroita ja kirjaamassa kellotettuja aikoja järjestelmään.

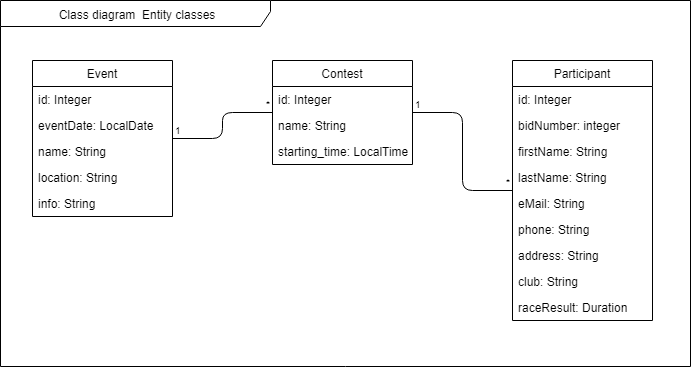
Kilpailutilan ”Maaliin saapuvien syöttö” otsikon alla oleviin kenttiin syötetään maaliin saapuvan osanottajan kilpailunumero/ lähtönumero ja aika joka kellolla saadaan. Aika täytyy kirjoittaa muodossa mm:ss tai hh:mm:ss. Syöte tallentuu, kun molemmat kentät on täytettynä ja painetaan ”Lisää” -nappia. Vaihtoehtoisesti myös Enterin painaminen näppäimistöltä tallentaa syötteen. Kun syöte on tallennettu, kohdistuu syöttö jälleen lähtönumero ruutuun. Näin tietojen syöttö sujuu nopeasti. Syötetään numero, vaihdetaan kenttää hiirellä tai näppäimistön Tab-näppäimellä aika kenttään ja painetaan Enter. Ja sama uudelleen sitä mukaan kun osanottajia saapuu maaaliin.

### Tulostila

Tulostilassa ohjelma muodostaa tuloslistan sen hetkisten maaliin saapuneiden perusteella. Voit viedä tuloslistan ulos ohjelmasta esimerkiksi tekstinkäsittelyohjelmaan tai nettisivuille kopioimalla tuloskentän tekstit hiirellä vetämällä ja painalla hiiren kakkospainiketta ja valitsemalla copy tai painamalla näppäimistöltä Ctrl+C.



## Ohjelmiston rakenteen ja toiminnan kuvaus yleisellä tasolla



Kuva Diagrammi kuvaa ohjelman kolmea entiteettiluokkaa, jotka toimivat myös tietokannan tauluina.

### Ohjelman ominaisuudet ja käsitteet

Ohjelman käynnistyessä käyttäjä valitsee urheilutapahtuman, jota haluaa muokata tai luo uuden tapahtuman. Urheilutapahtumaa ohjelmassa mallinnetaan käsiteluokalla Event.

***Tapahtumaolioilla*** *(*Event*) on attribuutteinaan nimi, päivämäärä, tapahtumapaikka, sekä vapaamuotoista lisätietoa.*

Tapahtumassa on tyypillisesti useita kilpailusarjoja, joita kuvataan luokalla . Uusia sarjoja voi lisätä ja muokata tapahtumanäkymässä (EventView), jossa tapahtuu myös tapahtuman tietojen muokkaus.

***Sarjaolioilla*** *(*Contest*) on nimi, alkamisaika ja tapahtuma, johon sarja liittyy.*

Jokainen tapahtuman osanottaja liittyy aina johonkin tapahtuman sarjaan. Osanottajia hallitaan näkymässä, jossa osanottajat listataan sarjoittain (ListParticipantsView). Käyttäjä voi valita näytettävän sarjan. Sarjaan liittyvät osanottajat näytetään taulussa, jossa näkyy myös tärkeimmät osanottajaan liittyvät tiedot. Tarkemmat tiedot saa näkyviin valitsemalla yhden osanottajan muokattavaksi (SinglePersonView). Sarjakohtaiset osanottajalistat voi myös tulostaa.

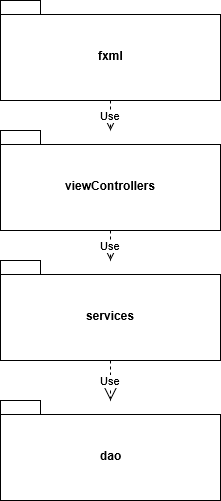
***Osanottajalla*** *(*Participant*) on attribuuttinaan nimen ja yhteystietojen lisäksi kilpailunumero.*

Yhdessä kilpailussa samaa numeroa ei voi olla kahdella kilpailijalla. Yhden osanottajan muokkaus tapahtuu uudessa ikkunassa, jossa tiedot voi tallentaa tai kilpailijan voi poistaa, samaa ikkunaa käytetään myös uuden kilpailijan luomiseen.

Kilpailutilassa (RaceView) tapahtumaan liittyvät osanottajat jaotellaan maaliin tulleisiin ja vielä maaliin saapumattomiin. Käyttäjä syöttää maaliin saapuvan kilpailunumeron, sekä ajan, joka osanottajalla on kestänyt suorituksessa. Tällöin osanottaja siirretään maaliin tulleisiin. Näkymässä voi myös siirtää osanottajia takaisin saapumattomiin, mikäli tietojen syöttämisessä on esimerkiksi tapahtunut virhe. Näkymässä näkyy myös saapumattomien lukumäärä.

Lopulta tapahtuman tuloksia voidaan tarkastella, tulostaa ja viedä ohjelmasta ulos tulosnäkymässä (ResultView).

### Ohjelmiston rakenne

Kyseessä on Windows-työpöytä sovellus. Ohjelman arkkitehtuuri noudattaa MVC-mallia, ja näkymät, niiden kontrollerit ja muut ohjelman toiminnallisuudet, on erotettu toisistaan. **Kerrosarkkitehtuurimaisesti** kukin toiminnallisuus on eriytetty omiin kerroksiinsa, jotka ovat entiteettiluokkia lukuun ottamatta yhteydessä vain viereisiin kerroksiin.

Kuva Ohjelma noudattaa kerrosarkkitehtuuria. Pakkausdiagrammissa näkyvät ohjelman kerrokset ja niiden riippuvuudet.

Käyttöliittymä on toteutettu **JavaFX**:llä ja käyttöliittymän näkymät on määritetty **FXML**-pohjaisesti (view(fxml)). Fxml-tiedostojen luonnissa on käytetty apuna Scene Builder ohjelmaa. Lisäksi ulkoasua on määritelty hieman **css**-tyylitiedostossa.

Jokaista näkymää vastaa yksi ohjausolio (pakkaus: viewControllers), joka vastaa kyseiseen näkymään liittyvästä toiminnallisuudesta, kuten tapahtumien käsittelystä, ja joka ohjaa pyyntöjä Service-luokille, jotka puolestaan ovat yhteydessä tietokantaa käsitteleviin luokkiin (pakkaus: dao).

Ohjelman service-oliot (pakkaus: services) tarjoavat käsiteolioihin liittyviä palveluita. Service- ja Utility -luokilla on pyritty keventämään kontrolleriluokkien työkuormaa, sekä mahdollistamaan sujuvampi siirtyminen mahdollisiin muihin käyttöliittymiin, mikäli ohjelmaa haluttaisiin kehittää esimerkiksi toimimaan web-käyttöliittymän kautta.

Tietokantatoiminnat on eristetty CRUD toiminnallisuuden tarjoaville dao-luokille. Ohjelma käyttää **H2-relaatiotietokantaa** JDBC-rajapinnan ja -ajureiden avulla. Tietokannan taulut vastaavat ohjelman entiteettiluokkia (pakkaus: domain).

## Tarkempi kuvaus sovelluslogiikasta

**Näkymät:** Kukin ohjelman toiminnallisuus tapahtuu omassa näkymässään, jonka komponentit on määritelty fxml-tiedostoissa.

* ChooceEvent
* EventView
* ListParticipantsView
* SinglePersonView
* RaceView
* ResultsView

Näiden lisäksi käytössä on MainView, jossa määritellään näkymien valintaan käytettävä valikko.

**Kontrollerit:** Näkymiin liittyviin kontrolleriluokkiin on sisällytetty käyttöliittymään liittyviä toimintoja ja ne on pyritty pitämään mahdollisimman selkeärajaisina. Kun kontrolleriolio käynnistyy, ajetaan ensimmäisenä initialize-metodi, jossa edelleen kutsutaan metodeja, jotka hoitavat tietojen lataamisen näkymiin ja määritetään metodit, jotka ajetaan käyttäjän painaessa jotain nappia. Näissä metodeissa toiminnallisuus on tietojen hakeminen service-olioilta, sekä näkymän päivittäminen. Näissä luokissa olevat metodit eivät palauta tyypillisesti mitään.

Kontrolleritason luokat ja metodit ovat:

* EventViewController
  + initialize
  + setSelectedEvent(Event event)
  + populateEventFields(Event event)
  + saveEvent()
  + populateContestTable()
  + populateContestFields(Contest contest)
  + addNewContest()
  + updateContest()
  + deleteContest()
  + validateFields()
* ListParticipantsViewController
  + initialize
  + setSelectedEvent(Event event)
  + refreshParticipantTable()
  + populateContestList()
  + populateParticipantTable(List<Participant> participants)
  + deleteParticipant()
  + showSinglePersonViewWindow(Participant participant)
* SinglePersonViewController
  + initialize
  + populateFields(Participant participant)
  + saveParticipant(Event event)
  + deleteParticipant(Event event)
  + boolean participantFieldsAreValid()
* RaceViewController
  + initialize
  + populateTables()
  + addToFinished()
  + removeFromFinished()
  + boolean fieldsAreValid()
* ResultsViewController
  + initialize
* MainViewController
* DialogUtil
  + boolean promptDelete(String message)
  + showErrorDialog(String errorMessage)
  + closeWindow(Event event)

**Service-tason** oliot toimivat kontrollerien ja tietokantatoimintojen välissä. **Käytettävät service-oliot tarjoavat nimenomaan käsiteolioihin liittyviä palveluita.** Näistä oleellisin luokka on ParticipantService, joka hoitaa useita välivaiheita käyttöliittymän ja dao-luokkien välillä. Useissa tapauksissa metodit ottavat vastaan käyttöliittymän lähettämät, käyttöliittymän kentistä kerätyt, String muotoiset parametrit, ja muodostavat näistä käsiteolioita tietokantatason käsiteltäviksi.

Servite-tason luokat ja metodit:

* EventService
  + Event findById(Integer id)
  + update(Event event)
* ContestService
  + boolean delete(Contest contest)
  + boolean update(Contest contest)
  + Integer addNew(String name, String startingTime)
  + List<Contest> findAll()
* ParticipantService
  + List<Participant> findByNameOrNumber(String searchWord)
  + List<Participant> findByContest(Contest contest)
  + delete(Participant participant)
  + Integer save(Integer id, String bidNumber, String firstName,

String lastName, String email, String phone, String address,

String club, Contest contest)

* + addToFinished(Participant participant, String durationString)
  + update(Participant participant)
  + List<Participant> findAll()
  + boolean bidNumberAlreadyInUse(String bidNumber, Participant participant)
  + Duration parseDuration(String original)

**Dao-luokat:** Tietokantaan kirjoittamisesta, lukemisesta, poistamisesta ja päivittämisestä vastaavat dao-tason oliot. Kutakin käsiteluokkaa kohden on oma dao-luokkansa. Näiden lisäksi pakkaukseen kuuluu DaoUtil, jota muut luokat käyttävät. getConnction palauttaa yhteyden tietokantaan, setValues valmistelee PreparedStatementin ja initialize luo ohjelman käyttämät tietokantataulut, mikäli niitä ei vielä ole olemassa (CREATE TABLE IF NOT EXISTS).

Dao-tason luokat ja metodit:

* EventDaoJdbc
  + Integer create(Event event)
  + update(Event event)
  + findById(Integer id)
  + delete(Integer key)
  + Event prepareEvent(ResultSet rs)
* ContestDaoJdbc
  + Integer create(Contest contest)
  + update(Contest contest)
  + delete(Integer key)
  + Contest findById(Integer key)
  + List<Contest> findAll()
  + Contest prepareContest(ResultSet rs)
* ParticipantDaoJdbc
  + Integer create(Participant participant)
  + update(Participant participant)
  + delete(Integer key)
  + Participant findById(Integer key)
  + List<Participant> findByNameLike(String name)
  + Participant findByBidNumber(Integer bidNumber)
  + List<Participant> listAll()
  + List<Participant> listByContest(Contest contest)
  + Integer countByContest(Contest contest)
  + Participant prepareParticipant(ResultSet rs)
* DaoUtil
  + Connection getConnection()
  + setValues (PreparedStatement preparedStatement, Object... values)
  + initialize()

Yksityiskohtaisempi metoditason kuvaus toiminnasta löytyy erillisestä JavaDoc-dokumentaatiosta.

### Keskeisten toimintojen tarkempi kuvaus

Ohjelman toiminta keskittyy tietokannan käytön ympärille. Käyttäjä tallentaa tietoja kantaan ja ohjelma hakee kannasta oleelliset tiedot kuhunkin näkymään. Muu toiminnallisuus enimmäkseen tukee tietokannan käyttöä ja syötteiden vastaanottoa. **Keskeisimpiä tehtäviä ovat sarjaan liittyvien osanottajien haku, osanottajien haku numeron tai kilpailunumeron perusteella, kilpailijoiden lisääminen ja ajan syöttäminen, sekä tapahtumien ja sarjojen hallinta.**

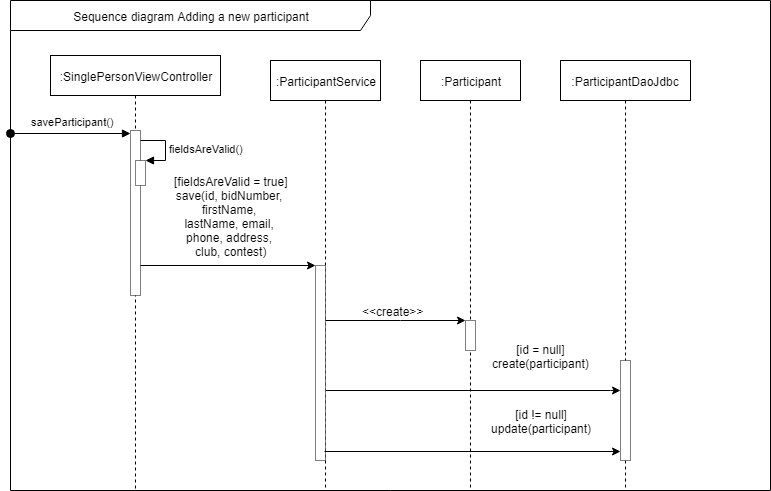
**Liikkuminen näkymien välillä:** Eri näkymien välillä siirrytään käyttäjän painaessa näkymän lataavaa nappia MainView:ssä ja ChooseEventView:ssä. Näkymien kontrollereille välitetään käytössä oleva tapahtuma(event) näkymän resurssien lataamisen yhteydessä. Tämä tapahtuu hakemalla ladatun näkymän kontrolleri, esimerkiksi:

fxmlLoader.<MainViewController>getController().setSelectedEvent(event);

ja kutsumalla kontrollerien metodia setSelectedEvent(Event event). Näin näkymät osaavat näyttää oikeaan tapahtumaan liittyvät sarjat(contest) ja niihin liittyvät osanottajat. Kontrollereiden initialize-metodi ajetaan jo resurssien lataamisen yhteydessä, joten setSelectedEvent tapahtuu vasta sen jälkeen. Jotta näkymän tiedot vastaisivat käytössä olevaa tapahtumaa, tässä metodissa tapahtuu myös tietojen lataaminen käyttöliittymäkomponentteihin.

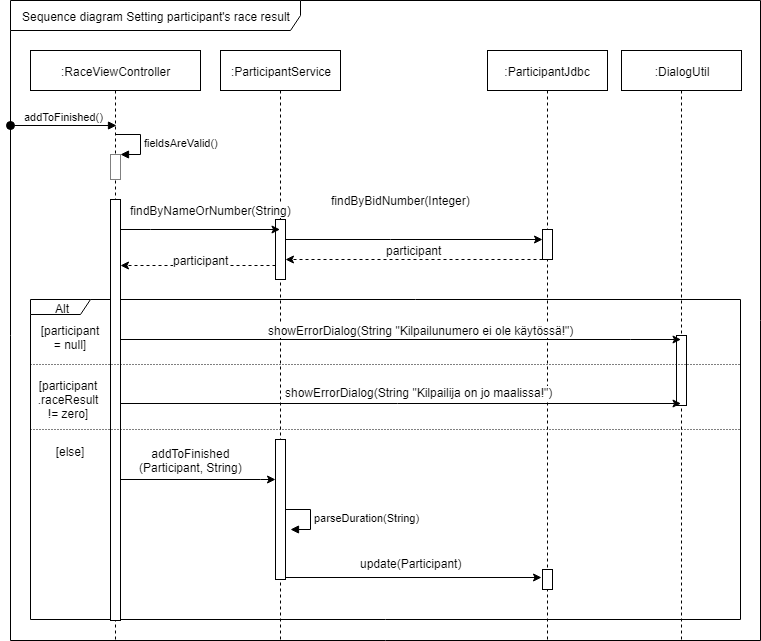
Saman tyyppistä tietojen välitystä käytetään, kun ListPartisipantViewissä käyttäjä valitsee muokattavan osanottajan ja tämän tiedot esiladataan populateFields(Participant participant) -metodilla.

**Uuden kilpailijat luominen:** Uutta kilpailijaa luodessa tarkistetaan ensin kenttien arvot kontrolleriluokassa (metodi participantFieldsAreValid), lähetetään sitten arvot kilpailijasta vastaavalle ParticipantService-luokalle ja sen metodille save, joka luo arvoja vastaavan osanottajan. Mikäli osanottajalla ei ole vielä id-attribuuttia, eli sitä ei ole tietokannassa, tallennetaan tietokantaan uusi osanottaja. Muussa tapauksessa päivitetään olemassa oleva osanottaja saatujen arvojen mukaiseksi.



Kuva Sekvenssi-diagrammi uuden osanottajan lisäämisestä järjestelmään

**Kilpailijan ajan syöttäminen:** Toisena erimerkkinä osanottajan tullessa maaliin käyttäjä syöttää osanottajan kilpailunumeron ja tälle kellotetun ajan ja painaa ”Lisää” nappia. Ensin tarkistetaan kenttien syötteiden oikeellisuus fieldsAreValid()-metodissa. Seuraavakasi haetaan participantServiceltä syötettyä numeroa vastaava osanottaja. Varmistetaan, että kilpailunumero on käytössä ja ettei osanottaja ole jo merkattu maaliin, muuten kutsutaan ParticipantServicen addToFinished-metodia. Seuraavaksi koetetaan muuttaa aikakentän String-muotoinen arvo Duration-muotoiseksi. Jos muunnos onnistuu, asetetaan osanottajan kilpailuaika vastaamana sitä ja kutsutaan dao-kerroksen metodia tietojen tallentamiseksi tietokantaan. Lopulta päivitetään näkymä vastaamaan uutta tilannetta.



Kuva Sekvenssi-diagrammi ajan asettamisesta osanottajalle

**Syötteiden validointi:** Käyttäjän antamat syötteet validoidaan tarkastamalla muodon oikeellisuus, sekä mm. varmistamalla, ettei annettu kilpailunumero ole käytössä tai kilpailija ole jo maalissa. Syötteiden muodon oikeellisuus tarkistetaan kussakin kontrolleriluokassa olevassa fieldsAreValid()-metodissa, joka palauttaa boolean-tyyppisen vastauksen. Epäkelvoista arvoista ilmoitetaan käyttäjälle lähettämällä virheteksti yleiskäyttöisen DialogUtil-luokan showErrorDialog(String)-metodille, joka näyttää havaitun virheen JavaFX:n Alert-ponnashdusikkunassa.

**Vaaralliset toiminnat:** Jos käyttäjä on poistamassa jotain, kuten osanottajaa tietokannasta, varmistetaan asia käyttäjältä DialogUtil-luokan promptDelete(String)-metodissa luotavalla popup-ikkunalla, joka palauttaa true, mikäli käyttäjä haluaa toteuttaa muutoksen (painaa ok) ja muuten false. Mikäli palautettu arvo oli true, lähetetään poisto pyyntö kontrolleriluokasta service-tasolle ja siitä eteenpäin.

ContestService luokan Delete-metodi tarkistaa ettei sarjaan kuulu osanottajia, ennen kuin jatkaa poistopyyntöä. showErrorDialog(String) – ikkuna avautuu informoimaan käyttäjää tilanteesta, mikä osanottajia löytyy.

**Käyttäjien haku:** Etsi toiminnallisuus ListParticipant-näkymässä puolestaan hakee ParticipantService-luokan findByNameOrNumber(String)avulla listan hakuehdot täyttävistä kilpailijoista ja täyttää taulunäkymän näillä tuloksilla.

## Testaus

Testaus on toteutettu JUnitilla ja testikattavuutta tarkastellaan JaCoco:n avulla.

Rivi- ja haaraumakattavuus domain, dao ja service pakkauksien osalta on hyvä, yli 90%. Sen sijaan käyttöliittymä jää vaille testausta. Kutakin metodia testataan tyypillisesti muutamalla assertEqualssilla tai assertTruella. Joitakin metodeja testataan myös null-arvoilla, jolla varmistetaan toiminta tilanteissa, joissa metodi ei jostain syystä saa kunollista parametriä.

Yksittäisten luokkien lisäksi ohjelmaa testataan Servicejen kohdalla hyödyntämällä service-luokkien lisäksi myös dao-luokkia. Service-tasolle tulevat pyynnöt ohjataan dao-luokille ja siitä tietokannalle.

Deo-luokkien testit kirjoitetaan varsinaiseen tietokantaan. Tämä ei toki ole hyvä asia, koska tällöin kantaan voi jäädä ylimääräisiä rivejä, vaikka testeissä pyritäänkin aina poistamaan se mikä on lisättykin. Tietokanta on paikallinen, eikä sen siten pitäisi päätyä muutenkaan käyttäjälle. Lisäksi tällä järjestelyllä tulee testattua todella käytettävän tietokannan toiminta. Joka tapauksessa tietokannan käsittely testeissä on rajoittunutta. Ongelman ratkaisemiseen voisi käyttää jotain tietokantojen testauskehystä.

Näiden lisäksi ohjelmaa on testattu manuaalisesti liikkumalla ohjelmassa tilasta toiseen ja syöttämällä kenttiin yllättäviäkin syötteitä, sekä painelemalla nappeja myös epäloogisessa järjestyksessä. Haluttujen tietojen päätymistä käyttöliittymästä tietokantaan on myös testattu tekemällä syötteet ja seuraamalla tietokannan sisältöä manuaalisesti konsolista. Tätä on tehty myös säännöllisesti koodimuutosten jälkeen.

## Kehitys ja itsereflektio

Ohjelman nykyisessä muodossa maaliin saapuvat syötetään käsin kirjoittamalla maaliin saapuvan kilpailunumero, sekä erillisellä kellolla saatu kesto. Tämä voi tapahtumasta riippuen olla ihan varteenotettava lähestymistapa, mutta jos ohjelmaa lähtisi kehittämään, tulisi toiminto muuttaa siten, että halutessa nämä tiedot tulisivat automaattisesti ajanottolaitteista ja kilpailijan tunnistus tapahtuisi RFID-tekniikan avulla.

Toinen selkeä kehityskohde olisi siirtää painopistettä nettiin niin, että tapahtuman esivalmisteluita pystyisi tekemään selaimen kautta. Samoin tulosten julkaiseminen olisi hyvä saada virtaviivaiseksi ja reaaliaikaiseksi. Tätä ajatusta lähdin valmistelemaan jo rakentamalla tulokset html-muotoiseksi. Tämä voisi palvella tulosten ohjelmasta ulos saamista, mutta ominaisuus jäi keskeneräiseksi ja rumaksi.

Ohjelma on vähän hidas. Tämä johtuu siitä, että näkymien resurssit ja tietokannan tiedot ladataan joka kerta uudelleen, kun käyttäjä painaa näkymän vaihtavaa nappia.

### Työnkulku ja pohdintaa

Työ sujui varsin sujuvasti suunnitelman mukaan. Koodia tuli aika paljon järjesteltyä uudelleen työn aikana. Vaikka ohjelma on periaatteessa yksinkertainen ja järjestys aika perus, niin niillä taidoilla ja tiedoilla, mitä projektia aloittaessa oli, en usko, että olisin tarkemmallakaan suunnittelulla osannut välttää muodon jatkuvaa hakemista.

Kun ensimmäiseen iteraatioon valmistuessa käsitteiden suhteet ja ohjelman perustoiminnallisuus oli mietittynä, tuntui, että ohjelma alkaakin olla jo valmis. Lopulta aika paljon oli kuitenkin tekemistä ennen kuin ohjelman käyttö oli saatu luontevaksi ja mm. kaikki virhemahdollisuudet käytyläpi.

Iteraatioittain lisättävät ominaisuudet muodostivat mukavia kokonaisuuksia, joita oli selkää lisäillä ohjelmaan. Toiseen iteraatioon sain käyttöön graafisen käyttöliittymän, mikä toisaalta auttoi itseäkin hahmottamaan paremmin mitä tästä on tulossa. Hieman alkuperäisestä suunnitelmasta poiketen tuntui luontevimmalta aloittaa osanottajia koskevista toiminnallisuuksista ja rakentaa sarja- ja tapahtuma- runkoa sitten siihen ympärille.

Kun osanottajien syöttäminen ja sarjojen käsittely oli valmis, olikin aika helppo lähteä lisäämään tulosten syöttöominaisuutta ja tuloslistojen käsittelyä. Lopulta keskeiset alkuperäiset toiminnalliset vaatimukset sain toteutettua ja jotain kaavailemiani lisäominaisuuksiakin, kuten jonkinlaisen tulostusmahdollisuuden.

Aika paljon lueskelin projektin aikana käytännön juttuja JavaFX:n käytöstä, esimerkiksi miten taulukoita täytetään, miten ikkunoita suljetaan jne. Samoin koetin lueskelle ohjelmien rakenteesta myös luentomonisteen ulkopuolelta. UML-kaavioissa oli opeteltavaa myös, mutta niiden piirtely kyllä tuntui hyödylliseltä myös omien ajatusten jäsentelyssä.

Jos lähtisin tekemään projektia uudelleen, ottaisin testaamisen kiinteästi ohjelman kehittämiseen mukaan heti alusta lähtien. Tämä oli tarkoitus nytkin, mutta asia tahtoi jäädä jatkuvasti tekemättä. Huomasin, että testien kirjoittaminen auttaisi kyllä selkeiden metodien kirjoittamisisessa ja toisaalta varmistamaan, että kaikki edelleen toimii, vaikka ohjelmaan tekee muutoksia.

Moni asia jää mietityttämään. Service-taso tässä jää aika ohueksi, joten sen mielekkyys tässä vaiheessa mietityttää. Toisaalta jos hommaa lähtisi jatkokehittämään sen merkitys varmasti kasvaisi. Jonkin sovelluskehyksen käyttöönotto selkeyttäisi ja tehostaisi varmasti joitain asioita. Näkymistä jäin miettimään taas taulukoiden kokonaan uudelleen lataamisen vaihtoehtoja niitä päivittäessä. Muutenkin tietokannasta haetaan turhan usein tietoa. Asian voisi ratkaista jotenkin sujuvamminkin.

Ohjelma on lopulta siinä kunnossa, että uskon sillä voisi periaatteessa olla käytännön käyttöäkin jossain pienemmissä tapahtumissa. Oikeasti mielenkiintoinen tästä kyllä tulisi kuitenkin vasta kun web-puoli ja ajanoton yhdistäminen olisivat mukana.

## Työaikakirjanpito

-> 19.3. 30 tuntia. Kurssin tehtävien tekoa, suunnittelua, tietokantakokeiluja, lukemista.

19.3. 6 tuntia

20.3. 2 tuntia

23.3. 2 tuntia. UML, työohjeita

24.3. 2 tuntia valmistelua palautusta varten. Sarjojen ja osanottajien käsittely alkaa olla hallinnassa.

25.3. 4 tuntia Työskentelyä. Lueskelua suunnittelumalleista ja JavaFX:stä

28.3. 4 tuntia työskentelyä,

29.4 1 tunti vertaisarvio

30.4. 1 tunti pientä näppäilyä

1.4 8 tuntia työskentelyä

3.4. ½ tutia. Lueskelin toisten töitä

4.4. 1 tunti

7.4. 1 tunti Valmistelin palautusta varten. Perusnäkymät, osanottajat, sarjat ja kilpailutila alkaa olla kunnossa. Tuloksetkin näkyy, mutta surkea. JavaDocia löytyy.

10.4. 1 tunti dokumenttia

11.4. 6 tuntia loppudokumenttia, kaavioiden päivitystä, koodin refaktorointia

12.4. ½ tuntia vertaisarvio

18.4. 1 tunti hyödytöntä näppäilyä

19.4. ½ tuntia 4. iteraation valmistelu ja palautus. Alkaa olla valmis. Vähän uusi käyttöliittymä tabeineen.

23.4. 8 tuntia testien kirjoittamista

26.4. 6 tuntia. Lisäsin eventin valinnan. Tabisysteemi pois.

28.4. 10 tuntia Käyttöliittymän hiomista, virheiden korjailua. Ei todellakaan ollut valmis. Loppudokumenttia

## Vertaisarviot

### Vertaisarvio Budjetointiohjelma

Harjoituksen vaatimusmäärittely on selkeä ja tavoitteet on helppo ymmärtää. Aloitettu toteutus myös toteuttaa tavoitteita siltä osin, mitä tähän mennessä on ehditty tekemään. Etenemissuunnitelman mukaisesti ainakin ensimmäisen iteraation tavoitteet on saavutettu. Koodia on ryhmitelty selkeästi luokkiin ja pakkauksiin. Käyttöliittymä on selkeästi erotettu muusta toiminnallisuudesta.

Tekstikäyttöliittymää on toki vähän työläs käyttää ainakin budjettia valmistellessa, eli graafista toteutusta jään odottamaan. Ja kuten suunnitelmissa näkyy olevankin, niin tästä aineistostahan on mukava piirrellä piirakoita ja pylväitä.

En tiedä kannattaako noita kategorioita määritellä enumeina. Entä jos ne olisivat ihan perus taulu tietokannassa, josta ne aina luetaan listaan? Helppo sinne olisi lisätä uusia kategorioita tai poistaa käyttäjän halutessa, jos tällaista ominaisuutta haluaa.

Olen itse painiskellut hyvin samanlaisen ratkaisun kanssa, kuin tässä on BudgetingApp-luokka. Jotenkin tuntuu, että joku tällainen voisi hyvä olla olemassa erottamassa käyttöliittymää ja dao-luokkia. Toisaalta BudgetingApp ei ainakaan vielä tee oikein mitään. Se vain välittää pyynnöt samanlaisina eteenpäin.

Vähän omaa tapahtumien älyämistä vaikeutti, kun luokkaa Budget vastaakin taulu, jonka nimi on BudgetLimit, BudgetLimit taas kuulostaa joltain liitostaululta. BudgetingApp.balance() -metodista tuli mieleen, että on sanottu, että mitään minkä tietokanta voi toteuttaa ei kannattaisi tehdä erikseen ohjelmassa. Nyt tämä metodi hakee tietokannasta ensin expencet, sitten budgetit ja sitten vielä ohjelmallisesti käy ne läpi. Joku tämmöisestä kehitetty systeemi varmaan toimisi nopeammin: SELECT SUM(Expence.amount) jne. Mutta ei kai näissä mitään väärää ole. Kunhan mietin.

JavaDoceja ei ole, eikä muutakaan dokumentointia. Testejäkin on heikonlaisesti. Muuten asiallinen homma. Jotain tästä copy-pastesinkin omaan juttuuni ja opin ehdollisen sijoituslauseen olemassaolosta Javassa.

### Vertaisarvio Jari Kuivaniemen Kirjanpito-projektista

Ohjelma on jaettu järkevän oloisiin luokkiin. On käsiteluokkia, repository-luokkia, ja käyttöliittymä-luokkia. Selkeyden vuoksi tiedostot voisi varmaan jakaa jotenkin pakkauksiin. Ainakin jos ohjelma tästä vielä laajenee. Olisiko se sitten toiminto kohtaisesti vai kerrosmaisesti vai jotenkin muuten.

Koodi on pintapuolisesti selkeän näköistä ja toiminnallisestikin seurattavissa. MainApp huolehtii kulloinkin halutun näkymän hallinnoinnista ja hakee halutun lomakkeen niistä vastaavalta luokalta. Itse lomakeluokissa aina määritellään käyttöliittymän komponentit ja napeille määritellään toiminta. Joitain mammuttimaisia metodeja näistä luokista löytyy. Taulukko voisi olla parempi toteuttaa jollain valmiilla taulukko-komponentilla, kuin GridPanella. Nappeja on paljon, eikä niiden kaikkien tarpeellisuus avaudu. Ehkä osa onkin vain kehitystä varten?

Virhetilanteisiin varaudutaan kiitettävästi try-catcheillä ja nullien tarkastuksilla. Väärät syötteet tarkistetaan myös.

Eipä juurikaan dokumentointia Javadocien muodossa, automaattisesti generoitujen juttujen lisäksi, enkä mitään mutakaan dokumentaatiota ainakaan löytänyt. Harmi, sillä ohjelman hahmottaminen ainakin itselleni kaipaisi vähän avitusta. Joissain kohdissa ohjelma kyllä antaakin vinkkiä miten pitäisi toimia. Varmaan kirjanpitoaiheen paremmin tuntevalle avautuisi paremmin. Toisaalta myös omassa työssä huomasin, että kaikki vaikutti selkeältä, kun oli valmiiksi täytettyä tietokantaa, mutta puhtaalta pöydältä lähdettäessä eri taulukoiden merkitys ei samalla tavalla auennutkaan.

Fxml:ää ei käytetä käyttöliittymän määrittelyyn, mutta jotain siihen liittyviä projektia luotaessa generoituja tiedostoja kummittelee mukana. Gitiin liittyviä tiedostoja ei ole mukana. Jar-tiedosto löytyi.

Mukavahan tätä oli lukea. Toimintoja tuntuu olevan, mutta toivoisi, että se olisi paketoitu käyttäjän kannalta vielä helpommin lähestyttävään muotoon.