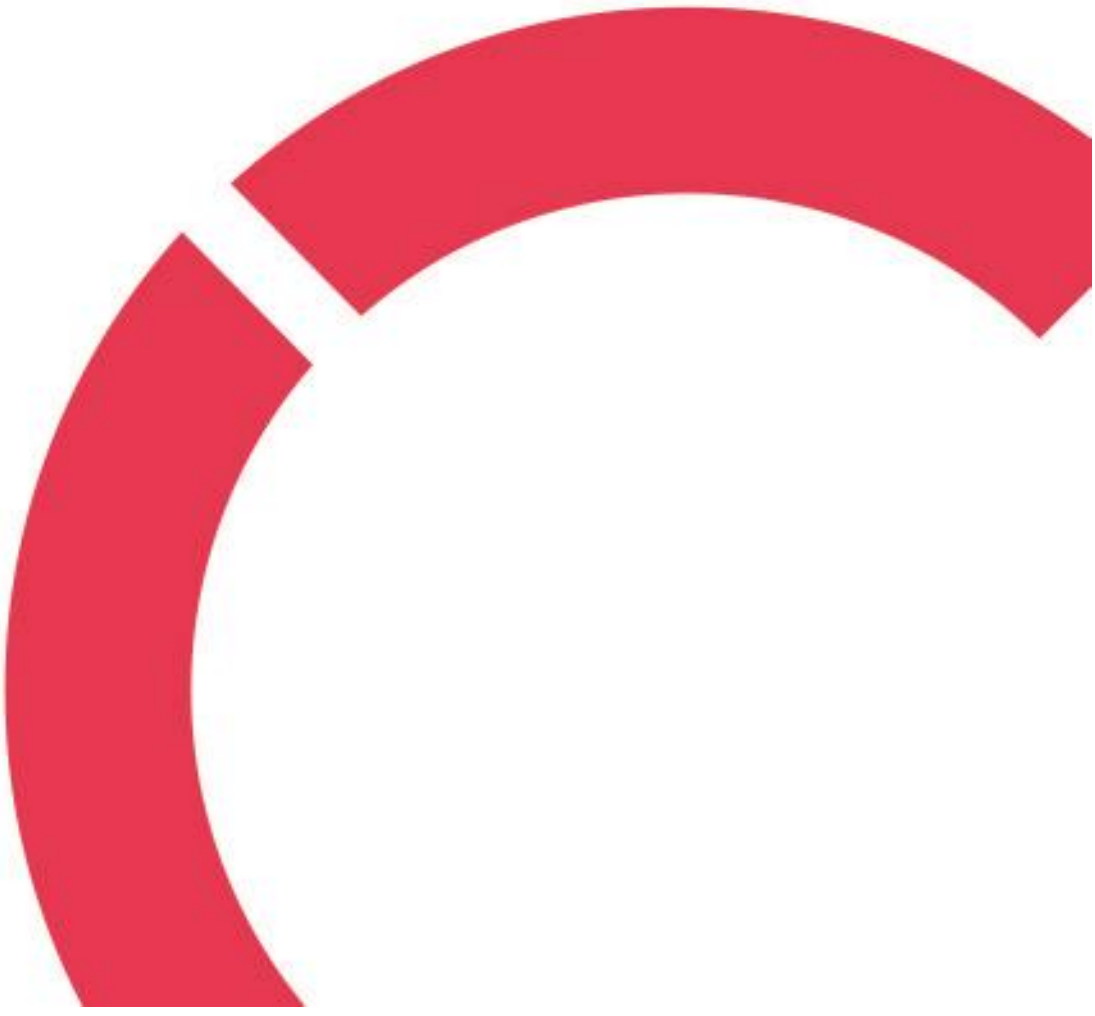
Juuso Niemi-Korpi

YRITYSSIMULAATTORIN TOTEUTUS 2D:NÄ

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutus
Marraskuu 2024



TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Marraskuu 2024	**Tekijä/tekijät** Juuso Niemi-Korpi
Koulutus Tieto- ja viestintätekniikka		☒ AMK ☐ YAMK
Työn nimi Yrityssimulaattorin toteutus 2D:nä		
Työn ohjaaja Kyösti Marjakangas		**Sivumäärä** 34

Tämän opinnäytetyön tavoitteena oli toteuttaa 2D-yrityssimulaattoriohjelma hyödyntäen Apache NetBeans-ohjelmointiympäristöä ja Java-ohjelmointikieltä. Projekti antoi mahdollisuuden soveltaa aikaisempia ohjelmointitaitojani ja käytännön kokemusta yrityssimulaattorin luomiseen.

Toteutetussa simulaattorissa pelaaja ylläpitää ja hallitsee yritystä ja kasvattaa sitä mahdollisuuksien mukaan, pelaajalle on luotu erilaisia haasteita vastaamaan todellisen yrityksen ylläpitoa. Simulaattorissa voi esimerkiksi valita vaikeusasteen, palkata työntekijöitä, myydä yrityksen tuotteita tai omaisuutta ja joutua konkurssiin. Konkurssin sattuessa yritys kaatuu ja simulaatio päättyy. Ohjelma on luotu ja suunniteltu pääasiassa Windows-alustalle.

Asiasanat
2D, Java, NetBeans, peli, yritys

ABSTRACT

Centria University of Applied Sciences	**Date** November 2024	**Author** Juuso Niemi-Korpi
Degree programme Information and Communications Technology		
Name of thesis Implementation of a company simulator in 2D		
Centria supervisor Kyösti Marjakangas		**Pages** 34

The aim of this project was to implement a 2D company simulator utilizing the Apache NetBeans-development environment and the Java programming language. The project provided an opportunity to apply my previous programming skills and practical experience in creating a company simulator.

In the implemented simulator, the player manages and operates a company, aiming to grow it as much as possible. Various challenges were designed to reflect the realities of running a business. The simulator allows the player to choose a difficulty level, hire employees, sell company products or assets and face bankruptcy. In the event of bankruptcy, the company collapses and the simulation ends. The program was primarily created and designed for the Windows platform.

Key words
2D, company, game, Java, NetBeans

KÄSITTEIDEN MÄÄRITTELY

2D

Kaksiulotteinen digitaalinen kuva

Apache NetBeans IDE

Integroitu ohjelmointiympäristö useille ohjelmointikielille

Java

Laitteistoriippumaton korkean tason ohjelmointikieli

Konsoli

Tulostaa viestejä ja virheitä erilliseen ikkunaan

KUVAT

KOODIT

1 JOHDANTO

Opinnäytetyön aikana on tarkoitus testata, voiko simulaattorin toteuttaa järkevästi yksilötyönä. Haluan tietää, onnistuvatko todellisen elämän tilanteen jäljittely ja tunne 2D:nä toteutettuna. Pohdin myös, voisiko toteutettua ohjelmaa käyttää muunneltuna johonkin muuhunkin käyttötarkoitukseen, kuten oikean elämän yrityksen hallintatyökaluna. Tavoitteena on kuitenkin rakentaa ohjelma, jonka käyttäjä saa kokemuksen yrittämisestä.

Opinnäytetyössä esitellään tekemääni ohjelmaa ja antaa tietoa ohjelman rakennusprosessista ja sen aikana käytettävistä työkaluista. Toteutettavan ohjelman päällimmäinen tarkoitus on viihdyttää käyttäjää, mutta kuitenkin jäljitellä todellisen elämän yritystoimintaa. Ihanteellisessa tilanteessa ohjelman käyttäjä viihtyy ja saa tietoa oikean elämän yrityksen riskeistä, sekä mahdollisuuksista. Opinnäytetyön nimeksi valikoitui yrityssimulaattori, sillä tavoitteiden mukainen ohjelma jäljittelee tosielämän yritystoimintaa.

Aluksi tutustutaan aiheeseen, jonka jälkeen käydään läpi lyhyesti ohjelmoinnin historiaa ja perusteita, jotta työn hahmottaminen ja rakennusprosessi olisi selkeämpää. Myöhemmin työssä esitellään ja käydään läpi kehittämäni ohjelman eli simulaattorin eri ominaisuuksia ja käyttötarkoituksia.

Koska simulaatio on pääroolissa toteutettavan ohjelman kehityksessä, käyn läpi sen haasteita ja ratkaisuja. Ymmärrettävästi simulaattoreiden toteutus on haasteellista, koska ne vaativat laajaa ymmärrystä simuloitavasta aiheesta ja ympäristöstä. Toteutuksessa pitää etenkin ottaa huomioon monimutkaiset muuttujat ja prosessit, joita voisi oikeasti tapahtua todellista tilannetta jäljennettäessä. Lisäksi käyttäjäkokemus ja käyttöliittymä on otettava huomioon kehityksessä. Yksi riskeistä simulaatioissa on epärealistisen kokemuksen antaminen ja tätä kokemusta seuraamalla käyttäjä saattaisi asettaa itsensä tai jonkun toisen vaaraan oikean elämän tilanteessa. Vääränlainen kokemus saattaa siis altistaa vaaratilanteelle. (Hurrell 2024.)

Ratkaisuna monimutkaisiin toteutuksiin simulaattorit jaetaan omiin osa-alueisiin ja luokkiinsa, jotta niiden käyttö ja mahdollinen kehitys olisi tehokkaampaa. Eri osa-alueet määräytyvät simulaattoreiden käyttötarkoituksien perusteella. Yleisesti simulaattoreita voidaan hyödyntää muun muassa koulutus- ja tutkimuskäytössä. Vaaratilanteiden ehkäisynä käyttäjille annetaan riittävästi tietoa simulaattoreiden

käyttötarkoituksista, joiden perusteella ne eivät ole kykeneviä tuottamaan täydellistä tosielämän tilannetta. Kuitenkin simulaattorien käyttö vähentää monenlaisia riskejä, kuten onnettomuusriskejä. Hyvä esimerkki ehkäisevästä onnettomuustekijästä on nykyään ajo-opetuksissa käytettävä ajosimulaattori. (CAP 2024.)

Vaikka simulaatiossa voikin tapahtua epärealistinen kokemus, se voi toisaalta tapahtua myös oikeassa elämässä. Oikeassa elämässä on joskus tilanteita, jotka ovat todella harvinaisia ja näin ollen epärealistisia. Myös ajo-opetuksissa käytettävä ajosimulaattori voi joissakin tapauksissa jopa lisätä oikean elämän riskejä, koska simulaattori ei nykypäivänä edelleenkään ole tarpeeksi realistinen. Jatkuvalla ohjelmistokehityksellä simulaattorit ovat kehittyneet siihen pisteeseen, että niitä käytetään laajasti eri aloilla. Uudet teknologiat ja niissä käytettävät algoritmit ja ohjelmointikielet mahdollistavat jatkuvasti realistisemmat ja monimutkaisemmat simulaatiot. (Jaber 2024.)

Omasta mielestäni realistisemmat ja monimutkaisemmat simulaatiot eivät välttämättä ole kuitenkaan parempi vaihtoehto, sillä niiden kehitys saattaa olla hitaampaa. Oletan, että pitkälle kehittyneet ja jatkuneet simulaattorit voisivat korvata joissakin tapauksissa kokonaan oikean elämän kokemukset, jolloin niiden hyötysuhde voisi olla jopa negatiivinen. Olen havainnut, että nykypäivänä pystytään luomaan erittäin monimutkaisia ja todellisia skenaarioita, joiden avulla voidaan kehittää käyttäjille todellisen ympäristön kokemusta ilman sen hetkisiä onnettomuusriskejä ja suuria kustannuksia. Tästä voin päätellä, että simulaattorin toteuttaminen ja sen erivaiheiden kehitys ja testaaminen on varsin hyvä ja haasteellinen työ kenelle tahansa riippuen sen käyttötarkoituksesta. Simulaattoreiden kehitystä voidaan jatkaa loputtomiin ja kehitystyö ei lopu koskaan.

2 TIETOA OHJELMOINNISTA JA TYÖSSÄ KÄYTETYISTÄ TYÖKALUISTA

Seuraavaksi lyhyesti kerrottavan ohjelmoinnin historian taustalla on idea kasvattaa ymmärrystä ja mielenkiintoa työtä kohtaan. Historian tarjoaa kontekstin, joka auttaa ymmärtämään ohjelmoinnin kehityskaaren. Historian jälkeen tulee tietoa ohjelmoinnista ja siinä käytettävistä eri työkaluista, sekä ohjelmointikielistä, jotka auttavat hahmottamaan ohjelman rakennusprosessin.

2.1 Ohjelmoinnin historiaa lyhyesti

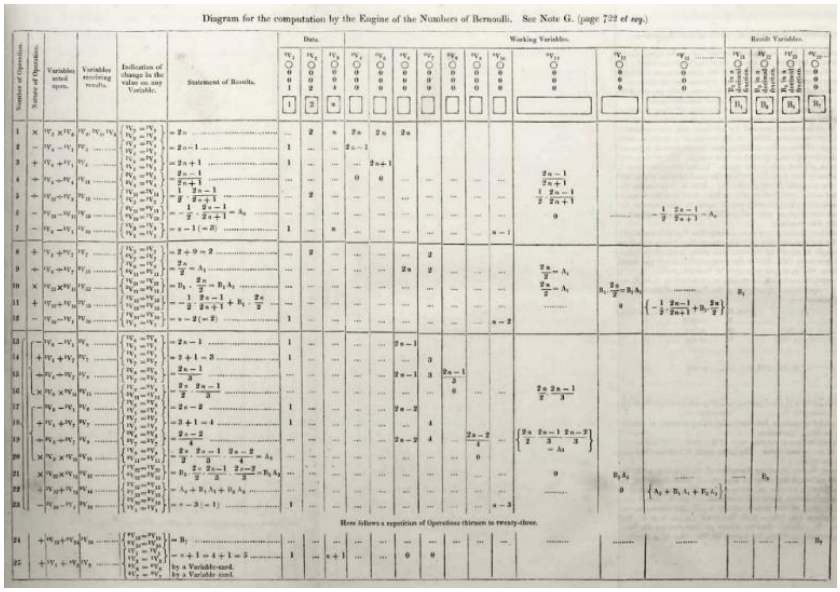
Ymmärtääksemme paremmin ohjelmoinnin laajuutta ja sen jatkuvaa kehitystä, on sen historiaa hyvä käydä hiukan läpi. Palataan siis ajassa 200-vuotta taaksepäin, jolloin ohjelmoinnin historia alkoi.

1800-luvun alussa englantilainen matemaatikko ja filosofi Charles Babbage kehitti yhteistyökumppaninsa Augusta Ada Kingin eli Ada Lovelacen kanssa ajatuksen ohjelmoitavasta tietokoneesta. Varsinainen ensimmäinen ohjelma on Lovelacen käsialaa, kun hän kirjoitti ohjelman Charlesin varhaista yleistietokonetta varten (KUVA 1). Lovelacea pidetään tästä syystä ensimmäisenä tietokoneohjelmoijana. (Project Lovelace 2024.)

Suuresta nerokkuudestaan huolimatta Lovelace ei saanut elinaikanaan ansaitsemaansa tunnustusta, koska 1800-luvun ihmiset ja tiedeyhteisö eivät ottaneet naisen ideoita tarpeeksi vakavasti. Lovelace kerkesi pohtimaan lyhyestä elämästään huolimatta jopa tekoälyn mahdollisuutta. Ada Lovelace menehtyi vain 36-vuotiaana vuoden 1852 alussa jättäen jälkeensä tärkeän osan digiajan perintöä. (Zwolak 2023.)

Lähes 100 vuotta myöhemmin tietojenkäsittelytieteen pioneeri Howard Aiken kiinnostui Charles Babbagen ja Ada Lovelacen ajatuksista ja huomasi, että uudet sähköiset ratkaisut mahdollistavat Charlesin ja Adan ideoiden toteutuksen. Howard Aiken suunnittelikin ensimmäisen sähkömekaanisen tietokoneen, joka pystyy suorittamaan automaattisia monimutkaisia laskelmia ohjelmoinnilla. (ETHW 2016.) Historiasta puhuttaessa on kuitenkin hyvä muistaa, että Ada Lovelacen ohjelma on pitkälti teoreettinen. Adan teoriaa ei keretty koskaan testaamaan käytännössä, koska Charles Babbagen suunnittelema kone jäi toteuttamatta. Voidaan myös miettiä, että olisiko sen ajan ihmiset ja tiedeyhteisö ottaneet miehen vastaavaa teoriaa yhtään vakavammin. Kyseessä oli kuitenkin uusi asia ja uusiin asioihin suhtaudutaan lähes aina jollakin tasolla kyseenalaistavasti tai negatiivisesti vielä nykypäivänäkin. Howard

Aikenin saavutuksia saattaa myös heijastaa negatiivisesti Babbagen ja Lovelacen työt, koska oletettavasti osa ihmisistä sanoo hänen työnsä olevan pelkkää kopiointia. Joka tapauksessa Aiken, Babbage ja Lovelace ovat historiassa erittäin merkittäviä ja viisaita ihmisiä, joiden aikaansaannokset tulevat aina olemaan osa ohjelmointia ja tietotekniikkaa.



KUVA 1. Ada Lovelacen algoritmin muistiinpanot. (Kärkkäinen 2018.)

2.1.1 Mitä on ohjelmointi

Ohjelmointi on prosessi, jossa tuotetaan ohje ongelman ratkaisemaksi ohjelmointikielellä, jonka tietokone voi suorittaa. Ihmisten elämässä esiintyy ohjelmointia vastaavia tapahtumia joka päivä. Esimerkiksi ihminen voi tehdä kotiaskareita järjestyksessä yksitellen ja näin saavuttaa tavoitteensa. (Jyväskylän yliopisto)

Ihmisen käyttämä kieli on liian monimutkainen suorilta käytettäväksi ohjelmointikielenä, joten ohjelmointiin on kehitetty helpommin käsitettäviä kieliä. Ohjelmointi ja siinä käytettävät kielet ovat suoria, eivätkä tulkinnanvaraisia. Ohjelma toimii juuri siten, kuten se on kirjoitettu. Ohjelman suoritusohjeita eli tietokoneohjelmia kutsutaan algoritmiksi. (Futurelearn 2024.)

Alla (Koodi 1) on pieni esimerkki Java-kielen algoritmista työssä tekemästäni ohjelmasta. Algoritmi määrittää muuttujalle ”valittulahja” käyttäjän hiirellä listasta valitun arvon. Käyttäjän valitessa listasta numeron 1, tulostuu kokonaisuudessaan arvo: ”Valitsit lahjan: 1” konsoliin (Koodi 1).

```
public void valittulahja ()  {
   String valittulahja = lahjalista.getSelectedValue();
   System.out.println("Valitsit lahjan: " +valittulahja);
  }
```

Koodi 1. Algoritmi tulostaa käyttäjän valitseman arvon konsoliin.

Algoritmi voidaan määrittää antamaan muuttujalle lähes mitä vain arvoja hyödyntäen eri ominaisuuksia, kunhan annetut arvot tai toiminnot ovat tuettuja käytetyssä ohjelmointikielessä.

2.1.2 Erilaiset ohjelmointikielet ja niiden käyttötarkoitukset

Ohjelmointikieliä käytetään kommunikointitapana ohjelmoinnissa ohjelmoijan ja tietokoneen välillä. Esimerkiksi työpöytäsovellukset, verkkosivustot ja mobiilisovellukset on kehitetty käyttäen eri ohjelmointikieliä. (Chakray 2024.)

Ohjelmointia varten on vuosien saatossa kehittynyt laaja valikoima erilaisia ohjelmointikieliä. Uusia kieliä luodaan ja vanhojen kielien tietoturvaa ja nopeutta kehitetään jatkuvasti sopimaan yhä monimutkaisempiin tulevaisuuden haasteisiin. (Upson 2023.)

Matalan tason ohjelmointikieli on tietokoneesta riippuvainen ohjelmointikieli. Prosessori kykenee suorittamaan tällaisia ohjelmia ilman kääntäjää tai tulkkia. (Tieturi 2024.)

Matalan tason ohjelmointikielet jaetaan kahteen eri osaan eli konekieli ja kokoonpanokieli (ASM). Konekieli on tietokoneen suoraan ymmärtävä kieli, koska se esitetään binääri- tai heksadesimaalimuodossa. Kokoonpanokieli (ASM) on kieli, joka on suunniteltu suoritettavaksi tietyille prosessoreille. (InetMedia 2024.)

Korkean tason ohjelmointikieli on tehty käyttäjäystävällisten ohjelmien ja verkkosivustojen kehittämiseen. Korkean tason ohjelmointikieli vaatii toimiakseen kääntäjän tai tulkin, joka kääntää käyttäjän kirjoittaman ohjelman konekielelle tietokoneen suorittamista varten. (Tieturi 2024.) Korkean tason ohjelmointikieliin kuuluvat esimerkiksi Java, Python, JavaScript, PHP, C#, C++, Objective C, Cobol ja Pascal. (Lenovo 2024.) Korkean tason ohjelmointikielet jaetaan kolmeen eri osaan eli proseduraalisesti orientoituneet ohjelmointikielet, oliokeskeiset ohjelmointikielet ja luonnolliset kielet. (Pedamkar 2024.)

Proseduraalista ohjelmointikieltä käytetään ohjelman luomiseen hyödyntäen esimerkiksi IDE-ohjelmointiympäristöä. Oliokeskeisessä ohjelmointikielessä ohjelmat jaetaan pieniin osiin ja niistä luodaan kokonaisuus. Luonnollisilla ohjelmointikielillä tarkoitetaan ihmisten käyttämiä kieliä, joita koneet käyttävät ymmärtämään ihmistä. (Datascientis 2024.)

2.2 API:n merkitys NetBeans ohjelmointiympäristössä

Avoimen lähdekoodin NetBeans-ohjelmointiympäristössä voi lisätä toiminnallisuuksia ja ominaisuuksia API:n avulla. API (Application Programming Interface) tarkoittaa ohjelmointirajapintaa, joka antaa mahdollisuuden määritellä, miten ohjelmiston eri palvelut ja komponentit toimivat keskenään. (Bautamo 2024.) API:n avulla tieto liikkuu kahden eri ohjelman tai sovelluksen välillä (KUVIO 1). Tekniikan avulla palveluita onnistutaan liittämään yhteen. (Alfame 2018.)

2.3 Avoin lähdekoodi

Opinnäytetyössä käytettävä Apache NetBeans IDE on avoimen lähdekoodin ohjelmointiympäristö. Avoimen lähdekoodin ohjelmistot ovat kaikille avoimesti saatavilla olevia ja ne ovat vapaita käytettäväksi, muokattavaksi ja levitettäväksi. Avoin lähdekoodi on tärkeä osa ohjelmistokehitystä, sillä se antaa laajalle yleisölle mahdollisuuden tarkastella ja tehdä muutoksia ohjelmaan haluamallaan tavalla. (Haltu 2023.)

OSI:n kymmenen avoimen lähdekoodin ohjelmiston tunnusmerkkiä:

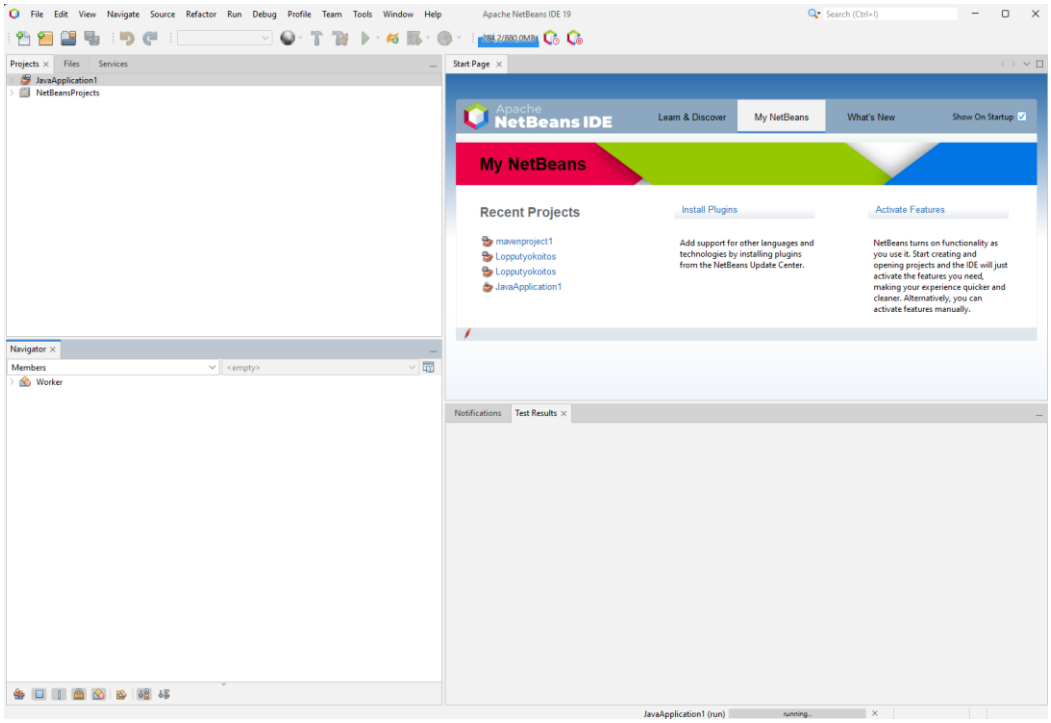
1. *Ohjelman täytyy olla vapaasti levitettävissä ja välitettävissä.*
2. *Lähdekoodin täytyy tulla ohjelman mukana tai olla vapaasti saatavissa.*
3. *Myös johdettujen teosten luominen ja levitys pitää sallia.*
4. *Lisenssi voi rajoittaa muokatun lähdekoodin levittämistä vain siinä tapauksessa, että lisenssi sallii erillisten korjaustiedostojen ja niiden lähdekoodin levittämisen. Voidaan myös vaatia, ettei johdettua teosta levitetä samalla nimellä tai versionumerolla kuin lähtöteosta.*
5. *Yksilöitä tai ihmisryhmiä ei saa asettaa eriarvoiseen asemaan.*
6. *Käyttötarkoituksia ei saa rajoittaa.*
7. *Kaikilla ohjelman käsiinsä saaneilla on samat oikeudet.*
8. *Lisenssi ei saa olla riippuvainen laajemmasta ohjelmistokokonaisuudesta, jonka osana ohjelmaa levitetään, vaan ohjelmaan liittyvät oikeudet säilyvät, vaikka se irrotettaisiin kokonaisuudesta.*
9. *Lisenssi ei voi asettaa ehtoja muille ohjelmille. Ohjelmaa saa levittää myös yhdessä sellaisten ohjelmien kanssa, joiden lähdekoodi ei ole avointa.*
10. *Lisenssin sisällön pitää olla riippumaton teknisestä toteutuksesta. Oikeuksiin ei saa liittää varaumia jakelutavan tai käyttöliittymän varjolla.* (Haltu 2023.)

2.4 Opinnäytetyössä käytettävä kehitysympäristö NetBeans ja sen historia

Opinnäytetyö on toteutettu IDE:llä. IDE tarkoittaa integroitua kehitysympäristöä, joka mahdollistaa eri osa-alueiden konsolidoinnin tietokoneohjelmia kirjoittaessa. (codecademy 2024.) IDEssä yhdistyvät ohjelmoinnin yleiset toiminnot yhteen sovellukseen. Näitä toimintoja ovat esimerkiksi lähdekoodin muokkaaminen, suoritettavien tiedostojen rakentaminen ja virheiden korjaaminen. (AWS 2024.)

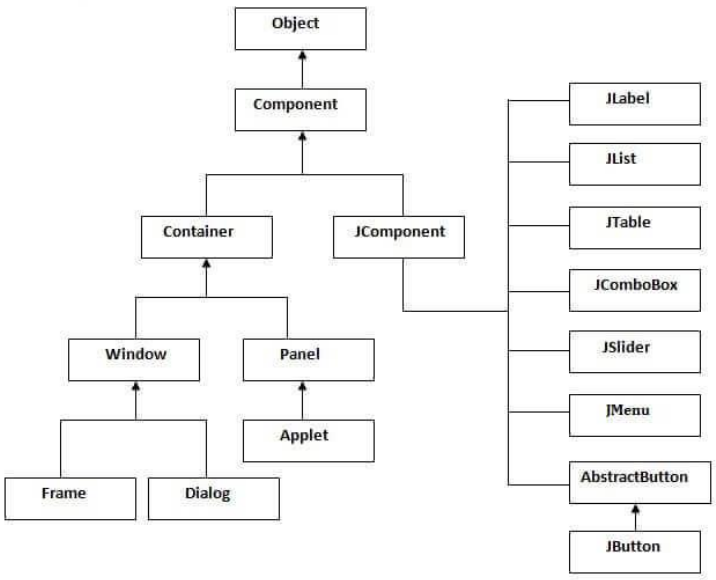
Apache NetBeans on avoimen lähdekoodin integroitu kehitysympäristö (IDE) ja alusta etenkin Java-kehittäjille. NetBeans sai alkunsa vuonna 2000, kun sen alkuperäinen luoja Sun Microsystems teki siitä avointa lähdekoodia. Sun Microsystems toimi NetBeans-ohjelmointiympäristön sponsorina vuoteen 2010, kunnes siitä tuli Oraclen tytäryhtiö. (Gooding 2019.)

Pitkän historiansa ajan NetBeans on ollut ilmainen ja avoimen lähdekoodin ohjelmisto. NetBeans on myös ollut mukana edistämässä Java-kehityksen kulkua. (Darryl 2019.)
Vuonna 2016 Oracle lahjoitti NetBeansin lähdekoodin Apache Software Foundationille ja vuonna 2019 Apache NetBeans nousi yhdeksi tärkeimmäksi Apache-projektiksi. (Apache 2024.)
Apache NetBeans IDE-ohjelmointiympäristö on kehitetty vastaamaan kehittäjien, käyttäjien ja yritysten tarpeita. Apache NetBeans IDE:ä on ladattu tähän mennessä yli 18 miljoonaa kertaa ja sen kehittämiseen on osallistunut vuosien varrella yli 800 000 kehittäjää. (Apache 2024.)



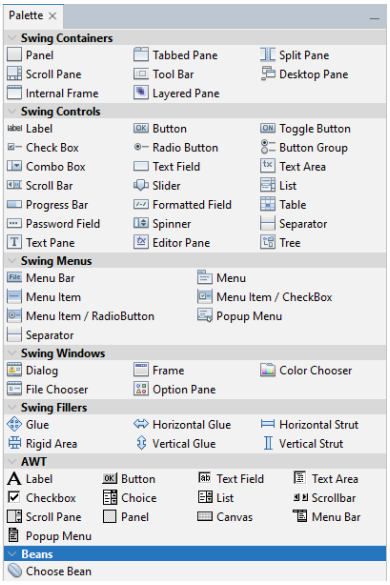
KUVA 2. Apache NetBeans IDE-ohjelmointiympäristö

NetBeans-ohjelmointiympäristöä käytetään usein Swing-sovelluksien kehittämiseen. NetBeans muodostaa yhdessä Swingin kanssa toimivan kokonaisuuden sovelluskehitystä varten. (Apache 2024.) Java Swing kuuluu Java Foundation Classes (JFC)-ohjelmiston osiin. Java Swingiä käytetään ikkuna-pohjaisten sovellusten tekemiseen. Swing on kirjoitettu vanhemman ohjelmointirajapinnan (AWT) päälle ja on kokonaan kirjoitettu Javalla. (Javatpoint 2024.)



KUVA 3. Java Swing-hierarkia (Javatpoint 2024).

Swing toimii hierarkia periaatteella, eli eri osien ja toimintojen täytyy olla oikeassa järjestyksessä, ohjelman toimimisen takaamiseksi (KUVA 3).



KUVA 4. Java Swing-paletti NetBeans

Vuonna 1996 julkaistu Java Swing on monipuolinen ja pitkään kehittynyt työkalu, joka sopii edelleen ammattilaiskäyttöön (KUVA 4). Swingillä on mahdollista luoda vaativia, mutta tarvittaessa yksinkertaisia ohjelmia tai toimintoja.

KUVA 5. Yksinkertainen Java Swing esimerkki (Javatpoint 2024).

2.5 Tietoa simulaatiosta

Simuloinnilla tarkoitetaan yleensä todellisten tapahtumien mallintamista. Simulaation avulla voidaan saavuttaa lähes todellinen tapahtuma kompromisseilla. Simulaatiossa on tarkoitus keskittyä jonkin tuotteen, järjestelmän tai prosessin jäljittelyyn. Simulaation kokonaisuus määritellään eri käyttötarkoituksen perusteella. (Räsänen 2004.)

Ohjelmoinnilla toteutetussa simulaatiossa käyttäjälle annetaan sille määritetyssä ympäristössä halutun jäljitelmän mukainen kokemus. Jäljitellyssä ympäristössä pyritään jäljittelemään mahdollisimman todenmukainen tapahtuma. (Räsänen 2004.) Simulaatio on käyttäjän näkökulmasta opettavainen ympäristö, jossa käyttäjä voi oppia uutta turvallisesti ilman riskejä. (SSH 2024.)

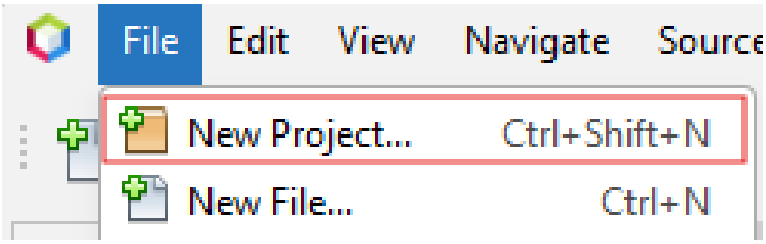
3 OHJELMAN RAKENTAMINEN

Tässä osassa käydään alkuun lyhyesti läpi Java-pohjaisen ohjelman rakentamisen aloitus NetBeans IDE-ohjelmointiympäristössä, jonka jälkeen työ esitellään valmiissa muodossa sen laajuuden vuoksi. Ohjelman kehittämisen alkuvaiheessa on tärkeää asettaa päämäärän mukainen tavoite ja pyrkiä toimimaan sen mukaan mahdollisimman tarkasti. Hyvä käsitys projektin tavoitteesta ja lopputuloksesta auttaa suunnittelemaan ja hahmottamaan ohjelman rakentamista ja kokonaisuutta. Työssä toteutettavan ohjelman rakentamisprosessissa pääosaa näyttelee aiemminkin mainittu NetBeans IDE-ohjelmointiympäristö, jossa ohjelma rakennetaan.

Ohjelman suorittamiseksi on hyvä osata kirjoittaa ja hallita suhteellisen vaativaa ohjelmointikokonaisuutta projektin luonteen takia. Ohjelma kehitetään IDE-ohjelmointiympäristössä, joten valmista ohjelmointikoodia on pienempi määrä verrattuna esimerkiksi yleisiin pelimoottoreihin.

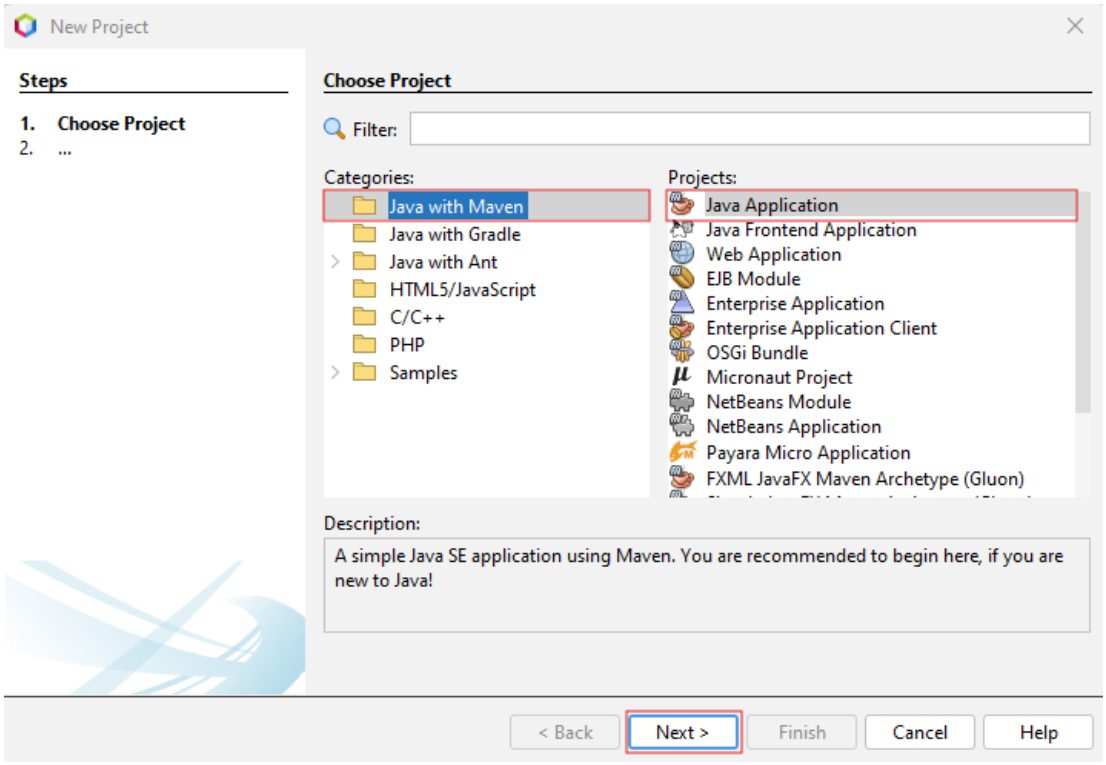
3.1 Ohjelman luominen

Käydään lyhyesti läpi, miten tämän kaltainen projekti ja ohjelman luominen aloitetaan, kun NetBeans on jo ladattu tietokoneelle. Ensimmäiseksi avataan ohjelmointiympäristö, eli tässä tapauksessa NetBeans, joka toimii projekti- ja koodieditorina koko työn ajan. Uusi projekti luodaan klikkaamalla (New Project) -kohdasta, joka avaa uuden ikkunan, jossa pääsee valitsemaan projektin asetukset ja sisältyvyydet (KUVA 6).



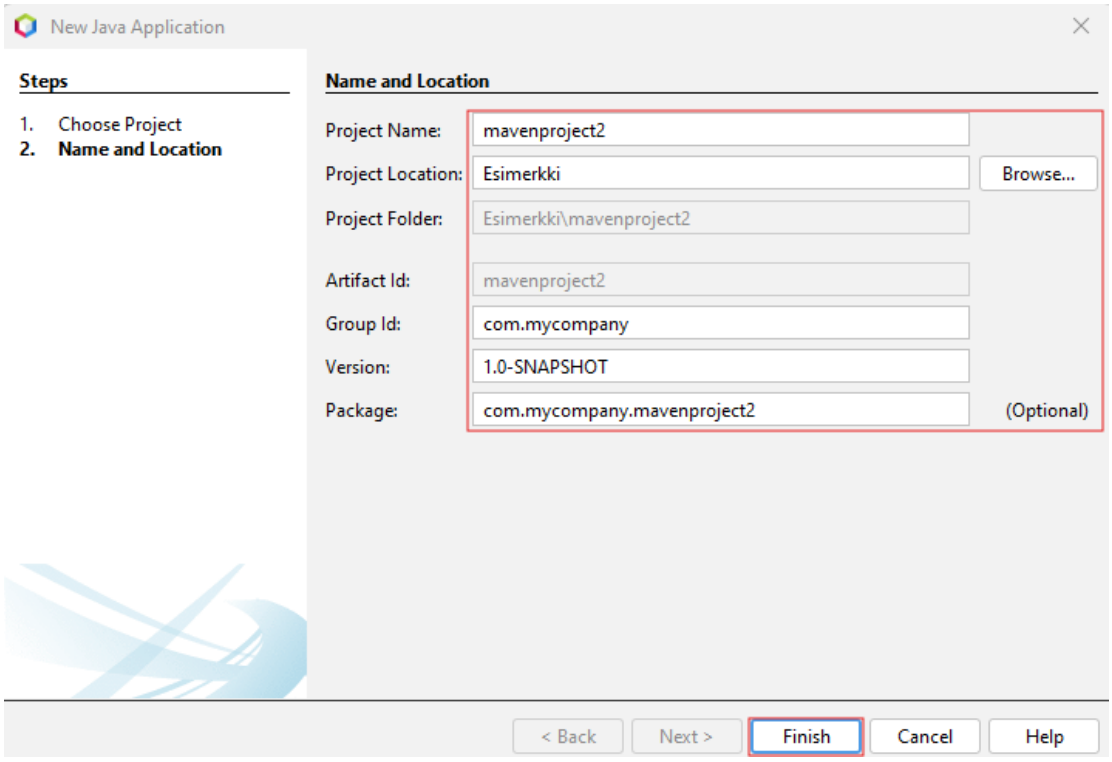
KUVA 6. NetBeans uusi projekti

Edeltävän klikkauksen jälkeen avautuu projektin määritykset. Valitaan kategoriat- ruudusta (Java with Maven) -kuvake ja projektit- ruudusta (Java Application) -kuvake, jonka jälkeen klikataan (Next >) -nappia (KUVA 7).



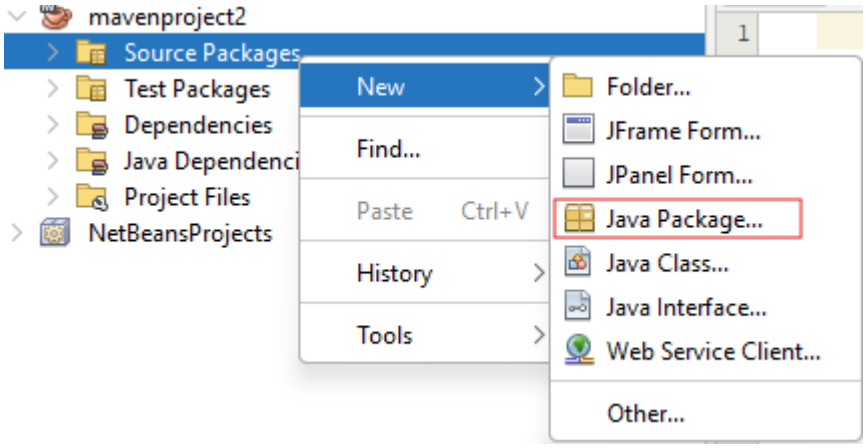
KUVA 7. Projektin kategorian määritys

Seuraavassa vaiheessa projektille valitaan nimi ja projektin tiedostosijainti. Projektin nimen ja sijainnin valitsemisen jälkeen voidaan klikata (Finish) -nappia (KUVA 8).



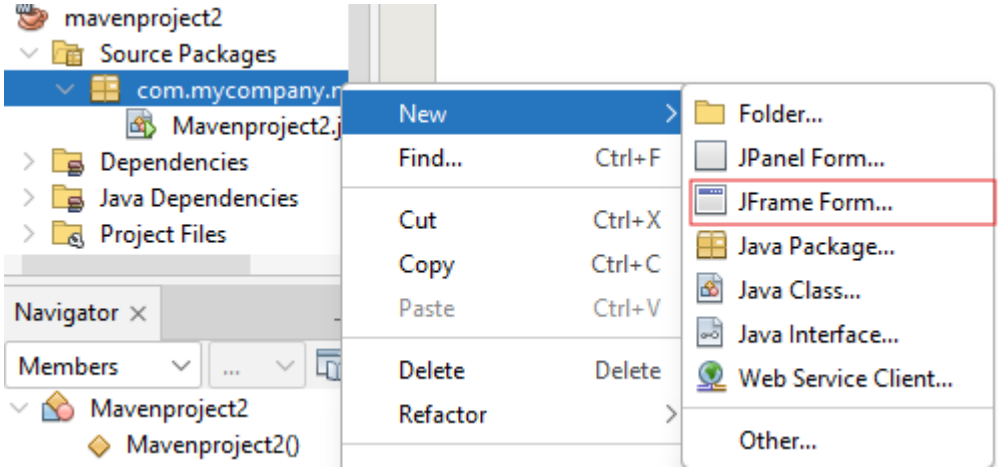
KUVA 8. Projektin nimen ja tiedoston määritys

Seuraavaksi tehdään juuri luotuun projektiin Java-paketti. Projektin kuvakkeen alla on (Source Packages) -kuvake, josta painetaan hiiren oikealla painikkeella. Kohdasta (New) valitaan (Java Package) -kuvake (KUVA 9).



KUVA 9. Java -paketin luonti

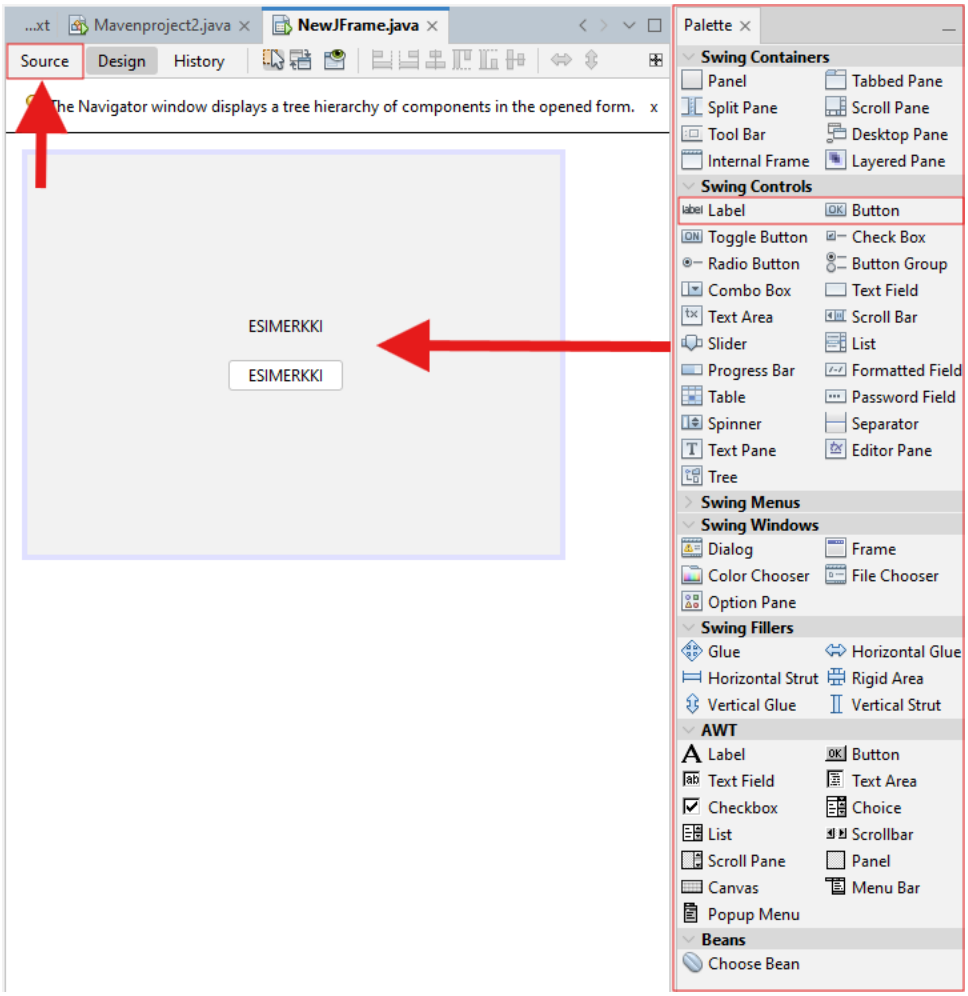
Nyt voidaan lisätä äsken luotuun Java-pakettiin uusi JFrame-muoto klikkaamalla hiiren oikealla painikkeella Java-paketista valitsemalla (New) ja sen jälkeen klikataan (JFrame Form) -kohdasta (KUVA 10).



KUVA 10. JFrame-muodon luonti

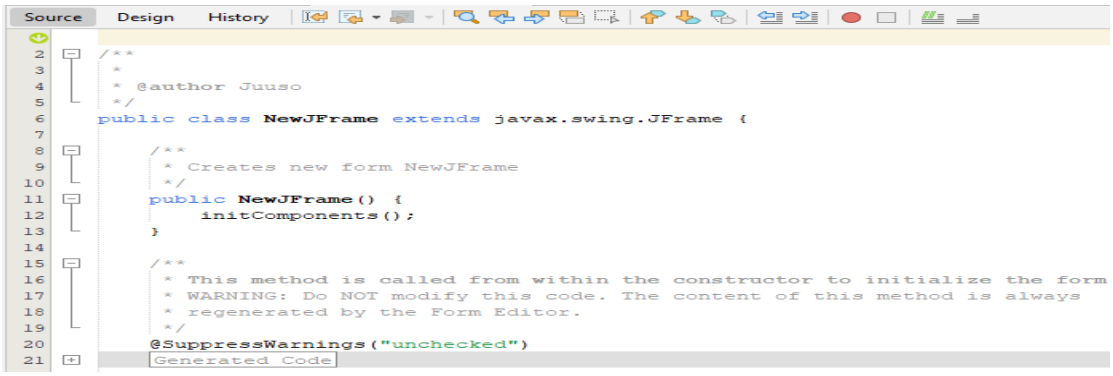
JFrame-muodon lisäämisen jälkeen avautuu Java Swing- paletti, josta löytyy erilaisia toiminnallisuuksia (KUVA 11). Esimerkkinä ruutuun on siirretty teksti- ruutu ja nappi, johon voi tehdä toiminnallisuuden koodieditorissa. (Source) -kohdasta löytyy koodieditori, jossa projektin koodi sijaitsee. Koodissa

määritellään suurin osa toiminnallisuuksista ja ohjelman ominaisuuksista, kuten mitä tapahtuu, jos nappia klikataan (KUVA 12).



KUVA 11. Paletti esimerkkejä

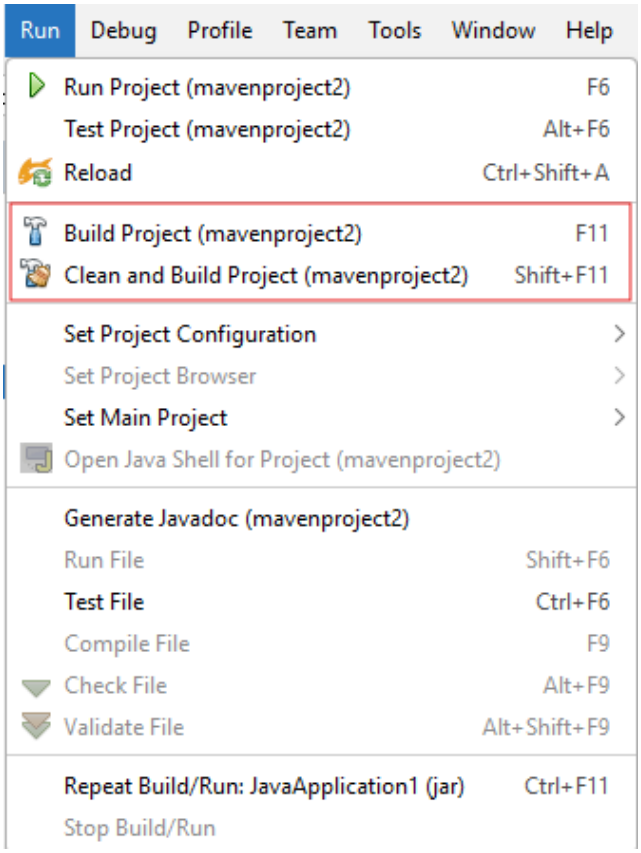
Koodiin ilmestyy automaattisesti osa koodista, kun Swing-valikosta valitaan vaikkapa nappi. Koodieditoriin täytyy kuitenkin itse kirjoittaa, mitä napista tapahtuu (KUVA 12).



KUVA 12. Koodieditori

3.1.1 Projektin julkaisu

Projektin ollessa valmis, se on mahdollista rakentaa kokonaiseksi paketiksi NetBeansin yläreunassa olevasta (Run) -valikosta, josta valitaan (Build Project) -kuvake (KUVA 13).



KUVA 13. Paketin rakentaminen

Paketti rakennetaan ”.jar” -tiedostoksi kansioon, josta se voidaan ajaa erillisenä tiedostona Javan avulla, tämä tiedosto ei ole enää riippuvainen NetBeans ohjelmointiympäristöstä.

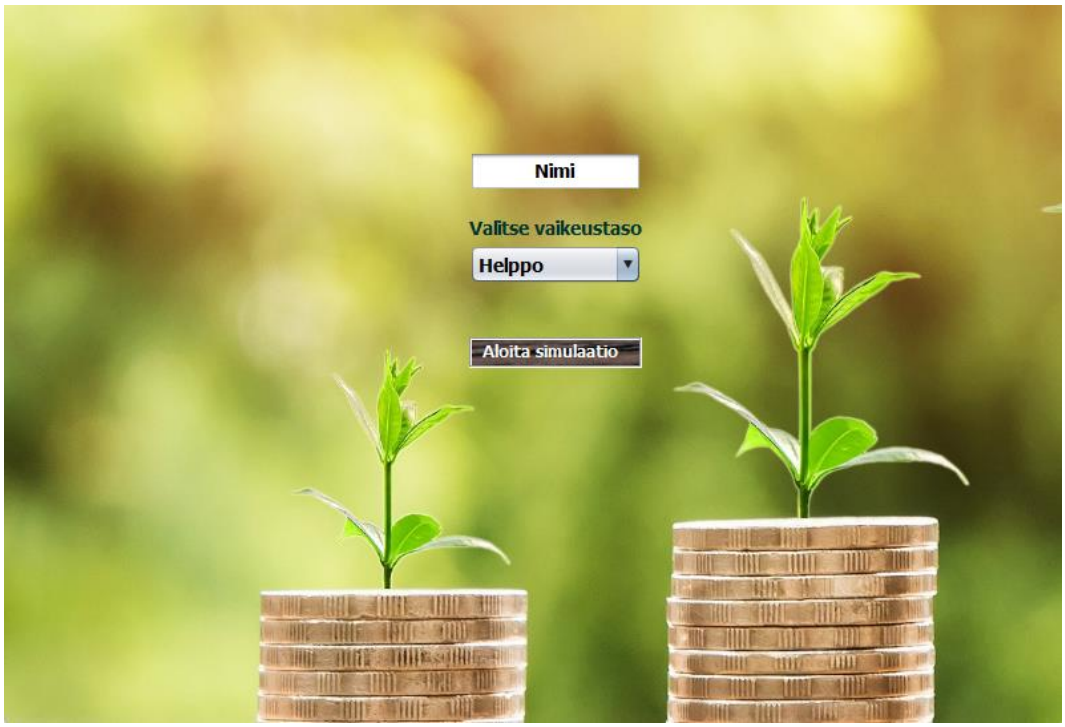
KUVA 14. JAR-tiedosto

Rakennettu projekti voidaan nyt helposti jakaa muille ihmisille, jos tekijä niin haluaa (KUVA 14). JAR-tiedosto vaatii kuitenkin Javan asentamista ohjelmaa suoritettavalle laitteelle.

3.2 Oman työn aloitus

Tästä eteenpäin projektia esitellään valmiissa muodossa vaiheittain. Projektissa on paljon pieniä yksityiskohtia ja toiminnallisuuksia, joiden esittely on selkeämpää, kun jokaista eri yksityiskohdan rakentamista ei selosteta ja näytetä erikseen.

Aloitin työn luomalla sopivan kokoisen aloitusruudun Swing- paletista. Lisäsin aloitusruutuun muutamia eri ominaisuuksia, kuten napin ja tekstiruudun, johon käyttäjä voi määrittää haluamansa nimen. Myös vaikeustaso on mahdollista määrittää (KUVA 15).

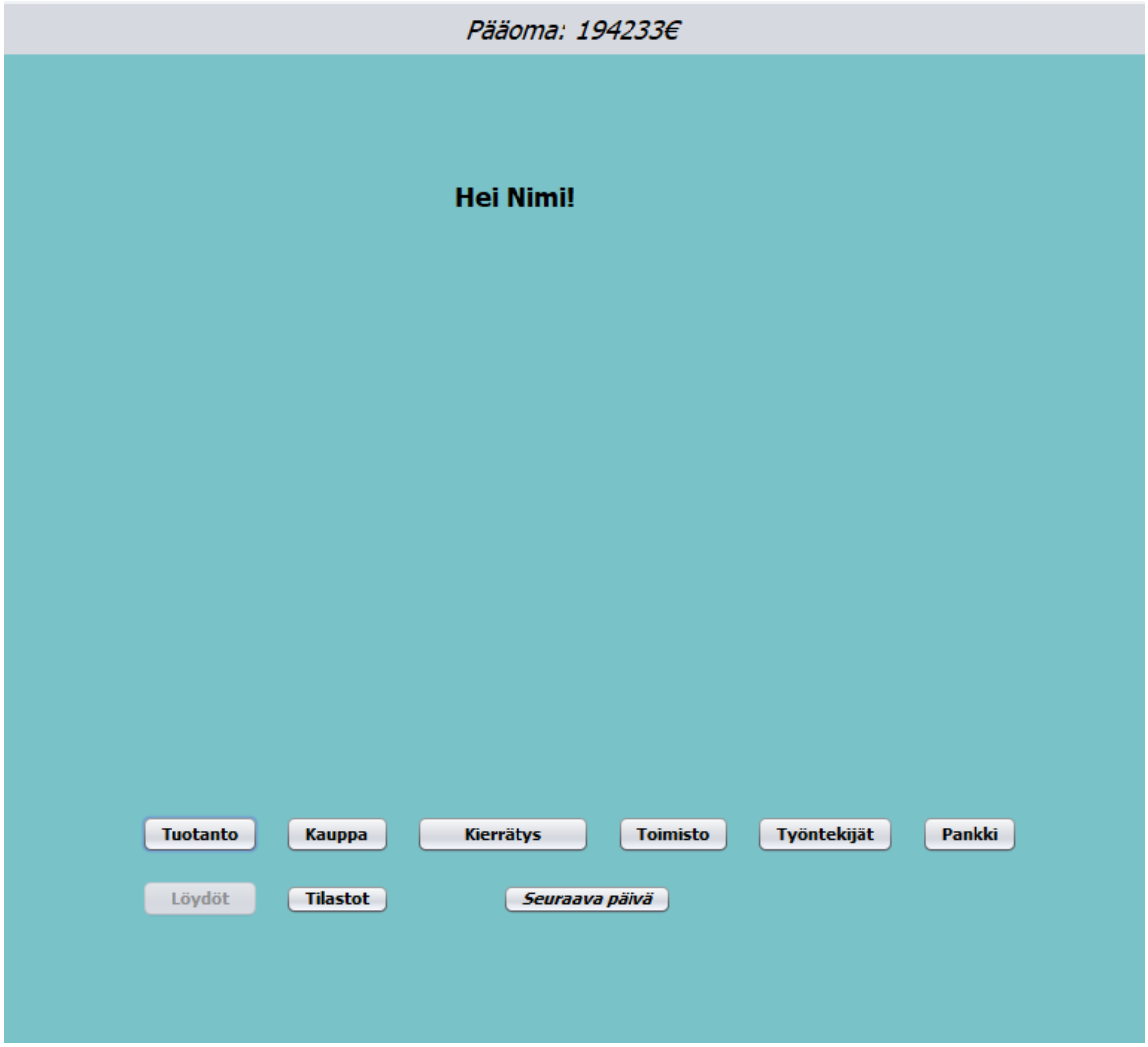


KUVA 15. Aloitusruutu ohjelmassa

Käyttäjän asettamat tiedot siirretään ohjelmassa eteenpäin seuraavaan ruutuun. Käyttäjän määrittämä nimi tulee olemaan käyttäjän nimi koko simulaation ajan.

3.3 Pääaula ja valikoiden suunnittelu ohjelmaan

Pääaulaan on luotu käyttäjälle mahdollisuuksia siirtyä eri ruutuihin. Valikoiksi muodostuivat oikean elämän yrityksessä olevat olennaiset paikat ja toiminnot (KUVA 16). Ohjelman ulkoasu ei ole pääroolissa, kuten kuvista voi päätellä.



KUVA 16. Valikot ja pääaula ohjelmassa

Valikkonapit ohjaavat käyttäjän valikon nimen mukaiselle uudelle ruudulle, jossa käyttäjä voi toteuttaa eri toimintoja halunsa ja mahdollisuuksiensa mukaan. Valikot voivat mennä pois käytöstä, mikäli käyttäjältä vaaditaan toimia.

3.4 Tuotanto ohjelmassa

Ohjelmassa on mahdollista jalostaa, sekä jatkojalostaa tuotteita, joilla yrityksen varallisuutta ja liikevaihtoa saadaan kasvatettua (KUVA 17). Valmiiksi ostettuja alkutuotteita on mahdollista myydä pois, mikäli varastotila loppuu kesken tai rahaa tarvitsee johonkin muuhun.

Jalostaakseen tuotteita, käyttäjä tarvitsee resursseja ja alkutuotteita. Tuotannon tehokkuuteen ja kyvykkyyteen vaikuttaa varaston koko, sekä työntekijöiden ja resurssien määrä.

KUVA 17. Tuotantoruutu ohjelmassa

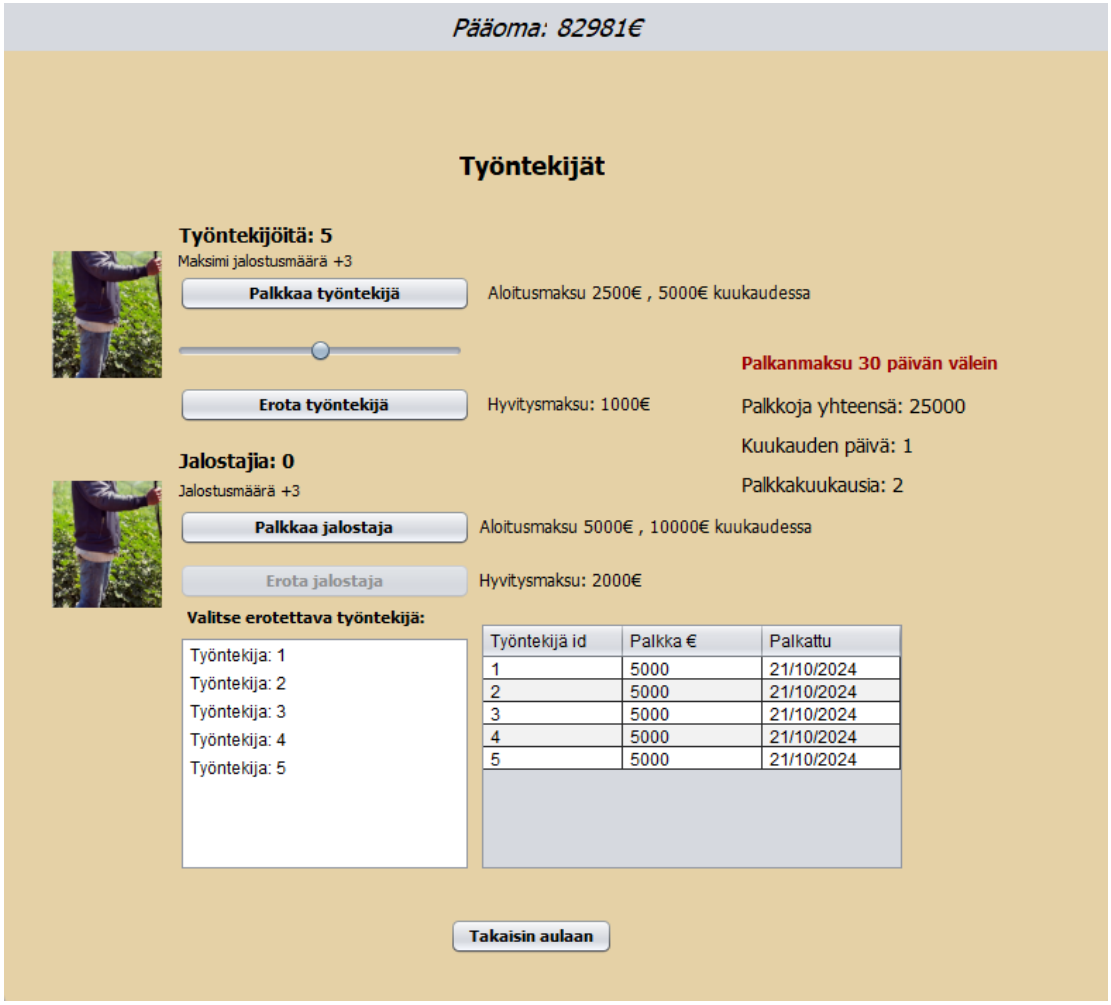
Käyttäjän suorittaessa toimintoja ohjelma tarkistaa jatkuvasti, että käyttäjän tekemiset ovat mahdollisia. Mikäli määritellyt säännöt eivät toteudu, käyttäjän pitää tehdä ohjeiden mukaisia asioita jatkaakseen toiminto loppuun.

3.5 Työntekijöiden luonti ja palkkausjärjestelmä ohjelmaan

Yritystä vastaavaa simulaatiota jäljiteltäessä on tärkeää, että henkilöstöä on mahdollista palkata. Työntekijöitä pitää pystyä palkata ja niistä pitää olla hyötyä yritykselle. Tässä ohjelmassa jokaisen työntekijän palkka on erikseen mahdollista valita tietyltä väliltä. Jokaisen työntekijän voi erikseen erottaa, jolloin kokonaispalkasta poistuu tietty summa.

3.5.1 Työntekijä taulu

Tässä kuvassa näkyy eroteltuna työntekijät. Työntekijöitä on mahdollista erikseen erottaa ja jokainen työntekijä on eriteltynä erikseen, myös työntekijän työsuhteen aloitusaika on merkattu ylös (KUVA 18).



KUVA 18. Työntekijäruutu ohjelmassa

Sivulla on myös mahdollista palkata ja erottaa jalostajia, mutta niiden tiedot ei siirry tauluun ja jokaiselle jalostajalle ei ole erikseen omakohtaisia tietoja ajan säästämiseksi. Kuukausikohtainen palkka ja jalostajien määrä on kuitenkin tiedossa.

3.5.2 Työntekijöiden palkkaus

Työntekijöitä voi palkata, mikäli käyttäjällä on siihen tarpeeksi varallisuutta. Palkatessaan työntekijän käyttäjä maksaa aloitusmaksun. Palkattuaan työntekijän käyttäjä näkee työntekijästä tietoja, kuten työntekijän palkan ja palkkaamisajan, sekä työntekijän eroteltuna ID-numerolla. Työntekijöiden kokonais- kuukausipalkka kasvaa työntekijän palkan verran.

Palkkaamalla työntekijöitä käyttäjä mahdollistaa suuremman tuotanto- ja jalostusmäärän, jonka avulla käyttäjä pystyy kasvattamaan yritystä nopeammin. Määritin työntekijöiden palkkaamisajan todellisen ajan perusteella (Koodi 2).

```
public void aika () {
DateTimeFormatter dtf = DateTimeFormatter.ofPat-
tern("dd/MM/yyyy");
LocalDateTime now = LocalDateTime.now();
       }
```

Koodi 2. Ajan määritys

Ajaksi olisi voinut helposti määrittää kuvitteellisen ajan, jolloin ohjelma toimisi täysin omassa ajassaan ja asioiden määritys olisi loogisempaa. Tämä ominaisuus on tulossa ohjelmaan jatkojalostuksessa opinnäytetyön ulkopuolella.

3.5.3 Työntekijöiden erotus

Palkattuaan työntekijän, käyttäjälle avautuu mahdollisuus erottaa haluamansa työntekijä. Käyttäjä voi erikseen valita, minkä työntekijän erottaa. Erottaessaan työntekijän, käyttäjän pitää maksaa hyvitysmaksu. Kokonais- palkka vähenee erotetun työntekijän palkan verran ja työntekijästä saadut hyödyt, kuten tuotanto- tai jalostusmäärä pienenee.

3.6 Pankki ohjelmassa

Pankin luominen alkoi miettimällä oikean elämän pankin tarjontaa. Pankkiin on luotu lainajärjestelmä, josta käyttäjä voi ottaa lainaa, mikäli omistaa vaaditun summan rahaa valmiiksi. Laina pitää maksaa

takaisin korkojen kera. Pankista saa myös otettua vakuutuksia, jolla voidaan ehkäistä ongelmia, kuten yllätysten tuomia vahinkoja.

3.6.1 Laina

Pankista on mahdollista ottaa lainaa, mutta siihen on oltava tarpeeksi varallisuutta (KUVA 19). Laina pitää maksaa takaisin tietyssä ajassa, jotta simulaatio voi jatkua ja yritys ei kaadu konkurssiin.

KUVA 19. Laina

Lainassa on aina takaisinmaksuaika, joka riippuu lainan summasta. Takaisinmaksuaika vähenee aina, kun päivä vaihtuu.

Pankki välilehdellä on laskuri, joka laskee päiviä maksuun. Päivien kuluessa takaisinmaksunaika vähenee ja viimein koittaa maksun aika, jatkaaksesi simulaatiota käyttäjän täytyy maksaa laina ja mahdolliset korkokulut. Osa koodista on määritelty alapuolella (Koodi 3).

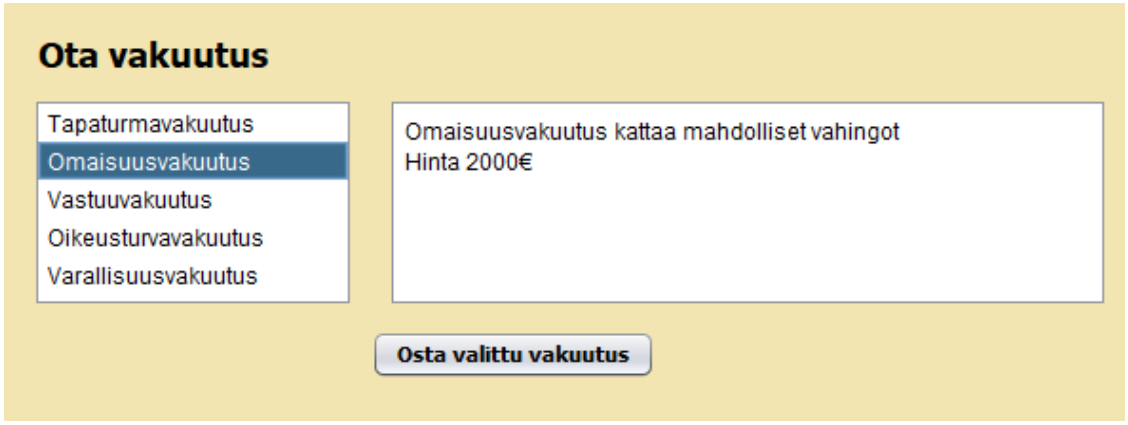
```
if (laina>0 && raha>=laina*1.1) {
    lainanotto1.setEnabled(true);
    lainanotto2.setEnabled(true);
    lainanotto5.setEnabled(true);
    seuraavapäivä.setEnabled(true);
    lainanmaksu.setEnabled(false);
    lainanmaksutarkistus = 0;
```

```
raha -= laina+(lainaprosentti*laina/100);
maksetutkorotyht =otetutlainatyht/10;
laina =laina - laina;
   lainalaskuri = 0;
```

Koodi 3. Lainan maksu

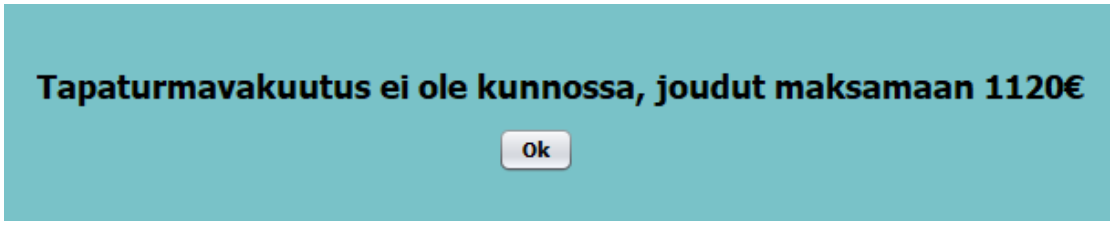
3.6.2 Vakuutuksen ostaminen pankista

Oikean elämän yritys voi ostaa itselleen vakuutuksia turvaamaan esimerkiksi henkilöiden tapaturmia (KUVA 20). Näin voi tehdä myös simulaatiossa. Vakuutukset maksavat kuukausitasolla tietyn summan ja vakuutuksien hinta muodostuu perustuen vakuutuksen kokoon.



KUVA 20. Vakuutusmahdollisuudet ohjelmassa

Käyttäjän ostaessa vakuutuksen turvataan mahdolliset kustannukset, jotka muuten tulisivat yllätyksistä (KUVA 21). Jos käyttäjä on esimerkiksi ostanut tapaturmavakuutuksen, ei työntekijöiden tapaturmista tarvitse enää korvata suuria summia, kuten aiemmin. Vakuutuksen hinta voidaan määritellä koodissa omistuksien mukaan.



KUVA 21. Tapaturmavakuutus ei ole kunnossa

3.7 Kierrätysjärjestelmän ohjelmassa

Tuotannosta ja yrityksen pyörittämisestä syntyy roskaa ja jätettä, vaikka kyseessä olisi pieni yritys. Kierrättäminen pitää ottaa huomioon nykypäivän yhteiskunnassa, joten kierrätys on luotuna mukaan simulaatioon (KUVA 22).

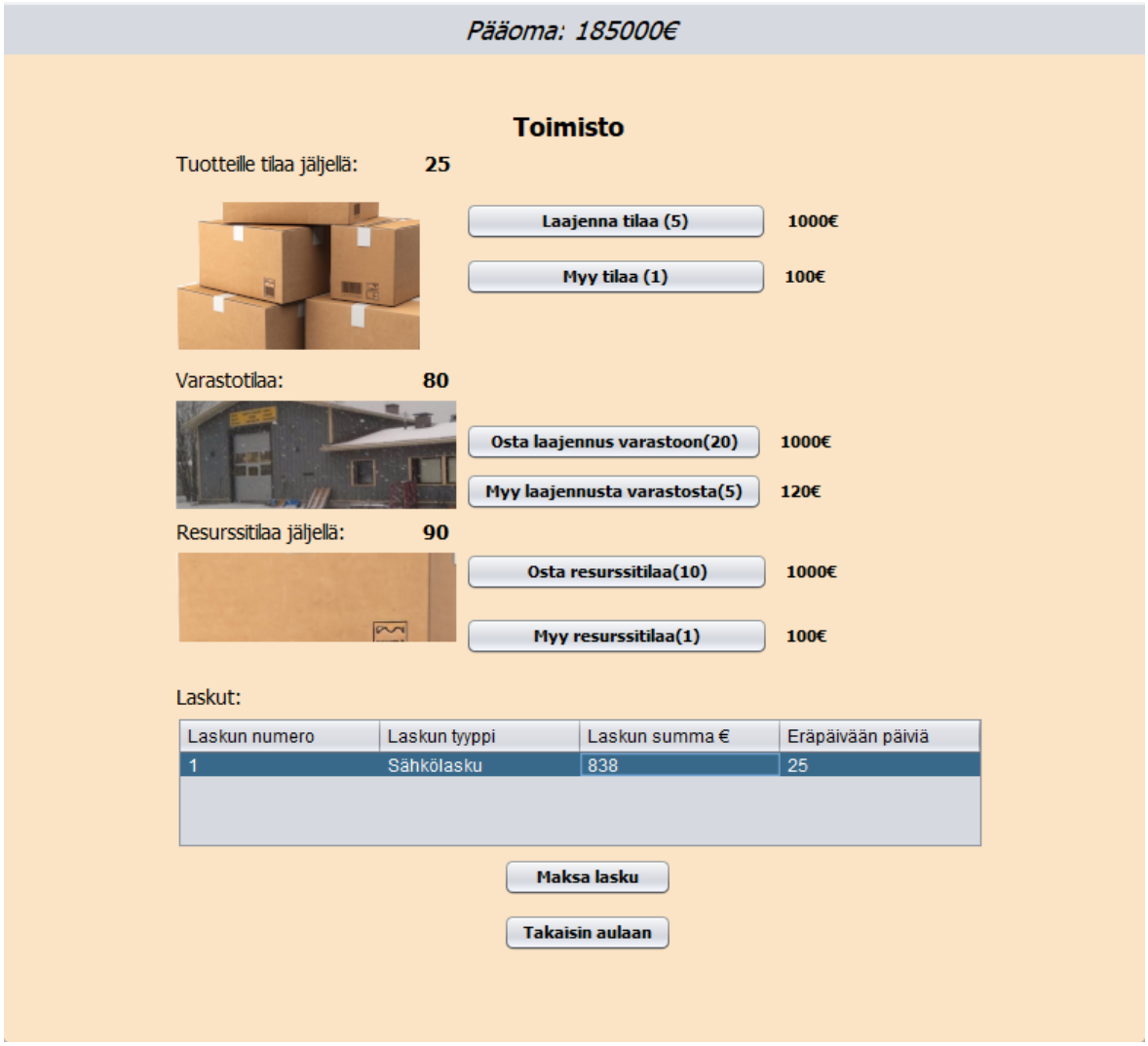
KUVA 22. Kierrätysruutu ohjelmassa

Kierrätysjärjestelmään saapuvat määrät on luotu keskimääräiseen oletukseen sen perusteella, minkä kokoinen yritys on. Järjestelmää on mahdollista kehittää esimerkiksi laittamalla eri jätteille omat nimensä.

Mikäli yrityksellä on jäänyt yli tavaraa, joka on mahdollista myydä, sen voi yrittää myydä. Ylijäämä tuotetta ei aina voi kuitenkaan myydä ja sen myyminen perustuu myös sattumaan. Ylijäämä tuotetta tarvitaan tietty määrä, että sen voi yrittää myydä eteenpäin.

3.8 Toimisto ohjelmassa

Toimistossa käyttäjä voi laajentaa yrityksen tiloja, jotta tuotantoa saadaan kasvatettua tarpeiden mukaisesti. Jokaisena kuukautena toimistoon tulee sähkölasku. Lasku ilmestyy toimistossa sijaitsevaan listaan (KUVA 23).



KUVA 23. Toimistoruutu ohjelmassa

Käyttäjä valitsee laskun, jonka haluaa maksaa. Lasku pitää maksaa eräpäivään mennessä. Lasku poistuu, kun se on maksettu. Pääomasta poistuu laskun summan suuruinen määrä.

3.9 Pörssikurssi ja kauppa ohjelmassa

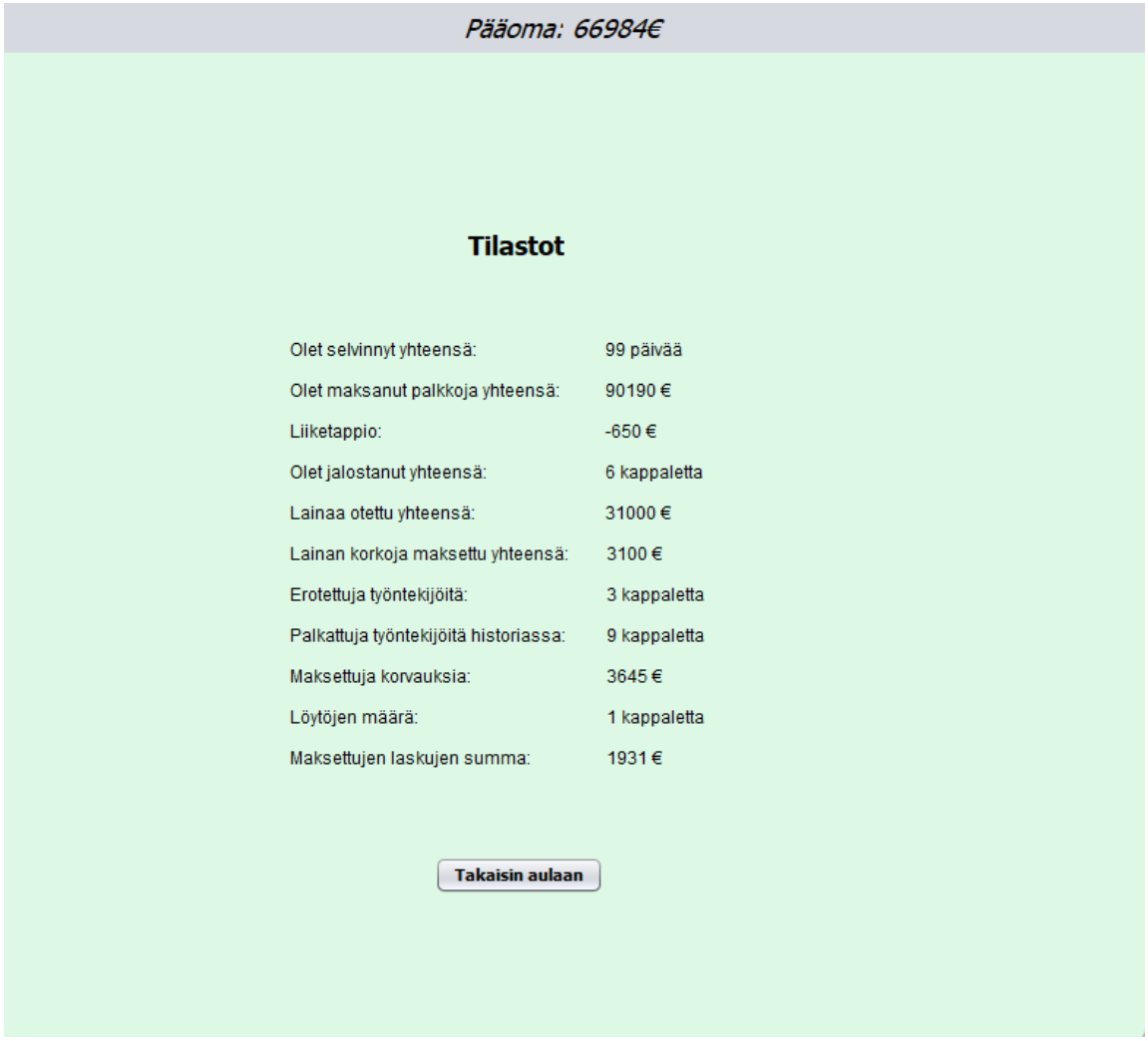
Tässä simulaatiossa materiaalien hinnat voivat vaihdella, eikä niiden perusteella pysty varmistamaan täydellistä tulosta. Hintoja on pyritty ohjaamaan kuvitteellisen kysynnän ja tarjonnan perusteella. Pörssikurssit vaihtelevat kuukauden välein ja satunnaiset tapahtumat voivat vaikuttaa kursseihin. Mikäli käyttäjä myy erittäin laajan määrän jotakin tuotetta, tuotteen hinta voi alkaa tippua. Tämä vaatisi kuitenkin erittäin pitkän simulaatioajan.

Myös jokainen kuukausi arpoo uudet tuotteiden hinnat vanhan hinnan perusteella. Hinta ei yleensä voi muuttua liian radikaalisti verrattuna vanhaan hintaan, koska se on säädetty erittäin harvinaiseksi.

KUVA 24. Pörssikurssi ohjelman kaupassa

4 TILASTOJEN SEURAAMINEN OHJELMASSA

Yritystä ylläpitäessä on mukava seurailla tilastoja ja yrityksen kehitystä. Tilastot on luotu sitä varten, että käyttäjä näkee sieltä tarvittavia tietoja (KUVA 25).



KUVA 25. Tilastoruutu ohjelmassa

Tilastojen taustalla on tehty aika paljon töitä, sillä jokainen toiminta on pitänyt miettiä erikseen. Tilastot toimivat, koska ohjelmoidut tiedot otetaan talteen välimuistiin ja ne säilötään tilastoihin. Erillinen tiedosto tai tietokanta on myös mahdollista tehdä tallennettavia tietoja varten, mutta sitä ei tällä hetkellä ole ajan säästämiseksi.

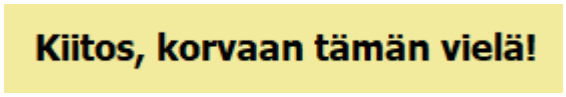
5 YLLÄTYKSELLISYYS OHJELMASSA

Simulaatiota tehdessä yksi tärkeimmistä asioista on muistaa luoda yllätyksellisyyttä, kuten oikeassa elämässä. Ohjelmassa yllätyksellisyys on pääasiassa toteutettu käyttäen satunnaisia generaattoreita, joiden lopputulosta on mahdotonta ennustaa. Esimerkkinä työntekijän kysyessä palkankorotusta lähtee liikkeelle vaihe, jossa tietokone arpoo satunnaisesti työntekijän vaatiman palkan (Koodi 4).

```
Random rand = new Random();
float kerto2 = (float)(10 + rand.nextInt(40))/1000;
Random randu = new Random();
työntekijanid = (1 + randu.nextInt(työntekijälaskuri));
palkankorotus = (int)(kerto2*(float) työntekijänpalkka);
maksakorotustyöntekijä = (int)(kerto2*(float) työntekijän-
palkka);
id =työntekijanid;
```

Koodi 4. Palkankorotuksen määritys

Käyttäjällä on mahdollisuus hyväksyä tai kieltäytyä palkankorotuksesta. Mikäli käyttäjä päättää olla hyväksymättä korkoa, siitä voi seurata mahdollisia ongelmia käyttäjälle. Käyttäjän hyväksyessä työntekijän vaatiman koron, työntekijä lupaa korvata hyvän teon (KUVA 26). Tämä on laitettu siksi, että käyttäjä saa tiedon mahdollisesta yllätyksestä ja hyvästä teosta.



KUVA 26. Työntekijän lupaus

Kieltäytyessään palkankorotuksesta taustalla alkaa pyöriä satunnainen laskenta erilaisista mahdollisuuksista ja palkankorotusta kysynyt työntekijä saattaa kostaa huonon kohtelunsa.

5.1 Löydöt

Halusin tehdä simulaatioon käyttäjän moraaliin perustuvia löytöjä, josta on yrityksen kasvun ja tulevaisuuden kannalta hyötyä. Löydöt ovat käytännössä kuvitteellisia, mutta joskus myös oikeassa elämässä hyvät teot tuottavat tulosta. Tällaisia löytöjä on mahdollista tutkia vasta, kun sellaisen on saanut.

Käyttäjän tehdessä hyviä tekoja, avautuu siis mahdollisuus saada löytöjä. Löydöt ovat sattumanvaraisia ja niiden avulla voi päästä ohjelmassa eteenpäin. Alla hiukan koodia, jossa arvotaan satunnainen lahjanumero (Koodi 5).

```
public void lahjasi() {
           if (lahja.size() < 10) {
            Random rand = new Random();
            float lahjalaskuri = 0;
            boolean run = true;
            while(run){
            System.out.println("loopin alku");
            lahjalaskuri = (float) (1 + rand.nextInt(10));
            run = checkGiftExists((int) lahjalaskuri);
            System.out.println(run);
            }
            salainenlahja = (int)(lahjalaskuri);
            lahja.add(salainenlahja);
            lahjaModel.addElement(salainenlahja +"");
            lahjalista.setModel(lahjaModel);
}
```

Koodi 5. Löydön määrityksiä

Löydetyt löydyt ilmestyvät erillisellä välilehdellä olevaan listaan, josta niitä voi tutkia. Löydöt eivät suoraan anna parempia mahdollisuuksia menestykseen, vaan ne pitää tutkia. Tutkiminen maksaa tietyn summan rahaa, mutta sen tehdessään käyttäjä saa uuden ominaisuuden käyttöönsä. Tutkitut löydöt voivat olla esimerkiksi koneita teollisuuteen, jonka avulla tuottavuus kasvaa.

5.2 Epäonniset tapahtumat

Epäonnisia tapahtumia lisättiin hyödyntäen satunnaisuutta. Myös moraalilla on vaikutuksia epäonnisiin tapahtumiin.

Käyttäjän kieltäytyessä palkankorotuksesta, lähtee liikkeelle satunnainen ryöstön mahdollisuus. Ryöstöä ei välttämättä kuitenkaan tapahdu. Ryöstön tapahtuessa käyttäjä menettää satunnaisen summan pääomastaan työntekijän tehdessä ryöstön. Ryöstöstä koodia alapuolella (Koodi 6).

```
public void yritaRyostoa()
{
  Random rand = new Random();
         if (tuhopäivälaskuri >= rand.nextInt(10) && pal-
         kankorotus1 == 2 &&etmaksanut==1){
```

```
            float kerroin = (float)(10 +
            rand.nextInt(40))/100;
            raha -= (int)(kerroin * (float)raha);
            etmaksanut=2;
            työntekijälaskuri-=1;
            tuhopäivälaskuri = 0;
            if(työntekijälaskuri > 0){
            palkankorotus1 = 1;
            työntekopäivät= 0;
        }
    }
```

Koodi 6. Ryöstön määritys

6 TOIMINNALLISUUDEN TOTEUTUMINEN OHJELMASSA

Työssä toteutettavassa simulaattorissa on useita toiminnallisuuksia, joiden pitää toimia keskenään oikein. Ohjelma on pyritty rakentamaan osissa, eli toimintoja on tehty yksitellen. Osat yhdistämällä rakennettiin toimiva kokonaisuus, jonka jälkeen toimintoja lisättiin pikkuhiljaa kasvattaen todellisuuden tuntua.

Useita toiminnallisuuksia ja osia on yhdistetty jälkikäteen yhdeksi kokonaisuudeksi. Esimerkiksi työntekijät lisättiin simulaation vasta tuotannon toiminnallisuuden varmistamisen jälkeen. Hyvänä esimerkkinä ovat ohjelmassa olleet listat, joita pystyi lisäämään jälkikäteen ohjelmaan (Koodi 7).

```
private ArrayList<Integer> todellasalainenlahja = new Ar-
rayList<>();
private ArrayList<Integer> lahja = new ArrayList<>();
private DefaultListModel<String> lahjaModel = new Default-
ListModel<>();
private ArrayList<Worker> tyontekijalista = new Ar-
rayList<>();
private DefaultListModel<String> tyontekijalistamodel = new
DefaultListModel<>();
```

Koodi 7. Listojen määrityksiä

Toiminnan varmistaminen ohjelmassa vaatii suuren määrän koodia ja säätämistä (KUVA 27). Ohjelman päätiedostossa on koodia yli 5000 riviä, jota voisi kuitenkin optimoida ja tiivistää pienempään tilaan.

```
5318        private javax.swing.JLabel velkaluku;
5319        private javax.swing.JLabel velkaluku2;
5320        private javax.swing.JButton älämaksa;
5321        // End of variables declaration
5322    }
5323
```

KUVA 27. Koodimäärä ohjelmassa

7 BUGIT JA NIIDEN KORJAAMINEN OHJELMASSA

Työtä tehdessä ylivoimaisesti suurimmaksi haasteeksi osoittautui bugien eli ohjelmointivirheiden korjaaminen. Toiminnallisuuksia ohjelmassa on sen verran paljon, että käyttäjä olisi pystynyt helposti löytämään bugeja ja virheitä simulaation aikana. Bugeja löytyy varmasti edelleen ja niitä on mahdollista korjata jatkokehityksessä.

Bugin eli ohjelmointivirheen pystyi havaitsemaan, jos jokin toiminnallisuus tai tapahtuma tapahtui, vaikka sen ei pitäisi olla mahdollista. Bugien korjaamisessa täytyi varmistaa jokaisen toiminnallisuuden kohdalla, että toiminto on todellisuudessa mahdollinen. Esimerkiksi jos käyttäjä yrittää ostaa tuotteita ja rahat ovat loppu, tuotetta voisi bugin takia ostaa ilman toiminnallisuuden varmistamista ja koko ohjelman idea olisi turha. Toinen hyvä esimerkki on muun muassa työntekijöiden erottaminen. Palkkalistoilla oleva työntekijä ja sen erottamiseen vaadittava hyvityssumma täytyy löytyä käyttäjältä pääomasta, ennen kuin erottaminen on mahdollista.

8 TULOKSET

Työn aikana minulla oli tarkoitus testata, voiko simulaattorin luoda 2D:nä järkevästi yksilötyönä. Työn tehtyäni voin todeta, että näin voi tehdä, eli sain kysymykseeni vastauksen. Todellisen tilanteen jäljittely onnistuu ainakin jossakin määrin ja tunteen oikeasta yrityksestä pystyy myös saavuttamaan. Ohjelma on myös mahdollista muuntaa oikean elämän yrityksen hallintatyökaluksi.

Ohjelma ja sen toteutus toimivat kohtuullisesti ja tässä mittakaavassa tuloksiin on päästy riittävin tavoitteisiin. Käyttäjä saa ohjelmasta vähintään karsitun kokemuksen yrityksen ylläpitämisestä. Ohjelmaa ei voi kuitenkaan pitää todellisuudessa opettavaisena, eikä sillä ei voi tässä vaiheessa opetella todellisen yrityksen pitämistä realistisesti. Huomattavasti paremman tuloksen saisi toteutettua hyödyntämällä 3D:tä, eli kolmiulotteista kuvaa, joka mahdollistaisi huomattavasti tarkemmat ja yksityiskohtaisemmat skenaariot ja todellisuuden tuntuman, eikä se välttämättä olisi sen vaikeampaa.

Omalta kohdaltani tuloksista voisin sanoa sen verran, että alkuinnostuksesta huolimatta toteutus ei lähtenyt halutun tavoitteiden mukaisesti liikkeelle, koska hyvää suunnitelmaa ei ollut tehty valmiiksi ja ohjelman toteuttaminen tuntui liian suurelta haasteelta. Ohjelman rakenteiden paloittelu ja laittaminen pienempiin palasiin saivat motivaation nousemaan ja tämän seurauksena ohjelma alkoi kehittyä todella nopeasti ja sen ominaisuudet lisääntyivät odotetulla tavalla.

Yhteenvetona totean, että ohjelman jatkokehitystä voisi jatkaa äärettömän pitkälle. Näitä kehityskohteita voisi olla muun muassa jalostuksen laajentaminen, erilaisten laskujen vastaanottaminen, verotuksen huomioon ottaminen ja monipuolisemmat laajentamismahdollisuudet. Ohjelman kehitys jatkuu opinnäytetyön jälkeen harrastustoimintana.

9 POHDINTA

Opinnäytetyön tarkoituksena oli tehdä mahdollisimman monipuolinen ohjelma, jossa pystyin käyttämään vanhoja taitojani hyödyksi. Simulaattorin lisäksi halusin samalla tehdä työstä ohjelman, jonka pystyisin tarvittaessa muuntamaan pienellä vaivalla mahdollisen oman yrityksen hallintaohjelmaksi. Ohjelma on tehty sillä periaatteella, että sitä voisi hiukan muuntamalla käyttää muuhunkin, kuin simulaatiotarkoitukseen.

Ohjelmaa varten minulle oli kehittynyt idea jo kauan ennen sen toteuttamista. En kuitenkaan todellisuudessa missään vaiheessa ole tehnyt riittävää etenemissuunnitelmaa, vaan ohjelma on tuotettu tuntemuksen, sekä mielessä olleen idean perusteella. Tiesin jo valmiiksi, että itse ohjelman toteutus ei tule olemaan ongelma laajan ohjelmointitaustani takia ja suurimmaksi ongelmaksi koituikin motivaation menetys. Projektin suuruuden kasvaminen ja jatkuvien uusien ideoiden keksiminen oli omiaan hidastamaan sen toteutusta. Hyvä ja tiukka suunnitelma olisi taannut ohjelman jatkuvan ja keskeytyksettömän toteutuksen, eli vähän on enemmän.

Ohjelman toteuttaminen vaati inspiraatiota ja erityisen suurta keskittymiskykyä ja inspiraationi katosi välillä hyvin pitkäksi aikaa ohjelmaa tehdessä. Omalta kohdaltani tiedän, että tällaista ohjelmaa ei kannata yksin lähteä tekemään motivaation säilyttämisen takaamiseksi, vaan tämän kaltainen työ kannattaisi suorittaa vaikkapa parityönä. Ohjelman tekeminen yksin ei sinänsä ole ongelma, vaan riski sen tekemättä jättämiseen on suuri, mikäli kohtaa haasteita matkalla monipuolisuuden vuoksi.

Opinnäytetyössä toteuttamani ohjelman tekeminen palautti mieleeni paljon vanhoja taitojani, joita on hyvä käyttää esimerkiksi työelämässä tai harrastustoiminnassa. Loppupäätelmänä on, että mikäli tulevaisuudessa perustan yrityksen, aion tehdä itse yritykselleni hallintaohjelman hyödyntäen opinnäytetyössä ja koulutuksessani saamaani taitoa ja inspiraatiota.

LÄHTEET

Alfame. 2018. API:t käytännössä – selkokielinen katsaus. Saatavissa: https://www.alfame.com/ajankohtaista/apit-kaytannossa-selkokielinen-katsaus. Viitattu 14.5.2024.

Apache. About Apache NetBeans. Saatavissa: https://netbeans.apache.org/front/main/about/. Viitattu 5.5.2024.

AWS. 2024. What is an IDE (Integrated Development Environment)? Saatavissa: https://aws.amazon.com/what-is/ide/. Viitattu 12.10.2024.

Bautamo. 2024. Rajapinta. Saatavissa: https://bautomo.com/sanastoa/rajapinta/. Viitattu: 18.10.2024

CAP. 2024. Simulaattoriopetus. Saatavissa: https://cap.fi/usein-kysyttya/ajosimulaattori/. Viitattu 20.10.2024.

Chakray. 2024. Programming Languages: Types and Features. Saatavissa: https://www.chakray.com/programming-languages-types-and-features/ Viitattu 19.10.2024.

Codecademy. 2024. What Is an IDE? Saatavissa: https://www.codecademy.com/article/what-is-an-ide. Viitattu 12.5.2024.

Darryl, K. T. 2019. NetBeans Java IDE becomes top-level Apache project. Saatavissa: https://www.theserverside.com/news/252462360/NetBeans-Java-IDE-becomes-top-level-Apache-project. Viitattu 20.10.2024.

Datascientist. 2024. Guide and Benefits of Procedural Programming for Developers. Saatavissa: https://datascientest.com/en/all-about-procedural-programming. Viitattu 18.10.2024.

ETHW. 2024. Howard Aiken. Saatavissa: https://ethw.org/Howard_Aiken. Viitattu 12.5.2024.

Finanssialalle. 2024. API-ohjelmistorajapinta. Saatavissa: https://www.finanssialalle.fi/opintomateriaalit/tulevaisuuden-finanssiala/tulevaisuuden-pankki/api-ohjelmistorajapinta.html. Viitattu 10.5.2024.

FutureLearn. 23.1.2024. What is programming? Computer programming structures and more. Saatavissa: https://www.futurelearn.com/info/blog/what-is-programming. Viitattu 21.10.2024.

Gooding, S. 2019. Apache NetBeans is Now a Top-Level Project of the Apache Software Foundation. Saatavissa: https://wptavern.com/apache-netbeans-is-now-a-top-level-project-of-the-apache-software-foundation. Viitattu 20.10.2024.

Haltu. 2023. Avoin lähdekoodi – mitä se on ja mitkä ovat sen edut ja haitat. Saatavissa: https://www.haltu.fi/blogi/avoin-lahdekoodi-mita-se-on#avoin-lahdekoodi. Viitattu 14.5.2024.

Hurrell, S. 2024. The eLearning benefits- and challenges – of simulation-based learning. Saatavissa: https://www.neovation.com/learn/90-simulation-based-learning-benefits-and-challenges-for-elearning. Viitattu 22.10.2024.

InetMedia. 2021. Matalan tason kielet. Saatavissa: https://inetmedia.fi/matalan-tason-kielet/. Viitattu 17.10.2024.

Jaber, S. 2024. 3 Generations of Simulation: The Evolution of Engineering Simulation Tools Saatavissa: https://www.simscale.com/blog/3-generations-of-simulation/. Viitattu 15.10.2024.

Javatpoint. Java Swing Tutorial. Saatavissa: https://www.javatpoint.com/java-swing. Viitattu 3.5.2024.

Kärkkäinen, M. 2018. Ada Lovelacen sivukommentista syntyi tietokone. AGRICOLA. Saatavissa: https://agricolaverkko.fi/tietosanomat/ada-lovelacen-sivukommentista-syntyi-tietokone/. Viitattu 12.5.2024.

Lenovo. 2024. What is a high-level language? Saatavissa: https://www.lenovo.com/us/en/glossary/high-level-language/. Viitattu 8.10.2024.

Pedamkar, P. 2024. Types of Computer Language. Saatavissa: https://www.educba.com/types-of-computer-language/. Viitattu 17.10.2024.

Project Lovelace. 2024. Ada Lovelace's Note G. Saatavissa: https://projectlovelace.net/problems/ada-lovelaces-note-g/. Viitattu 12.5.2024.

Räsänen, S. 2004. Verkko-opetuksen tietotekniikkaa - Simulaatio opetuksessa. Kuopion yliopisto, tietojenkäsittelytieteen laitos, PL 1627, 70211. Saatavissa: https://www.cs.uku.fi/tutkimus/publications/reports/B-2004-3.pdf Viitattu 14.5.2024.

SSH. 2024. About Simulation. Saatavissa: https://www.ssih.org/About-SSH/About-Simulation. Viitattu 14.10.2024.

TIM. 2024. Mitä ohjelmointi on? Ohjelmointi on ongelmanratkaisua. Saatavissa: https://tim.jyu.fi/view/kurssit/tie/ohj1/materiaali/mita-ohjelmointi-on#ohjelmointi-on-ongelmanratkaisua. Viitattu 12.5.2024.

Tieturi. 2024. Ohjelmointikielet. Saatavissa: https://www.tieturi.fi/koulutusala/ohjelmistokehitys/ohjelmointikielet/. Viitattu 13.5.2024.

Upson, M. 2023. How many Coding Languages Are There? Saatavissa: https://www.bestcolleges.com/bootcamps/guides/how-many-coding-languages-are-there/ Viitattu 21.10.2024

Zwolak, J. 2023. Ada Lovelace: The World's First Computer Programmer Who Predicted Artificial Intelligence. Blogi. Nist. Saatavissa: https://www.nist.gov/blogs/taking-measure/ada-lovelace-worlds-first-computer-programmer-who-predicted-artificial. Viitattu 19.10.2024.