Jari Axen, Jaakko Hyvärinen

Suunnittelumallit

Graafistenkäyttöliittymien ohjelmointi 2003

Seminaarityö

25.11.2003

1 SUUNNITTELUMALLIT

1.1 Johdanto

Tässä kirjoitelmassa esitellään suunnittelumallikäsite ja muutaman mallin käyttö Graafisten käyttöliittymien ohjelmointi 2003 kurssille tekemässämme harjoitustyössä. Kirjoitelma **referoi**[1] Gamman ja kumppanien Olio-ohjelmointi suunnittelumallit (Design Patterns Elements of Reusable Object-oriented Software) kirjassa esiteltyjä asioita. Malleista kertovat esimerkit on muokattu vastaamaan harjoitustyötämme. Suunnittelumalleja koskevia näkökulmia on myös otettu seuraavien henkilöiden teoksista. Rintala & Jokinen, Haikala & Märijärvi, Koskimies ja Budd.

Haikala & Märijärvi kertoo kirjassaan seuraavasti: Viime vuosina oliomenetelmiin liittyvässä keskustelussa ovat nousseet voimakkaasti esille suunnittelumallit (Design Patterns). Suunnittelumalleja käytettäessä suunnittelua pyritään jäsentämään yhtä luokkaa suurempina kokonaisuuksina, usean luokan muodostamina suunnittelumalleina. Suunnittelumallien idea perustuu havaintoon, jonka mukaan saman tapaiset ratkaisut luokkarakenteen ja luokkien yhteistyön suunnittelussa näyttävät toistuvan yhä uudelleen. Kokeneet suunnittelijat yleensä tuntevat nämä ”patenttiratkaisut” ja soveltavat niitä jopa huomaamattaan. Kun mallit nimetään ja dokumentoidaan vakiomuotoon, niistä tulee suunnittelijoiden yhteinen ”kieli” ohjelmiston suunnitteluun ja dokumentointiin.[Haikala & Märijärvi 2000, 344-345]

1.2 Mikä on suunnittelumalli

Gamma ja kumppanit määrittelevät suunnittelumallin suunnilleen seuraavasti: Suunnittelumalli on kuvaus keskenään vuorovaikutuksessa olevista olioista ja luokista, jotka on muotoiltu ratkaisemaan tietyssä yhteydessä esiintyvä yleinen suunnitteluongelma.

Suunnittelumalli nimeää, abstrahoi ja määrittää yleisen suunnittelurakenteen avainkohdat, jotka hyödyttävät uudelleenkäytettävän oliopohjaisen suunnitteluratkaisun luomista. Suunnittelumalli märittää osallistuvat luokat ja ilmentymät, niiden roolit ja yhteistyön sekä vastuiden jakamisen. Kukin suunnittelumalli keskittyy tiettyyn oliopohjaiseen suunnittelun ongelmaan tai kohteeseen. Malli kertoo missä tilanteessa se on sovellettavissa ja voidaanko sitä soveltaa muiden suunnitteluraajoitteiden yhteydessä sekä mitkä ovat mallin seuraukset ja sen hyvät ja huonot puolet. Suunnittelumallit esitettään ko. kirjassa ohjelmaesimerkein ja OMT-kuvauksin.

[1] Lupaa kysymättä

Suunnittelu malleissa on yleensä neljä keskeistä osaa.

1. Mallin nimi kuvaa muutamalla sanalla suunnittelun kohteena olevan ongelman, sen ratkaisun ja ratkaisun seuraamukset. Mallin nimeäminen rikastuttaa suunnittelusanastoa ja auttaa meitä suunnittelemaan korkeammalla abstraktiotasolla. Malleihin liittyvää sanastoa käyttämällä pystytään keskustelemaan malleista muiden suunnittelijoiden kanssa ja sisällyttämään mallien kuvauksia dokumentteihin. Sanasto helpottaa suunnitteluratkaisujen arviointia ja niiden hyvistä ja huonoista puolista keskustelemista.

2. Ongelma ja ympäristö, jossa ongelma esiintyy, kuvaa mallin soveltamiskohteita. Se voi kuvata erityisiä suunnitteluongelmia kuten esittää algoritmi oliona. Se voi tuoda esiin luokkia ja oliorakenteita, jotka ovat oireita joustamattomasta suunnittelusta. Joskus ongelma sisältää listan ehtoja, joiden on täytyttävä, jotta mallin soveltaminen olisi järkevää.

3. Ratkaisussa kuvataan elementit, josta suunnitteluratkaisu koostuu, niiden vastuut, niiden väliset suhteet ja niiden keskinäinen yhteistyö. Ratkaisu ei kuvaa mitään yksittäistä konkreettista toteutusta, sillä malli on ikään kuin runko, jota voidaan soveltaa usealla eri tavalla. Malli tarjoaa abstraktin kuvauksen ongelmasta ja siitä, millaisella luokkien ja olioiden yleisrakenteella se ratkaistaan.

4. Seuraukset kuvaavat mallin soveltamisen tuloksia ja sen hyötyjä sekä haittoja. Vaikka seurauksia ei yleensä erikseen mainita suunnitteluratkaisuja kuvattaessa, ne ovat oleellisia mallin hyvien ja huonojen puolien ymmärtämisessä ja vaihtoehtoisten ratkaisujen arvioimisessa. Koska uudelleenkäyttö on usein oliopohjaisen suunnittelun tavoite, seurauksissa selostetaan mallin käytön vaikutus ratkaisun muunneltavuuteen, laajennettavuuteen ja siirrettävyyteen.

Budd kirjoittaa kirjassaan An introduction to Object-Oriented Programming, malleista seuraavasti:

When faced with the task of solving a new problem, most people will consider first previous problems they have encountered that seem to have characteristics in common with the new assignment. There previous problems can be used as a model, and the new problem can be attacked in a similar fashion, making changes as necessary to fit the different circumstances.

This insight lies behind the idea of a software design pattern. A pattern is really nothing more than an attempt to document a proven solution to a problem so that future problems can be more easily handled in a similar fashion.[Budd 2002, 463]

1.3 Suunnittelumallien edut ja haitat

Rintala & Jokinen kirjoittaa kirjassaan Olioiden ohjelmointi c++:lla, seuraavasti suunnittelumallien eduista ja haitoista. Suunnittelumalliksi voitaisiin periaatteessa julistaa melkein mikä tahansa ”ei-triviiali” ohjelmakoodin tai suunnitelman osa. Tämä ei kuitenkaan ole tarkoitus. Yleisiksi suunnittelumalleiksi on tarkoitus kerätä käytännössä usein käytettyjä malliratkaisuja. Malleiksi valitaan ainoastaan sellaisia suunnittelumalleja, joille on löytynyt vähintään kaksi toisistaan riippumatonta käyttäjäkuntaa(ohjelmistotaloa tai –projektia). Tällaiset yleiset suunnittelumallit tulisi pikkuhiljaa saada osaksi jokaisen ohjelmistosuunnittelijan tietämystä -- tällä hetkellä on vain hyvin vaikea arvioida, mitkä mallit ohjelmistosuunnittelijan yleissivistykseen tulisi kuulua. [Rintala & Jokinen 2001,190-191]

Parhaimmillaan suunnittelumalli on abstrakti suunnitteludokumentti, joka voidaan ottaa käyttöön ohjelmiston suunnittelussa heti, kun suunnittelija tunnistaa ongelmasta kohdan, johon malli sopii. Yleistä tietämystä(osa ohjelmoijan yleissivistystä) olevien mallien lisäksi suunnittelumallit ovat erinomainen tapa kerätä talteen organisaatiossa olevaa sovellusaluekohtaista tietoa, joka näin säilyy, vaikka ihmiset vaihtuvatkin. [Rintala & Jokinen 2001,190-191]

Jotta suunnittelumalleista olisi hyötyä, ne tulisi dokumentoida huolellisesti ja niiden tulisi olla helposti omaksuttavissa. Dokumentointiin on määrämuotoisten dokumentointipohjien lisäksi kehitetty muodollisempia menetelmiä -- niin sanottuja mallikieliä(pattern language). Suunnittelumallit itsessään eivät auta mitään, elleivät niitä käyttävät suunnittelija tiedä, mitä mikin malli tarkoittaa. Nykyisin uusien suunnittelumallien löytäminen ja julkaiseminen vaikuttaa hieman riistäytyneen käsistä eikä kukaan yksittäinen ohjelmoija ehdi opetella kaikkia uusia malleja. Mallien muistamista ja opettelua haittaa myös usein niiden sidonnaisuus englannin kieleen -- vaatii erinomaista kielitaitoa saada oikea mielleyhtymä, jos suunnittelumallin nimenä on esimerkiksi Memento.[Rintala & Jokinen 2001,190-191]

1.4 Suunnittelumallien kuvaaminen

Graafiset esitystavat ja ohjelma koodi ovat tärkeitä ja hyödyllisiä kuvauksia. Ne eivät yksistään silti ole riittäviä, sillä ne ainoastaan tallentavat suunnittelu prosessin lopputuotteen, luokkien ja olion välisinä suhteina. Voidaksemme käyttää suunnitteluratkaisua uudelleen on meidän kirjattava ylös myös ratkaisuun johtaneet päätökset, suunnittelun aikana eteen tulleet vaihtoehdot, ja ratkaisun hyvät ja huonot puolet.

Gamma ja kumppanit esittelevät mallit käyttäen yhtenäistä formaattia. Jokainen malli jaetaan osiin seuraavan rungon mukaisesti. Runko luo tiedoille yhtenäisen rakenteen, joka helpottaa suunnittelumallien oppimista, ymmärtämistä ja käyttämistä.

1.4.1 Mallin nimi ja luokittelu

Mallin nimi kertoo olemuksen ytimekkäästi.

1.4.2 Tarkoitus

Lyhyt lause, joka kertoo mitä suunnittelumalli tekee ja mikä on sen perusajatus ja tarkoitus. Lisäksi kuvataan mihin tiettyyn suunnittelukohteeseen tai ongelmaan se soveltuu.

1.4.3 Alias

Mallin muita yleisesti tunnettuja nimiä, jos sellaisia on.

1.4.4 Perustelut

Esimerkkitilanne, jossa kuvataan suunnitteluongelma ja se, miten mallin luokka- ja oliorakenteet ongelman ratkaisevat. Esimerkki auttaa ymmärtämään jäljessä seuraavan abstraktimman kuvauksen.

1.4.5 Soveltuvuus

Mihin tilanteisiin suunnittelumallia voidaan soveltaa. Esimerkkejä huonoista suunnitteluratkaisuista, joita mallilla voidaan korjata, ja siitä, miten nämä tilanteet tunnistetaan.

1.4.6 Rakenne

Mallin luokkien graafinen esitys OMT-notaatiolla. Lisäksi vuorovaikutus suhteita on kuvattu sekvenssikaavioilla.

1.4.7 Osallistujat

Malliin liittyvät luokat ja/tai oliot sekä niiden vastuut.

1.4.8 Yhteistyösuhteet

Miten osallistujat toimivat yhteistyössä toteuttaakseen vastuunsa.

1.4.9 Seuraukset

Miten hyvin malli tukee tavoitteitaan? Mitä etuja mallin käytöllä saavutetaan ja mitä menetetään? Mitä systeemirakenteita malli sallii muutettavan riippumattomasti?

1.4.10 Toteutus

mitkä sudenkuopat, vinkit ja tekniikat on tiedettävä mallia toteuttaessa. Onko eri ohjelmointikielillä tehtyjen toteutusten välillä ohjelmointikielistä johtuvia eroja?

1.4.11 Mallikoodia

Koodiesimerkkejä mallin toteutuksesta C++:lla tai Smalltalkilla.

1.4.12 Tunnettuja käyttökohteita

Sisältää esimerkkejä mallin käytöstä todellisissa järjestelmissä. Annamme vähintään kaksi esimerkkiä, jotka ovat eri alueilta.

1.4.13 Läheiset mallit

Mitkä suunnittelumallit liittyvät läheisesti tähän malliin? Mitkä ovat tärkeimmät erot? Minkä muiden mallien kanssa tätä mallia tulisi käyttää?

1.5 Suunnittelumallien hakemisto

1.5.1 Abstrakti tehdas - Abstract Factory

Tuottaa rajapinna, jolla luodaan toisiinsa liittyvien olioiden muodostamia olioperheitä, määrittelemättä olioiden konkreettisia luokkia.

1.5.2 Ainokainen - Singleton

Varmistaa, että luokasta luodaan vain yksi ilmentymä, ja tarjoaa globaalin tavan päästä käsiksi tähän ilmentymään.

1.5.3 Edustaja - Proxy

Tuottaa olioille korvikkeen tai paikanpitäjän, joka kontrolloi olioon kohdistuvia pyyntöjä.

1.5.4 Hiutale - Flyweight

Mahdollistaa yhteiskäytön kautta runsaslukuisten hienojakoisten olioiden tehokkaan käytön.

1.5.5 Iteraattori - Iterator

Tarjoaa tavan, jolla kokoelmaolion alkiot saadaan läpikäytyä peräkkäisjärjestyksessä paljastamatta rakenteen sisäistä esitystapaa.

1.5.6 Julkisivu - Facade

Tarjoaa yhtenäisen rajapinnan alijärjestelmän rajapintojen joukolle. Julkisivu tarjoaa korkeamman tason rajapinnan, jonka kautta alijärjestelmää on helpompi käyttää.

1.5.7 Komento - Command

Kapseloi pyynnön olioksi. Tämän ratkaisun avulla voidaan asiakas parametroida lähettämään erilaisia pyyntöjä, laittaa pyynnöt jonoon, pitää niistä lokia ja peruuttaa operaatioita.

1.5.8 Kuorruttaja - Decorator

Lisää oliolle dynaamisesti uusia vastuita. Kuorruttaja tarjoaa joustavan vaihtoehdon perinnälle, kun on tarpeen laajentaa toiminnallisuutta.

1.5.9 Muisto - Memento

Rikkomatta kapselointia kokoaa ja ulkoistaa olion sisäisen tilan siten, että olio voidaan myöhemmin palauttaa tähän tilaan.

1.5.10 Operaatiorunko - Template Method

Määrittelee algoritmin rungon operaatiossa ja jättää jotkin sen osat aliluokkien toteuttavaksi. Operaatiorunko sallii aliluokkien uudelleenmäärittelyllä tietyt algoritmin kohdat, mutta algoritmin rakenne pysyy muuttumattomana.

1.5.11 Protyyppi – Prototype

Määrittelee prototyyppi-ilmentymää käyttämällä millainen olio luodaan, ja luo uusia olioita tätä prototyyppiä kopioimalla.

1.5.12 Rakentaja - Builder

Malli erottaa toisistaan monimutkaisen olion rakentamisessa käytetyn prosessin ja olion esitysmuodon, jolloin samalla rakentamisprosessilla voidaan tuottaa erilaisia esitysmuotoja.

1.5.13 Rekursiokooste – Composite

Malli esittää oliot rekursiivisesti koostettuna puurakenteena(part-whole hierarchy). Yksittäisiä olioita ja oliokoosteita voidaan käsitellä samalla tavalla.

1.5.14 Silta – Bridge

Erottaa rajapinnan toteutuksesta, jolloin kumpaakin voidaan muutella toisistaan riippumattomasti.

1.5.15 Sovitin – Adapter

Konvertoi luokan rajapinnan toiseksi rajapinnaksi, joka vastaa sovelluksen tarpeita. Sovittimen avulla saadaan sellaiset luokat, joilla on epäyhteensopivat rajapinnat, toimimaan yhdessä.

1.5.16 Strategia – Strategy

Määrittelee algoritmiperheen, kapseloi kunkin algoritmin ja tekee, niistä keskenään vaihdettavia. Algoritmia voidaan muuttaa muuttamatta sovellusta, joka sitä käyttää.

1.5.17 Tarkkailija – Observer

Määrittelee olioiden välille yksi moneen -riippuvuuden siten, että kun yhden olion tila muuttuu, siitä riippuvat oliot saavat ilmoituksen ja päivittyvät automaattisesti.

1.5.18 Tehdasmetodi – Factory Method

Määrittelee olion luontioperaation kutsumuodon, mutta jättää aliluokkien tehtäväksi päättää, mistä luokasta ilmentymä luodaan. Tehdasmetodia käyttämällä luokka voi siirtää ilmentymien luonnin aliluokille.

1.5.19 Tila – State

Malli saa olion muuttumaan käyttäytymistään, kun sen sisäinen tila muuttuu. Näyttää siltä kuin olio olisi vaihtanut luokkaansa.

1.5.20 Tulkki - Interprete

Määrittelee annetun kielen kieliopille esitysmuodon ja tulkin, joka käyttää esitysmuotoa kielen lauseiden tulkitsemiseen.

1.5.21 Vastuuketju – Chain of Responsibility

Pyynnön lähettäjän sitominen vastaanottajaan vältetään antamalla useammalle kuin yhdelle oliolle mahdollisuus pyynnön käsittelyyn. Vastaanottavista olioista muodostetaan ketju, ja pyyntöä siirretään ketjussa, kunnes joku olioista käsittelee sen.

1.5.22 Vierailija – Visitor

Edustaa operaatiota, joka suoritetaan oliorakenteen elementeille. Elementtien luokkia ei tarvitse muuttaa, kun luodaan uusi niihin kohdistuva operaatio.

1.5.23 Välittäjä – Mediator

Esittelee olion, johon kapseloidaan oliojoukon väliset vuorovaikutustavat. Välittäjä edistää löyhää sidontaa estämällä olioita viittaamasta suoraan toisiinsa ja mahdollistaa vuorovaikutuksen muuttamisen kaikkia yhteyksiä muuttamatta.

1.6 Mallien luokittelu

Gamma ja kumppanit luokittelevat mallit seuraavalla tavalla ”perheet”.(lihavoituna harjoitustyössä käytetyt mallit).

		TARKOITUS		
		Luonti	Rakenne	Käyttäytyminen
KOHDE	Luokka	Tehdasmetodi	Sovitin(luokka)	Tulkki Operaatiorunko
	Olio	Abstrakti tehdas **Rakentaja** Prototyyppi **Ainokainen**	Sovitin(olio) Silta Rekursiokooste Kuorruttaja Julkisivu Hiutale Edustaja	Vastuuketju Komento Iteraattori Välittäjä Muistio Tarkkailija Tila **Strategia** **Vierailija**

Suunnittelumallit luokitellaan oheisessa luokituksessa kahden kriteerin mukaan. Ensimmäinen kriteeri, tarkoitus, kuvaa mallin toimintaa. Malli kohdistuu joko luontiin, rakenteeseen tai käyttäytymiseen. Luontimallit käsittelevät olioiden luontiprosessia. Rakennemallit koskevat luokkien ja olioiden koosteita. Käyttäytymismallit käsittelevät tapoja, joilla luokat ja oliot ovat vuorovaikutuksessa jo joilla ne jakavat vastuita.

Toinen tärkeä kriteeri on kohde, joka kertoo liittyykö malli ensisijaisesti luokkiin vai olioihin. Luokkamallit käsittelevät luokkien ja niiden aliluokkien välisiä suhteita. Nämä suhteet kiinnitetään periytymisen kautta, joten ne ovat staattisia eli käännösaikaisia. Oliomallit käsittelevät olioiden välisiä suhteita, joita voidaan muuttaa ajonaikaisesti ja jotka ovat dynaamisempia. Melkein kaikki mallit käyttävät perintää jossain määrin. Niinpä luokkamalleiksi on luokiteltu vain ne mallit, jotka keskittyvät luokkien välisiin suhteisiin.

Luokkien luontimallit siirtävät osan olioiden luonnista aliluokille, kun taas olioihin liittyvät luontimallit siirtävät sen toiselle oliolle. Luokkiin liittyvät rakennemallit käyttävät perintää luokkakoosteiden muodostamiseen, kun taas olioihin liittyvät rakennemallit esittävät, miten olioita koostetaan. Luokkiin liittyvät käyttäytymismallit käyttävät perintää algoritmien ja kontrollin ohjauksen esittämiseen, kun taas olioihin liittyvät käyttäytymismallit määrittävät, miten olioiden joukko tekee yhteistyötä sellaisen tehtävän suorittamiseksi, jota yksi olio ei yksinään pystyisi tekemään.

Lisäksi Gamma ja kumppanit esittelevät muutaman muun tavan luokitella malleja. Jotain malleja käytetään usein yhdessä. Esimerkiksi Rekursiokoostetta ja Iteraattoria tai Vierailijaa käytetään usein yhdessä. Jotkut mallit taas ovat vaihtoehtoja: Prototyyppi on usein Abstraktin tehtaan vaihtoehto. Joidenkin mallien tuloksena on samanlainen ratkaisu, vaikka niiden tarkoitus on eri. Esimerkiksi Rekursiokoosteen ja Kuorruttajan rakennekaaviot ovat samanlaiset.

2 HARJOITUSTYÖSSÄMME KÄYTTÄMÄMME MALLIT

2.1 Strategia

2.1.1 Tarkoitus

Määrittelee algoritmiperheen, kapseloi kunkin algoritmin ja tekee niistä keskenään vaihdettavia. Algoritmia voidaan muuttaa muuttamatta sovellusta, joka sitä käyttää.

2.1.2 Perustelut

On olemassa useita erilaisia algoritmeja joilla voidaan pelilaudan geometria muodostaa. Kaikkien tällaisten algoritmien sisällyttäminen pelisovellukseen on huono ratkaisu useasta syystä:

- Pelilautakomponentista tulee erittäin monimutkainen, jos siihen sisällytetetään geometrian luonti algoritmi.
- Eri geometria algoritmit sopivat eri peleille esimerkiksi monopolipeliin olisi järjetöntä sisällyttää afrikantähtilaudan geometrian luomisalgoritmi. Pelilautakomponenttiin ei kannata sisällyttää kaikkia geometria algoritmeja, mikäli niitä ei käytetä.
- Mikäli geometria algoritmi liitetään kiinteästi pelilautaan on sen muuntaminen hankalaa. Monopolipelilauta esimerkiksi tarjoaa mahdollisuuden tuottaa vain monopolipelilautoja, eikä toimi yleisenä geometriasta vapaana pelilautana.

2.1.3 Soveltuvuus

Joukko toisiinsa liittyviä luokkia eroaa toisistaan vain käyttäytymiseltään. Strategian avulla luokkaan saadaan liitettyä yksi useasta tarjolla olevasta käyttäytymisestä.

Tarvitset algoritmista erilaisia variaatioita. Algoritmeilla voi esimerkiksi olla erilaisia muisti- ja suoritusaikavaatimuksia. Strategia-mallia voidaan käyttää, jos variaatiot toteutetaan algoritmeista muodostetussa luokkahierarkiassa.

Algoritmi käyttää tietoja, joista sovelluksen ei pidä tietää mitään. Käytä Strategia-mallia välttääksesi paljastamasta monimutkaista algoritmikohtaisia tietorakenteita.

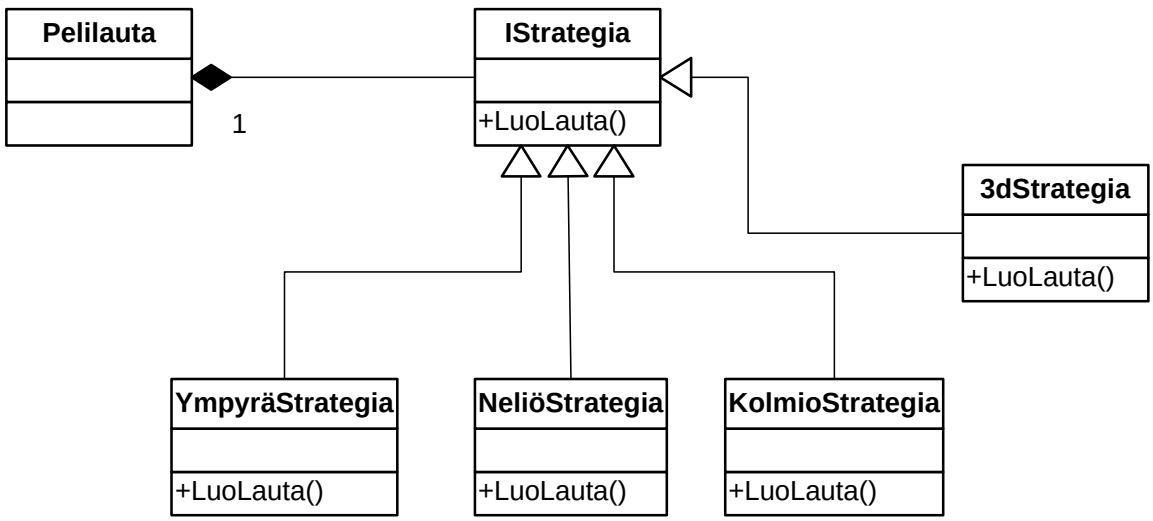
Luokka määrittelee erilaisia käyttäytymisiä ja käyttäytymiset ilmenevät sen operaatioissa useina ehtolausekkeina. Useiden ehtolausekkeiden käytön sijaan siirretään ehdolliset haarat omiin strategialuokkiinsa.

2.1.4 Osallistujat

- Strategy(Strategia,”geometria”)
    - Määrittelee kaikille algoritmeille yhteisen rajapinnan
- ConcreteStrategy(YmpyräStrategia,NelioStrategia)
    - Toteuttaa yksittäisen algoritmin strategy-rajapinnan mukaisesti
- Context(pelilauta)
    - Konfiguroidaan yhteen concretyStrategyOlion kanssa
        - Vie esimerkiksi rakentajan strategialle
    - Pitää yllä viitettä strategiaan

2.1.5 Seuraukset

- Strategiat tekevät ehtolauseet tarpeettomiksi
    - Strategy-malli tarjoaa vaihtoehdon ehtolausekkeiden käytölle halutun käyttäytymisen valinnassa. Jos yhteen luokkaan on kasattu erilaisia käyttäytymistapoja, on vaikea välttyä käyttämästä ehtolausekkeita oikean käyttäytymisen valitsemiseksi. Käyttäytymisen kapselointi erilaisiin Strategy-luokkiin tekee ehtolausekkeet tarpeettomiksi.
- Yhteen liittyvien algoritmien perheet
    - Strategy-luokkien hierarkia määrittelevät algoritmiperheen ja käyttäytymisjoukon, jota contexti voi käyttää.
- Vaihtoehto periytymisen käyttämiselle



2.2 Vierailija

2.2.1 Tarkoitus

Edustaa operaatiota, joka suoritetaan oliorakenteen elementeille. Elementtien luokkia ei tarvitse muuttaa, kun luodaan uusi niihin kohdistuva operaatio.

2.2.2 Perustelut

Vierailija-mallin avulla pystymme lisäämään laudan ruutuihin operaatioita koskematta itse ruutuihin. Riittää, että määrittelemme uusia Vierailija-olioita, näin saamme uusia operaatioita. Esimerkiksi jos haluaisimme ruudun näkyvän eri päätelaitteissa ja emme tiedä missä kaikissa haluaisimme sen näkyvän, niin Vierailijan avulla ei tarvitse koskea itse ruutuihin. Monesti on ongelmallista se, että useiden erilaisten operaatioiden sijoittaminen solmuluokkiin tekee järjestelmästä vaikeasti ymmärrettävän ja ylläpidettävän. Tämä ei välttämättä toteudu meidän ratkaisussa, koska kyseessä on kuitenkin pieni luokka. Ohjaajamme suositteli meitä käyttämään kyseistä mallia laudan ruutujen logiikan toteuttamisessa, mutta tämä osoittautui vääräksi menetelmäksi. Koska kumminkin teimme kyseisen mallin mukaiset ominaisuudet luokkiimme, päätimme jättää ne niihin, todellisen hyödyn jäädessä vähäiseksi.

2.2.3 Soveltuvuus

Käytä vierailija-mallia, kun oliorakenne sisältää useita luokkia, joilla on erilaiset rajapinnat, ja haluat suorittaa näihin olioihin kohdistuvia operaatioita, jotka riippuvat olioiden konkreettisista luokista.

Oliorakenteen olioille on suoritettava useita erillisiä ja toisiinsa liittymättömiä operaatioita ja olioluokkia ei haluta ”roskata” näillä operaatioilla. Vierailija-malli auttaa pitämään toisiinsa liittyvät operaatiot yhdessä kokoamalla ne yhteen luokkaan. Jos useat sovellukset jakavat oliorakennetta, käytä Vierailija-mallia sovelluskohtaisten operaatioiden niputtamiseen.

Oliorakenteen luokat muuttuvat harvoin, mutta uusia rakenteeseen kohdistuvia operaatioita lisätään usein. Oliorakenteen luokkien muuttaminen vaatii kaikkien vierailijoiden rajapintojen muuttamista, mikä voi olla kallista. Jos oliorakenteen luokat muuttuvat usein, on luultavasti parempi määritellä operaatiot noissa luokissa.

2.2.4 Osallistujat

- Visitor(TVisitor)
    - Määrittelee visit-operaation kullekin peliruudulle
- concreteVisitor(TPiirtoVisitor)
    - Toteuttaa jokaisen visitorluokan operaation
- Element(TRuutu)

- Määrittelee accept-operaation jolle vierailija on parametrina

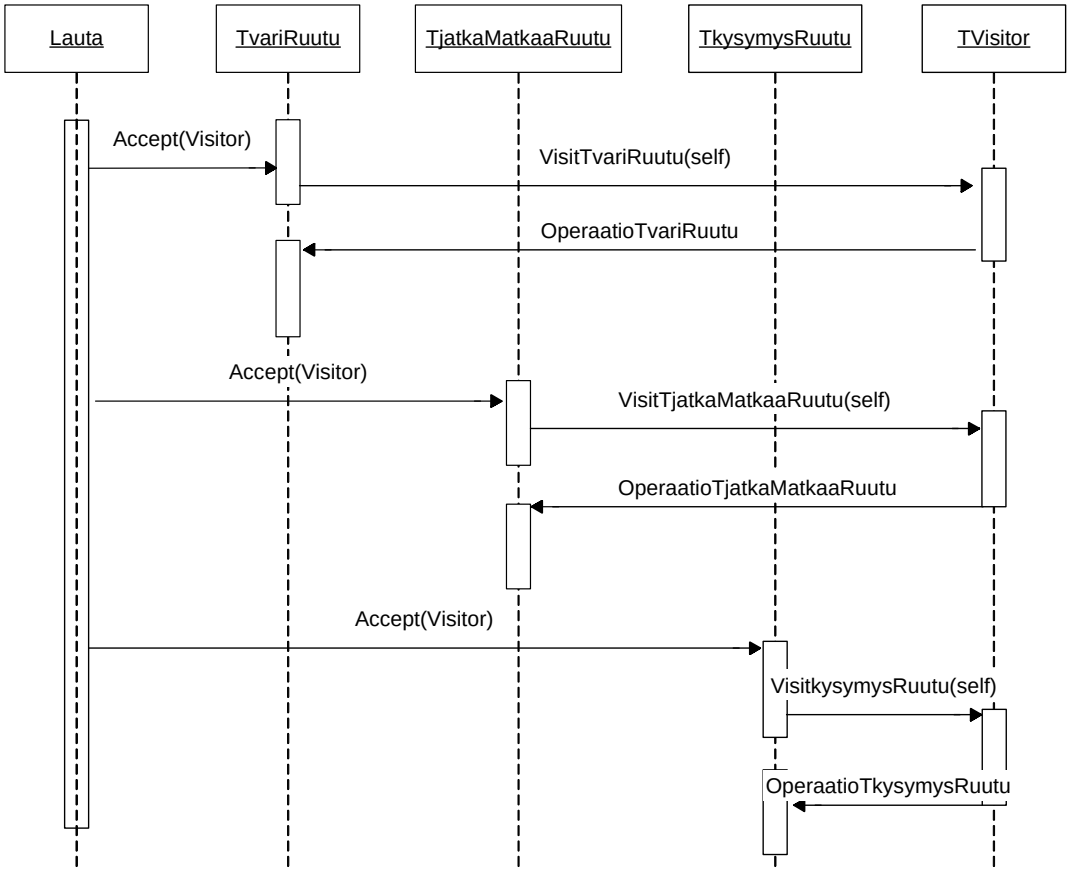
- ConcreteElement(esim.TVariRuutu)
  - Toteuttaa accept-operaation jolle vierailija on parametrina
- Objectstructure(TLauta)
  - Osaa luetella alkionsa(Ruudut)
  - Tarjoaa vierailijalle rajapinnan, jonka avulla vierailija voi vierailla alkioissa
  - Usein kooste(rekursiokooste) tai kokoelma esim. lista

2.2.5 Seuraukset

- Uusien operaatioiden lisääminen on helppoa.
- Vierailija kokoaan yhteen kuuluvat operaatiot ja eristää yhteen kuulumattomat operaatiot.

Uuden konkreetti elementti luokan lisääminen on vaikeaa.

- Vierailu luokkahierarkioiden yli mahdollista.



2.3 Ainokainen

2.3.1 Tarkoitus

Varmistaa, että luokasta luodaan vain yksi ilmentymä, ja tarjoaa globaalin tavan päästä käsiksi tähän ilmentymään.

2.3.2 Perustelut

Olio-ohjelmoinnissa on luokkia, joiden rooli on sellainen, että niistä voidaan luoda vain yksi ilmentymä. Pelissä esimerkiksi sekoittelun hoitava sotkija-luokka on tällainen. Sotkijaa käyttävät sekä kysymyskorttipakat, että itse kysymykset. Perinteisesti funktionaalisessa ohjelmoinnissa sotkija olisi funktio, mutta olio-ohjelmoinnissa emme käytä funktioita.

Miten varmistetaan, että luokalla on vain yksi ilmentymä ja että sovellukset pääsevät tähän ilmentymään helposti käsiksi? Olioon päästään käsiksi globaalin muuttujan kautta, mutta globaalin muuttuja käyttö ei yksin riitä estämään useampien ilmentymien luontia. Parempi ratkaisu on antaa olion itse pitää huolta siitä, että useampia ilmentymiä ei luoda. Ainokainen-malli toimii juuri näin.

2.3.3 Soveltuvuus

Käytä Ainokainen-mallia, kun

Luokalla täytyy olla täsmälleen yksi ilmentymä ja sen täytyy olla kaikkien sovellusten saatavilla.

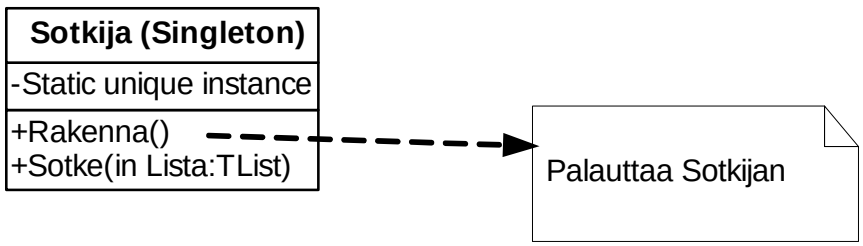
Ilmentymää on voitava laajentaa aliluokkien avulla ja sovellusten on kyettävä käyttämään laajennettua ilmentymää ilman, että niiden koodia pitää muuttaa.

2.3.4 Osallistujat

- Singelton(Sotkija) määrittelee instance-operaation(rakenna), jolla sovellukset pääsevät käsiksi sen uniikkiin ilmentymään. Instance on luokkaoperaatio.

2.3.5 Seuraukset

- Ainoan ilmentymän käyttöä kontrolloidaan
- Nimiavaruus pienenee
- Operaatioita ja esitysmuotoa voidaan jalostaa
- Ilmentymien lukumäärää voidaan vaihdella
- Ratkaisu on joustavampi kuin luokkaoperaation käyttö



2.4 Rakentaja

2.4.1 Tarkoitus

Malli erottaa toisistaan monimutkaisen olion rakentamisessa käytetyn prosessin ja olion esitysmuodon, jolloin samalla rakentamisprosessilla voidaan tuottaa erilaisia esitysmuotoja.

2.4.2 Perustelut

Saadaksemme pelilautakomponentista yleisen pelilautakomponentin täytyi pelilaudan ruutujen luominen erottaa itse pelilaudasta. Strategia(”geometria”) ohjaa pelilaudan ruutujen rakentamiseen ohjelmoitua rakentajaa. Erilaisia peliruutuja on niin paljon, kun on erilaisia pelejä. Joten pelilautaan on pystyttävä helposti määrittelemään erilaisten pelien ruutujen luominen. Ongelma ratkaistaan siten, että määritellään strategia, joka luo pelilaudan ruudut rakentajan avulla. Erilaiset rakentajat erikoistuvat eri peleihin esimerkiksi triviaali rakentaja on erikoistunut trivialpursuitin ruutujen rakentamiseen, kun taas monopolirakentaja on erikoistunut monopolin ruutujen rakentamiseen.

2.4.3 Soveltuvuus

Käytä Builder-mallia, kun

Haluat pitää erillään algoritmin, jolla monimutkainen osista koostuva olio luodaan, ja osat, joista olio koostuu sekä tavan, jolla osat yhdistetään.

Rakentamisprosessin on pystyttävä tuottamaan esitysmuodoltaan erilaisia olioita.

2.4.4 Osallistujat

- Builder(Rakentaja) määrittelee abstactin rajapinnan jolla luodaan tuotteen(pelilaudan) osat.

- ConcreteBuilder(TriviaaliRakentaja) Toteuttaa builder rajapinnan ja käyttää rajapintaa tuotteen rakentamiseen osista kokoamalla. Pitää kirjaa tuotteen rakenteen edistymisestä rakentamisen kuluesssa(kerää peliruudut listaan). Toteuttaa tuotteen hakuoperaation(AnnaRuudut).

- Director(Strategia) Kokoaa laudan käyttäen builder rajapintaa.

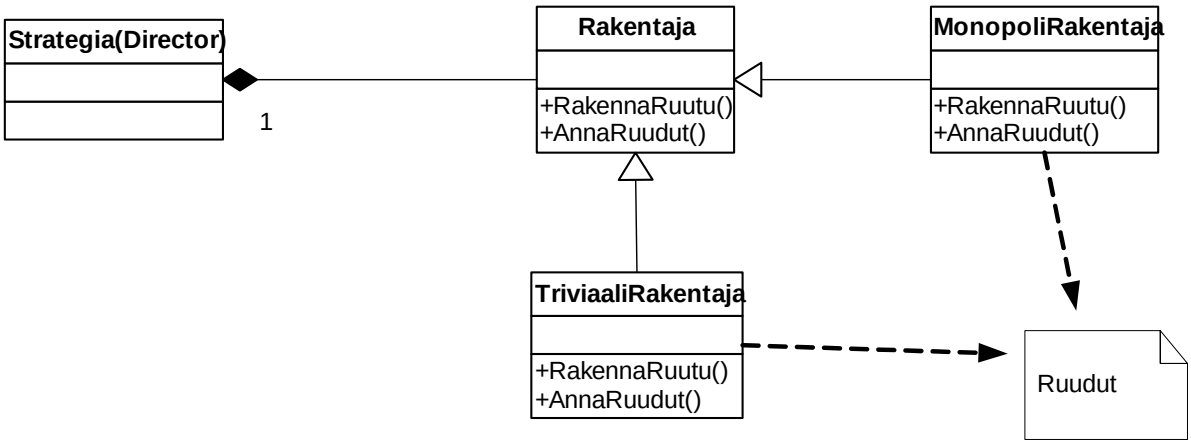
- Product(Pelilauta) Kuvaa rakentamisen kohteena olevan tuotteen. ConcreteBuilder rakentaa tuotteen sisäisen esitys muodon ja määrittelee prosessin, jonka mukaisesti kokoaminen tehdään.

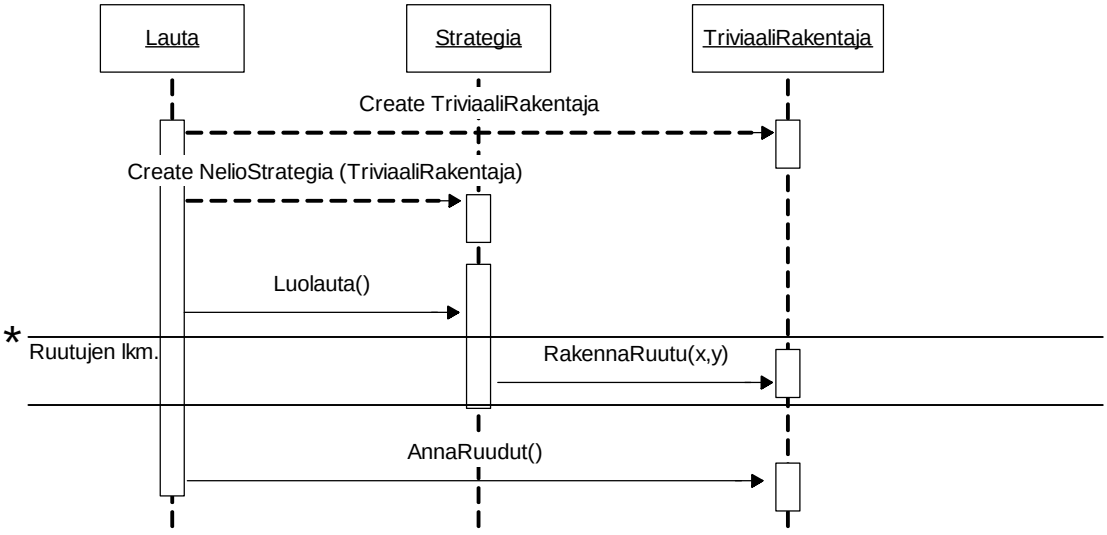
2.4.5 Seuraukset

- Mahdollistaa tuotteen(pelilaudan) sisäisen esitysmuodon varioinnin.

- Eristää toisistaan tuotteen rakentamiseen ja esittämiseen käytetyn koodin.

- Tarjoaa hyvän kontrollin rakentamisprosessille.





3 LOPPUSANAT

Koskimiehen sanoihin on hyvä lopettaa.

Siihen, mikä voidaan katsoa suunnittelumalliksi, vaikuttaa myös valittu kieli. Oletamme tässä, että kieli on oliokieli. Mikäli esimerkiksi perinteistä proseduraalista kieltä käytettäisiin toteutusvälineenä, saattaisivat periytyminen ja dynaaminen sidonta olla suunnittelumalleja: tällöin ohjelmoija joutuisi ne tarvittaessa toteuttamaan itse kyseisen mallin mukaisesti. Myöskin mallin kuvauksessa joudutaan tekemään jotain oletuksia toteutuskielen ominaisuuksista. Periaatteessa suunnittelumallit pyritään kuitenkin antamaan mahdollisimman yleisinä sitoutumatta tiettyyn kieleen. [Koskimies 2001,246]

On syytä korostaa, että suunnittelumalli on sellaisenaan varsin löyhä käsite. Itse asiassa koko tietojenkäsittelytiede on suurelta osalta erilaisten yleisesti esiintyvien mallien identifiointia ja abstrahointia. Tämä tulee erityisen selvästi näkyviin ohjelmointikielten kehityksessä: havaittaessa ohjelmissa usein esiintyvä muoto pyritään kielen tasoa nostamaan antamalle tälle muodolle oma nimi ja kielellinen rakenne. Näin ovat aikanaan kehittyneet vaikkapa ohjausrakenteet ja tietotyypit; joitakin kymmeniä vuosia sitten näitäkin olisi voitu kutsua suunnittelumalleiksi. Kiintoisa kysymys on se, kuinka monet nykyisistä suunnittelumalleista löytävä tiensä tulevaisuuden oliokieliin. [Koskimies 2001,246]

Olennaista suunnittelumalleissa ei ole itse suunnittelumallin käsite, vaan kaikki se tietotaito ja suunnittelukokemus, joka on saatu kerättyä tämän otsakkeen alle. Tärkeätä on myös, että tämä informaatio esitetään mahdollisimman selkeässä, systemaattisessa ja käytännönläheisessä muodossa ja että suunnittelumalleilla on yleisesti tunnetut, kuvaavat nimet. Näin suunnittelumallit tarjoavat käytännölliseen ohjelmistokehitykseen valmiin, organisoidun kokoelman ratkaisuja, jotka sopivat tiettyihin yleisiin ongelmiin. Suunnittelumalleja ei pidä kuitenkaan väkisin yrittää sovittaa ohjelmiinsa: niillä ei ole arvoa muussa mielessä kuin tiettyjen, hyvin tarkasti spesifioitujen ongelmien ratkaisuina. Ei siis voi sanoa, että jos ohjelmistossa on sovellettu runsaasti suunnittelumalleja, se olisi ilman muuta laadukas. Suunnittelumallin soveltamisen tulee aina lähteä tietyn ongelman identifioimisesta. [Koskimies 2001,246]

4 LÄHTEET

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, ”Olio-ohjelmointi suunnittelumallit”, IT Press, 2001.

Kai Koskimies, ”Oliokirja”, Satku, 2001

Matti Rintala, Jyke Jokinen, ”Olioiden Ohjelmointi C++:lla”. Satku, 2001

Ilkka Haikala, Jukka Märijärvi, ”Ohjelmistotuotanto”, Satku, 2000

Timothy A. Budd, ” An Introduction to Object Oriented Programming”, Addison-Wesley, 2002

5 LIITTEET (OHJELMAKOODIT)

```
unit strategia;

interface

uses rakentaja,ruutu,Classes,types;

type IStrategia= interface
    procedure LuoLauta;
    function AnnaLista:TList;
    procedure AsetaKoko(IClientrect: Trect);
end;

implementation

end.


unit trivaalistrategia;
//laudan geometrian muodastaminen neliöksi

interface

uses
  SysUtils, Classes, strategia, ruutu, rakentaja,types;

type
  TNelioStrategia = class(TComponent,iStrategia)
private
    FRuutujaSivulla:integer;
    FKoko:integer;
    aloitus:TTriviaaliRuutu;
    ruudut:TList;
    FRakentaja :IRakentaja;
    procedure SetRakentaja(const Value: IRakentaja);
public
  constructor Create (AOwner: Tcomponent); override;
  procedure LuoLauta;
  function AnnaAloitus:TRuutu;
  function AnnaLista:TList;
  procedure AsetaKoko(IClientrect: Trect);
published
  property Rakentaja: IRakentaja read FRakentaja write SetRakentaja;
  property Ruudunkoko: integer read FKoko write FKoko default 60;
  property Ruutujasivulla: integer read FRuutujaSivulla write FRuutujasivulla
default 0;
end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Vaivaiset', [TNelioStrategia]);
end;



function TNelioStrategia.AnnaAloitus: TRuutu;
begin
  result:=aloitus;
end;

function TNelioStrategia.AnnaLista: TList;
begin
 result := nil;
```

```
 if not assigned (Rakentaja) then exit;
 ruudut:= Rakentaja.AnnaRuudut;
 result :=ruudut;
end;

procedure TNelioStrategia.AsetaKoko(IClientrect: Trect);
begin

end;

constructor TNelioStrategia.create(AOwner: Tcomponent);
begin
  inherited create(AOwner);
  ruutujaSivulla:=0;
  ruudut:=TList.Create;
  ruudunkoko:=60;
end;

procedure TNelioStrategia.LuoLauta;
var i:integer;
begin
  if not assigned (Rakentaja) then exit;
  for i:=0 to RuutujaSivulla do begin
    Rakentaja.RakennaRuutu(ruudunkoko*i,0);
   end;
  for i:=1 to RuutujaSivulla-1 do begin
   Rakentaja.RakennaRuutu(ruudunkoko*RuutujaSivulla,Ruudunkoko*i);
  end;
  for i:=0 to RuutujaSivulla do begin
   Rakentaja.RakennaRuutu(Ruudunkoko*Ruutujasivulla-Ruudunkoko*i,
Ruudunkoko*RuutujaSivulla);
  end;
  for i:=1 to RuutujaSivulla-1 do begin
    Rakentaja.RakennaRuutu(0,Ruudunkoko*RuutujaSivulla-Ruudunkoko*i);
  end;
end;
procedure TNelioStrategia.SetRakentaja(const Value: IRakentaja);
begin
  FRakentaja := Value;
end;

end.



unit YmpyraStrategia;
//laudan geometrian muodastaminen ympyräksi

interface

uses
  SysUtils, Classes, strategia, ruutu, rakentaja,m3d, Graphics, Controls, Forms,
 Dialogs, StdCtrls,
  Windows, Messages,lauta;

type
  TYmpyraStrategia = class(TComponent,iStrategia)
  private
    FRuutujaSivulla:integer;
    FKoko:integer;
    FClientrect:Trect;
    aloitus:TRuutu;
    ruudut:TList;
    FRakentaja :IRakentaja;
    A:cTMatrix;
    DRect:cDRect;
    procedure SetRakentaja(const Value: IRakentaja);
```

```
    procedure Scale;
  public
    constructor Create (AOwner: TComponent); override;
    procedure LuoLauta;
    function AnnaAloitus:TRuutu;
    function AnnaLista:TList;
    procedure AsetaKoko(IClientrect: Trect);
  published
    property ClientRect: TRect read FClientRect write FClientRect;
    property Rakentaja: IRakentaja read FRakentaja write SetRakentaja;
    property Ruudunkoko: integer read FKoko write FKoko default 60;
    property Ruutujasivulla: integer read FRuutujaSivulla write FRuutujasivulla
default 0;
end;

procedure Register;

implementation
 uses math;

procedure Register;
begin
  RegisterComponents('Vaivaiset', [TYmpyraStrategia]);
end;

function TYmpyraStrategia.AnnaAloitus: TRuutu;
begin
  result:=aloitus;
end;

function TYmpyraStrategia.AnnaLista: TList;
begin
 ruudut:= rakentaja.AnnaRuudut;
 result :=ruudut;
end;

constructor TYmpyraStrategia.Create(AOwner: Tcomponent);
begin
  inherited create(AOwner);
  RuutujaSivulla:=0;
  ruudut:=TList.Create;
  Ruudunkoko:=60;
  A := cTMatrix.Create;
  dRect := cDRect.Create;
  dRect.p1.SetPoint(-10,-10);
  dRect.p2.SetPoint(10,10);
end;
procedure TYmpyraStrategia.Scale;
begin
  A.Scale(dRect,clientRect);
end;

procedure TYmpyraStrategia.LuoLauta;
var x:double;
    apu:TPoint;
    sade:double;
    askellusvali:integer;
begin
  Scale;
  sade:=round((2*PI*((clientRect.BottomRight.X-clientRect.TopLeft.X)/2)));
  askellusvali:=round(sade/ruudunkoko);
  x := 0 ;
  while ( x < 360 ) do begin
    apu:= A.Map(cos(x*PI/180)*8,sin(x*PI/180)*8);
    rakentaja.rakennaRuutu(apu.X,apu.Y);
    x := x + 360/askellusvali*1.5;
  end;
```

```
end;
procedure TYmpyraStrategia.SetRakentaja(const Value: IRakentaja);
begin
  FRakentaja := Value;
end;
procedure TYmpyraStrategia.AsetaKoko(IClientrect: Trect);
begin
 ClientRect:=IClientrect;
end;

end.


unit Kolmiostrategia;
//laudan geometrian muodastaminen kolmioksi


interface

uses
  SysUtils, Classes, strategia, ruutu, rakentaja,types;

type
  TKolmioStrategia = class(TComponent,iStrategia)
private
    FRuutujaSivulla:integer;
    FKoko:integer;
    aloitus:TRuutu;
    ruudut:TList;
    FRakentaja :IRakentaja;
    procedure setRakentaja(const Value: IRakentaja);
public
  constructor create (AOwner: Tcomponent); override;
  procedure LuoLauta;
  function AnnaAloitus:TRuutu;
  function AnnaLista:TList;
  procedure AsetaKoko(IClientrect: Trect);
published
  property Rakentaja: IRakentaja read FRakentaja write SetRakentaja;
  property Ruudunkoko: integer read FKoko write FKoko default 60;
  property Ruutujasivulla: integer read FRuutujaSivulla write FRuutujasivulla
default 0;
end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Vaivaiset', [TKolmioStrategia]);
end;




function TKolmioStrategia.AnnaAloitus: TRuutu;
begin
  result:=aloitus;
end;

function TKolmioStrategia.AnnaLista: TList;
begin
 ruudut:= Rakentaja.AnnaRuudut;
 result :=ruudut;
end;
```

```
procedure TKolmioStrategia.AsetaKoko(IClientrect: Trect);
begin

end;

constructor TKolmioStrategia.Create(AOwner: Tcomponent);
begin
  inherited Create(AOwner);
  RuutujaSivulla:=0;
  Ruudut:=TList.Create;
  Ruudunkoko:=60;
end;

procedure TKolmioStrategia.LuoLauta;
var i:integer;
begin
 Rakentaja.Rakennaruutu(0,0);
 i:=1;
 Rakentaja.Rakennaruutu(Ruudunkoko*i,0);

   for i:=2 to RuutujaSivulla do begin
    Rakentaja.Rakennaruutu(Ruudunkoko*i,0);
   end;
  for i:=1 to round((RuutujaSivulla-1)/2) do begin
   Rakentaja.Rakennaruutu(Ruudunkoko*RuutujaSivulla-Ruudunkoko*i,Ruudunkoko*i);
  end;
  for i:=round((RuutujaSivulla)/2) downto 1 do begin
   Rakentaja.Rakennaruutu(Ruudunkoko*round((RuutujaSivulla)/2)-Ruudunkoko*i,
Ruudunkoko*round((RuutujaSivulla)/2)-Ruudunkoko*i);
 end;
end;
procedure TKolmioStrategia.SetRakentaja(const Value: IRakentaja);
begin
  FRakentaja := Value;
end;

end.


-------------------------------------------------------------

unit ruutu;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls;


type
  TRuutu = class(TImage)
  private
    FX:integer;
    FY:integer;
    FYlos:TRuutu;
    FAlas:TRuutu;
    FVasen:TRuutu;
    FOikea:TRuutu;
  public
    constructor Create(ix:integer;iy:integer); virtual;
    procedure AcceptVisitor(visitori:TObject); virtual; abstract;
  published
    property Vasen:TRuutu read FVasen write FVasen;
    property Ylos:TRuutu read FYlos write FYlos;
    property Alas:TRuutu read FAlas write FAlas;
    property Oikea:TRuutu read FOikea write FOikea;
```

```
    property X:integer read FX;
    property Y:integer read FY;

end;

TTriviaaliRuutu =class(TRuutu)
  private
    FVilkkuvaKuva :TBitmap;
    FAlkuperainenKuva :TBitmap;
    aika:TTimer;
    FSaavilkkua:Boolean;
    tagi:boolean;
  public
    constructor Create(ix:integer;iy:integer); virtual;
    procedure Vilkuta(Sender: TObject);
  published
    property VilkkuvaKuva:Tbitmap read FVilkkuvaKuva write FVilkkuvaKuva;
    property Saavilkkua:Boolean read FSaaVilkkua write FSaaVilkkua;
    property AlkuperainenKuva:Tbitmap read FAlkuperainenKuva write
FAlkuperainenKuva;

end;


TJatkaMatkaaRuutu = class(TTriviaaliRuutu)
  public
   constructor Create(ix:integer;iy:integer);  override;
   procedure AcceptVisitor(visitori: TObject); override;

end;

TKysymysRuutu = class(TTriviaaliRuutu)
  private

  public
    tyyppi:integer;
    constructor Create(ix:integer;iy:integer;ityyppi:integer); virtual;
    procedure AcceptVisitor(visitori: TObject); override;

end;

TVariRuutu = class(TKysymysRuutu)
  private

  public
    constructor Create(ix:integer;iy:integer;ityyppi:integer); override;
    procedure AcceptVisitor(visitori: TObject); override;
end;



implementation

uses visitor;


{ TRuutu }

constructor TRuutu.Create(ix:integer;iy:integer);
begin
inherited Create(nil);
FX:=ix;
FY:=iy;
end;
```

```
procedure TJatkaMatkaaRuutu.AcceptVisitor(visitori: TObject);
begin
  if(visitori is TVisitor) then
  (visitori as TVisitor).VisitTJatkaMatkaaRuutu(self);
end;
procedure TKysymysRuutu.AcceptVisitor(visitori: TObject);
begin
  if(visitori is TVisitor) then
  (visitori as TVisitor).VisitTKysymysRuutu(self);
end;
procedure TVariRuutu.AcceptVisitor(visitori: TObject);
begin
  if(visitori is TVisitor) then
  (visitori as TVisitor).VisitTVariRuutu(self);
end;



constructor TTriviaaliRuutu.Create(ix, iy: integer);
begin
inherited create(ix,iy);
FVilkkuvakuva:=TBitmap.Create;
FAlkuperainenKuva :=TBitmap.Create;
aika := TTimer.Create(self);
aika.Interval:=500;
aika.OnTimer := vilkuta;
end;


procedure TTriviaaliRuutu.Vilkuta(Sender: TObject);
begin
  if not SaaVilkkua then exit;
  if(tagi=true) then begin
      picture.Bitmap := FVilkkuvaKuva;
      tagi:= false;
    end
  else
    begin
      picture.Bitmap := fAlkuperainenKuva;
      tagi:=true;
    end;
end;

{ TVariRuutu }

constructor TVariRuutu.Create(ix:integer;iy:integer;ityyppi:integer);
begin
  inherited Create(ix,iy,ityyppi);
  left:=ix;
  top:=iy;
end;

{ TKysymysRuutu }

constructor TKysymysRuutu.Create(ix:integer;iy:integer;ityyppi:integer);
begin
  inherited Create(ix,iy);
  tyyppi:=ityyppi;
  left:=ix;
  top:=iy;
end;

{ TJatkaMatkaaRuutu }

constructor TJatkaMatkaaRuutu.Create(ix:integer;iy:integer);
begin
inherited Create(ix,iy);
```

```
 left:=ix;
 top:=iy;
end;

end.


unit visitor;

//Vierailija

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Ruutu;

type
TVisitor = class
  public
   procedure VisitTJatkaMatkaaRuutu(apu:TJatkaMatkaaRuutu);virtual; abstract;
   procedure VisitTVariRuutu(apu:TVariRuutu);virtual; abstract;
   procedure VisitTKysymysRuutu(apu:TKysymysRuutu); virtual; abstract;
   procedure VisitTRuutu(apu:TRuutu);virtual; abstract;
   procedure VisitTTriviaaliRuutu(apu:TTriviaaliRuutu);virtual; abstract;


end;

TPiirtoVisitor = class(TVisitor)
  private
    kanvas:TCanvas;
    koko:integer;
    id:integer;
  public
   constructor Create(apu:TCanvas;ikoko:integer);
   procedure VisitTJatkaMatkaaRuutu(apu:TJatkaMatkaaRuutu);  override;
   procedure VisitTVariRuutu(apu:TVariRuutu);  override;
   procedure VisitTKysymysRuutu(apu:TKysymysRuutu); override;
   procedure VisitTRuutu(apu:TRuutu);  override;
   procedure VisitTTriviaaliRuutu(apu:TTriviaaliRuutu);  override;
end;

implementation

{ TVisitor }

constructor TPiirtoVisitor.Create(apu:TCanvas;ikoko:integer);
begin
 kanvas:=apu;
 koko:=ikoko;
 id:=0;
end;

procedure TPiirtoVisitor.VisitTRuutu(apu: TRuutu);
begin
end;

procedure TPiirtoVisitor.VisitTJatkaMatkaaRuutu(apu: TJatkaMatkaaRuutu);
begin
  kanvas.Rectangle(apu.x,apu.y,apu.x+koko,apu.y+koko);
  kanvas.TextOut(apu.x,apu.y,'-'+IntToStr(id));
  id:=id+1;

end;

procedure TPiirtoVisitor.VisitTVariRuutu(apu: TVariRuutu);
```

```
begin
  kanvas.Rectangle(apu.x,apu.y,apu.x+koko,apu.y+koko);
  kanvas.TextOut(apu.x,apu.y,IntToStr(id)+' Vari '+IntToStr(apu.tyyppi));
  id:=id+1;
end;

procedure TPiirtoVisitor.VisitTKysymysRuutu(apu: TKysymysRuutu);
begin
  kanvas.Rectangle(apu.x,apu.y,apu.x+koko,apu.y+koko);
  kanvas.TextOut(apu.x,apu.y,IntToStr(id)+' '+IntToStr(apu.tyyppi));
  id:=id+1;
end;

procedure TPiirtoVisitor.VisitTTriviaaliRuutu(apu: TTriviaaliRuutu);
begin

end;

end.



------------------------------------------------------------

unit sotkija;

//sekoittaa tietorakenteen

interface

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;

type

  Tsotkija = class
  private
    tietorakenne : TList;
    function AnnaObject(i: integer):TObject;
    procedure LisaaLoppuun(Objecti: TObject);
  protected
    constructor Create;
  public
    procedure Sotke (itietorakenne: TList);
    class function Rakenna:TSotkija;

  end;

  implementation

var Globaalisotkija:Tsotkija;  //ei delphi tuo luokkamuuttujia, siksi tälläinen
ratkaisu

function Tsotkija.AnnaObject(i: integer): TObject;
var apu:TObject;
begin
  apu:=tietorakenne[i];
  tietorakenne.Remove(apu);
  result := apu;
end;

constructor Tsotkija.Create;
begin
  Globaalisotkija := self;
end;
```

```
procedure Tsotkija.LisaaLoppuun(Objecti: TObject);
begin
  tietorakenne.Insert(tietorakenne.Count,Objecti);
end;

class function Tsotkija.Rakenna: TSotkija;
begin
   if not assigned(Globaalisotkija) then self.create;
   result := Globaalisotkija;
end;

procedure Tsotkija.Sotke(itietorakenne: TList);
var i:integer;
begin
   tietorakenne:=itietorakenne;
  for i:= tietorakenne.count downto 1 do begin
    LisaaLoppuun(Annaobject(random(i)));
  end;

end;

end.

-------------------------------------------------------------

unit rakentaja;

interface

uses Classes;

type

Irakentaja = interface
['{E067D502-94EE-4885-B2FC-D7521569D0FD}']
   function AnnaRuudut:TList;
   procedure RakennaRuutu(x:integer;y:integer);
end;

implementation

end.


unit triviialiRakentaja;

//rakentaa triviaaliruutuja strategian ohjaamana

interface

uses
SysUtils, Classes, ruutu, rakentaja, lauta;

type
  TRakennusMetodi = procedure(x:integer;y:integer) of object; //osoitin taulukko
   //että voidaan helposti lisätä
  TTriviialiRakentaja = class(TComponent,Irakentaja)
  private
    FRuutujenKuvat: TStringlist;
    FKysymysTyyppienLukumaara:integer;
    kysymysCounter:integer;   //hoitaa kysymysruutujen syklin
    variRuutuCounter:integer; //hoitaa väriruutujen syklin laudalle(nappula)
    variCounter:integer; //hoitaa kaikkien ruutujen syklin
    ruudut:TList;
    FMesta:TLauta;
    RakennusMetodit : array of TRakennusMetodi;
    procedure RakennaJatkaMatkaaRuutu(x:integer;y:integer);
```

```
    procedure RakennaVariRuutu(x:integer;y:integer);
    procedure RakennaKysymysRuutu(x:integer;y:integer);
    procedure Linkita(kohdalla:TTriviaaliRuutu);
 public
    function AnnaRuudut:TList;
    constructor Create (AOwner:TComponent); override;
    procedure RakennaRuutu(x:integer;y:integer);
   destructor Destroy; override;
  published
   property RuutujenKuvat: TStringlist read FRuutujenKuvat write FRuutujenKuvat;
   property KysymysTyyppienLukumaara : integer read fkysymysTyyppienLukumaara
write fkysymysTyyppienLukumaara default 4;
   property Mesta:TLauta read FMesta write fMesta;
 end;

procedure Register;

implementation


procedure Register;
begin
  RegisterComponents('Vaivaiset', [TTriviialiRakentaja]);
end;


{ TTriviialiRakentaja }

function TTriviialiRakentaja.AnnaRuudut: TList;
var i:integer;
begin
  i:=ruudut.Count;
  TTriviaaliRuutu(ruudut.Items[0]).alas:=TTriviaaliRuutu(ruudut.Items[i-1]);
  TTriviaaliRuutu(ruudut.Items[i-1]).ylos:=TTriviaaliRuutu(ruudut.Items[0]);
  result:=ruudut;
end;

constructor TTriviialiRakentaja.Create(AOwner: TComponent);
var i:integer;
begin
  inherited Create(AOwner);
  if(AOwner is TLauta) then mesta := TLauta(AOwner);

  fRuutujenKuvat:=TStringlist.Create;
  RuutujenKuvat.Add('./vihrea');      //oletusarvot
  RuutujenKuvat.Add('./sininen');
  RuutujenKuvat.Add('./punainen');
  RuutujenKuvat.Add('./keltainen');
  RuutujenKuvat.Add('./harmaa');

  kysymysTyyppienLukumaara:=4;
  setlength(RakennusMetodit,kysymysTyyppienLukumaara+2);

  for i:=0 to kysymysTyyppienLukumaara do  begin
  RakennusMetodit[i] := rakennaKysymysRuutu;
  end;
  i:=kysymysTyyppienLukumaara;
  RakennusMetodit[i] := RakennaVariRuutu;
  RakennusMetodit[i+1] := RakennaJatkaMatkaaRuutu;
  ruudut:=TList.Create;

end;

destructor TTriviialiRakentaja.Destroy;
begin
  fruutujenkuvat.Free;
  ruudut.Free;
```

```
  inherited Destroy;
end;

procedure TTriviialiRakentaja.Linkita(kohdalla: TTriviaaliRuutu);
var ruutuja:integer;
begin
  ruutuja:=ruudut.Count;
  if(ruutuja = 0) then exit;
  ruutuja:=ruutuja-1;
  TTriviaaliRuutu(ruudut.Items[ruutuja]).ylos := kohdalla;
  kohdalla.alas:=TTriviaaliRuutu(ruudut.Items[ruutuja]);
end;

procedure TTriviialiRakentaja.RakennaJatkaMatkaaRuutu(x, y: integer);
var  apuRuutu:TTriviaaliRuutu;
begin
  apuRuutu:=TJatkaMatkaaRuutu.Create(x,y);
  Linkita(apuruutu);
  ruudut.Add(apuRuutu);
  if assigned(Mesta) then apuruutu.Parent := mesta;
  if not assigned (Ruutujenkuvat) then exit;
  apuruutu.AlkuperainenKuva.
LoadFromFile(ruutujenkuvat[kysymystyyppienlukumaara]+'.bmp');
  apuruutu.VilkkuvaKuva.LoadFromFile(ruutujenkuvat[kysymystyyppienlukumaara]+'v.
bmp');
  apuruutu.Picture.Bitmap := apuruutu.AlkuperainenKuva;
end;

procedure TTriviialiRakentaja.RakennaKysymysRuutu(x, y: integer);
var  apuRuutu:TTriviaaliRuutu;
begin
  apuRuutu:=TKysymysRuutu.Create(x,y,kysymysCounter);
  Linkita(apuruutu);
  ruudut.Add(apuRuutu);
  if assigned(Mesta) then apuruutu.Parent := mesta;
  if not assigned (Ruutujenkuvat) then exit;
  apuruutu.AlkuperainenKuva.LoadFromFile(RuutujenKuvat[kysymysCounter]+'.bmp');
  apuruutu.VilkkuvaKuva.LoadFromFile(RuutujenKuvat[kysymysCounter]+'v.bmp');
  apuruutu.Picture.Bitmap := apuruutu.alkuperainenKuva;
  kysymysCounter:=kysymysCounter+1;
  if(kysymysCounter=kysymysTyyppienLukumaara) then kysymysCounter := 0;
end;

procedure TTriviialiRakentaja.RakennaRuutu(x, y: integer);
begin
  RakennusMetodit[variCounter](x,y);  //rakentaa tietyn ruudun
  inc(variCounter);
  if(variCounter = kysymysTyyppienLukumaara+2) then //jos mennään loppuun, niin
0
  variCounter :=0;
end;

procedure TTriviialiRakentaja.RakennaVariRuutu(x, y: integer);
var  apuRuutu:TTriviaaliRuutu;
begin
  apuRuutu:= TVariRuutu.Create(x,y,variRuutuCounter);
  Linkita(apuruutu);
  ruudut.Add(apuRuutu);
  if assigned(Mesta) then apuruutu.Parent := mesta;
  if not assigned (Ruutujenkuvat) then exit;

apuruutu.AlkuperainenKuva.LoadFromFile(RuutujenKuvat[variRuutuCounter]+'2.bmp');
  apuruutu.Vilkkuvakuva.LoadFromFile(RuutujenKuvat[variRuutuCounter]+'2v.bmp');
  apuruutu.Picture.Bitmap := apuruutu.alkuperainenKuva;
  variRuutuCounter:=variRuutuCounter+1;
  if(variRuutuCounter=kysymysTyyppienLukumaara) then variRuutuCounter :=0;
end;
```

```
end.
```